
**Industrial automation systems and
integration — COLLADA™ digital
asset schema specification for 3D
visualization of industrial data**

*Systèmes d'automatisation industrielle et intégration —
Spécifications du schéma des actifs numériques COLLADA™ pour la
visualisation 3D des données industrielles*



STANDARDSISO.COM : Click to view the full PDF of ISO 17506:2022



COPYRIGHT PROTECTED DOCUMENT

© ISO 2022

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword	xi
Introduction	xii
1 Scope	1
2 Normative references	1
3 Terms and definitions	2
3.1 Terms and definitions	2
3.2 Abbreviated terms	4
4 Notational conventions	5
4.1 Typographic conventions and notation	5
4.2 Notation and organization in the reference sections	5
4.2.1 Child element conventions	5
4.2.2 Child element order	6
5 COLLADA format	7
5.1 Design considerations	7
5.1.1 Introduction	7
5.1.2 Assumptions and dependencies	7
5.1.3 Goals and guidelines	7
5.2 Tool requirements and options	11
5.2.1 Introduction	11
5.2.2 Exporters	11
5.2.3 Importers	14
5.2.4 Archive packaging	15
5.3 Schema concepts	15
5.3.1 Introduction	15
5.3.2 XML overview	15
5.3.3 Address syntax	16
5.3.4 Instantiation and external referencing	19
5.3.5 The Common profile	21
5.3.6 Common profile elements	21
5.3.7 Example and discussion on techniques	22
5.3.8 Common glossary	22
5.4 Programming guideline	24
5.4.1 General	24
5.4.2 Parameters in COLLADA	24
5.4.3 Curve interpolation	25
5.4.4 Skin deformation (or skinning) in COLLADA	31
5.5 Core elements reference	33
5.5.1 General	33
5.5.2 Elements by category	34
5.5.3 accessor	38
5.5.4 ambient	44

5.5.5	animation	45
5.5.6	animation_clip	47
5.5.7	asset	49
5.5.8	bool_array	52
5.5.9	camera	53
5.5.10	channel	55
5.5.11	COLLADA	56
5.5.12	color	58
5.5.13	contributor	58
5.5.14	controller	59
5.5.15	control_vertices	61
5.5.16	directional	63
5.5.17	evaluate_scene	64
5.5.18	extra	65
5.5.19	float_array	67
5.5.20	formula	68
5.5.21	geographic_location	70
5.5.22	geometry	72
5.5.23	IDREF_array	74
5.5.24	imager	75
5.5.25	input (shared)	77
5.5.26	input (unshared)	79
5.5.27	instance_animation	81
5.5.28	instance_camera	82
5.5.29	instance_controller	83
5.5.30	instance_formula	85
5.5.31	instance_geometry	87
5.5.32	instance_light	88
5.5.33	instance_node	90
5.5.34	instance_visual_scene	92
5.5.35	int_array	93
5.5.36	joints	94
5.5.37	library_animation_clips	95
5.5.38	library_animations	96
5.5.39	library_cameras	97
5.5.40	library_controllers	98
5.5.41	library_formulas	100
5.5.42	library_geometries	101
5.5.43	library_lights	102
5.5.44	library_nodes	103
5.5.45	library_visual_scenes	104
5.5.46	light	105
5.5.47	lines	107
5.5.48	linestrips	108

5.5.49	lookat.....	110
5.5.50	matrix.....	112
5.5.51	mesh	113
5.5.52	morph.....	115
5.5.53	Name_array.....	117
5.5.54	newparam	119
5.5.55	node	121
5.5.56	optics	123
5.5.57	orthographic.....	125
5.5.58	param (data flow)	126
5.5.59	param (reference).....	127
5.5.60	perspective	130
5.5.61	point.....	131
5.5.62	polygons.....	133
5.5.63	polylist.....	136
5.5.64	rotate.....	138
5.5.65	sampler.....	139
5.5.66	scale	146
5.5.67	scene.....	147
5.5.68	setparam	149
5.5.69	SIDREF_array.....	150
5.5.70	skeleton.....	151
5.5.71	skew.....	153
5.5.72	skin.....	154
5.5.73	source (Core)	158
5.5.74	spline	160
5.5.75	spot.....	161
5.5.76	targets	163
5.5.77	technique (Core).....	164
5.5.78	technique_common	165
5.5.79	Token_array.....	166
5.5.80	translate.....	167
5.5.81	triangles	168
5.5.82	trifans.....	170
5.5.83	tristrips.....	172
5.5.84	vertex_weights	173
5.5.85	vertices	175
5.5.86	visual_scene.....	176
5.6	Physics Reference	178
5.6.1	General.....	178
5.6.2	Elements by category	178
5.6.3	attachment.....	181
5.6.4	box.....	182
5.6.5	capsule.....	183

5.6.6	convex_mesh	184
5.6.7	cylinder.....	186
5.6.8	force_field	187
5.6.9	instance_force_field	188
5.6.10	instance_physics_material.....	189
5.6.11	instance_physics_model	190
5.6.12	instance_physics_scene.....	192
5.6.13	instance_rigid_body.....	193
5.6.14	instance_rigid_constraint.....	196
5.6.15	library_force_fields	198
5.6.16	library_physics_materials.....	199
5.6.17	library_physics_models	200
5.6.18	library_physics_scenes.....	201
5.6.19	physics_material.....	202
5.6.20	physics_model	204
5.6.21	physics_scene.....	206
5.6.22	plane	209
5.6.23	rigid_body.....	211
5.6.24	rigid_constraint.....	215
5.6.25	shape	220
5.6.26	sphere	222
5.7	Getting started with COLLADA FX.....	223
5.7.1	General	223
5.7.2	Using profiles for platform-specific effects	223
5.7.3	About parameters in COLLADA FX	227
5.7.4	Shaders.....	228
5.7.5	Rendering.....	228
5.7.6	Texturing.....	230
5.8	FX reference	232
5.8.1	General	232
5.8.2	Elements by category	232
5.8.3	About COLLADA FX	235
5.8.4	alpha.....	235
5.8.5	annotate.....	236
5.8.6	argument.....	237
5.8.7	array	239
5.8.8	binary	241
5.8.9	bind (FX)	242
5.8.10	bind_attribute	243
5.8.11	bind_material.....	244
5.8.12	bind_uniform.....	246
5.8.13	bind_vertex_input	248
5.8.14	blinn.....	250
5.8.15	code.....	252

5.8.16	color_clear	253
5.8.17	color_target	254
5.8.18	compiler	256
5.8.19	constant (FX)	257
5.8.20	create_2d	259
5.8.21	create_3d	261
5.8.22	create_cube	263
5.8.23	depth_clear	265
5.8.24	depth_target	266
5.8.25	draw	268
5.8.26	effect	269
5.8.27	evaluate	271
5.8.28	format	273
5.8.29	fx_common_color_or_texture_type	275
5.8.30	fx_common_float_or_param_type	277
5.8.31	fx_sampler_common	278
5.8.32	image	282
5.8.33	include	284
5.8.34	init_from	285
5.8.35	instance_effect	287
5.8.36	instance_image	288
5.8.37	instance_material (geometry)	290
5.8.38	instance_material (rendering)	292
5.8.39	lamert	294
5.8.40	library_effects	296
5.8.41	library_images	297
5.8.42	library_materials	298
5.8.43	linker	299
5.8.44	material	300
5.8.45	modifier	302
5.8.46	pass	303
5.8.47	phong	305
5.8.48	profile_BRIDGE	307
5.8.49	profile_CG	309
5.8.50	profile_COMMON	311
5.8.51	profile_GLES	313
5.8.52	profile_GLES2	316
5.8.53	profile_GLSL	320
5.8.54	program	321
5.8.55	render	323
5.8.56	RGB	324
5.8.57	sampler1D	325
5.8.58	sampler2D	326
5.8.59	sampler3D	327

5.8.60	samplerCUBE	327
5.8.61	samplerDEPTH	328
5.8.62	samplerRECT	329
5.8.63	sampler_image	330
5.8.64	sampler_states	331
5.8.65	semantic	332
5.8.66	shader	333
5.8.67	sources	335
5.8.68	states	336
5.8.69	stencil_clear	342
5.8.70	stencil_target	343
5.8.71	technique (FX)	344
5.8.72	technique_hint	346
5.8.73	texcombiner	347
5.8.74	texenv	350
5.8.75	texture_pipeline	352
5.8.76	usertype	354
5.9	B-Rep Reference	355
5.9.1	General	355
5.9.2	Elements by category	356
5.9.3	About B-rep in COLLADA	357
5.9.4	brep	362
5.9.5	circle	364
5.9.6	cone	365
5.9.7	curve	367
5.9.8	curves	368
5.9.9	cylinder (B-Rep)	369
5.9.10	edges	370
5.9.11	ellipse	372
5.9.12	faces	373
5.9.13	hyperbola	375
5.9.14	line	376
5.9.15	nurbs	377
5.9.16	nurbs_surface	380
5.9.17	orient	382
5.9.18	origin	383
5.9.19	parabola	384
5.9.20	pcurves	385
5.9.21	shells	387
5.9.22	solids	389
5.9.23	surface	390
5.9.24	surfaces	392
5.9.25	surface_curves	393
5.9.26	swept_surface	394

5.9.27	torus.....	396
5.9.28	wires.....	397
5.9.29	Complete B-Rep example.....	399
5.10	Kinematics reference.....	409
5.10.1	General.....	409
5.10.2	Elements by Category.....	409
5.10.3	articulated_system.....	412
5.10.4	attachment_end.....	413
5.10.5	attachment_full.....	414
5.10.6	attachment_start.....	415
5.10.7	axis_info.....	417
5.10.8	bind (kinematics).....	420
5.10.9	bind_joint_axis.....	421
5.10.10	bind_kinematics_model.....	422
5.10.11	connect_param (kinematics).....	423
5.10.12	effector_info.....	424
5.10.13	frame_object, frame_origin, frame_tcp, frame_tip.....	426
5.10.14	instance_articulated_system.....	427
5.10.15	instance_joint.....	428
5.10.16	instance_kinematics_model.....	429
5.10.17	instance_kinematics_scene.....	431
5.10.18	joint.....	432
5.10.19	kinematics.....	434
5.10.20	kinematics_model.....	436
5.10.21	kinematics_scene.....	439
5.10.22	library_articulated_systems.....	440
5.10.23	library_joints.....	441
5.10.24	library_kinematics_models.....	442
5.10.25	library_kinematics_scenes.....	443
5.10.26	link.....	444
5.10.27	motion.....	446
5.10.28	prismatic.....	447
5.10.29	revolute.....	449
5.11	Types.....	450
5.11.1	General.....	450
5.11.2	Simple Value Types.....	450
5.11.3	Parameter-type elements.....	451
5.11.4	Other simple types.....	452
5.11.5	Value-or-param types.....	453
Annex A (informative) COLLADA example		454
Annex B (informative) Profile GLSL and GLES2 examples		457
B.1 Example: <profile-GLSL>		457
B.2 Example: <profile-GLES2>		462

Annex C (informative) Index of COLLADA elements.....	469
Bibliography.....	476

STANDARDSISO.COM : Click to view the full PDF of ISO 17506:2022

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 184, *Automation systems and integration*, Subcommittee SC 4, *Industrial data*.

This first edition cancels and replaces ISO/PAS 17506:2012, which has been technically revised.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document describes the COLLADA™¹ schema. COLLADA is a COLLABorative Design Activity that defines an XML-based schema to enable 3D authoring applications to freely exchange digital assets without loss of information, enabling multiple software packages to be combined into extremely powerful tool chains.

The purpose of this document is to provide a specification for the COLLADA schema in sufficient detail to enable software developers to create tools to process COLLADA resources. In particular, it is relevant to those who import to or export from digital content creation (DCC) applications, 3D interactive applications and tool chains, prototyping tools, real-time visualization applications such as those used in the video game and movie industries, and CAD tools.

This document covers the initial design and specifications of the COLLADA schema, as well as a minimal set of requirements for COLLADA exporters. A short example of a COLLADA instance document is presented in Annex A.

STANDARDSISO.COM : Click to view the full PDF of ISO 17506:2022

¹ COLLADA is the trademark of a product supplied by the Khronos Group Inc. (<http://khronos.org>) This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named. Equivalent products may be used if they can be shown to lead to the same results.

Industrial automation systems and integration — COLLADA™ digital asset schema specification for 3D visualization of industrial data

1 Scope

This document describes the COLLADA™ schema. COLLADA is a Collaborative Design Activity that defines an XML-based schema to enable 3D authoring applications to freely exchange digital assets without loss of information, enabling multiple software packages to be combined into extremely powerful tool chains.

The purpose of this document is to provide a specification for the COLLADA schema in sufficient detail to enable software developers to create tools to process COLLADA resources. In particular, it is relevant to those who import to or export from digital content creation (DCC) applications, 3D interactive applications and tool chains, prototyping tools, real-time visualization applications such as those used in the video game and movie industries, and CAD tools.

This document covers the initial design and specifications of the COLLADA schema, as well as a minimal set of requirements for COLLADA exporters.

This document covers the following information:

- initial design and specifications of the COLLADA schema;
- requirements of COLLADA tools and a minimal set of requirements for COLLADA exporters;
- detailed explanations for COLLADA programming;
- core elements that describe geometry, animation, skinning, assets, and scenes;
- physics model, visual effects (FX), boundary representation (B-rep) of animation, kinematics.

The document does not specify the implementation of, or definition of a run-time architecture for viewing or processing of COLLADA data.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

XML, Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation, 26 November 2008 [viewed 2020-09-01]. Available at <https://www.w3.org/TR/2008/REC-xml-20081126/>

XPath, XML Path Language (XPath) Version 1.0, W3C Recommendation, 16 November 1999 [viewed 2020-09-01]. Available at <https://www.w3.org/TR/1999/REC-xpath-19991116/>

XPointer, XML Pointer Language (XPointer) 1.0, W3C Recommendation, January 2001 [viewed 2020-09-01]. Available at <https://www.w3.org/TR/2001/WD-xptr-20010108/>

IETF RFC 3986, Uniform Resource Identifier (URI), January 2005 [viewed 2020-09-01]. Available at <https://datatracker.ietf.org/doc/html/rfc3986>

3 Terms and definitions

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://www.electropedia.org/>

3.1.1

COLLADA

digital asset XML schema for 3D visualization

Note 1 to entry: COLLADA is an abbreviation of Collaborative Design Activity.

3.1.2

COLLADA document

DAE file

file containing COLLADA XML elements that describe certain digital assets

3.1.3

COLLADA schema

XML schema document that defines all valid COLLADA elements

3.1.4

profile

<COLLADA> structure in which to gather effects information for a specific platform or environment

3.1.5

instance

<COLLADA> occurrence of an object, the result of instantiating a copy or version of the object

3.1.6

instance document

COLLADA document

3.1.7

instantiation

creation of an instance of an object

3.1.8**node**

<COLLADA> point of information within a scene graph

Note 1 to entry: COLLADA uses node to refer to interior (branch) nodes rather than to exterior (leaf) nodes.

3.1.9**path**

<COLLADA> connection between nodes

3.1.10**shorthand pointer**

<COLLADA> value of the id attribute of an element in an instance document

Note 1 to entry: This is a URI fragment identifier that conforms to XPointer 1.0 syntax.

3.1.11**animation curve****function curve**

<COLLADA> 2D function defined by a set of key frames and the interpolation among them

3.1.12**effect scope**

<COLLADA> declaration space that is inside an <effect> element but not within any specific profile

3.1.13**FX runtime**

<COLLADA> library of code that handles the creation, use, and management of shaders, source code, parameters, and other effects properties

3.1.14**morph target**

<COLLADA> mesh that can be blended with other meshes

3.1.15**multiple render target****MRT**

<COLLADA> rendering to multiple drawing buffers simultaneously

3.1.16**scene graph**

<COLLADA> hierarchical structure of a scene represented

Note 1 to entry: scene graph is defined by the scene element contents in a COLLADA document.

Note 2 to entry: scene graph is a directed acyclic graph or tree data structure that contains nodes of visual information and related data.

3.1.17**tone mapping**

<COLLADA> combination of spectral sampling and dynamic range remapping which is performed as the last step of rendering.

3.1.18

viewing frustum

<COLLADA> region of space that appears in a camera's view

Note 1 to entry: In this document, frustum means viewing frustum.

3.1.19

shader

<COLLADA> user defined program executed in graphics pipeline

Note 1 to entry: shader is programmed with shading language.

Note 2 to entry: According to profile, number and kind of shader are varying.

3.2 Abbreviated terms

For the purposes of this document, the following abbreviated terms apply.

API Application Programming Interface

COLLADA Collaborative Design Activity

CTS Conformance Test Suite

CV Control Vertex

DAG Directed Acyclic Graph

DCC Digital Content Creation

DAE Digital Asset Exchange

ID Identifier

IDREF Reference to an ID

HDR High Dynamic Range

LOD Level Of Detail

SID Scoped Identifier

SIDREF Reference to an SID

URI Uniform Resource Identifier

MRT Multiple Render Targets

XML eXtended Markup Language

4 Notational conventions

4.1 Typographic conventions and notation

Certain typographic conventions are used throughout this document to clarify the meaning of the text.

Table 1 — Typographic conventions

Conventions	Description
Regular text	Descriptive text
<code>Courier-type font</code>	code, variable and attributes in descriptive text
Courier bold	Elements or file names except within code
Blue underline	Hyperlinks
Italic text	Emphasis
<code>element1/element2</code>	element1 is the parent, element2 is the child (Element hierarchy shall conform XPath 1.0 specification)

4.2 Notation and organization in the reference sections

The schema reference clauses describe each feature of the COLLADA schema syntax. Each XML element in the schema has the following subclauses in Table 2.

Table 2 — Notation and organization in reference subclause

Subclause	Description
Introduction	Name and purpose of the element
Concepts	Background and rationale for the element
Attributes	Attributes applicable to the element
Related elements	Lists of parent elements and of other related elements
Child elements	Lists of valid child elements and descriptions of each
Details	Information concerning the usage of the element
Example	Example usage of the element

Some subclauses, especially some “Details” subclause may be omitted. Each subclause presents features with form of tables, however caption and index of tables are not provided.

4.2.1 Child element conventions

The Child Elements table lists all child elements for the specified element. For each child:

- “See main entry” means that one of the reference subclauses has a main entry for the child element. Refer to this for details about the child’s usage, attributes, and children.
- If there is not a main entry in the reference subclauses, or if the local child element’s properties vary from the main entry, information about the child element is given either in the Child Elements table or in an additional element-specific subclause.

Table 3 is examples of notation and organization of child elements.

Table 3 — Notation and organization in reference subclause — Child elements

Name/example	Description	Default	Occurrences
<camera>	Brief description. See main entry. (This means that there is a main Reference entry for camera, so look there for details.)		1 or more
<technique_common>	Brief description. See the following subclause. (This means that details are given here but in a separate table.)	N/A (means not applicable)	
<yfov sid="...">	Description, including discussion of attributes, content, and relevant child elements. (This means that there is no main Reference entry for yfov . Details are given here.)	None means none assigned NONE (Courier Font) means the value NONE	

4.2.2 Child element order

The Child elements table lists all child elements.

XML allows a schema definition to include notation that requires elements to occur in a certain order within their parent element [4]. When this reference states that child elements shall appear in the following order, it refers to a declaration similar to the following, in which the XML **<sequence>** element states that **<extra>** shall follow **<asset>**:

```
<xs:sequence>
  <xs:element ref="asset" minOccurs="0"/>
  <xs:element ref="extra" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
```

XML also provides notation indicating that two or more child elements can occur in any order. When this reference states that two child elements can appear in any order, it refers to the XML **<choice>** element with an unbounded maximum. For example, in the following, **<image>** and **<newparam>** shall appear before **<extra>** and after **<asset>**, but in that position, they can occur in any order, and the unbounded attribute specifies that you can include as many of them as needed in any combination:

```
<xs:sequence>
  <xs:element ref="asset" minOccurs="0"/>
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="image"/>
    <xs:element ref="newparam"/>
  </xs:choice>
  <xs:element ref="extra" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
```

5 COLLADA format

5.1 Design considerations

5.1.1 Introduction

Development of the COLLADA Digital Asset Exchange schema involves designers and software engineers from many companies in a collaborative design activity. Subclause 5.1 reviews the more important design goals, thoughts, and assumptions made by the designers during the inception of the project.

5.1.2 Assumptions and dependencies

During the first design phase of COLLADA, the contributors discussed and agreed on the following assumptions:

- This is not a game engine or run-time delivery format. It is assumed that COLLADA will be beneficial to users of authoring tools and to content-creation pipelines for interactive applications. It is assumed that most interactive applications will use COLLADA in the production pipeline, but not as a final delivery mechanism. For example, most games will use proprietary, size-optimized, streaming-friendly binary files.
- Artists and end users will want to quickly develop and test relatively simple content and test models that still include advanced rendering techniques such as vertex and pixel programs (shaders). It is assumed that rapid prototyping of content is important to artists and developers and that a human-readable, text-based format, along with the ability to create valid “empty” or partial content, is essential.

5.1.3 Goals and guidelines

5.1.3.1 General

Design goals for the COLLADA Digital Asset Exchange schema include the following:

- To liberate digital assets from proprietary binary formats into a well-specified, XML-based, royalty-free, open-standard format.
- To provide a standard common language format so that COLLADA assets can be used directly in existing content toolchains, and to facilitate this integration.
- To be adopted by as many digital-content users as possible.
- To provide an easy integration mechanism that enables all the data to be available through COLLADA.
- To be a basis for common data exchange among 3D applications.
- To be a catalyst for digital-asset schema design among developers and DCC, hardware, and middleware vendors.

The following subclauses explain the goals and discuss their consequences and rationales.

5.1.3.2 Liberate digital assets from proprietary binary formats

Goal: To liberate digital assets from proprietary binary formats into a well-specified, XML-based, royalty free, open-standard format.

Digital assets are the most valuable artifact for most 3D application users.

Developers have enormous investment in assets that are stored in opaque proprietary formats. Exporting the data from the tools requires considerable investment to develop software for proprietary, complex software development kits. Even after this investment has been made, it is still impossible to modify the data outside of the tool and import it again later. It is necessary to permanently update the exporters with the ever-evolving tools, with the risk of seeing the data become obsolete.

Hardware vendors need increasingly more-complex assets to take advantage of new hardware. The data needed may exist inside a tool, but there is often no way to export this data from the tool or exporting this data is a complex process that is a barrier to developers using advanced features, and a problem for hardware vendors in promoting new products.

Middleware and tool vendors have to integrate with every tool chain so that it can be used by developers, which is an impossible mission. Successful middleware vendors have to provide their own extensible tool chain and framework, and have to convince developers to adopt it. That makes it impossible for developers to use several middleware tools in the same project, just as it is difficult to use several DCC tools in the same project.

This goal led to several decisions, including:

- COLLADA will use XML.

XML provides a well-defined framework for structured content. Issues such as character sets (ASCII, Unicode, shift-jis) are already covered by the XML standard, making any schema that uses XML instantly internationally useful. XML is also fairly easy to understand given only a sample instance document and no documentation, something that is rarely true for other formats. There are XML parsers and text editors for nearly every language on every platform, making the documents easily accessible to almost any application. Therefore, valid COLLADA document shall conform XML 1.0 syntax.

- COLLADA will not use binary data inside XML.

Some discussion often occurs about storing vertices and animation data in some kind of binary representation for ease of loading, speed, and reduced asset size. Unfortunately, that goes counter to the desire of being useful to the greatest number of users on development teams. Furthermore, storing binary data within XML documents is problematic and well supported only using a base-64 encoding that contrarily increases the size of the data. Keeping COLLADA completely text based supports the most options. COLLADA does provide mechanisms to store external binary data and to reference it from a COLLADA asset.

- The COLLADA common profile will be expand over time to include as much common data as possible.

5.1.3.3 Provide a standard common language format

Goal: To provide a standard common language format so that COLLADA assets can be used directly in existing content toolchains, and to facilitate this integration.

This goal led to the COMMON profile. The intent is that, if a user's tools can read a COLLADA asset and use the data presented by the common profile, the user should be able to use any DCC tool for content creation.

To facilitate the integration of COLLADA assets into tool chains, it appears that COLLADA must provide not only a schema and a specification, but also a well-designed API (a COLLADA API) that helps integrate COLLADA assets in existing tool chains. This new axis of development can open numerous new possibilities, as well as provide a substantial savings for developers. Its design has to be such as to facilitate its integration with specific data structures used by existing content tool chains.

COLLADA can enable the development of numerous tools that can be organized in a toolchain to create a professional content pipeline. COLLADA will facilitate the design of a large number of specialized tools, rather than a monolithic, hard-to-maintain tool chain. Better reuse of tools developed internally or externally will provide economic and technical advantages to developers and tools/middleware providers, and therefore strengthen COLLADA as a standard language for digital-asset exchange.

5.1.3.4 Adoption by many digital-content users

Goal: To be adopted by as many digital-content users as possible.

To be adopted, COLLADA needs to be useful to artists and developers. For a developer to measure the utility of COLLADA to their problem, we need to provide the developer with the right information and enable the measurement of the quality of COLLADA tools. This includes:

- Provide a conformance test suite to measure the level of conformance and quality of tools.
- Provide a list of requirements in the specification for the tool providers to follow in order to be useful to most developers. (These goals are specified in 5.2: Tool Requirements and Options.)
- Collect feedback from users and add it to the requirements and conformance test suite.
- Manage bug-reporting problems and implementation questions to the public. This involves prioritizing bugs and scheduling fixes among the COLLADA partners.
- Facilitate asset-exchange and asset-management solutions.
- Engage DCC tool and middleware vendors to directly support COLLADA exporters, importers, and other tools. Developers win because they can now use every package in their pipeline. Tool vendors win because they have the opportunity to reach more users.
- Provide a command-line interface to DCC tool exporters and importers so that those tasks can be incorporated into an automated build process.

5.1.3.5 Provide an easy integration mechanism

Goal: To provide an easy integration mechanism that enables all the data to be available through COLLADA.

COLLADA is fully extensible, so it is possible for developers to adapt COLLADA to their specific needs. This leads to the following goals:

- Design the COLLADA API and future enhancements to COLLADA to ease the extension process by making full use of XML schema capabilities and rapid code generation.
- Encourage DCC vendors to make exporters and importers that can be easily extended.

- If developers need functionality that is not yet ready to be in the COMMON profile, encourage vendors to add this functionality as a vendor-specific extension to their exporters and importers.
 - This applies to tools-specific information, such as undo stack, or to concepts that are still in the consideration for inclusion in COLLADA, but that are urgently needed, such as complex shaders.
- Collect this information and lead the working group to solve the problem in the COMMON profile for the next version of COLLADA.

Make COLLADA asset-management friendly:

- Select a part of the data in a DCC tool and export it as a specific asset.
- Enable asset identification and have the correct metadata.
- Enforce the asset metadata usage in exporters and importers.

5.1.3.6 Serve as basis for common data exchange

Goal: To be a basis for common data exchange among 3D applications

The biggest consequence of this goal is that the COLLADA common profile will be an ongoing exercise. Originally, it covered polygon-based models, materials and shaders, and some animations and DAG-based scene graphs. In the future it will cover other more complex data types in a common way that makes exchanging that information among tools a possibility.

5.1.3.7 Catalyze digital asset schema design

Goal: To be a catalyst for digital-asset schema design among developers and DCC, hardware, and middleware vendors.

There is fierce competition among and within market segments: the DCC vendors, the hardware vendors, and the middleware vendors. But all of them need to communicate to solve the digital-content problems for developers. Not being able to collaborate on a common digital-asset language format has a direct impact on the overall optimization of the market solutions:

- Hardware vendors are suffering from the lack of features exposed by DCC tools.
- Middleware vendors suffer because they lack compatibility among the tool chains.
- DCC vendors suffer from the amount of support and specific development required to make developers happy.
- Developers suffer by the huge amount of investment necessary to create a working tool-chain.

None of these actors can lead the design of a common language format without being seen by the others as factoring a commercial or technical advantage into the design. No one can provide the goals that will make everybody happy, but it is necessary that everybody accept the format. It is necessary for all major actors to be happy with the design of this format for it to have wide adoption and be accepted.

5.2 Tool requirements and options

5.2.1 Introduction

Any fully conformant COLLADA tool shall support the entire specification of data represented in the schema. What may not be so obvious is the need to require more than just adherence to the schema specification. Some such additional needs are the uniform interpretation of values, the necessity of offering crucial user-configurable options, and details on how to incorporate additional discretionary features into tools. The goal of this subclause is to prioritize those issues.

Each “Requirements” subclause details options that shall be implemented completely by every compliant tool. One exception to this rule is when the specified information is not available within a particular application. An example is a tool that does not support layers, so it would not be required to export layer information (assuming that the export of such layer information is normally required); however, every tool that did support layers would be required to export them properly.

The “Optional” subclause describes options and mechanisms for things that are not necessary to implement but that probably would be valuable for some subset of anticipated users as advanced or esoteric options.

The requirements explored in this subclause are placed on tools to ensure quality and conformance to the purpose of COLLADA. These critical data interpretations and options aim to satisfy interoperability and configurability needs of cross-platform application-development pipelines. Ambiguity in interpretation or omission of essential options could greatly limit the benefit and utility to be gained by using COLLADA. This subclause has been written to minimize such shortcomings.

It is recommended that features required in this clause is tested by one or more test cases in the COLLADA Conformance Test Suite (CTS). The COLLADA CTS had developed by The Khronos Group, is a set of tools that automate the testing of exporters and importers of applications for digital content creation. Each test case compares the native content against that content after it has gone through the tool’s COLLADA import/export plug-in [10].

5.2.2 Exporters

5.2.2.1 General

The responsibility of a COLLADA exporter is to write all the specified data according to certain essential options.

5.2.2.2 Requirements

5.2.2.2.1 Hierarchy and transforms

Table 4 is minimum requirements for exporters of hierarchy and transforms.

Table 4 — Requirements for exporters of hierarchy and transforms

Data	Requirements
Translation	Translations
Scaling	Scales
Rotation	Rotations
Parenting	Parent relationships
Static object instantiation	Instances of static objects. Such an object can have multiple transforms
Animated object instantiation	Instances of animated objects. Such an object can have multiple transforms
Skewing	Skews
Transparency / reflectivity	Additional material parameters for transparency and reflectivity
Texture-mapping method	A texture-mapping method (cylindrical, spherical, etc.)
Transform with no geometry	It shall be possible to transform something with no geometry (for example locator, NULL)

5.2.2.2.2 Materials and textures

Table 5 is minimum requirements for exporters of materials and textures.

Table 5 — Requirements for exporter of materials and textures

Data	Requirements
RGB textures	An arbitrary number of RGB textures
RGBA textures	An arbitrary number of RGBA textures
Baked procedural texture coordinates	Baked procedural texture coordinates
Common profile material	A common profile material (PHONG, LAMBERT, etc.)
Per-face material	Per-primitive materials

5.2.2.2.3 Vertex attributes

Table 6 is minimum requirements for exporter of vertex attributes.

Table 6 — Requirements for exporter of vertex attributes

Data	Requirements
Vertex texture coordinates	An arbitrary number of Texture Coordinates per vertex
Vertex normals	Vertex normals
Vertex binormals	Vertex binormals
Vertex tangents	Vertex tangents
Vertex UV coordinates	Vertex UV coordinates (distinct from texture coordinates)
Vertex colors	Vertex colors
Custom vertex attributes	Custom vertex attributes

5.2.2.2.4 Animation

All of the following kinds of animations (that don't specifically state otherwise) shall be exported using samples or key frames (according to a user-specified option).

Animations are usually represented in an application by the use of sparse key frames and complex controls and constraints. These are combined by the application when the animation is played, providing final output. When parsing animation data, it is possible that an application will not be able to implement the full set of constraints or controllers used by the tool that exported the data, and thus the resulting animation will not be preserved. Therefore, it is necessary to provide an option to export fully resolved transformation data at regularly defined intervals. The sample rate shall be specifiable by the user when samples are preferred to key frames.

Exporting all available animated parameters is necessary. This includes:

- Material parameters;
- Texture parameters;
- UV placement parameters;
- Light parameters;
- Camera parameters;
- Shader parameters;
- Global environment parameters;
- Mesh-construction parameters;
- Node parameters;
- User parameters.

5.2.2.2.5 Scene data

Table 7 is minimum requirements for exporters of scene data.

Table 7 — Requirements for exporters of scene data

Data	Requirements
Empty nodes	Empty nodes
Cameras	Cameras
Spotlights	Spotlights
Directional lights	Directional lights
Point lights	Point lights
Ambient lights	Ambient lights

5.2.2.2.6 Exporter user interface options

Table 8 is minimum requirements for exporters of user interface options.

Table 8 — Requirements for exporter of user interface options

Data	Requirements
Export triangle list	Triangle lists
Export polygon list	Polygon lists
Bake matrices	Baked matrices
Single <matrix> element	An instance document that contains only a single <matrix> element for each node. (See the following “Single <matrix> Element Option” discussion)

5.2.2.2.7 Single **<matrix>** element option

COLLADA allows transformations to be represented by a stack of different transformation element types, which shall be composed in the specified order. This representation is useful for accurate storage and/or interchange of transformations in the case where an application internally uses separate transformation stages. However, if this is implemented by an application, it should be provided as a user option, retaining the ability to store only a single baked **<matrix>**.

A side effect of this requirement is that any other data that target specific elements inside a transformation stack (such as animation) shall target the matrix instead.

5.2.2.2.8 Command-line operation

It shall be possible to execute an exporter supporting all features from a command line.

5.2.2.3 Optional

5.2.2.3.1 Shader export

An exporter may export shaders (for example: Cg [14], GLSL [16], HLSL [15]).

5.2.3 Importers

5.2.3.1 General

The responsibility of a COLLADA importer is to read all the specified data according to certain essential options.

Importers shall provide the inverse function operation of every export option described in the “5.2.2 Exporters” subclause where it is possible to do so. This subclause describes only issues where the requirements placed on importers diverge or need clarification from the obvious inverse method of exporters.

5.2.3.2 Requirements

Importer shall support to import all conforming COLLADA data, even if some data is not understood by the tool, and retained for later export. The **<asset>** element will be used by external tools to recognize that some exported data may require synchronization.

5.2.3.3 Optional

There are no unique options for importers.

5.2.4 Archive packaging

On import and export, DCC tools shall support the .dae format, which is a zip archive of one or several .dae files (COLLADA documents) and all the referenced content (textures).

The archive shall include a file named manifest.xml, an XML-encoded file that contains a **<dae_root>** element. This element is a UTF8 encoding of the relative URI pointing to a .dae file. If the URI contains a fragment, then the indicated element is the starting point for application loading of the .dae archive. Otherwise, the **<scene>** element will be the starting point for application loading the .dae archive. If neither of these conditions is met, then the behaviour is undefined.

The URIs in the .dae files can reference any other file in the archive using relative paths from the root of the archive, in accordance with the URI standard.

The archive itself may include other archives (zip, rar, kmz, zae). The URI to reference a document inside a nested archive, itself inside the .dae archive, will use the name of the nested archive in the path, such as:

```
./internal_archive.zip/directory/document.dae#element
```

It is not possible to reference content outside of an archive using a relative URI, but it is valid to reference content using an absolute URI, such as:

```
file:///other_directory/other_document.dae#element
```

5.3 Schema concepts

5.3.1 Introduction

The COLLADA schema is an eXtensible Markup Language (XML) database schema. The XML Schema language is used to describe the COLLADA feature set.

Documents that use the COLLADA schema, i.e. that contain XML elements that are valid according to the COLLADA schema, are called COLLADA instance documents. The file extension chosen for these documents is .dae (an acronym for Digital Asset Exchange).

This subclause briefly introduces basic XML terminology, describes how COLLADA elements can refer to other COLLADA elements, and provides additional conceptual information about how COLLADA works.

5.3.2 XML overview

XML provides a standard language for describing the content, structure, and semantics of files, documents, or datasets. An XML document consists primarily of *elements*, which are blocks of information surrounded by start and end *tags*. For example:

```
<node id="here">
  <translate sid="trans"> 1.0 2.0 3.0 </translate>
  <rotate sid="rot"> 1.0 2.0 3.0 4.0 </rotate>
  <matrix sid="mat">
    1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0
    9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0
  </matrix>
</node>
```

This example contains four elements: **<node>**, **<translate>**, **<rotate>**, and **<matrix>**. The latter three elements are nested within the **<node>** element; elements can be nested to an arbitrary depth.

Elements can have attributes, which describe some aspect of the element. For example, the `id` attribute of the `<node>` element in the example has the value "here"; this might differentiate it from another `<node>` element whose `id` attribute value is "there". In this case, the attribute's name is "id"; its value is "here".

For additional information about XML vocabulary, refer to Clause 3 Terms and definitions.

5.3.3 Address syntax

5.3.3.1 General

COLLADA uses two mechanisms to address elements and values within an instance document:

- URI addressing: Refers to the `id` attribute of an element. Used in `url` and `source` attributes. Described in the following "URI Addressing" subclause.
- Scoped-Identifier (SID) addressing: Refers to the `sid` attribute of an element. Used in `target` and `ref` attributes, `<SIDREF>` and `<SIDREF_array>` elements, and every other attribute or element that contains or references an SID. Described in the following "Scoped-Identifier (SID) Addressing" subclause.

5.3.3.2 URI addressing

5.3.3.2.1 General

The `url` and `source` attributes of many elements use the URI addressing scheme that locates instance documents and elements within them by their `id` attribute values.

5.3.3.2.2 URI fragment identifier

Many COLLADA elements have an `id` attribute. These elements can be addressed using the Uniform Resource Identifier (URI) fragment identifier notation. The XML specification defines the syntax for a URI fragment identifier within an XML document. The URI fragment identifier shall conform to the XPointer syntax. As COLLADA addresses only unique identifiers with URI, the XPointer syntax used is called the shorthand pointer notation. A shorthand pointer is the value of the `id` attribute of an element in the instance document.

In a `url` or `source` attribute, the URI fragment identifier is preceded with the literal pound sign or hash character ("#"). In a `target` attribute, there is no pound sign because it is not a URI. For example, the same `<source>` element is addressed as follows using each notation:

```
<source id="here" />
<input source="#here" />
<bind target="here" />
```

Within a COLLADA instance document, a light defined with the ID "Lt-Light" can be referenced using `<instance_light url = "#Lt-Light">`. In the following example, the light node element refers to the light element found in the light library as following example code.

```
<library_lights>
  <light id="Lt-Light" name="light">
    <technique_common>
      <ambient>
        <color>1 1 1</color>
```



```

    </ambient>
  </technique_common>
</light>
</library>
...
<node id="Light" name="Light">
  <translate>-5.000000 10.000000 4.000000</translate>
  <rotate>0 0 1 0</rotate>
  <rotate>0 1 0 0</rotate>
  <rotate>1 0 0 0</rotate>
  <instance_light url="#Lt-Light" />
</node>

```

5.3.3.2.3 URI path syntax

URIs used in COLLADA document shall conform to URI syntax IETF RFC 3986. It defines a generic URI as consisting of five hierarchical parts: the scheme, authority, path, query, and fragment. In BNF, the syntax is:

```

scheme ":" hierarchy-part ["?" query ] ["#" fragment ]
hierarchy-part = "//" authority path-abempty
               / path-absolute
               / path-rootless
               / path-empty

```

The scheme and the hierarchy-part are required for an absolute URI. The hierarchy-part, however, can be an empty path.

URI syntax requires that the hierarchical pathname levels (such as directories) be separated with forward slashes ("/") and that other special characters within the path be escaped, for example, converting spaces to their hexadecimal representation "%20". An absolute path, such as a native file-system path, that does not conform to IETF format shall be adjusted to do so. For example, the absolute Windows path "C:\foo\bar\my file#27.dae", by URI syntax definition, could be interpreted as a relative path (starting with "C") to the current base URI – and, furthermore, backslashes could be treated the same as any other text character, not as valid separators. Although some applications look for Windows paths and convert them to valid URLs, not all applications do. Therefore, always use valid URI syntax, which for this example would be "/C:/foo/bar/my%20file%2327.dae".

NOTE Whenever possible, it is better encoding practice to use paths that are relative to the location of the document that references them rather than to use absolute paths.

5.3.3.3 Scoped-identifier (SID) addressing

5.3.3.3.1 General

A scoped identifier (SID) is a value of `sid_type`, defined as an XML Schema Language `NCName` type (`xs:NCName`) with the added constraint that its value is unique within the scope of its parent element, among the set of SIDs at the same hierarchical level, as found using a breadth-first traversal. An SID might be ambiguous across **<technique>** elements.

Every attribute and element that contains or references a path to an SID uses values of type `sidref_type`, which is a COLLADA-defined addressing scheme of `id` and `sid` attributes to locate elements within an instance document. This includes the `target` and `ref` attributes, the **<SIDREF>** and **<SIDREF_array>** elements, and every other attribute or element that contains or references an SID path.

5.3.3.3.2 SID addressing syntax

The syntax of scoped-identifier addressing (previously called target addressing) has several parts:

- The first part is the `id` attribute of an element in the instance document or a dot segment (`"."`) indicating that this is a relative address;
- Zero or more scoped identifiers (SIDs) follow. Each is preceded by a literal slash (`"/"`) as a path separator; if this part is empty, then there is no literal slash. The scoped identifiers are taken from a child of the element identified by the first part. For nested elements, multiple scoped identifiers can be used to identify the path to the targeted element;
- The final part is optional. This is a C/C++-style structure-member selection syntax for addressing element values. If this part is absent, all member values of the target element are targeted (for example, all values of a matrix). If this part is present, it can take one of two forms:
 - The name of the member value (field) indicating symbolic access. This notation consists of:
 - A literal period (`"."`) indicating member selection access;
 - The symbolic name of the member value (field). The "Common Glossary" subclause later in this clause documents values for this field under the common profile. Application-defined values are also permitted, although, in that case, interoperability is not assured.
 - The cardinal position of the member value (field) indicating array access. The array-access syntax can be used to express fields only in one-dimensional vectors and two-dimensional matrices, where both are column-major order. This notation consists of:
 - A literal left parenthesis (`"("`) indicating array selection access;
 - A number of the field, starting at zero for the first field;
 - A literal right parenthesis (`")"`) closing the expression.

5.3.3.3.3 SID addressing examples

Here are examples of scoped-identifier addressing:

```
<channel target="cube/translate.X" />
<connect_param ref="cube/translate.X" />
<SIDREF>cube/translate.X</SIDREF>
<SIDREF_array>cube/translate.X cube/translate.Y</SIDREF_array>
```

In the following example, three of the **<channel>** elements target one component of the **<translate>** element's member values denoted by X, Y, and Z. Likewise, the **<rotate>** element's **ANGLE** member is targeted twice using symbolic and array syntax, respectively:

```
<channel target="here/trans.X" />
<channel target="here/trans.Y" />
<channel target="here/trans.Z" />
```

```

<channel target="here/rot.ANGLE" />
<channel target="here/rot(3)" />
<channel target="here/mat(3)(2)" />

<node id="here">
  <translate sid="trans"> 1.0 2.0 3.0 </translate>
  <rotate sid="rot"> 1.0 2.0 3.0 4.0 </rotate>
  <matrix sid="mat">
    1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0
    9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0
  </matrix>
</node>

```

For increased flexibility and concision, the SID addressing mechanism allows for skipping XML elements. It is not necessary to assign `id` or `sid` attributes to all in-between elements.

For example, you can target the Y value of a camera without adding `sid` attributes for **<optics>** and the **<technique>** elements. Some elements don't even allow `id` and `sid` attributes.

It is also possible to target the **<yfov>** of that camera in multiple techniques without having to create extra animation channels for each targeted technique (techniques are “switches”: One or the other is picked on import, but not both, so it still resolves to a single target).

For example:

```

<channel source="#YFOVSampler" target="Camera01/YFOV"/>
...
<camera id="#Camera01">
  <optics>
    <technique_common>
      <perspective>
        <yfov sid="YFOV">45.0</yfov>
        <aspect_ratio>1.33333</aspect_ratio>
        <znear>1.0</znear>
        <zfar>1000.0</zfar>
      </perspective>
    </technique_common>
    <technique profile="OTHER">
      <param sid="YFOV" type="float">45.0</param>
      <otherStuff type="MySpecialCamera">DATA</otherStuff>
    </technique>
  </optics>
</camera>

```

Notice that the same `sid="YFOV"` attribute is used even though the name of the parameter is different in each technique. This is valid.

Without allowing for skipping, targeting elements would be a brittle mechanism and require long attributes and potentially many extra animation channels. Of course, you can still use separate animation channels if the targeted parameters under different techniques require different values.

5.3.4 Instantiation and external referencing

The actual data representation of an object might be stored only once. However, the object can appear in a scene more than once. The object can be transformed in various ways each time it appears. Each appearance in the scene is called an instance of the object. The family of `instance_*` elements enables a COLLADA document to efficiently describe the instantiation and sharing of object data.

Each instance inherits the local coordinate system from its parent, including the applicable **<unit>** and **<up_axis>** settings, to determine its position, orientation, and scale.

Each instance of the object can be unique or can share data with other instances. A unique instance has its own copy of the object's data and can be manipulated independently. A non-unique (shared) instance shares some or all of its data with other instances of that object. Changes made to one shared instance affect all the other instances sharing that data.

When the mechanism to achieve this effect is local to the current scene or resource, it is called instantiation. When the mechanism to achieve this effect is external to the current scene or resource, it is called external referencing.

NOTE COLLADA does not dictate the policy of data sharing for each instance. This decision is left to the run-time application.

COLLADA contains several `instance_*` elements, which instantiate their related elements. For example, **<instance_animation>** describes an instance of **<animation>**. The url attribute of an instance element points to an element of the appropriate related type.

In core COLLADA, these elements are:

```
<instance_animation>
<instance_camera>
<instance_controller>
<instance_formula>
<instance_geometry>
<instance_light>
<instance_node>
<instance_visual_scene>
```

In COLLADA Physics, these elements are:

```
<instance_force_field>
<instance_physics_material>
<instance_physics_model>
<instance_physics_scene>
<instance_rigid_body>
<instance_rigid_constraint>
```

In COLLADA FX, these elements are:

```
<instance_effect>
<instance_image>
<instance_material>
```

In COLLADA kinematics, these elements are:

```
<instance_articulated_system>
<instance_joint>
<instance_kinematics_model>
<instance_kinematics_scene>
```

5.3.5 The Common profile

5.3.5.1 General

The COLLADA schema defines **<technique>** elements that establish a context for the representation of information that conforms to a configuration profile. This profile information is currently outside the scope of the COLLADA schema.

One aspect of the COLLADA design is the presence of techniques for a common profile. The **<technique_common>** and **<profile_COMMON>** elements explicitly invoke this profile. All tools that parse COLLADA content shall understand this common profile. Therefore, COLLADA provides a definition for the common profile.

5.3.5.2 Naming conventions

The COLLADA common profile uses the following naming conventions for canonical names:

- Parameter names are uppercase. For example, the values for the **<param>** element's name attribute are all uppercase letters:

```
<param name="X" type="float"/>
```

- Parameter types are lowercase when they correspond to a primitive type in the COLLADA schema, in the XML Schema, or in the C/C++ languages. Type names are otherwise inter-capitalized. For example, the values for the **<param>** element's type attribute follow this rule:

```
<param name="X" type="float"/>
```

- Input and parameter semantic names are uppercase. For example, the values for the **<input>** and **<newparam>** elements' semantic attribute are all uppercase letters:

```
<input semantic="POSITION" source="#grid-Position"/>
<newparam sid="blah">
  <semantic>DOUBLE_SIDED</semantic>
  <float>1.0</float>
</newparam>
```

5.3.6 Common profile elements

The COLLADA common profile is declared by the **<technique_common>** or **<profile_COMMON>** elements. For example:

```
<technique_common>
<!-- This scope is in the common profile -->
</technique_common>
```

Elements that appear outside the scope of a **<technique_common>** element are not in any profile, much less the common profile. For example, an **<input>** element that appears within the scope of the **<polygons>** element is not in the common profile; rather, it is invariant to all techniques and profiles.

5.3.7 Example and discussion on techniques

More generally, **<technique_common>** and **<technique>** together represent the design idiom for COLLADA multiple representation and extensibility by alternative profiles. COLLADA enables multiple representations of many elements using one **<technique_common>** and zero or more **<technique>** elements. The common technique, which is required, is a strongly typed representation of the element.

Other techniques are defined by a vendor supplying an alternative representation. Each **<technique>** has a **profile** attribute that indicates the platform (product name or similar) to which the extension applies.

Each alternative representation of an extensible element should contain values that describe the element for that profile. The representations may have coherency among profiles, although that is not required, i.e. if **<technique_common>** describes an ambient light, it is valid if a **<technique>** for that light describes something else, such as an area light, for that profile.

```
<channel source="#YFOVSampler" target="Camera01/YFOV"/>
...
<camera id="#Camera01">
  <optics>
    <technique_common>
      <perspective>
        <yfov sid="YFOV">45.0</yfov>
        <aspect_ratio>1.33333</aspect_ratio>
        <znear>1.0</znear>
        <zfar>1000.0</zfar>
      </perspective>
    </technique_common>
    <technique profile="OTHER">
      <param sid="YFOV" type="float">45.0</param>
      <otherStuff type="MySpecialCamera">DATA</otherStuff>
    </technique>
  </optics>
</camera>
```

- All consuming applications shall recognize **<technique_common>**. Information in this technique is designed to be used as the reliable fallback when no other technique is recognized by the current runtime.
- If an exporting application uses any **<technique>** elements, it shall include a **<technique_common>**.
- All techniques in a specific location represent the same concept, object, or process, but might provide entirely different information for that representation depending on the target application.
- A consuming application can choose among the **<technique>**s; if it doesn't explicitly choose, **<technique_common>** is the default.

5.3.8 Common glossary

This subclause lists the canonical names of parameters and semantics that are within the common profile. Also listed are the member-selection symbolic names for the target attribute addressing scheme.

The common **<param>** (data flow) name attribute and **<newparam>** semantic values are in Table 9.

Table 9 — Common <param> (data flow) name attribute <newparam> semantic value

Value of name or semantic attribute	Type	Typical Context	Description	Default
A	float_type	<material> , <texture>	Alpha color component	N/A
ANGLE	float_type	<animation> , <light>	Euler angle	N/A
B	float_type	<material> , <texture>	Blue color component	N/A
DOUBLE_SIDED	Boolean	<material>	Rendering state	N/A
G	float_type	<material> , <texture>	Green color component	N/A
P	float_type	<geometry>	Third texture coordinate	N/A
Q	float_type	<geometry>	Fourth texture coordinate	N/A
R	float_type	<material> , <texture>	Red color component	N/A
S	float_type	<geometry>	First texture coordinate	N/A
T	float_type	<geometry>	Second texture coordinate	N/A
TIME	float_type	<animation>	Time in seconds	N/A
U	float_type	<geometry>	First generic parameter	N/A
V	float_type	<geometry>	Second generic parameter	N/A
W	float_type	<animation> , <controller> , <geometry>	Fourth Cartesian coordinate	N/A
X	float_type	<animation> , <controller> , <geometry>	First Cartesian coordinate	N/A
Y	float_type	<animation> , <controller> , <geometry>	Second Cartesian coordinate	N/A
Z	float_type	<animation> , <controller> , <geometry>	Third Cartesian coordinate	N/A

The common **<channel1>** target attribute member selection values are in Table 10.

Table 10 — The common <channel> target attribute member selection values

Value of target attribute	Type	Description
'(' # ')' ' ['(' # ')']	float_type	Matrix or vector field
A	float_type	Alpha color component
ANGLE	float_type	Euler angle
B	float_type	Blue color component
G	float_type	Green color component
P	float_type	Third texture coordinate
Q	float_type	Fourth texture coordinate
R	float_type	Red color component
S	float_type	First texture coordinate
T	float_type	Second texture coordinate
TIME	float_type	Time in seconds
U	float_type	First generic parameter
V	float_type	Second generic parameter
W	float_type	Fourth Cartesian coordinate
X	float_type	First Cartesian coordinate
Y	float_type	Second Cartesian coordinate
Z	float_type	Third Cartesian coordinate

Recall that array index notation can be used to target vector and matrix fields.

5.4 Programming guideline

5.4.1 General

This subclause provides detailed explanations for COLLADA programming.

5.4.2 Parameters in COLLADA

In COLLADA FX, a **<param>** (Core) or **<newparam>** element declares a bindable parameter within the given scope. Parameter types do not have to strictly match to be successfully bound. The types shall be compatible, however, through simple (and sensible as defined by the application) conversion or promotion, such as integer to floating-point, or float3_type to float4_type, or Boolean to integer.

COLLADA provides the following element for working with general parameters:

- **<param>** (Core): Defines a parameter and sets its type and value for immediate use.

COLLADA provides the following basic elements for working with parameters in FX and kinematics:

- **<newparam>**: Creates a parameter;
- **<setparam>**: Changes or sets the type and value of a parameter;
- **<param>** (reference): Refers to an existing parameter created by **<newparam>**.

Details about parameters are in 5.7 Getting started with COLLADA FX.

5.4.3 Curve interpolation

5.4.3.1 General

This subclause provides information to describe an unambiguous implementation of **<geometry>/<spline>** and **<animation>/<sampler>** curves.

Both **<geometry>/<spline>** and **<animation>/<sampler>** define curves. The first represents curves that can be displayed; the second represents curves that are used to create animations.

COLLADA defines a semantic attribute for the **<input>** element that identifies the data needed for interpolating curves. The values for this attribute include POSITION, INTERPOLATION, LINEAR_STEPS, INPUT, OUTPUT, IN_TANGENT, OUT_TANGENT, and CONTINUITY. In addition, the **<Name_array>** within a source allows an application to specify the type of curve to be processed; the common profile defines the values BEZIER, LINEAR, BSPLINE, and HERMITE. This subclause describes how COLLADA uses these semantics and curve names.

5.4.3.2 Spline curves (<geometry>/<spline>)

5.4.3.2.1 General

The COLLADA specification of animated curves (**<animation>/<sampler>**) is derived from the dataflow definition of the drawing primitive for cubic polynomial curves (**<geometry>/<spline>**).

A curve is defined in segments. Each segment is defined by two endpoints. Each endpoint of a segment is also the beginning point of the next segment. The endpoints for segment[i] are given by POSITION[i] and POSITION[i+1]. Therefore, a curve with n segments will have n+1 positions. Points can be defined in two or in three dimensions (2D or 3D).

The behaviour of a curve between its endpoints is given by a specific interpolation method and additional coefficients. Each segment can be interpolated with a different method. By convention, the interpolation method for a segment is attached to the first point, so the interpolation method for segment[i] is stored in INTERPOLATION[i]. If an n-segment curve is open, then INTERPOLATION[v+1] is not used, but if the curve is closed (the endpoint of the last segment is connected to the beginning point of the first segment), then INTERPOLATION[n+1] is the interpolation method for this extra segment. The closed attribute of the **<spline>** element indicates whether the curve is closed (true) or open (false; this is the default).

LINEAR_STEPS is an optional **<input>** semantic that indicates how precisely a curve needs to be interpolated. In general, a complex curve inside a segment is done by approximations using a subdivision of line segments. The number of subdivisions is given by LINEAR_STEPS.

The following is an example of what a spline definition might look in COLLADA:

```
<spline closed="true">
  <source id="positions" >
    <!-- contains n+1 values --> </source>
  <source id="interpolations" >
    <!-- contains n+1 values --> </source>
  <source ... >
    <!-- n+1 values --> </source>
  <source ... >
    <!-- n+1 values --> </source>
  <control_vertices>
    <input semantic="POSITION" source="#positions"/>
    <input semantic="INTERPOLATION" source="#interpolations"/>
    <input ... <!--additional inputs depending on interpolation methods -->
```

CONTINUITY is an optional **<input>** semantic that indicates how the tangents were constrained when the curve was designed. Valid **CONTINUITY** values are:

5.4.3.2.2 Linear splines

Linear interpolation is the simplest; it means that the curve for the given segment is a straight line between the beginning and end points. It does not require any additional control points within a segment.

Figure 1 illustrates a three-segment closed **<spline>** with **LINEAR** interpolation between each pair of the four positions (P_0, P_1, P_2, P_3). Because it is a closed spline, there is a final (fourth) segment between P_3 and P_0 .

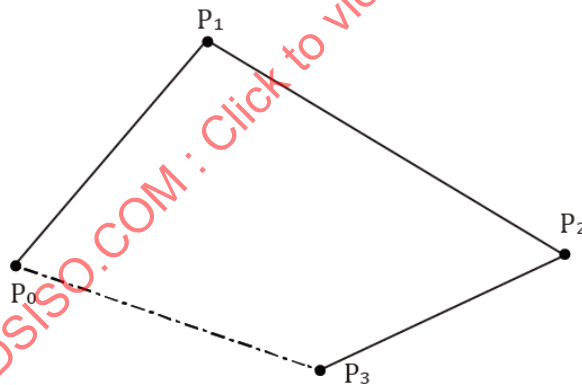


Figure 1 — Linear spline

A linear spline formula is given by:

$$L(s) = P_0 + (P_1 - P_0)s, \quad s \in [0,1]$$

Another way to represent this formula is to use the matrix form:

$$L(s) = S M C$$

$$S = [s \quad 1]$$

$$M = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$C = [s \quad 1]^T$$

In COLLADA, a geometry vector for `LINEAR` segment[i] is defined by:

- P_0 is `POSITION[i]`
- P_1 is `POSITION[i+1]`

5.4.3.2.3 Bézier and Hermite splines

A segment can be a cubic Bézier spline, which requires two additional points to control the behaviour of the curve. The following example shows one segment interpolated using `BEZIER`. It has the same beginning and end points as previously (P_0 and P_1) but has two additional control points (C_0 and C_1) that provide the additional information to calculate the curve.

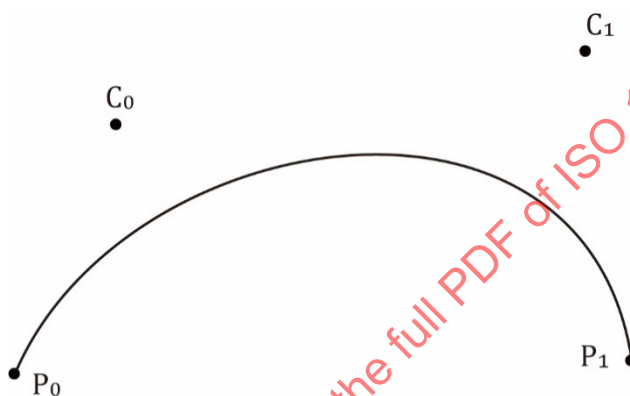


Figure 2 — Bézier Curve

`HERMITE` is equivalent to `BEZIER`, but instead of providing the control points C_0 and C_1 , tangents T_0 and T_1 are provided.

The following figure illustrates a two-segment curve with cubic Hermite interpolation:

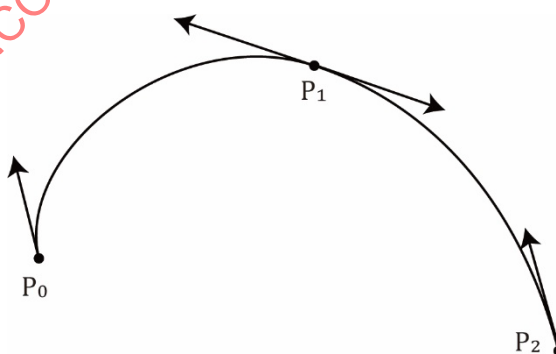


Figure 3 — Hermite Spline

Two tangents are attached to the point P_1 . The tangent that defines the beginning of the second segment is called the `OUT_TANGENT`, because it is for the segment that begins by coming out of P_1 . The tangent that defines the end of the first segment is called the `IN_TANGENT`, because this is for the segment that ends by going in to P_1 , i.e. `IN_TANGENT[1]` is the end tangent of segment[0] and `OUT_TANGENT[1]` is the beginning tangent of segment[1].

HERMITE and BEZIER are identical polynomial interpolations, with a variable change given by:

$$T_0 = 3(C_0 - P_0)$$

$$T_1 = 3(P_1 - C_1)$$

Equations are given as parametric equations. A parameter s that goes from 0 to 1 is used to calculate all the points on the curve. If s is 0, then the equation gives P_0 ; if s is 1, then the equation gives P_1 . The equation is not defined outside of those values.

In COLLADA, the **<input>** semantics IN_TANGENT and OUT_TANGENT are used to store either the tangents or the control points depending on the interpolation method.

A cubic Bézier spline formula is given by:

$$B(s) = P_0 (1-s)^3 + 3 C_0 s (1-s)^2 + 3 C_1 s^2 (1-s) + P_1 s^3, \quad s \in [0,1]$$

Another popular way of representing this equation is with the matrix form:

$$B(s) = S M C$$

$$S = [s^3 \quad s^2 \quad s \quad 1]$$

$$M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$C = [P_0 \quad C_0 \quad C_1 \quad P_1]^T$$

In COLLADA, a geometry vector for Bézier segment $[i]$ is defined by:

- P_0 is POSITION $[i]$;
- C_0 is OUT_TANGENT $[i]$;
- C_1 is IN_TANGENT $[i+1]$;
- P_1 is POSITION $[i+1]$;

The following is a COLLADA example of a 2D (X,Y) Bézier curve with two segments:

```
<spline>
  <source id="positions">
    <float_array count="6" ...> </float_array>
    <technique_common>
      <accessor>
        ... <param name="X" offset="0" type="float" ...
        ... <param name="Y" offset="1" type="float" ...
      </accessor>
    </technique_common>
  </source>
  <source id="interpolations" >
    <Name_array count="3"> BEZIER BEZIER BEZIER</Name_array>    <!-- last one
ignored for open curves -->
    <technique_common>
      <accessor>
        ... <param name="INTERPOLATION" type="Name" ...
      </accessor> </technique_common> </source>
```

```

    </technique_common>
  </source>
  <source id="in_tangents" >
    <float_array count="6" ...> <!-- first one ignored for open curves -->
    <technique_common>
      <accessor>
        ... <param name="X" offset="0" type="float" ...
        ... <param name="Y" offset="1" type="float" ...
      </accessor>
    </technique_common>
  </source>
  <source id="out_tangents">
    <float_array count="6" ...> <!-- last one ignored for open curves -->
    </float_array>
    <technique_common>
      <accessor>
        ... <param name="X" offset="0" type="float" ...
        ... <param name="Y" offset="1" type="float" ...
      </accessor>
    </technique_common>
  </source>
  <control_vertices>
    <input semantic="POSITION" source = "#positions"/>
    <input semantic="INTERPOLATION" source="#interpolations"/>
    <input semantic="IN_TANGENT" source="#in_tangents"/>
    <input semantic="OUT_TANGENT" source="#out_tangents"/>
  </control_vertices>

```

A cubic Hermite spline formula is given by:

$$H(s) = P_0(2s^3 - 3s^2 + 1) + T_0(s^3 - 2s^2 + s) + P_1(-2s^3 + 3s^2) + T_1(s^3 - s^2), \quad s \in [0,1]$$

In its matrix form, this is:

$$H(s) = S M C$$

$$S = [s^3 \quad s^2 \quad s \quad 1]$$

$$M = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$C = [P_0 \quad P_1 \quad T_0 \quad T_1]^T$$

Where

$$T_0 = 3(C_0 - P_0),$$

$$T_1 = 3(P_1 - C_1)$$

In COLLADA, a geometry vector for HERMITE segment [i] is defined by:

- P_0 is POSITION[i];
- P_1 is POSITION[i+1];
- T_0 is OUT_TANGENT[i];
- T_1 is IN_TANGENT[i+1];

5.4.3.2.4 B-Splines

Basis splines (B-splines) are defined by a series of control points, for which the curve is guaranteed only to go through the first and the last point, such as in the following Figure 4:

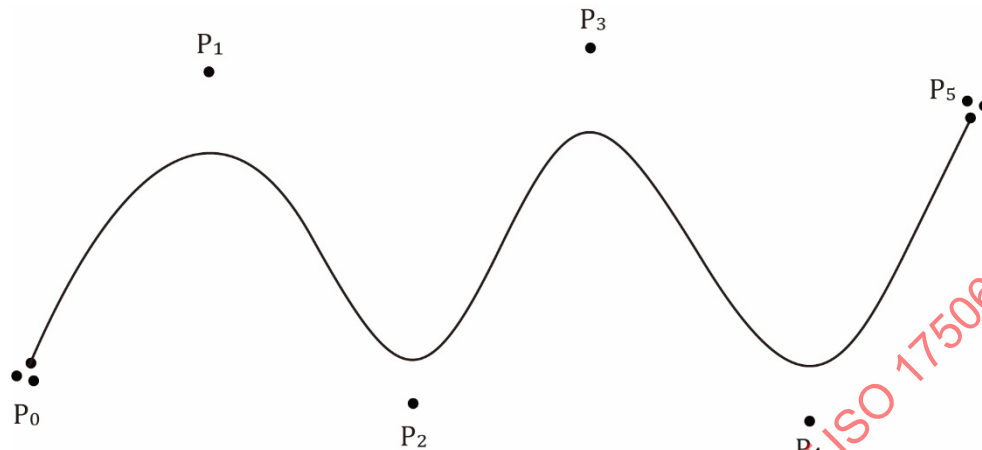


Figure 4 — Example of B-splines

COLLADA defines uniform cubic B-spline interpolations. The behaviour of a curve points $P_2 \sim P_3$ is given by the following formula, using the previous and next points. In endpoints and P_1 and P_4 , it is assumed that control points are added in the same location as endpoints.

A B-spline is described by the following matrix-form formulae, for the segment going from P_i to P_{i+1} .

$$B_i(s) = S M C$$

$$S = [s^3 \quad s^2 \quad s \quad 1]$$

$$M = u * \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

$$u = \frac{1}{6}$$

$$C = [P_{i-1} \quad P_i \quad P_{i+1} \quad P_{i+2}]^T$$

In COLLADA, defining this B-spline geometry vector requires using only two **<input>** elements, POSITION and INTERPOLATION, with:

- `Pi = POSITION[i]`
- `INTERPOLATION[i] = BSPLINE`

5.4.3.2.5 Cardinal splines

The cardinal spline is a cubic Hermite spline whose tangents are defined by the endpoints and a tension parameter. The tangents are calculated with the previous and the next point following the segment:

$$T_i = \frac{1}{2} * (1 - c)(P_{i+1} - P_{i-1})$$

where c is the tension parameter, which is a constant that modifies the length of the tangent. This parameter is not specified separately in COLLADA and is instead baked into the tangents that are provided by the `OUT_TANGENT` and `IN_TANGENT` inputs to the sampler.

The cardinal spline can be put into matrix form, using the same geometry vector C as for the BSPLINE:

$$D_i(s) = S M C$$

$$S = [s^3 \quad s^2 \quad s \quad 1]$$

$$t = \frac{(1 - c)}{2}$$

$$M = u * \begin{bmatrix} -t & 2-t & t-2 & t \\ 2t & t-3 & 3-2t & 1-t \\ -t & 0 & t & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$C = [P_{i-1} \quad P_i \quad T_{i+1} \quad T_{i+2}]^T$$

5.4.4 Skin deformation (or skinning) in COLLADA

5.4.4.1 General

Skinning is a technique for deforming geometry by linearly weighting vertices to a set of transformations, represented by `<node>` elements. Nodes that affect a particular geometry are usually organized into a single hierarchy called a “skeleton,” although the influencing nodes may come from unrelated parts of the hierarchy. The nodes of such a hierarchy represents the “joints” of the skeleton.

NOTE The term “joints” should not be confused with the term “bones,” which are the imaginary line segments connecting two joints.

This subclause provides a description of and equations for skinning in COLLADA.

A skinning `<controller>` associates a geometry with a skeleton. The skeleton is considered to be in its resting position or bind pose. The bind pose is the world-space position and orientation of each joint when the skeleton was bound to the geometry. This world space is also called the bind-pose space to distinguish it from other world-space coordinate systems.

A skinning `<instance_controller>` instantiates a skinning `<controller>` and associates it with a run-time skeleton. COLLADA defines skinning in object space, so the `<instance_controller>`'s placement in the `<node>` hierarchy contributes to the final vertex location. Object-space skinning provides the maximum amount of flexibility. The output of object-space skinning is vertices in the object space of the `<node>` coordinate system that contains the `<instance_controller>`.

When vertices are skinned in object space, it is easy and efficient to render the same skinned geometry in multiple locations. This is important when multiple actors are displayed simultaneously in the same pose but in different locations. Events like this happen most frequently in the animation of large crowds, parallel machines, and multi actor choreography. Each actor in the same pose shares the same skinned vertex data.

5.4.4.2 Skinning definitions

Definitions related to skinning in COLLADA:

- **Bind shape:** The vertices of the mesh referred to by the `source` attribute of the `<skin>` element;
- **Bind-shape matrix:** A single matrix that represents the transform of the bind-shape at the time when the mesh was bound to a skeleton. This matrix transforms the bind-shape from object space to bind-space;
- **Joints:** The bones of a skeleton are defined by their joints; the base of each bone extends to the next joint. In bind space, joints are in their bind pose: the position and orientation at the time the joints of the skeleton were bound to the bind shape. In the `<visual_scene>`, the joints are oriented according to the poses and animations of the actor. The world-space location of the joints may not directly match the mesh; it is dependent on the root matrix used to convert the mesh back into object-space;
- **Weights:** How much a joint influence the final destination of a vertex. A vertex is typically weighted to one or more joints, where the sum of the weights equals 1. A vertex is transformed by each joint independently. The multiply transformed vertex results are linearly combined according to their weights to generate the skinned vertex;
- **Inverse bind-pose matrix:** The inverse of the joint's bind-space transformation matrix at the time the bind shape was bound to this joint.

5.4.4.3 Skinning equations

The skinning calculation for each vertex v in a bind shape is:

$$outv = \sum_{i=0}^n \{ (v * BSM) * IBM_i * JM_i * JW \}$$

where:

- n is the number of joints that influence vertex v ;
- BSM is bind-shape matrix;
- IBM_i is inverse bind-pose matrix of joint i ;
- JM_i is transformation matrix of joint i ;
- JW is weight of the influence of joint i on vertex v .

NOTE v , BSM , IBM_i , and JW are constants with regards to some skeletal animation. Depending on the application used, it can be beneficial to premultiply BSM with IBM_i or v with BSM .

5.4.4.4 Equation notes

The main difference between world-space skinning and object-space skinning lies in the definition of JMi:

- For world-space skinning, JMi is the transformation matrix of the joint from object space to world space;
- For object-space skinning, JMi is a transformation matrix of the joint from object space to another object space. The first object-space transformation is the geometry's object-space transformation where the bind shape was originally defined. The second object-space transformation is the destination object space, which is selected by the `<instance_controller> <skeleton>`.

It is easiest to conceptualize this transformation by considering the other spaces that may fall between these spaces to construct this final matrix. One method is to go from geometry object space to world space as you might see with world-space skinning, then transform from world space to the skeleton's object space using the inverse of the skeleton's world-space matrix.

It is important to note that the skeleton's matrix referred to here is not the bind-shape matrix. It is the `<node>` in the `<visual_scene>` referenced by `<instance_controller> <skeleton>` and that might not have the same values. Using the `<node>` referenced by `<instance_controller> <skeleton>` provides maximum flexibility for locating and animating skeletons in the scene. It removes all restrictions over the bind space of the skin and the object space of the skeleton in the scene. This is because the animation is always relative to what you pick as the root node.

If the bind-shape matrix was hypothetically used instead, then the skeleton would always have to be located and animated relative to the bind-shape matrix's location and orientation in the scene. If multiple characters were animated at once, this can be disorienting because there is a high probability of overlap. It is worth noting that the node's world-space matrix, referenced by `<instance_controller><skeleton>`, can be equal to a skin's bind-shape matrix and that would match the behaviour just mentioned, or it can be equal to an identity matrix to match the behaviour of world-space skinning. Enabling these options makes object-space skinning the most flexible model.

The result of the preceding equation is a vertex in skeleton-relative object space, so it shall still be multiplied by a transform from object space to world space to produce the final vertex. This last step is typically done in a vertex shader and this matrix is the world-space transformation matrix for the node that owns the `<instance_controller>`.

There is a simple trick to animating a skeleton and its `<instance_controller>` simultaneously. If you place the `<instance_controller>` inside the root of `<skeleton>` then the last two matrices cancel each other, which gives a solution much like world-space skinning. The mesh always follows the skeleton.

5.5 Core elements reference

5.5.1 General

This clause covers the core elements that represent the basic functionality and infrastructure of the COLLADA schema, outside of the effects (FX) and physics frameworks. This includes elements that describe geometry, animation, skinning, assets, and scenes.

5.5.2 Elements by category

5.5.2.1 General

This subclause lists elements in alphabetical order. The following tables list elements by category, for ease in finding related elements.

5.5.2.2 Animation

Element	Description
animation	Categorizes the declaration of animation information.
animation_clip	Defines a section of the animation curves to be used together as an animation clip.
channel	Declares an output channel of an animation.
instance_animation	Instantiates a COLLADA animation resource.
library_animation_clips	Provides a library in which to place <animation_clip> elements.
library_animations	Provides a library in which to place <animation> elements.
sampler	Declares an interpolation sampling function for an animation.

5.5.2.3 Camera

Element	Description
camera	Declares a view into the scene hierarchy or scene graph. The camera contains elements that describe the camera's optics and imager.
imager	Represents the image sensor of a camera (for example, film or CCD).
instance_camera	Instantiates a COLLADA camera resource.
library_cameras	Provides a library in which to place <camera> elements.
optics	Represents the apparatus on a camera that projects the image onto the image sensor.
orthographic	Describes the field of view of an orthographic camera.
perspective	Describes the field of view of a perspective camera.

5.5.2.4 Controller

Element	Description
controller	Categorizes the declaration of generic control information.
instance_controller	Instantiates a COLLADA controller resource.
joints	Associates joint, or skeleton, nodes with attribute data.
library_controllers	Provides a library in which to place <controller> elements.
morph	Describes the data required to blend between sets of static meshes.
skeleton	Indicates where a skin controller is to start searching for the joint nodes that it needs.
skin	Contains vertex and primitive information sufficient to describe blend-weight skinning.
targets	Declares morph targets, their weights, and any user-defined attributes associated with them.
vertex_weights	Describes the combination of joints and weights used by a skin.

5.5.2.5 Data flow

Element	Description
accessor	Declares an access pattern to one of the array elements <float_array> , <int_array> , <Name_array> , <bool_array> , and <IDREF_array> .
bool_array	Declares the storage for a homogenous array of Boolean values.
float_array	Declares the storage for a homogenous array of floating-point values.
IDREF_array	Declares the storage for a homogenous array of ID reference values.
int_array	Stores a homogenous array of integer values.
Name_array	Stores a homogenous array of symbolic name values.
param (Core)	Declares parametric information for its parent element.
SIDREF_array	Declares the storage for a homogenous array of scoped-identifier reference values.
Source	Declares a data repository that provides values according to the semantics of an <input> element that refers to it.
input (shared)	Declares the input semantics of a data source.
input (unshared)	Declares the input semantics of a data source.

5.5.2.6 Extensibility

Element	Description
extra	Provides arbitrary additional information about or related to its parent element.
technique (Core)	Declares the information used to process some portion of the content. Each technique conforms to an associated profile.
technique_common	Specifies the information for a specific element for the common profile that all COLLADA implementations shall support.

5.5.2.7 Geometry

Element	Description
control_vertices	Describes the control vertices (CVs) of a spline.
geometry	Describes the visual shape and appearance of an object in a scene.
instance_geometry	Instantiates a COLLADA geometry resource.
library_geometries	Provides a library in which to place <geometry> elements.
lines	Declares the binding of geometric primitives and vertex attributes for a <mesh> element.
linestrips	Declares a binding of geometric primitives and vertex attributes for a <mesh> element.
mesh	Describes basic geometric meshes using vertex and primitive information.
polygons	Declares the binding of geometric primitives and vertex attributes for a <mesh> element.
polylist	Declares the binding of geometric primitives and vertex attributes for a <mesh> element.
spline	Describes a multisegment spline with control vertex (CV) and segment information.
triangles	Provides the information needed to bind vertex attributes together and then organize those vertices into individual triangles.
trifans	Provides the information needed to bind vertex attributes together and then organize those vertices into connected triangles.
tristrips	Provides the information needed to bind vertex attributes together and then organize those vertices into connected triangles
vertices	Declares the attributes and identity of mesh-vertices.

5.5.2.8 Lighting

Element	Description
ambient (Core)	Describes an ambient light source.
color	Describes the color of its parent light element.
directional	Describes a directional light source.
instance_light	Instantiates a COLLADA light resource.
library_lights	Provides a library in which to place <image> elements.
light	Declares a light source that illuminates a scene.
point	Describes a point light source.
spot	Describes a spot light source.

5.5.2.9 Mathematics

Element	Description
formula	Defines a formula.
instance_formula	Instantiates a COLLADA formula resource.
library_formulas	Provides a library in which to place <formula> elements.

5.5.2.10 Metadata

Element	Description
asset	Defines asset-management information regarding its parent element.
COLLADA	Declares the root of the document that contains some of the content in the COLLADA schema.
contributor	Defines authoring information for asset management.
geographic_location	Defines an asset's location for asset management.

5.5.2.11 Parameters

Element	Description
newparam	Creates a new, named parameter object and assigns it a type and an initial value.
param (reference)	References a predefined parameter.
setparam	Assigns a new value to a previously defined parameter.

5.5.2.12 Scene

Element	Description
evaluate_scene	Declares information specifying how to evaluate a <visual_scene>.
instance_node	Instantiates a COLLADA node resource.
instance_visual_scene	Instantiates a COLLADA visual_scene resource.
library_nodes	Provides a library in which to place <node> elements.
library_visual_scenes	Provides a library in which to place <visual_scene> elements.
node	Embodies the hierarchical relationship of elements in a scene.
Scene	Embodies the entire set of information that can be visualized from the contents of a COLLADA resource.
visual_scene	Embodies the entire set of information that can be visualized from the contents of a COLLADA resource.

5.5.2.13 Transform

Element	Description
lookat	Contains a position and orientation transformation suitable for aiming a camera.
matrix	Describes transformations that embody mathematical changes to points within a coordinate system or the coordinate system itself.
rotate	Specifies how to rotate an object around an axis.
scale	Specifies how to change an object's size.
skew	Specifies how to deform an object along one axis.
translate	Changes the position of an object in a coordinate system without any rotation.

5.5.3 accessor

Category: Data flow

5.5.3.1 General

Describes a stream of values from an array data source.

5.5.3.2 Concepts

The **<accessor>** element declares an access pattern into one of the array elements **<float_array>**, **<int_array>**, **<Name_array>**, **<bool_array>**, and **<IDREF_array>** or into an external array source. The arrays can be organized in either an interleaved or non-interleaved manner, depending on the offset and stride attributes.

The output of the accessor is described by its child **<param>** elements.

5.5.3.3 Attributes

count	uint_type	The number of times the array is accessed. Required.
offset	uint_type	The index of the first value to be read from the array. The default is 0. Optional.
source	xs:anyURI	The location of the array to access using a URI expression. Required. This element may refer to a COLLADA array element or to an array data source outside the scope of the instance document; the source does not need to be a COLLADA document.
stride	uint_type	The number of values that are to be considered a unit during each access to the array. The default is 1, indicating that a single value is accessed. Optional.

5.5.3.4 Related elements.

The **<accessor>** element relates to the following elements:

Parent elements	Source / technique_common
Child elements	See the following subclause.
Other	bool_array, float_array, IDREF_array, int_array, Name_array, mesh, convex_mesh, SIDREF_array

5.5.3.5 Child elements

Name/example	Description	Default	Occurrences
<param> (data flow)	The type attribute of the <param> element, when it is a child of the <accessor> element, is restricted to the set of array types: int, float, Name, bool, IDREF, and SIDREF. See main entry.	N/A	0 or more

5.5.3.6 Details

The number and order of **<param>** elements define the output of the **<accessor>** element. Parameters are bound to values in the order in which both are specified. No reordering of the data can occur. A **<param>** element without a name attribute indicates that the value is not part of the output, so the element is unbound.

The stride attribute shall have a value equal to or greater than the number of **<param>** elements. If there are fewer **<param>** elements than indicated by the stride value, the unbound array data source values are skipped.

5.5.3.7 Example

The following is an example of a basic **<accessor>** element and its illustration is in Figure 5.

```
<int_array name="values" count="9">
  200 210 220 230 240 250 260 270 280
</int_array>
<technique_common>
  <accessor source="#values" count="9">
    <param name="Adata" type="int"/>
  </accessor>
</technique_common>
</source>
```

default stride = 1										
	array offset	0	1	2	3	4	5	6	7	8
	values	200	210	220	230	240	250	260	270	280
accessor count	1st pass	Adata								
	2nd pass		Adata							
	3rd path			Adata						
	...									
	9th path									Adata

Figure 5 — Illustration of <accessor> element example

The following is an example of an <accessor> element that describes a stream of three pairs of integer values, while skipping every second value in the array because the second <param> element has no name attribute as illustration in Figure 6:

```
<source>
  <int_array name="values" count="9">
    200 210 220 230 240 250 260 270 280
  </int_array>
  <technique_common>
    <accessor source="#values" count="3" stride="3">
      <param name="Adata" type="int"/>
      <param type="int"/>
      <param name="Bdata" type="int"/>
    </accessor>
  </technique_common>
</source>
```

stride = 3, Adata, int, Bdata										
	array offset	0	1	2	3	4	5	6	7	8
	values	200	210	220	230	240	250	260	270	280
accessor count	1st pass	Adata		Bdata						
	2nd pass				Adata		Bdata			
	3rd pass							Adata		Bdata

Figure 6 — Illustration of <acceesor> element example with stride=3

The following is another example showing every third value being skipped because there is no <param> element binding it to the output although the stride attribute is still three and its illustration is in Figure 7:

```
<source>
  <int_array name="values" count="9">
    200 210 220 230 240 250 260 270 280
  </int_array>
  <technique_common>
    <accessor source="#values" count="3" stride="3">
      <param name="Bdata" type="int"/>
      <param name="Adata" type="int"/>
    </accessor>
  </technique_common>
</source>
```


stride = 3, Bdata, Adata										
	array offset	0	1	2	3	4	5	6	7	8
	values	200	210	220	230	240	250	260	270	280
accessor count	1st pass	Bdata	Adata							
	2nd pass				Bdata	Adata				
	3rd pass							Bdata	Adata	

Figure 7 — Illustration of basic <accessor> element example with stride=3, order=Bdata,Adata

In this example, every third value is skipped because there is no <param> element binding it to the output, although the stride attribute is still three. It also disregards the first three values (because the offset=3 begins at the fourth value) and the last three values (because the count is only 4, so only 12 values are read). See Figure 8.

```

<source>
  <int_array name="values" count="18">
    20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
  </int_array>
  <technique_common>
    <accessor source="#values" offset="3" count="4" stride="3">
      <param name="Ad" type="int"/>
      <param name="Cd" type="int"/>
    </accessor>
  </technique_common>
</source>

```

count=4 stride = 3, offset=3, Ad, Cd																
	array offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	values	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
accessor count	1st pass				Ad	Cd										
	2nd pass							Ad	Cd							
	3rd pass										Ad	Cd				
	4th pass													Ad	Cd	

Figure 8 — Illustration basic <accessor> element with offset=3

Semantics in an <input> imply a specific data ordering in a source (such as X, Y, Z or R, G, B); the actual names of the <param>s in the <source>'s <accessor> are not significant. The names in <param>s do not imply any kind of binding, but the absence of a name or whole <param> (if it is at the end of the list) indicates data to be skipped.

To properly read a source through an <accessor>, the program has to consider the data expected by a particular semantic and compare it to stride, offset, and the number of params with nonnull names to decide how many values to read. Then, when reading, it has to skip over the data that corresponds to <param>s with no name.

Assume that the program has a vertex-map-like array that it is trying to fill with geometry:

```

struct
{
    float x_pos, y_pos, z_pos, x_norm, y_norm, z_norm, text1_U,
    text1_V, tex2_U tex2_V;
} my_array[1000];

```

Given this source:

```
<source id=test1>
  <float_array name="values" count="9">
    1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
  </float_array>
  <technique_common>
    <accessor source="#values" count="3" stride="3">
      <param name="A" type="float"/>
      <param name="F" type="float"/>
      <param name="X" type="float"/>
    </accessor>
  </technique_common>
</source>
```

with the following input:

```
<triangles count="1">
  <input semantic="POSITION" source="#test1" offset="0"/>
  <p>0 1 2</p>
```

If the data is read into my_array sequentially, because the stride of the accessor is 3 and all the **<param>**s have names, the **<source>** is assumed to contain 3D positions and my_array would be filled in like this:

x_pos	y_pos	z_pos	x_norm	y_norm	z_norm	tex1_U	tex1_V	tex2_U	tex2_V
1.0	2.0	3.0							
4.0	5.0	6.0							
7.0	8.0	9.0							

Change the accessor to this:

```
<accessor source="#values" count="3" stride="3">
  <param name="A" type="float"/>
  <param type="float"/>
  <param name="X" type="float"/>
</accessor>
```

Because the second **<param>** has no name, it is skipped. With only two named **<param>**s, the **<source>** is assumed to contain 2D positions and is read like this:

x_pos	y_pos	z_pos	x_norm	y_norm	z_norm	tex1_U	tex1_V	tex2_U	tex2_V
1.0	3.0								
4.0	6.0								
7.0	9.0								

If the equivalent of an entire vertex array is packed into one floating-point array:

```
<source id=positions>
  <float_array name="values" count="30">
    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
    29 30
  </float_array>
  <technique_common>
    <accessor source="#values" count="3" stride="10">
```

```

        <param name="A" type="float"/>
        <param name="F" type="float"/>
        <param name="X" type="float"/>
    </accessor>
</technique_common>
</source>

<source id="normals">
    <technique_common>
        <accessor source="#values" count="3" stride="10">
            <param type="float"/>
            <param type="float"/>
            <param type="float"/>
            <param name="A" type="float"/>
            <param name="F" type="float"/>
            <param name="X" type="float"/>
        </accessor>
    </technique_common>
</source>

<source id="texture1">
    <technique_common>
        <accessor source="#values" count="3" stride="10">
            <param type="float"/>
            <param type="float"/>
            <param type="float"/>
            <param type="float"/>
            <param type="float"/>
            <param type="float"/>
            <param name="A" type="float"/>
            <param name="F" type="float"/>
        </accessor>
    </technique_common>
</source>
<source id="texture2">
    <technique_common>
        <accessor source="#values" count="3" stride="10">
            <param type="float"/>
            <param type="float"/>
            <param type="float"/>
            <param type="float"/>
            <param type="float"/>
            <param type="float"/>
            <param type="float"/>
            <param name="F" type="float"/>
            <param name="X" type="float"/>
        </accessor>
    </technique_common>
</source>

<triangles count="1">
    <input semantic="POSITION" source="#positions" offset="0"/>
    <input semantic="NORMAL" source="#normals" offset="0"/>
    <input semantic="TEXCOORD" source="#texture1" offset="0"/>
    <input semantic="TEXCOORD" source="#texture2" offset="0"/>
    <p>1 2 3</p>

```

Based on the **<param>** count in each **<accessor>**, it is assumed that it is a reading of 3D positions, 3D normals, and 2D texture coordinates.

x_pos	y_pos	z_pos	x_norm	y_norm	z_norm	tex1_U	tex1_V	tex2_U	tex2_V
1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
11.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	20.0
21.0	22.0	23.0	24.0	25.0	26.0	27.0	28.0	29.0	30.0

Note that the **<accessor>** **offset** attribute can also be used to skip leading fields of data. For example, the **<accessor>** in the source id=texture1 in the preceding example could be written the following way and it would work the same way:

```
<accessor source="#values" count="3" stride="10" offset="6">
  <param name="A" type="float"/>
  <param name="F" type="float"/>
</accessor>
```

5.5.4 ambient

Category: Lighting

5.5.4.1 General

Describes an ambient light source.

NOTE There are two **<ambient>** variants; see also "fx_common_color_or_texture_type" in subclause 5.8 FX reference.

5.5.4.2 Concepts

The **<ambient>** element declares the parameters required to describe an ambient light source. An ambient light is one that lights everything evenly, regardless of location or orientation.

5.5.4.3 Attributes

The **<ambient>** element has no attributes.

5.5.4.4 Related elements

The **<ambient>** element has no attributes.

Parent elements	light / technique_common
Child elements	See the following subclause.
Other	None

5.5.4.5 Child elements

Name/example	Description	Default	Occurrences
<color>	Contains three floating-point numbers specifying the color of the light. See main entry.	None	1

5.5.4.6 Example

The following is an example of an **<ambient>** element:

```
<light id="blue">
  <technique_common>
    <ambient>
      <color>0.1 0.1 0.5</color>
    </ambient>
  </technique_common>
</light>
```

5.5.5 animation

Category: Animation

5.5.5.1 General

Categorizes the declaration of animation information.

5.5.5.2 Concepts

The animation hierarchy contains elements that describe the animation's key-frame data and sampler functions, ordered in such a way as to group animations that should be executed together.

Animation describes the transformation of an object or value over time. A common use of animation is to give the illusion of motion. A common animation technique is key-frame animation.

A key frame is a two-dimensional (2D) sampling of data. The first dimension is called the input and is usually time but can be any other real value. The second dimension is called the output and represents the value being animated. Using a set of key frames and an interpolation algorithm, intermediate values are computed for times between the key frames, producing a set of output values over the interval between the key frames. The set of key frames and the interpolation between them define a 2D function called an animation curve or function curve, represented by an **<animation>** element.

For more information about interpolating animation curves, see "Curve Interpolation" in Subclause 5.4 Programming guideline.

5.5.5.3 Attributes

The **<animation>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <animation> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.5.4 Related elements

The **<animation>** element relates to the following elements:

Parent elements	library_animations, animation
Child elements	See the following subclause.
Other	instance_animation

5.5.5.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<animation>	Allows the formation of a hierarchy of related animations. See main entry.	N/A	0 or more (see “Details”)
<Source> (Core)	See main entry.	N/A	0 or more (see “Details”)
<sampler>	Describes the interpolation sampling function for the animation. See main entry.	N/A	0 or more (see “Details”)
<channel>	Describes an output channel for the animation. See main entry.	None	0 or more (see “Details”)
<extra>	See main entry.	N/A	0 or more

5.5.5.6 Details

An **<animation>** element contains the elements that describe animation data to form an animation tree. The actual type and complexity of the data is represented in detail by the child elements.

The child elements follow these rules:

The **<animation>** element shall contain at least one of the following:

- **<animation>;**
- **<sampler>** and **<channel>;**
- **<sampler>** and **<channel>** shall always be used together.

<animation>s that are not referenced by **<animation_clip>** elements can be applied to the scene at playback time; otherwise, see **<animation_clip>**s for playback details.

See “Details” in **<animation_clip>** for information about resolving animation targets.

5.5.5.7 Example

The following is an example of an empty **<animation>** element with the allowed attributes:

```
<library_animations>
  <animation name="walk" id="Walk123">
    <!-- use appropriate id, source, target values -->
    <source id="..." />
    <source id="..." />
    <sampler> ... </sampler>
    <channel source="..." target="..." />
  </animation>
</library_animations>
```

This next example describes a simple animation tree defining a “jump” animation:

```
<library_animations>
  <animation name="jump" id="jump">
    <animation id="skeleton_root_translate">
      <source/><source/><sampler/><channel/>
    </animation>
    <animation id="left_hip_rotation">
      <source.../><source.../>
      <sampler>...</sampler><channel .../>
    </animation>
    <animation id="left_knee_rotation">
      <source .../><source .../>
      <sampler>...</sampler><channel .../>
    </animation>
    <animation id="right_hip_rotation">
      <source .../><source .../>
      <sampler>...</sampler><channel .../>
    </animation>
    <animation id="right_knee_rotation">
      <source .../><source .../>
      <sampler>...</sampler><channel .../>
    </animation>
  </animation>
</library_animations>
```

The next example shows a more complex animation tree, with some of the animations left undefined.

```
<library_animations>
  <animation name="elliots_animations" id="all_elliott">
    <animation name="elliott's spells" id="spells_elliott">
      <animation id="elliott_fire_blast"/>
      <animation id="elliott_freeze_down"/>
      <animation id="elliott_ferocity"/>
    </animation>
    <animation name="elliott's moves" id="moves_elliott">
      <animation id="elliott_walk"/>
      <animation id="elliott_run"/>
      <animation id="elliott_jump"/>
    </animation>
  </animation>
</library_animations>
```

5.5.6 animation_clip

Category: Animation

5.5.6.1 General

Defines a section of a set of animation curves and/or formulas to be used together as an animation clip.

5.5.6.2 Concepts

Animation clips can be used to separate different pieces of a set of animation curves, formulas, or both. For example, an animation might have a character walk, then run. The walking and running animations can be separated as two different clips. Clips can also be used to separate the animations and formulas of different characters in the same scene, or even different parts of the same character (such as upper and lower body).

Currently, animation clips cannot be instantiated inside a COLLADA document. They are for use by engines and other tools.

5.5.6.3 Attributes

The **<animation_clip>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <animation> element. This value shall be unique within the instance document. Optional.
start	xs:double	The time in seconds of the beginning of the clip. This time is the same as that used in the key-frame data and is used to determine which set of key frames will be included in the clip. The start time does not specify when the clip will be played. If the time falls between two key frames of a referenced animation, an interpolated value shall be used. The default is 0.0. Optional.
end	xs:double	The time in seconds of the end of the clip. This is used in the same way as the start time. If end is not specified, the value is taken to be the end time of the longest animation. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.6.4 Related elements

The **<animation_clip>** element relates to the following elements:

Parent elements	library_animation_clips
Child elements	See the following subclause.
Other	None

5.5.6.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<instance_animation>	See main entry.	N/A	1 or more
<instance_formula>	See main entry.	N/A	1 or more
<extra>	See main entry.	N/A	0 or more

5.5.6.5.1 Animation targets and scene

Two or more `<animation_clip>`'s can refer to `<animation>`s or `<formula>`s that have the same target; in addition, it is possible to have an `<animation>` or a `<formula>` with the same target that is not referred to by an `<animation_clip>` but is meant to be applied and used when the playback occurs.

`<animation>`s and `<formula>`s that are referred to and used in `<animation_clip>`s shall not be applied to the scene at playback time; instead, apply only unreferenced `<animation>`s or `<formula>`s to the scene (used for playback).

Plug-in implementors shall support this strategy even if they do not fully support `<animation_clip>`. For example, DCC tools can store the contents of `<library_animations>` and `<library_animation_clips>` in banks or palettes. Any unreferenced `<animation>` or `<formula>` is left to be processed according to the application run-time; these are the ones to load and play.

5.5.6.6 Example

The following is an example of two `<animation_clip>` elements with the allowed attributes:

```
<library_animation_clips>
  <animation_clip id="GuyWalking" start="0.25" end="1.25">
    <instance_animation url="#Guy1MoveAnim"/>
  </animation_clip>
  <animation_clip id="GuyRunning" start="2.5" end="4.5">
    <instance_animation url="#Guy1MoveAnim"/>
    <instance_animation url="#Guy1BreatheAnim"/>
  </animation_clip>
</library_animation_clips>
```

5.5.7 asset

Category: Metadata

5.5.7.1 General

Defines asset-management information regarding its parent element.

5.5.7.2 Concepts

Computers store vast amounts of information. An asset is a set of information that is organized into a distinct collection and managed as a unit. A wide range of attributes describes assets so that the information can be maintained and understood both by software tools and by humans. Asset information is often hierarchical, where the parts of a large asset are divided into smaller pieces that are managed as distinct assets themselves.

5.5.7.3 Attributes

The `<asset>` element has no attributes.

5.5.7.4 Related elements

The **<asset>** element relates to the following elements:

Parent elements	In core: animation , animation_clip , camera , COLLADA , controller , evaluate_scene , extra , geometry , light , node , Source , visual_scene In FX: material , image , effect , profile_* (see "Profiles"), technique (FX) (in profile_CG , profile_COMMON , and profile_GLES) In Physics: force_field , physics_material , physics_scene , physics_model In Kinematics: kinematics_scene In all clauses: library_*
Child elements	See the following subclause.
Other	None

5.5.7.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<contributor>	Provides data related to a contributor who worked on the parent element. See main entry.	N/A	0 or more
<coverage> <geographic_location> ... </geographic_location> </coverage>	Provides information about the location of the visual scene in physical space. Element has no attributes, but it can contain 0 or 1 <geographic_location> child elements. See <geographic_location> main entry.	N/A	0 or 1
<created>	Contains the date and time that the parent element was created. Represented in an ISO 8601 format as per the XML Schema xs:dateTime primitive type. Element has no attributes.	None	1
<keywords>	Contains a list of words used as search criteria for the parent element. Element has no attributes.	None	0 or 1
<modified>	Contains the date and time that the parent element was last modified. Represented in an ISO 8601 format [1] as per the XML Schema xs:dateTime primitive type. Element has no attributes.	None	1
<revision>	Contains revision information for the parent element. Element has no attributes.	None	0 of 1
<subject>	Contains a description of the topical subject of the parent element. Element has no attributes.	None	0 or 1
<title>	Contains title information for the parent element. Element has no attributes.	None	0 or 1

Name/example	Description	Default	Occurrences
<pre><unit> meter=... name=... /></pre>	Defines unit of distance for COLLADA elements and objects. This unit of distance applies to all spatial measurements within the scope of <asset>'s parent element, unless overridden by a more local <asset>/<unit>. The value of the unit is self-describing and does not have to be consistent with any real-world measurement. Its optional attributes are: • name: The name (xs:NMTOKEN) of the distance unit. For example, "meter", "centimeter", "inches", or "parsec". This can be the real name of a measurement, or an imaginary name. • meter: How many real-world meters in one distance unit as a floating-point number. For example, 1.0 for the name "meter"; 1000 for the name "kilometer"; 0.3048 for the name "foot". For more information, see "About physical units" in "0 Physics Reference". • name: meter	name: meter meter: 1.0	0 or 1
<up_axis>	Contains descriptive information about the coordinate system of the geometric data. All coordinates are right-handed by definition. Valid values are X_UP , Y_UP or Z_UP . This element specifies which axis is considered upward, which is considered to the right, and which is considered inward. See "Details." Element has no attributes.	Y_UP	0 or 1
<extra>	See main entry.	N/A	0 or more

5.5.7.6 Details

In the case of hierarchical **<asset>** elements, where both the parent and child assets supply a value for the same metadata, such as for **<unit>**, the child asset's value supersedes the parent's value within the scope of the child element. This applies recursively.

5.5.7.6.1 Up axis values

The **<up_axis>** element's values have the following meanings:

Value	Right Axis	Up Axis	In Axis
X_UP	Negative y	Positive x	Positive z
Y_UP	Positive x	Positive y	Positive z
Z_UP	Positive x	Positive z	Negative y

5.5.7.7 Example

The following is an example of an **<asset>** element that describes the parent **<COLLADA>** element, and hence the entire document:

```
<COLLADA>
  <asset>
    <created>2005-06-27T21:00:00Z</created>
    <keywords>COLLADA interchange</keywords>
    <modified>2005-06-27T21:00:00Z</modified>
    <unit name="nautical_league" meter="5556.0" />
    <up_axis>Z_UP</up_axis>
  </asset>
</COLLADA>
```

5.5.8 bool_array

Category: Data flow

5.5.8.1 General

Declares the storage for a homogenous array of Boolean values.

5.5.8.2 Concepts

The **<bool_array>** element stores data values for generic use within the COLLADA schema. The arrays themselves are strongly typed but without semantics. They simply describe a sequence of XML Boolean values.

5.5.8.3 Attributes

The **<bool_array>** element has the following attributes:

count	uint_type	The number of values in the array. Required.
id	xs:ID	A text string containing the unique identifier of this element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.8.4 Related elements

The **<bool_array>** element relates to the following elements:

Parent elements	Source (Core)
Child elements	None
Other	accessor

5.5.8.5 Details

A **<bool_array>** element contains a list of XML Boolean values. These values are a repository of data for **<source>** elements.

5.5.8.6 Example

The following is an example of a **<bool_array>** element that describes a sequence of four Boolean values:

```
<bool_array id="flags" name="myFlags" count="4">
  true true false false
</bool_array>
```

5.5.9 camera

Category: Camera

5.5.9.1 General

Declares a view of the visual scene hierarchy or scene graph. The camera contains elements that describe the camera's optics and imager.

5.5.9.2 Concepts

A camera embodies the eye point of the viewer looking at the visual scene. It is a device that captures visual images of a scene. A camera has a position and orientation in the scene. This is the viewpoint of the camera as seen by the camera's optics or lens.

The camera optics focuses the incoming light onto an image. The image is focused onto the plane of the camera's imager or film. The imager records the resulting image.

5.5.9.3 Attributes

The **<camera>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <camera> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.9.4 Related elements

The **<camera>** element relates to the following elements:

Parent elements	library_cameras
Child elements	See the following subclause.
Other	instance_camera

5.5.9.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	Defines the directions of the axes and the units of measurement for the camera's view. Also contains information about the creation of this element. See main entry.	N/A	0 or 1
<optics>	Describes the field of view and viewing frustum using canonical parameters. See main entry.	N/A	1
<imager>	Represents the image sensor of a camera (for example, film or CCD). See main entry.	N/A	0 or 1
<extra>	See main entry.	N/A	0 or more

5.5.9.6 Details

For simple cameras, a generic technique needs to contain only an optics element.

The camera is defined such that the local +X axis is to the right, the lens looks towards the local -Z axis, and the top of the camera is aligned with the local +Y axis (also see the <lookat> element). This orientation is affected by the <asset> element's <up_axis> value.

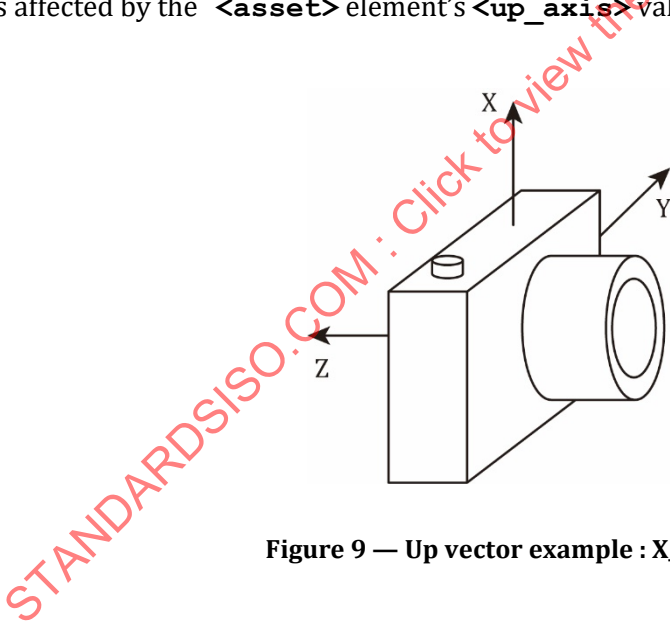


Figure 9 — Up vector example : X_UP

5.5.9.7 Example

The following is an example of a <camera> element that describes a perspective view of a scene with a 45-degree field of view:

```
<camera name="eyepoint">
  <optics>
    <technique_common>
      <perspective>
        <yfov>45</yfov>
        <aspect_ratio>1.33333</aspect_ratio>
        <znear>1.0</znear>
        <zfar>1000.0</zfar>
```

```

    </perspective>
  </technique_common>
</optics>
</camera>

```

5.5.10 channel

Category: Animation

5.5.10.1 General

Declares an output channel of an animation.

5.5.10.2 Concepts

As an animation's sampler transforms values over time, those values are directed out to channels. The animation channels describe where to store the transformed values from the animation engine. The channels target the data structures that receive the animated values.

5.5.10.3 Attributes

The **<channel>** element has the following attributes:

source	urifragment_type	The location of the animation sampler using a URL expression. Required.
target	sidref_type	A reference to the SID of the element bound to the output of the sampler. This text string is a path name following a simple syntax described in the "Address syntax" subclause in 5.3 Schema concepts. Required.

5.5.10.4 Related elements

The **<channel>** element relates to the following elements:

Parent elements	<animation>
Child elements	None
Other	None

5.5.10.5 Details

This element encloses no data.

5.5.10.6 Example

The following is an example of a **<channel>** element that targets the translated values of an element whose id is "Box":

```

<animation>
  <channel source="#Box-Translate-X-Sampler" target="Box/Trans.X"/>
  <channel source="#Box-Translate-Y-Sampler" target="Box/Trans.Y"/>
  <channel source="#Box-Translate-Z-Sampler" target="Box/Trans.Z"/>
</animation>

```

5.5.11 COLLADA

Category: Metadata

5.5.11.1 General

Declares the root of the document that contains some of the content in the COLLADA schema.

5.5.11.2 Concepts

The COLLADA schema is XML based; therefore, it shall have exactly one document root element or document entity to be a well-formed XML document. The COLLADA element serves that purpose.

5.5.11.3 Attributes

The **<COLLADA>** element has the following attributes:

version	Enumeration	The COLLADA schema revision with which the instance document conforms. The only valid value is 1.5.0. Required.
xmlns	xs:anyURI	This XML Schema namespace attribute applies to this element to identify the schema for an instance document.
base	xs:anyURI	The XML Base specification describes a facility, similar to that of HTML BASE, for defining base URIs for parts of XML documents. It defines a single attribute, xml:base, and describes in detail the procedure for its use in processing relative URI references. Refer to [6].

5.5.11.4 Related elements

The **<COLLADA>** element relates to the following elements:

Parent elements	No parent elements
Child elements	See the following subclause.
Other	None

5.5.11.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	1
library_element	Any quantity and combination of any library elements can appear in any order: <library_animation_clips> <library_animations> <library_articulated_systems> (in Kinematics) <library_cameras> <library_controllers> <library_effects> (in FX) <library_force_fields> (in Physics) <library_formulas> <library_geometries> <library_images> (in FX) <library_joints> (in Kinematics) <library_kinematics_models> (in Kinematics) <library_kinematics_scenes> (in Kinematics) <library_lights> <library_materials> (in FX) <library_nodes> <library_physics_materials> (in Physics) <library_physics_models> (in Physics) <library_physics_scenes> (in Physics) <library_visual_scenes> See main entries.	N/A	0 or more
<Scene>	See main entry.	N/A	0 or 1
<extra>	See main entry.	N/A	0 or more

5.5.11.6 Details

The **<COLLADA>** element is the document entity (root element) in a COLLADA instance document.

5.5.11.7 Example

The following example outlines an empty COLLADA instance document whose schema version is "1.5.0":

```
<?xml version="1.0" encoding="utf-8"?>
<COLLADA xmlns="http://www.collada.org/2008/03/COLLADASchema" version="1.5.0">
  <asset>
    <created/>
    <modified/>
  </asset>
  <library_geometries/>
  <library_visual_scenes/>
  <scene />
</COLLADA>
```

5.5.12 color

Category: **Lighting**

5.5.12.1 Concepts

In the context of **<light>**, the **<color>** element contains three floating-point values describing the RGB color of its parent light element.

In the context of **<profile_COMMON>**, it contains four floating-point values describing the RGBA color of its parent element.

5.5.12.2 Attributes

The **<color>** element has the following attribute:

sid	sid_type	Optional. For <profile_COMMON> parent elements only. A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. For details, see “Address syntax” in 5.3 Schema concepts.

5.5.12.3 Related elements

The **<color>** element relates to the following elements:

Parent elements	In <light> : ambient (Core), directional, point, spot In <profile_COMMON> : elements of type fx_common_color_or_texture_type (ambient, emission, diffuse, reflective, specular, transparent)
Child elements	None
Other	None

5.5.13 contributor

Category: Metadata

5.5.13.1 General

Defines authoring information for asset management.

5.5.13.2 Concepts

In modern production pipelines, especially as art teams are steadily increasing in size, it is becoming more likely that a single asset may be worked on by multiple authors, possibly even using multiple tools. This information may be important for an asset management system and its content format is application-defined.

5.5.13.3 Attributes

The **<contributor>** element has no attributes.

5.5.13.4 Related elements

The **<contributor>** element relates to the following elements:

Parent elements	asset
Child elements	See the following subclause.
Other	None

5.5.13.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<author>	Contains a string with the author's name. This element has no attributes.	None	0 or 1
<author_email>	Contains a string with the author's full email address compliant with RFC 2822 clause 3.4. This element has no attributes. [3]	None	0 or 1
<author_website>	Contains an <code>xs:anyURI</code> for the URL of this contributor's website. This element has no attributes.	None	0 or 1
<authoring_tool>	Contains a string with the name of the authoring tool. This element has no attributes.	None	0 or 1
<comments>	Contains a string with comments from this contributor. This element has no attributes.	None	0 or 1
<copyright>	Contains a string with copyright information. This element has no attributes.	None	0 or 1
<source_data>	Contains a URI reference (<code>xs:anyURI</code>) to the source data used for this asset. This element has no attributes.	None	0 or 1

5.5.13.6 Example

The following is an example of a **<contributor>** element for an asset:

```
<asset>
  <contributor>
    <author>Bob the Artist</author>
    <author_email>bob@bobartist.com</author_email>
    <author_website>http://www.bobartist.com</author_website>
    <authoring_tool>Super3DmodelMaker3000</authoring_tool>
    <comments>This is a big Tank</comments>
    <copyright>Bob's game shack: all rights reserved</copyright>
    <source_data>c:/models/tank.s3d</source_data>
  </contributor>
</asset>
```

5.5.14 controller

Category: Controller

5.5.14.1 General

Defines generic control information for dynamic content.

5.5.14.2 Concepts

A controller is a device or mechanism that manages and directs the operations of another object. A **<controller>** element is a general, generic mechanism for describing active or dynamic content. It contains elements that describe the manipulation of the data. The actual type and complexity of the data is represented in detail by the child elements.

COLLADA describes two types of controllers for active mesh geometry in the visual scene: vertex skinning and mesh morphing. The controller concept is not limited to geometry and visualization, however, and other types of controllers may be introduced in future versions of the specification, which describe animation blending, physical simulation, dynamics, or user interaction.

5.5.14.3 Attributes

The **<controller>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <controller> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.14.4 Related elements

The **<controller>** element relates to the following elements:

Parent elements	library_controllers
Child elements	See the following subclause.
Other	instance_controller

5.5.14.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
control_element	The element that contains control data. Shall be either: <skin> <morph> See main entries.	N/A	1
<extra>	See main entry.	N/A	0 or more

5.5.14.6 Details

The **<controller>** element is similar to the **<geometry>** element in how it is used and instantiated in the scope of a **<node>** element.

A **<morph>** element provides the information for a mesh morphing controller that deforms meshes and blends them.

A **<skin>** element provides the information for a vertex skinning controller that transforms vertices based on weighted influences to produce a smoothly changing mesh.

5.5.14.6.1 Multiple controller interaction

More than one controller can be applied simultaneously. To do this, a controller can use another controller as its source. When applying a controller, if the source of the controller currently being applied is another controller, that other controller (the source) shall be applied first, i.e. the controller execution starts with the one that has the non-controller object as its source (usually a geometry) and the controller's execution pipeline continues from there.

5.5.14.7 Example

The following is an example of an empty **<controller>** element with the allowed attributes:

```
<library_controllers>
  <controller name="skinner" id="skinner456">
    <skin/>
  </controller>
</library_controllers>
```

The following is a typical example where a **<morph>** controller is applied first and a **<skin>** is applied after:

```
<library_controllers>
  <controller id="controllers_0">
    <morph source="#geometries_0" method="NORMALIZED">
    </morph>
  </controller>
  <controller id="controllers_1">
    <skin source="#controllers_0">
    </skin>
  </controller>
</library_controllers>
```

5.5.15 control_vertices

Category: Geometry

5.5.15.1 General

Describes the control vertices (CVs) of a spline.

5.5.15.2 Concepts

Information about both a control vertex and its related segment are stored on the control vertices. Segment data applies to the spline segment that starts at the given control vertex.

Each control vertex shall provide a position. It is recommended to provide a source of interpolation methods to be used on the segment that starts at this control vertex. Otherwise, the linear interpolation is assumed. For Bézier and Hermite interpolations, input and output tangents shall be provided for each control vertex.

Additionally, each control vertex may have an arbitrary amount user-specific information, through custom data sources. The custom data sources shall provide enough data to have one value – however large – for each control vertex.

5.5.15.3 Attributes

The **<control_vertices>** element has no attributes.

5.5.15.4 Related elements

The **<control_vertices>** element relates to the following elements:

Parent elements	spline, nurbs, nurbs_surface
Child elements	See the following subclause.
Other	Source

5.5.15.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (unshared)	At least one <input> (unshared) element shall have a semantic attribute whose value is POSITION . See main entry.	None	1 or more
<extra>	See main entry.	N/A	0 or more

5.5.15.6 Details

COLLADA recognizes the following polynomial interpolation types for **<control_vertices>**: **LINEAR**, **BEZIER**, **CARDINAL**, **HERMITE**, and **BSPLINE**. These symbolic names are used in a **<Name_array>** within a **<Source>** element. These values are fed into the control vertices by an **<input>** element that includes a semantic attribute with a value of **POSITION**.

For more information, “Curve Interpolation” in 5.4 Programming guideline.

The COMMON profile defines the following **<input>** semantics for **<control_vertices>**:

<input> semantic value	Type	Description	Default
POSITION	any multidimensional floating-point	The position of the control vertex.	N/A
INTERPOLATION	xs:Name	The type of polynomial interpolation to represent the segment starting at the CV. Common-profile types are: LINEAR, BEZIER, HERMITE, CARDINAL, and BSPLINE.	LINEAR
IN_TANGENT	any multidimensional floating-point	The tangent that controls the shape of the segment preceding the CV (BEZIER and HERMITE). The number of dimensions to the values of this source shall match the number of dimensions in the POSITION source.	N/A
OUT_TANGENT	any multidimensional floating-point	The tangent that controls the shape of the segment following the CV (BEZIER and HERMITE). The number of dimensions to the values of this source shall match the number of dimensions in the POSITION source.	N/A
CONTINUITY	xs:Name	(Optional) Defines the continuity constraint at the CV. The common-profile types are C0, C1, G1.	N/A
LINEAR_STEPS	int_type	(Optional) The number of piece-wise linear approximation steps to use for the spline segment that follows this CV.	N/A

5.5.15.7 Details

For mixed interpolation splines, if at least one segment has the BEZIER or HERMITE interpolation type, then one IN_TANGENT value and one OUT_TANGENT value shall be provided for every control vertex.

The data types of all child elements shall be the same. For example, when the interpolation type is BEZIER or HERMITE:

- Valid: POSITION, IN_TANGENT, and OUT_TANGENT all defined as float2_type;
- Invalid: POSITION as float2_type and IN_TANGENT or OUT_TANGENT as float3_type.

There are constraints among child elements. For example, the quantity of POSITION child elements shall equal the quantity of INTERPOLATION elements. For details, see **<spline>**.

5.5.15.8 Example

See **<spline>**.

5.5.16 directional

Category: Lighting

5.5.16.1 General

Describes a directional light source.

5.5.16.2 Concepts

The **<directional>** element declares the parameters required to describe a directional light source. A directional light is one that lights everything from the same direction, regardless of location.

The light's default direction vector in local coordinates is [0,0,-1], pointing down the negative z axis. The actual direction of the light is defined by the transform of the node where the light is instantiated.

5.5.16.3 Attributes

The **<directional>** element has no attributes.

5.5.16.4 Related elements

The **<directional>** element relates to the following elements:

Parent elements	light / technique_common
Child elements	See the following subclause.
Other	None

5.5.16.5 Child elements

Name/example	Description	Default	Occurrences
<color>	Contains three floating-point numbers specifying the color of the light. See main entry.	None	1

5.5.16.6 Example

The following is an example of a **<directional>** element:

```
<light id="blue">
  <technique_common>
    <directional>
      <color>0.1 0.1 0.5</color>
    </directional>
  </technique_common>
</light>
```

5.5.17 evaluate_scene

Category: Scene

5.5.17.1 General

Declares information specifying how to evaluate a **<visual_scene>**.

5.5.17.2 Attributes

The **<evaluate_scene>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.
sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.
enable	xs:boolean	Whether evaluation is enabled. Disabling evaluation can be useful for debugging. The default is true. Optional.

5.5.17.3 Related elements

The **<evaluate_scene>** element relates to the following elements:

Parent elements	visual_scene
Child elements	See the following subclause.
Other	None

5.5.17.4 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	For asset management information. See main entry.	N/A	0 or 1
<render>	Describes one effects pass to render a scene. See main entry in FX.	N/A	0 or more
<extra>	See main entry.	N/A	0 or more

5.5.18 extra

Category: Extensibility

5.5.18.1 General

Provides arbitrary additional information about or related to its parent element.

5.5.18.2 Concepts

An extensible schema requires a means for users to specify arbitrary information. This extra information can represent additional real data or semantic (meta) data to the application.

COLLADA represents extra information as techniques containing arbitrary XML elements and data.

5.5.18.3 Attributes

The **<extra>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <extra> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.
Type	xs:NMTOKEN	A hint as to the type of information that the particular <extra> element represents. This text string shall be understood by the application. Optional.

5.5.18.4 Related elements

The **<extra>** element relates to the following elements:

Parent elements	In core: animation, animation_clip, camera, COLLADA, controller, control_vertices, evaluate_scene, geometry, imager, joints, light, lines, linestrips, mesh, morph, node, optics, polygons, polylist, Scene, skin, spline, targets, triangles, trifans, tristrips, vertex_weights, vertices, visual_scene In Physics: attachment, box, capsule, convex_mesh, cylinder, force_field, physics_material, physics_model, physics_scene, plane, ref_attachment, rigid_body, rigid_constraint, shape, sphere In FX: bind_material, effect, image, material, pass, profile_BRIDGE, profile_CG, profile_COMMON, profile_GLES, profile_GLES2, profile_GLSL, render, sampler_state, sampler_* (see "Texturing"), shader, technique (FX) (in profile_*), texture, texture_pipeline In B-Rep: brep, circle, cone, curves, cylinder (B-Rep), edges, ellipse, faces, hyperbola, line, nurbs, nurbs_surface, parabola, pcurves, shells, solids, surfaces, surface_curves, swept_surface, torus, wires In Kinematics: articulated_system, joint, kinematics, kinematics_model, kinematics_scene, motion All clauses: instance_*, library_*
Child elements	See the following subclause.
Other	None

5.5.18.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<technique> (Core)	See main entry.	N/A	1 or more

5.5.18.6 Example

The following is an example of an **<extra>** element that outlines both structured and unstructured additional content:

```
<geometry>
  <extra>
    <technique profile="Max" xmlns:max="some/max/schema">
      <param name="wow" sid="animated" type="string">a validated string parameter
from the COLLADA schema.</param>
      <max:someElement>defined in the Max schema and validated.</max:someElement>
      <uhoh>something well-formed and legal, but that can't be validated because
there is no schema for it!</uhoh>
    </technique>
  </extra>
</geometry>
```

The following example shows how **<extra>** and **<technique>** can work together:

```
<light>
  <!-- Application chooses one of the following three techniques -->
  <technique_common> ... </technique_common>
  <technique profile="ProductA"> ... </technique>
  <technique profile="ProductB"> ... </technique>
  <!-- Application chooses zero or more of the following two extras -->
  <!-- and one technique within each extra -->
  <extra type="basic">
    <technique profile="ProductA"> ... </technique>
    <technique profile="ProductB"> ... </technique>
  </extra>
  <extra type="bonus">
    <technique profile="ProductB"> ... </technique>
  </extra>
```

Examples of two choices are:

- **technique_common**, extra "basic" (product B), and extra "bonus" (product B);
- **technique** (product B), extra "basic" (product B), and extra "bonus" (product B).

5.5.19 float_array

Category: Data flow

5.5.19.1 General

Declares the storage for a homogenous array of floating-point values.

5.5.19.2 Concepts

The **<float_array>** element stores the data values for generic use within the COLLADA schema. The arrays themselves are strongly typed but without semantics. They simply describe a sequence of floating-point values.

5.5.19.3 Attributes

The **<float_array>** element has the following attributes:

count	uint_type	The number of values in the array. Required.
id	xs:ID	A text string containing the unique identifier of this element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.
digits	xs:unsignedByte	The number of significant decimal digits of the floating-point values that can be contained in the array. The minimum value is 1; the maximum is 17. The default is 6. Optional.
magnitude	xs:short	The largest exponent of the floating-point values that can be contained in the array. The maximum value is 308; the minimum is -324. The default is 38. Optional.

5.5.19.4 Related elements

The **<float_array>** element relates to the following elements:

Parent elements	Source (Core)
Child elements	None
Other	accessor

5.5.19.5 Details

A **<float_array>** element contains a list of floating-point values. These values are a repository of data for **<Source>** elements.

5.5.19.6 Example

The following is an example of a **<float_array>** element that describes a sequence of nine floating-point values:

```
<float_array id="floats" name="myFloats" count="9">
  1.0 0.0 0.0
  0.0 0.0 0.0
  1.0 1.0 0.0
</float_array>
```

5.5.20 formula

Category: Mathematics

5.5.20.1 General

Defines a formula.

5.5.20.2 Concepts

There are many ways to describe a formula. COLLADA uses MathML as its common technique.

5.5.20.3 Attributes

The **<formula>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of this element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of the element. Optional.
sid	sid_type	A text string value containing the scoped identifier of this element. This shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.

5.5.20.4 Related elements

The **<formula>** element relates to the following elements:

Parent elements	library_formulas , animation_clip , kinematics_model/technique_common , kinematics/axis_info
Child elements	See the following subclause.
Other	instance_formula , library_formulas

5.5.20.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<newparam>	See main entry.	N/A	0 or more
<target>	Contains a common_float_or_param_type (see 5.11 Types) that specifies the result variable of the formula. Usually a parameter.	N/A	1
<technique_common>	Specifies a formula for the common profile that all COLLADA implementations shall support. See the following subclause for child element details, “The Common Profile” clause for usage information, and main entry.	N/A	1
<technique>	Each <technique> specifies a formula for a specific profile as designated by the <technique> ’s profile attribute. See main entry.	N/A	0 or more

5.5.20.6 Child elements for <formula> / <technique_common>

MathML has two major parts: presentation and content. COLLADA supports only the content part. The **<technique_common>** child element can contain any valid MathML XML if the appropriate schema is made available.

5.5.20.7 Details

COLLADA Mathematics provides the capability of defining functions in COLLADA. It is implemented based on MathML and provides representation and content definitions.

MathML was chosen because:

- MathML is an accepted Standard (W3C);
- MathML can be schema validated.

To use MathML within COLLADA, you shall have access to the MathML schema. In the common profile, COLLADA uses MathML version 2.0.

5.5.20.8 Example

The following is an example of a **<formula>** element.

```
<formula id="formula">
  <newparam sid="target">
    <float>0</float>
  </newparam>
  <newparam sid="value">
    <float>0</float>
  </newparam>
  <newparam sid="pitch">
    <float>0</float>
  </newparam>
  <target><param>target</param></target>
  <!-- target = (value/360) * pitch -->
  <technique_common>
    <math:math>
      <math:apply>
        <math:times />
        <math:apply>
          <math:divide />
          <math:csymbol encoding="COLLADA">
            value
          </math:csymbol>
          <math:ci>360</math:ci>
        </math:apply>
        <math:csymbol encoding="COLLADA">
          pitch
        </math:csymbol>
      </math:apply>
    </math:math>
  </technique_common>
  <technique profile="test">
    <any>
      ...
    </any>
  </technique>
</formula>
```

5.5.21 geographic_location

Category: Metadata

5.5.21.1 General

Defines geographic location information regarding the parent of the **<asset>** element in which it resides.

5.5.21.2 Attributes

The **<geographic_location>** element has no attributes.

5.5.21.3 Related elements

The **<geographic_location>** element relates to the following elements:

Parent elements	asset/coverage
Child elements	See the following subclause.
Other	None

5.5.21.4 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<longitude>	Contains a floating-point number that specifies the longitude of the asset as defined by the WGS 84 world geodetic system. Valid values range from -180.0 to 180.0.	N/A	1
<latitude>	Contains a floating-point number that specifies the latitude of the asset as defined by the WGS 84 world geodetic system. Valid values range from -90.0 to 90.0.	None	1
<altitude mode=" ">	Specifies the altitude of the asset as a floating-point number. Altitude follows the Keyhole Markup Language (KML) standard rather than the WGS 84 calculation of height. That is, it can be relative to terrain, or relative to sea level. Its required attribute is: • mode: Indicates whether the altitude value shall be interpreted as the distance in meters from sea level or from the altitude of the terrain at the latitude/longitude point. Valid values are : • absolute • relativeToGround; this is the default.	None	1

5.5.21.5 Example

The following is an example of a visual scene that contains an asset that specifies the geographic location of the visual scene:

```
<library_visual_scene>
  <visual_scene id="SketchUpScene" name="EiffelTower">
    <asset>
      <coverage>
        <geographic_location>
          <longitude>-105.2830</longitude>
          <latitude>40.0170</latitude>
```

```

        <altitude mode="relativeToGround">0</altitude>
    </geographic_location>
</coverage>
<created>2008-01-28T20:51:36Z</created>
<modified>2008-01-28T20:51:36Z</modified>
<up_axis>Z_UP</up_axis> <!-- Optional; otherwise inherited from parent asset
-->
</asset>
<node id="Model" name="Model">
    <rotate>0, 0, 1, -10</rotate>
    <scale>0.75 0.75 0.75</scale>
    <node id="Component_1" name="Component_1">
        <matrix>
            1.0 0.0 0.0 28.8084
            0.0 1.0 0.0 311.67
            0.0 0.0 1.0 0.0
            0.0 0.0 0.0 1.0
        </matrix>
    </node>
</node>
</library_visual_scene>

```

5.5.22 geometry

Category: Geometry

5.5.22.1 General

Describes the visual shape and appearance of an object in a scene.

5.5.22.2 Concepts

The **<geometry>** element categorizes the declaration of geometric information. Geometry is a branch of mathematics that deals with the measurement, properties, and relationships of points, lines, angles, surfaces, and solids. The **<geometry>** element contains a declaration of a mesh, convex mesh, or spline.

There are many forms of geometric description. Computer graphics hardware has been normalized, primarily, to accept vertex position information with varying degrees of attribution (color, normals, etc.). Geometric descriptions provide this vertex data with relative directness or efficiency. Some of the more common forms of geometry are:

- B-Spline;
- Bézier;
- Mesh;
- NURBS;
- Patch.

This is by no means an exhaustive list. Currently, COLLADA supports only polygonal meshes and splines.

5.5.22.3 Attributes

The **<geometry>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <geometry> element. This value shall be unique within the instance document. Optional.
name	xs:token	A text string containing the name of this element. Optional.

5.5.22.4 Related elements

The **<geometry>** element relates to the following elements:

Parent elements	library_geometries
Child elements	See the following subclause.
Other	instance_geometry

5.5.22.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
geometric_element	The element that describes geometric data. Shall be exactly one of: <convex_mesh> (see 0 Physics Reference) <mesh> <spline> <brep> (see 5.9 B-Rep Reference) See main entries.	N/A	1
<extra>	Provides arbitrary additional information about or related to the <geometry> element. See main entry.	N/A	0 or more

5.5.22.6 Details

A **<geometry>** element contains elements that describe geometric data. The actual type and complexity of the data is represented in detail by the child elements.

5.5.22.7 Example

The following is an example of an empty **<geometry>** element with the allowed attributes:

```
<library_geometries>
  <geometry name="cube" id="cube123">
    <mesh>
      <source id="box-Pos"/>
      <vertices id="box-Vtx">
        <input semantic="POSITION" source="#box-Pos"/>
      </vertices>
    </mesh>
  </geometry>
</library_geometries>
```

```

    </vertices>
  </mesh>
</geometry>
</library_geometries>

```

5.5.23 IDREF_array

Category: Data flow

5.5.23.1 General

Declares the storage for a homogenous array of ID reference values.

5.5.23.2 Concepts

The **<IDREF_array>** element stores string values that reference IDs within the instance document.

5.5.23.3 Attributes

The **<IDREF_array>** element has the following attributes:

count	uint_type	The number of values in the array. Required.
id	xs:ID	A text string containing the unique identifier of this element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.23.4 Related elements

The **<IDREF_array>** element relates to the following elements:

Parent elements	Source (Core)
Child elements	None
Other	accessor

5.5.23.5 Details

An **<IDREF_array>** element contains a list of XML IDREF values. These values are a repository of data for **<Source>** elements.

5.5.23.6 Example

The following is an example of an **<IDREF_array>** element that refers to **<node>** elements in the document:

```

<library_nodes>
  <node id="Node1"/>
  <node id="Node2"/>
  <node id="Joint3"/>
  <node id="WristJoint"/>

```

```

</library_nodes>
<IDREF_array id="refs" name="myRefs" count="4">
  Node1 Node2 Joint3 WristJoint
</IDREF_array>

```

5.5.24 imager

Category: Camera

5.5.24.1 General

Represents the image sensor of a camera (for example, film or CCD).

5.5.24.2 Concepts

The optics of a camera projects an image onto a (usually planar) sensor.

The **<imager>** element defines how this sensor transforms light colors and intensities into numerical values.

Real light intensities may have a very high dynamic range. For example, in an outdoor scene, the sun is many orders of magnitude brighter than the shadow of a tree. Also, real light may contain photons with an infinite variety of wavelengths.

Display devices use a much more limited dynamic range, and they usually consider only three wavelengths within the visible range: red, green, and blue (primary colors). This is usually represented as three 8-bit values.

An image sensor therefore performs two tasks:

- Spectral sampling;
- Dynamic range remapping.

The combination of these is called *tone mapping*, which is performed as the last step of image synthesis (rendering).

High-quality renderers, e.g. as ray tracers, represent spectral intensities as floating-point numbers internally and store the actual pixel colors as float3_types, or even as arrays of floating-points (multispectral renderers), then perform tone mapping to create a 24-bit RGB image that can be displayed by the graphics hardware and monitor.

Many renderers can also save the original high dynamic range (HDR) image to allow for “re-exposing” it later.

5.5.24.3 Attributes

The **<imager>** element has no attributes.

5.5.24.4 Related elements

The **<imager>** element relates to the following elements:

Parent elements	camera
Child elements	See the following subclause.
Other	None

5.5.24.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<technique> (Core)	See main entry.	N/A	1 or more
<extra>	Provides arbitrary additional information about or related to the <imager> element. See main entry.	N/A	0 or more

5.5.24.6 Details

The **<imager>** element is optional. The COMMON profile omits it (there is no **<technique_common>** for this element) and the default interpretation is:

- Linear mapping of intensities;
- Clamping to the 0 ... 1 range (in terms of an 8-bit per component frame buffer, this maps to 0...255);
- R,G,B spectral sampling.

Multispectral renderers need to specify an **<imager>** element to at least define the spectral sampling.

5.5.24.7 Example

The following is an example of a **<camera>** element that describes a realistic camera with a CCD sensor:

```
<camera name="eyepoint">
  <optics>
    <technique_common>....</technique_common>
    <technique profile="MyFancyGIRenderer">
      <param name="FocalLength" type="float">180.0</param>
      <param name="Aperture" type="float">5.6</param>
    </technique>
  </optics>
  <imager>
    <technique profile="MyFancyGIRenderer">
      <param name="ShutterSpeed" type="float">200.0</param>
      <!-- "White-balance" -->
      <param name="RedGain" type="float">0.2</param>
      <param name="GreenGain" type="float">0.22</param>
      <param name="BlueGain" type="float">0.25</param>
      <param name="RedGamma" type="float">2.2</param>
      <param name="GreenGamma" type="float">2.1</param>
    </technique>
  </imager>
</camera>
```

```

    <param name="BlueGamma" type="float">2.17</param>
    <param name="BloomPixelLeak" type="float">0.17</param>
    <param name="BloomFalloff" type="Name">InvSquare</param>
  </technique>
</imager>
</camera>

```

5.5.25 input (shared)

Category: Data flow

5.5.25.1 General

Declares the input semantics of a data source and connects a consumer to that source.

NOTE There are two **<input>** variants; see also “**<input>** (unshared).”

5.5.25.2 Concepts

The **<input>** element declares the input connections to a data source that a consumer requires. A data source is a container of raw data that lacks semantic meaning so that the data can be reused within the document. To use the data, a consumer declares a connection to it with the desired semantic information.

The **<Source>** and **<input>** elements are part of the COLLADA dataflow model. This model is also known as stream processing, pipe, or producer-consumer. An input connection is the dataflow path from a **<Source>** to a sink (the dataflow consumers, which are **<input>**’s parents, such as **<polylist>**).

In COLLADA, all inputs are driven by index values. A consumer samples an input by supplying an index value to an input. Some consumers have multiple inputs that can share the same index values. Inputs that have the same `offset` attribute value are driven by the same index value from the consumer. This is an optimization that reduces the total number of indexes that the consumer shall store. These inputs are described in this clause as shared inputs but otherwise operate in the same manner as unshared inputs.

5.5.25.3 Attributes

The **<input>** element has the following attributes:

offset	uint_type	The offset into the list of indices defined by the parent element’s <p> or <v> sub element. If two <input> elements share the same offset, they are indexed the same. This is a simple form of compression for the list of indices and also defines the order in which the inputs are used. Required.
semantic	xs:NMTOKEN	The user-defined meaning of the input connection. Required. See “Details” for the list of common <input> <code>semantic</code> attribute values enumerated in the COLLADA schema.
source	urifragment_type	The location of the data source. Required.
set	uint_type	Which inputs to group as a single set. This is helpful when multiple inputs share the same semantics. Optional.

5.5.25.4 Related elements

The **<input>** element relates to the following elements:

Parent elements	In core: lines , linestrips , polygons , polylist , triangles , trifans , tristrips , vertex_weights In B-Rep: edges , faces , pcurves , shells , solids , wires
Child elements	None
Other	p (in lines , linestrips , polygons , polylist , triangles , trifans , tristrips); v (in vertex_weights)

5.5.25.5 Details

Each input connection can be uniquely identified by its `offset` attribute within the scope of its parent element.

The common **<input>** `semantic` attribute values are:

Value of semantic attribute	Description
BINORMAL	Geometric binormal (bitangent) vector
COLOR	Color coordinate vector. Color inputs are RGB (float3_type)
CONTINUITY	Continuity constraint at the control vertex (CV). See also “Curve Interpolation” in 5.4 Programming guideline.
IMAGE	Raster or MIP-level input.
INPUT	Sampler input. See also “Curve Interpolation” in 5.4 Programming guideline.
IN_TANGENT	Tangent vector for preceding control point. See also “Curve Interpolation” in 5.4 Programming guideline.
INTERPOLATION	Sampler interpolation type. See also “Curve Interpolation” in 5.4 Programming guideline.
INV_BIND_MATRIX	Inverse of local-to-world matrix.
JOINT	Skin influence identifier
LINEAR_STEPS	Number of piece-wise linear approximation steps to use for the spline segment that follows this CV. See also “Curve Interpolation” in 5.4 Programming guideline.
MORPH_TARGET	Morph targets for mesh morphing
MORPH_WEIGHT	Weights for mesh morphing
NORMAL	Normal vector
OUTPUT	Sampler output. See also “Curve Interpolation” in 5.4 Programming guideline.
OUT_TANGENT	Tangent vector for succeeding control point. See also “Curve Interpolation” in 5.4 Programming guideline.
POSITION	Geometric coordinate vector. See also “Curve Interpolation” in 5.4 Programming guideline.
TANGENT	Geometric tangent vector

Value of semantic attribute	Description
TEXBINORMAL	Texture binormal (bitangent) vector
TEXCOORD	Texture coordinate vector
TEXTANGENT	Texture tangent vector
UV	Generic parameter vector
VERTEX	Mesh vertex
WEIGHT	Skin influence weighting value

5.5.25.6 Example

The following is an example of six **<input>** elements that describe the sources of vertex positions, normals, and two sets of texture coordinates along with their texture space tangents for a **<polygons>** element. The **offset** attribute indicates the index from the **<p>** element that the input will use to sample the source data. When two or more **<input>** elements have the same offset value, it means that they share the same index in the **<p>** element. This is a simple form of index compression that saves space in the document.

The **set** attribute indicates the logical organization of **<input>** elements that belong in the same logical set of information. In this example, there are two sets of **TEXCOORD** and **TEXTANGENT** pairs:

```
<mesh>
  <source name="grid-Position"/>
  <source name="grid-0-Normal"/>
  <source name="texCoords1"/>
  <source name="grid-texTangents1"/>
  <source name="texCoords2"/>
  <source name="grid-texTangents2"/>
  <vertices id="grid-Verts">
    <input semantic="POSITION" source="#grid-Position"/>
  </vertices>
  <polygons count="1" material="Bricks">
    <input semantic="VERTEX" source="#grid-Verts" offset="0"/>
    <input semantic="NORMAL" source="#grid-Normal" offset="1"/>
    <input semantic="TEXCOORD" source="#texCoords1" offset="2" set="0"/>
    <input semantic="TEXCOORD" source="#texCoords2" offset="2" set="1"/>
    <input semantic="TEXTANGENT" source="#texTangents1" offset="2" set="0"/>
    <input semantic="TEXTANGENT" source="#texTangents2" offset="2" set="1"/>
    <p>0 0 0 2 1 1 3 2 2 1 3 3</p>
  </polygons>
</mesh>
```

5.5.26 input (unshared)

Category: Data flow

5.5.26.1 General

Declares the input semantics of a data source and connects a consumer to that source.

NOTE There are two **<input>** variants; see also "**<input>** (shared)."

5.5.26.2 Concepts

The **<input>** element declares the input connections that a consumer requires. A data source is a container of raw data that lacks semantic meaning so that the data can be reused within the document. To use the data, a consumer declares a connection to it with the desired semantic information.

The **<Source>** and **<input>** elements are part of the COLLADA dataflow model. This model is also known as stream processing, pipe, or producer-consumer. An input connection is the dataflow path from a **<Source>** to a sink (the dataflow consumers, which are **<input>**'s parents, such as **<vertices>**).

In COLLADA, all inputs are driven by index values. A consumer samples an input by supplying an index value to an input. Some consumers have simple inputs that are driven by unique index values. These inputs are described in this clause as unshared inputs but otherwise operate in the same manner as shared inputs.

5.5.26.3 Attributes

The **<input>** element has the following attributes:

semantic	xs:NMTOKEN	The user-defined meaning of the input connection. Required. See the list of common <input> semantic attribute values in the "Common glossary" in 5.3 Schema concepts. See " <input> (shared)" for the list of common <input> semantic attribute values enumerated in the COLLADA schema.
source	urifragment_type	The location of the data source. Required.

5.5.26.4 Related elements

The **<input>** element relates to the following elements:

Parent elements	joints, sampler, targets, vertices, control_vertices
Child elements	None
Other	None

5.5.26.5 Details

Each input can be uniquely identified by its `offset` attribute within the scope of its parent element.

See "Curve Interpolation" in 5.4 Programming guideline for a description of the semantic values that are useful in in **<sampler>** animation curves.

5.5.26.6 Example

The following is an example for **<sampler><input>**:

```
<animation>
  <source id="translate_X-input">
    ...
  </source>
  <source id="translate_X-output">
    ...
```



```

</source>
<source id="translate_X-intangents">
  ...
</source>
<source id="translate_X-outtangents">
  ...
</source>
<source id="translate_X-interpolations">
  ...
</source>
<sampler id="translate_X-sampler">
  <input semantic="INPUT" source="#translate_X-input"/>
  <input semantic="OUTPUT" source="#translate_X-output"/>
  <input semantic="IN_TANGENT" source="#translate_X-intangents"/>
  <input semantic="OUT_TANGENT" source="#translate_X-outtangents"/>
  <input semantic="INTERPOLATION" source="#translate_X-interpolations"/>
</sampler>
<channel source="#translate_X-sampler" target="pCube1/translate.X"/>
</animation>

```

5.5.27 instance_animation

Category: Animation

5.5.27.1 General

Instantiates a COLLADA animation resource.

5.5.27.2 Concepts

An **<instance_animation>** element instantiates an object described by an **<animation>** element. Multiple **<instance_animation>** elements are grouped together to create animation clips in the **<animation_clip>** element.

For details about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

5.5.27.3 Attributes

The **<instance_animation>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URL of the location of the <animation> element to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the “#” character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.5.27.4 Related elements

The **<instance_animation>** element relates to the following elements:

Parent elements	animation_clip
Child elements	See the following subclause.
Other	animation

5.5.27.5 Child elements

Name/example	Description	Default	Occurrences
<extra>	Provides arbitrary additional information about or related to the <animation> element. See main entry.	N/A	0 or more

5.5.27.6 Example

The following is an example of an **<instance_animation>** element that refers to a locally defined **<animation>** element identified by the ID "anim". The instance is translated some distance from the original:

```
<library_animations>
  <animation id="anim"/>
</library_animations>
<library_animation_clips>
  <animation_clip start="1.0" end="5.0"/>
    <instance_animation url="#anim"/>
  </animation_clip>
</library_animation_clips>
```

5.5.28 instance_camera

Category: Camera

5.5.28.1 General

Instantiates a COLLADA camera resource.

5.5.28.2 Concepts

The **<instance_camera>** element instantiates an object described by a **<camera>** element to activate it in the visual scene. A camera object is instantiated within the local coordinate system of its parent **<node>** and that determines its position, orientation, and scale.

For details about instance elements in COLLADA, see "Instantiation and External Referencing" in 5.3 Schema concepts.

5.5.28.3 Attributes

The **<instance_camera>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URI of the location of the <camera> element to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the "#" character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.5.28.4 Related elements

The **<instance_camera>** element relates to the following elements:

Parent elements	node
Child elements	See the following subclause.
Other	camera

5.5.28.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<extra>	Provides arbitrary additional information about or related to the <instance_camera> element. See main entry.	N/A	0 or more

5.5.28.6 Example

```
<library_cameras>
  <camera id="cam"/>
</library_cameras>
<node>
  <node>
    <translate>11.0 12.0 13.0</translate>
    <instance_camera url="#cam"/>
  </node>
</node>
```

5.5.29 instance_controller

Category: Controller

5.5.29.1 General

Instantiates a COLLADA controller resource.

5.5.29.2 Concepts

The **<instance_controller>** element instantiates an object described by a **<controller>** element. A controller object is instantiated within the local coordinate system of its parent **<node>** and that determines its position, orientation, and scale. The controller allows deformations of meshes based on skinning animations or morphing animations.

For details about instance elements in COLLADA, see “Instantiation and External Referencing” in 5.3 Schema concepts.

5.5.29.3 Attributes

The **<instance_controller>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value must be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URI of the location of the <controller> element to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the “#” character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.5.29.4 Related Elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<skeleton>	Indicates where a skin controller is to start to search for the joint nodes it needs. This element is meaningless for morph controllers. See main entry.	None	0 or more
<bind_material>	See main entry in FX.	N/A	0 or 1
<extra>	Provides arbitrary additional information about or related to the <instance_controller> element. See main entry.	N/A	0 or more

5.5.29.5 Example

The following is an example of an **<instance_controller>** element that refers to a locally defined **<controller>** element identified as “skin”. The instance is translated some distance from the original:

```
<library_controllers>
  <controller id="skin"/>
</library_controllers>
<node id="skel"/>
  ...
</node>
<node>
  <translate>11.0 12.0 13.0</translate>
  <instance_controller url="#skin"/>

```

```

    <skeleton>#skel</skeleton>
  </instance_controller>
</node>

```

The following is an Example of **two** **<instance_controller>** elements that refer to the same locally defined **<controller>** element identified as “skin”. The two skin instances are bound to different instances of a skeleton using the **<skeleton>** element:

```

<library_controllers>
  <controller id="skin">
    <skin source="#base_mesh">
      <source id="Joints">
        <Name_array count="4"> Root Spine1 Spine2 Head </Name_array>
        ...
      </source>
      <source id="Weights"/>
      <source id="Inv_bind_mats"/>
      <joints>
        <input source="#Joints" semantic="JOINT"/>
      </joints>
      <vertex_weights/>
    </skin>
  </controller>
</library_controllers>
<library_nodes>
  <node id="Skeleton1" sid="Root">
    <node sid="Spine1">
      <node sid="Spine2">
        <node sid="Head"/>
      </node>
    </node>
  </node>
</library_nodes>
<node id="skel01">
  <instance_node url="#Skeleton1"/>
</node>
<node id="skel02">
  <instance_node url="#Skeleton1"/>
</node>
<node>
  <instance_controller url="#skin"/>
    <skeleton>#skel01</skeleton>
  </instance_controller>
</node>
<node>
  <instance_controller url="#skin"/>
    <skeleton>#skel02</skeleton>
  </instance_controller>
</node>

```

5.5.30 instance_formula

Category: Mathematics

5.5.30.1 General

Instantiates a COLLADA formula resource.

5.5.30.2 Concepts

For details about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

5.5.30.3 Attributes

The **<instance_formula>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of the element. Optional.
url	xs:anyURI	The URL of the location of the object to instantiate. Required. Refers to a local instance using a relative URI fragment identifier that begins with the “#” character. The fragment identifier is a XPointer shorthand pointer that consists of the URI of the element to instantiate. Refers to an external reference using an absolute or relative URL when it contains a path to another resource.

5.5.30.4 Related elements

The **<instance_formula>** element relates to the following elements:

Parent elements	animation_clip , kinematics/axis_info , kinematics_model/technique_common
Child elements	See the following subclause.
Other	formula, library_formulas

5.5.30.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<setparam>	Specifies the source (for arguments) or the destination (for the result) of the instantiated formula. See main entry.	N/A	0 or more

5.5.30.6 Example

The following is an example of an **<instance_formula>** element.

```
<instance_formula url="#formula">
  <setparam ref="target">
    <connect_param ref="joint.trans.target"/>
  </setparam>
  <setparam ref="value">
    <connect_param ref="joint.trans.value"/>
  </setparam>
  <setparam ref="pitch">
```

```

    <connect_param ref="pitch"/>
  </setparam>
</instance_formula>

```

5.5.31 instance_geometry

Category: Geometry

5.5.31.1 General

Instantiates a COLLADA geometry resource.

5.5.31.2 Concepts

The **<instance_geometry>** element instantiates an object described by a **<geometry>** element. A geometry object is instantiated within the local coordinate system of its parent **<node>** or **<shape>** and that determines its position, orientation, and scale. COLLADA supports convex mesh, mesh, and spline primitives.

For details about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

5.5.31.3 Attributes

The **<instance_geometry>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URL of the location of the <geometry> element to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the “#” character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.5.31.4 Related elements

The **<instance_geometry>** element relates to the following elements:

Parent elements	node , shape
Child elements	See the following subclause.
Other	geometry

5.5.31.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<bind_material>	Binds material symbols to material instances. This allows a single geometry to be instantiated into a scene multiple times each with a different appearance. See main entry.	None	0 or 1
<extra>	Provides arbitrary additional information about or related to the <instance_geometry> element. See main entry.	N/A	0 or more

5.5.31.6 Example

The following is an example of an **<instance_geometry>** element that refers to a locally defined **<geometry>** element identified by the ID “cube”. The instance is translated some distance from the original:

```
<library_geometries>
  <geometry id="cube"/>
</library_geometries>
<node>
  <node>
    <translate>11.0 12.0 13.0</translate>
    <instance_geometry url="#cube"/>
  </node>
</node>
```

5.5.32 instance_light

Category: Lighting

5.5.32.1 General

Instantiates a COLLADA light resource.

5.5.32.2 Concepts

The **<instance_light>** element instantiates an object described by a **<light>** element to activate it in the visual scene. Directional, point, and spot light objects are instantiated within the local coordinate system of their parent **<node>** and that determines their position, orientation, and scale. The exception is ambient light; because ambient light radiates in all directions equally, it is not affected by these spatial transformations.

For details about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

5.5.32.3 Attributes

The **<instance_light>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URL of the location of the <light> element to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the “#” character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.5.32.4 Related elements

The **<instance_light>** element relates to the following elements:

Parent elements	node
Child elements	See the following subclause.
Other	light

5.5.32.5 Child elements

Name/example	Description	Default	Occurrences
<extra>	Provides arbitrary additional information about or related to the <instance_light> element. See main entry.	N/A	0 or more

5.5.32.6 Details

COLLADA does not dictate the policy of data sharing for each instance. This decision is left to the run-time application.

5.5.32.7 Example

The following is an example of an **<instance_light>** element that refers to a locally defined **<light>** element identified by the id “light”. The instance is translated some distance from the original:

```
<library_lights>
  <light id="light"/>
</library_lights>
<node>
  <node>
    <translate>11.0 12.0 13.0</translate>
    <instance_light url="#light"/>
  </node>
</node>
```

5.5.33 instance_node

Category: Scene

5.5.33.1 General

Instantiates a COLLADA node resource.

5.5.33.2 Concepts

An **<instance_node>** creates an instance of an object described by a **<node>** element. Each instance of a **<node>** element refers to an element in the **<node>** hierarchy that has its own local coordinate system defined for placing objects in the scene.

For details about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

5.5.33.3 Attributes

The **<instance_node>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URL of the location of the <node> element to instantiate. Required. Can refer to a local instance or external reference.
proxy	xs:anyURI	Optional. The mechanism and use of this attribute are application-defined. For example, it can be used for bounding boxes or level of detail. See “Details.” For a local instance, this is a relative URI fragment identifier that begins with the “#” character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.5.33.4 Related elements

The **<instance_node>** element relates to the following elements:

Parent elements	node
Child elements	See the following subclause.
Other	node

5.5.33.5 Child elements

Name/example	Description	Default	Occurrences
<extra>	Provides arbitrary additional information about or related to the <instance_node> element. See main entry.	N/A	0 or more

5.5.33.6 Details

The application can decide to resolve either the URL in the `url` attribute or the URL in the `proxy` attribute. Both resolve into a **<node>** element. Because the mechanism and use of this attribute are application defined, more information about how applications can decide which path to follow shall be stored in the **<extra>** element of **<instance_node>**.

For example:

```
<instance_node url="URL1" proxy="URL2">
  <extra ...>
```

In the following example, an application may decide to use the simple box node hierarchy, which may be a simple **<instance_geometry>**, rather than load a new document, parse it, and manage a complex building. The complex building might itself contain an **<instance_node>** with both a url and a proxy, allowing for the hierarchical management of data:

```
<instance_node url="file:///some_place/doc.dae#complex_building" proxy="#box">
```

The following example has both url and proxy to reference **<node>** and **<instance_node>**, all defined in the same COLLADA document. This construct provides the application with multiple choices of which node to use, which is the basic construct for Level of Detail (LOD).

```
<node id="NODE0"/>
<node id="NODE1"/>
<node id="NODE2"/>

<node id="LOD1">
  <instance_node url="#NODE1" proxy="#LOD2"/>
</node>

<node id="LOD2">
  <instance_node url="#NODE2"/>
</node>

<visual_scene>
  <node>
    <instance_node url="#NODE0" proxy="#LOD1">
  </node>
</visual_scene>
```

5.5.33.7 Example

The following is an example of an **<instance_node>** element that refers to a locally defined **<node>** element identified by the ID "myNode". The instance is translated some distance from the original:

```
<library_nodes>
  <node id="myNode"/>
</library_nodes>
<node>
  <node>
    <translate>11.0 12.0 13.0</translate>
    <instance_node url="#myNode"/>
  </node>
</node>
```

5.5.34 instance_visual_scene

Category: Scene

5.5.34.1 General

Instantiates a COLLADA visual_scene resource.

5.5.34.2 Concepts

An **<instance_visual_scene>** instantiates the visual aspects of a scene. The **<Scene>** element can contain, at most, one **<instance_visual_scene>** element. This constraint creates a one-to-one relationship between the document, the top-level scene, and its visual description. This provides applications and tools, especially those that support only one scene, an indication of the primary scene to load.

For details about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

5.5.34.3 Attributes

The **<instance_visual_scene>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URL of the location of the <visual_scene> element to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the “#” character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.5.34.4 Related elements

The **<instance_visual_scene>** element relates to the following elements:

Parent elements	Scene
Child elements	See the following subclause.
Other	visual_scene

5.5.34.5 Child elements

Name/example	Description	Default	Occurrences
<extra>	Provides arbitrary additional information about or related to the <instance_visual_scene> element. See main entry.	N/A	0 or more

5.5.34.6 Example

The following is an example of an **<instance_visual_scene>** element that refers to a locally defined **<visual_scene>** element identified by the ID “vis_scene”:

```
<library_visual_scenes>
  <visual_scene id="vis_scene"/>
</library_visual_scenes>
<scene>
  <instance_visual_scene url="#vis_scene"/>
</scene>
```

5.5.35 int_array

Category: Data flow

5.5.35.1 General

Stores a homogenous array of integer values.

5.5.35.2 Concepts

The **<int_array>** element stores the data values for generic use within the COLLADA schema. The arrays themselves are strongly typed but without semantics. They simply describe a sequence of integer values.

5.5.35.3 Attributes

The **<int_array>** element has the following attributes:

count	uint_type	The number of values in the array. Required.
id	xs:ID	A text string containing the unique identifier of this element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.
minInclusive	xs:integer	The smallest integer value that can be contained in the array. The default is -2147483648. Optional.
maxInclusive	xs:integer	The largest integer value that can be contained in the array. The default is 2147483647. Optional.

5.5.35.4 Related elements

The **<int_array>** element relates to the following elements:

Parent elements	Source (Core)
Child elements	None
Other	accessor

5.5.35.5 Details

An **<int_array>** element contains a list of integer values. These values are a repository of data to **<Source>** elements.

5.5.35.6 Example

The following is an example of an **<int_array>** element that describes a sequence of five integer numbers:

```
<int_array id="integers" name="myInts" count="5">
  1 2 3 4 5
</int_array>
```

5.5.36 joints

Category: Controller

5.5.36.1 General

Declares the association between joint nodes and attribute data.

5.5.36.2 Concepts

Associates joint, or skeleton, nodes with attribute data. In COLLADA, this is specified by the inverse bind matrix of each joint (influence) in the skeleton.

5.5.36.3 Attributes

The **<joints>** element has no attributes.

5.5.36.4 Related elements

The **<joints>** element relates to the following elements:

Parent elements	skin
Child elements	See the following subclause.
Other	None

5.5.36.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (unshared)	At least one <input> element shall have the semantic JOINT . The <Source> referenced by the input with the JOINT semantic shall contain a <Name_array> that contains SIDs to identify the joint nodes. SIDs are used instead of IDREFs to allow a skin controller to be instantiated multiple times, where each instance can be animated independently. See main entry. See also Address syntax” in 5.3 Schema concepts.	None	2 or more
<extra>	Provides arbitrary additional information about or related to the <joints> element. See main entry.	N/A	0 or more

5.5.36.6 Example

The following is an example of a **<joints>** element that associates joints and their bind positions:

```
<skin source="#geometry_mesh">
  <joints>
    <input semantic="JOINT" source="#joints"/>
    <input semantic="INV_BIND_MATRIX" source="#inv-bind-matrices"/>
  </joints>
</skin>
```

5.5.37 library_animation_clips

Category: Animation

5.5.37.1 General

Provides a library in which to place **<animation_clip>** elements.

5.5.37.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.5.37.3 Attributes

The **<library_animation_clips>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <library_animation_clips> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.37.4 Related elements

The **<library_animation_clips>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.5.37.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	None	0 or more
<animation_clip>	See main entry.	N/A	1 or more
<extra>	Provides arbitrary additional information about or related to the <library_animation_clips> element. See main entry.	N/A	0 or more

5.5.37.6 Example

The following is an example of a **<library_animation_clips>** element:

```
<library_animation_clips>
  <animation_clip>
    <instance_animation url="#animation1" />
  </animation_clip>
  <animation_clip>
    <instance_animation url="#animation2" />
    <instance_animation url="#animation3" />
  </animation_clip>
</library_animation_clips>
```

5.5.38 library_animations

Category: Animation

5.5.38.1 General

Provides a library in which to place **<animation>** elements.

5.5.38.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.5.38.3 Attributes

The **<library_animations>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <library_animations> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.38.4 Related elements

The **<library_animations>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.5.38.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<animation>	See main entry.	N/A	1 or more
<extra>	Provides arbitrary additional information about or related to the <library_animations> element. See main entry.	N/A	0 or more

5.5.38.6 Example

The following is an example of a **<library_animations>** element:

```
<library_animations>
  <animation name="animation1" />
  <animation name="animation2" />
  <animation name="animation3" />
</library_animations>
```

5.5.39 library_cameras

Category: Camera

5.5.39.1 General

Provides a library in which to place **<camera>** elements.

5.5.39.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.5.39.3 Attributes

The **<library_cameras>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <library_cameras> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.39.4 Related elements

The **<library_cameras>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.5.39.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<camera>	See main entry.	N/A	1 or more
<extra>	See main entry.	N/A	0 or more

5.5.39.6 Example

The following is an example of a **<library_cameras>** element:

```
<library_cameras>
  <camera name="eyepoint">
    ...
  </camera>
  <camera name="overhead">
    ...
  </camera>
</library_cameras>
```

5.5.40 library_controllers

Category: Controller

5.5.40.1 General

Provides a library in which to place **<controller>** elements.

5.5.40.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.5.40.3 Attributes

The **<library_controllers>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <library_controllers> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.40.4 Related elements

The **<library_controllers>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.5.40.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<controller>	See main entry.	N/A	1 or more
<extra>	See main entry.	N/A	0 or more

5.5.40.6 Example

The following is an example of a **<library_controllers>** element:

```
<library_controllers>
  <controller>
    <skin source="#geometry_mesh">
      ...
    </skin>
  </controller>
</library_controllers>
```

5.5.41 library_formulas

Category: Mathematics

5.5.41.1 General

Provides a library in which to place **<formula>** elements.

5.5.41.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.5.41.3 Attributes

The **<library_formulas>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <library_formulas> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of the element. Optional.

5.5.41.4 Related elements

The **<library_formulas>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	instance_formula

5.5.41.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<formula>	Specifies a formula with its argument and result parameters. See main entry.	N/A	1 or more
<extra>	See main entry.	N/A	0 or more

5.5.41.6 Example

The following is an example of a **<library_formulas>** element.

```
<library_formulas>
  <formula id="formula">
    ...
  </formula>
</library_formulas>
```

5.5.42 library_geometries

Category: Geometry

5.5.42.1 General

Provides a library in which to place **<geometry>** elements.

5.5.42.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.5.42.3 Attributes

The **<library_geometries>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <library_geometries> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.42.4 Related elements

The **<library_geometries>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.5.42.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<geometry>	See main entry.	N/A	1 or more
<extra>	See main entry.	N/A	0 or more

5.5.42.6 Example

The following is an example of a **<library_geometries>** element:

```
<library_geometries>
  <geometry name="cube" id="cube123">
    <mesh>
      <source id="box-Pos"/>
      <vertices id="box-Vtx">
        <input semantic="POSITION" source="#box-Pos"/>
      </vertices>
    </mesh>
  </geometry>
</library_geometries>
```

```

    </vertices>
  </mesh>
</geometry>
</library_geometries>

```

5.5.43 library_lights

Category: Lighting

5.5.43.1 General

Provides a library in which to place **<light>** elements.

5.5.43.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.5.43.3 Attributes

The **<library_lights>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.43.4 Related elements

The **<library_lights>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.5.43.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<light>	See main entry.	N/A	1 or more
<extra>	See main entry.	N/A	0 or more

5.5.43.6 Example

The following is an example of a **<library_lights>** element:

```
<library_lights>
  <light id="light1">
    ...
  </light>
  <light id="light2">
    ...
  </light>
</library_lights>
```

5.5.44 library_nodes

Category: Scene

5.5.44.1 General

Provides a library in which to place **<node>** elements.

5.5.44.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.5.44.3 Attributes

The **<library_nodes>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.44.4 Related elements

The **<library_nodes>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.5.44.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<node>	See main entry.	N/A	1 or more
<extra>	See main entry.	N/A	0 or more

5.5.44.6 Example

The following is an example of a <library_nodes> element:

```
<library_nodes>
  <node id="node1">
    ...
  </node>
  <node id="node2">
    ...
  </node>
</library_nodes>
```

5.5.45 library_visual_scenes

Category: Scene

5.5.45.1 General

Provides a library in which to place <visual_scene> elements.

5.5.45.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.5.45.3 Attributes

The <library_visual_scenes> element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.45.4 Related elements

The **<library_visual_scenes>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.5.45.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<visual_scene>	See main entry.	N/A	1 or more
<extra>	See main entry.	N/A	0 or more

5.5.45.6 Example

The following is an example of a **<library_visual_scenes>** element:

```
<library_visual_scenes>
  <visual_scene id="vis_sce1">
    ...
  </visual_scene>
  <visual_scene id="vis_sce2">
    ...
  </visual_scene>
</library_visual_scenes>
```

5.5.46 light

Category: Lighting

5.5.46.1 General

Declares a light source that illuminates a scene.

5.5.46.2 Concepts

A light embodies a source of illumination shining on the visual scene. A light source can be located within the scene or infinitely far away. Light sources have many different properties and radiate light in many different patterns and frequencies. COLLADA supports:

- An ambient light source radiates light from all directions at once. The intensity of an ambient light source is not attenuated.
- A point light source radiates light in all directions from a known location in space. The intensity of a point light source is attenuated as the distance to the light source increases.

- A directional light source radiates light in one direction from a known direction in space that is infinitely far away. The intensity of a directional light source is not attenuated.
- A spot light source radiates light in one direction from a known location in space. The light radiates from the spot light source in a cone shape. The intensity of the light is attenuated as the radiation angle increases away from the direction of the light source. The intensity of a spot light source is also attenuated as the distance to the light source increases.

5.5.46.3 Attributes

The **<light>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.46.4 Related elements

The **<light>** element relates to the following elements:

Parent elements	library_lights
Child elements	See the following subclause.
Other	instance_light , ambient (Core), directional , point , spot

5.5.46.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<technique_common>	Specifies light information for the common profile that all COLLADA implementations shall support. See "The common profile" clause for usage information. Shall contain exactly one <ambient> (Core), <directional> , <point> , or <spot> element; see their main entries.	N/A	1
<technique> (Core)	Each <technique> specifies light information for a specific profile as designated by the <technique> 's profile attribute. See main entry.	N/A	0 or more
<extra>	See main entry.	N/A	0 or more

5.5.46.6 Example

The following is an example of a **<library_lights>** element that contains a directional **<light>** element that is instantiated in a visual scene, rotated to portray a sunset:

```
<library_lights>
  <light id="sun" name="the-sun">
    <technique_common>
      <directional>
        <color>1.0 1.0 1.0</color>
```

```

        </directional>
    </technique_common>
</light>
</library_lights>
<library_visual_scenes>
    <visual_scene>
        <node>
            <rotate>1 0 0 -10</rotate>
            <instance_light url="#sun"/>
        </node>
    </visual_scene>
</library_visual_scenes>

```

5.5.47 lines

Category: Geometry

5.5.47.1 General

Provides the information needed for a mesh to bind vertex attributes together and then organize those vertices into individual lines.

5.5.47.2 Concepts

The **<lines>** element declares the binding of geometric primitives and vertex attributes for a **<mesh>** element.

The vertex array information is supplied in distinct attribute arrays of the **<mesh>** element that are then indexed by the **<lines>** element.

Each line described by the mesh has two vertices. The first line is formed from the first and second vertices. The second line is formed from the third and fourth vertices, and so on.

5.5.47.3 Attributes

The **<lines>** element has the following attributes:

name	xs:token	The text string name of this element. Optional.
count	uint_type	The number of line primitives. Required.
material	xs:NCName	Declares a symbol for a material. This symbol is bound to a material at the time of instantiation; see <instance_geometry> and <bind_material> . Optional. If not specified, then the lighting and shading results are application defined.

5.5.47.4 Related elements

The **<lines>** element relates to the following elements:

Parent elements	mesh , convex_mesh
Child elements	See the following subclause.
Other	None

5.5.47.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (shared)	When at least one input is present, one input shall specify semantic="VERTEX". See main entry.	None	0 or more
<p>	Contains indices that describe the vertex attributes for an arbitrary number of individual lines. The indices in a <p> ("primitives") element refer to different inputs depending on their order. The first index in a <p> element refers to all inputs with an offset of 0. The second index refers to all inputs with an offset of 1. Each vertex of the line is made up of one index into each input. After each input is used, the next index again refers to the inputs with offset of 0 and begins a new vertex.	None	0 or 1
<extra>	See main entry.	N/A	0 or more

5.5.47.6 Example

The following is an example of the **<lines>** element: collating three **<input>** elements into two separate lines, where the last two inputs use the same offset:

```
<mesh>
  <source id="position"/>
  <source id="texcoord0"/>
  <source id="texcoord1"/>
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <lines count="2">
    <input semantic="VERTEX" source="#verts" offset="0"/>
    <input semantic="TEXCOORD" source="#texcoord0" offset="1"/>
    <input semantic="TEXCOORD" source="#texcoord1" offset="1"/>
    <p>10 10 11 11 21 21 22 22</p>
  </lines>
</mesh>
```

5.5.48 linestrips

Category: Geometry

5.5.48.1 General

Provides the information needed to bind vertex attributes together and then organize those vertices into connected line-strips.

5.5.48.2 Concepts

The **<linestrips>** element declares a binding of geometric primitives and vertex attributes for a **<mesh>** element.

The vertex information is supplied in distinct attribute arrays of the **<mesh>** element that are then indexed by the **<linestrips>** element.

Each line-strip described by the mesh has an arbitrary number of vertices. Each line segment within the line-strip is formed from the current vertex and the preceding vertex.

5.5.48.3 Attributes

The **<linestrips>** element has the following attributes:

name	xs:token	The text string name of this element. Optional.
count	uint_type	The number of line-strip primitives. Required.
material	xs:NCName	Declares a symbol for a material. This symbol is bound to a material at the time of instantiation; see <instance_geometry> and <bind_material> . Optional. If not specified, then the lighting and shading results are application defined.

5.5.48.4 Related elements

The **<linestrips>** element relates to the following elements:

Parent elements	mesh , convex_mesh
Child elements	See the following subclause.
Other	None

5.5.48.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (shared)	When at least one input is present, one input shall specify semantic="VERTEX". See main entry.	None	0 or more
<p>	Contains indices that describe the vertex attributes for an arbitrary number of connected line segments. The indices in a <p> ("primitive") element refer to different inputs depending on their order. The first index in a <p> element refers to all inputs with an offset of 0. The second index refers to all inputs with an offset of 1. Each vertex of the line is made up of one index into each input. After each input is used, the next index again refers to the inputs with offset of 0 and begins a new vertex.	None	0 or more
<extra>	See main entry.	N/A	0 or more

5.5.48.6 Details

A **<linestrips>** element contains a sequence of **<p>** elements.

5.5.48.7 Example

The following is an example of the **<linestrips>** element that describes two line segments with three vertex attributes, where all three inputs use the same offset:

```
<mesh>
  <source id="position"/>
  <source id="normals"/>
  <source id="texcoord"/>
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <linestrips count="1">
    <input semantic="VERTEX" source="#verts" offset="0"/>
    <input semantic="NORMAL" source="#normals" offset="0"/>
    <input semantic="TEXCOORD" source="#texcoord" offset="0"/>
    <p>0 1 2</p>
  </linestrips>
</mesh>
```

5.5.49 lookat

Category: Transform

5.5.49.1 General

Contains a position and orientation transformation suitable for aiming a camera.

5.5.49.2 Concepts

The **<lookat>** element contains a float3x3_type, which is three mathematical vectors that describe:

- The position of the object;
- The position of the interest point;
- The direction that points up.

Positioning and orienting a camera or object in the scene is often complicated when using a matrix. A lookat transform is an intuitive way to specify an eye position, interest point, and orientation.

5.5.49.3 Attributes

The **<lookat>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.

5.5.49.4 Related elements

The **<lookat>** element relates to the following elements:

Parent elements	node
Child elements	None
Other	None

5.5.49.5 Details

The **<lookat>** element contains a list of 9 floating-point values. As in the OpenGL® Utilities (GLU) implementation [7], these values are organized into three vectors as follows:

1. Eye position is given as **Px, Py, Pz**;
2. Interest point is given as **Ix, Iy, Iz**;
3. Up-axis direction is given as **UPx, UPy, UPz**.

When computing the equivalent (viewing) matrix, the interest point is mapped to the negative z axis and the eye position to the origin. The up-axis is mapped to the positive y axis of the viewing plane (See Figure 10).

The values are specified in local object coordinates.

For more information about how transformation elements are applied, see **<node>**.

5.5.49.6 Example

The following is an example of a **<lookat>** element indicating a position of [10,20,30], centred on the local origin, with the y axis rotated up (yellow triangle in Figure 10):

```
<node id="Camera">
  <lookat>
    6.0  4.0  5.0 <!-- eye position (X,Y,Z)      -->
    0.0  0.0  0.0 <!-- interest position (X,Y,Z) -->
    0.0  1.0  0.0 <!-- up-vector position (X,Y,Z) -->
  </lookat>
  <instance_camera url="#camera1"/>
  ...
```

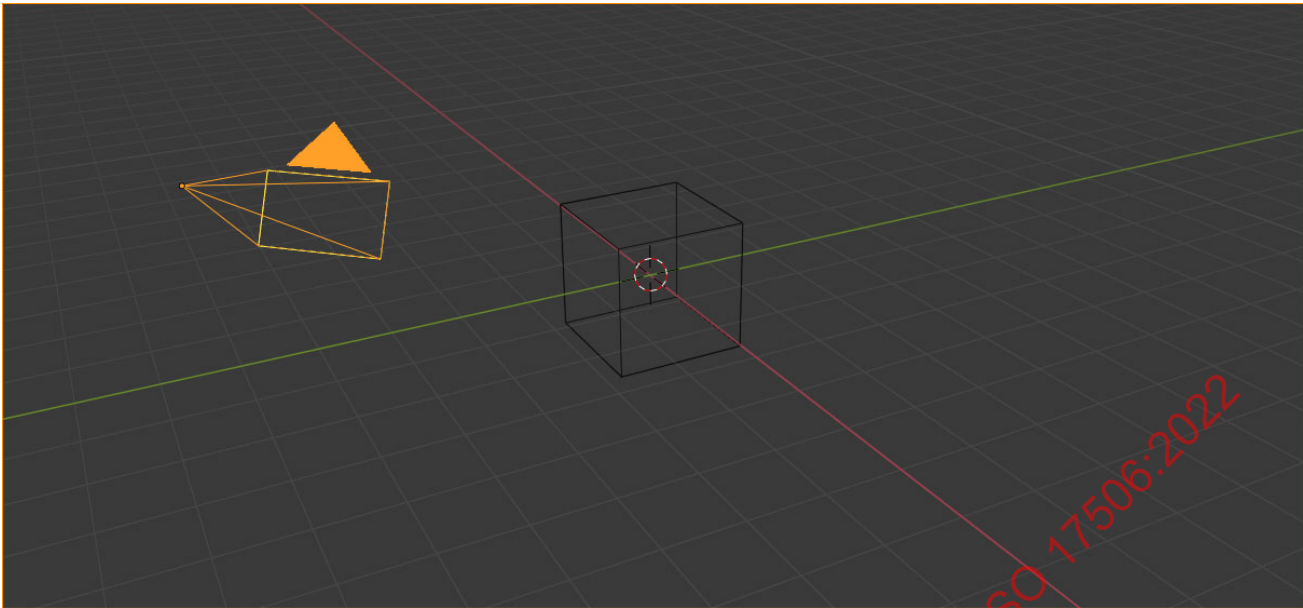


Figure 10 — <lookat> element

5.5.50 matrix

Category: Transform

5.5.50.1 General

Describes transformations that embody mathematical changes to points within a coordinate system or the coordinate system itself.

5.5.50.2 Concepts

The <matrix> element contains a float4x4_type, which is a 4-by-4 matrix of floating-point values.

Computer graphics employ linear algebraic techniques to transform data. The general form of a 3D coordinate system is represented as a 4-by-4 matrix. These matrices can be organized hierarchically, via the scene graph, to form a concatenation of coordinated frames of reference.

Matrices in COLLADA are column matrices in the mathematical sense. These matrices are written in row-major order to aid the human reader. See the example.

5.5.50.3 Attributes

The <matrix> element has the following attribute:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.

5.5.50.4 Related elements

The **<matrix>** element relates to the following elements:

Parent elements	node
Child elements	None
Other	None

5.5.50.5 Details

The **<matrix>** element contains a list of 16 floating-point values. These values are organized into a 4-by-4 column-order matrix suitable for matrix composition.

For more information about how transformation elements are applied, see **<node>**.

5.5.50.6 Example

The following is an example of a **<matrix>** element forming a translation matrix that translates 2 units along the x axis, 3 units along the y axis, and 4 units along the z axis:

```
<matrix>
  1.0 0.0 0.0 2.0
  0.0 1.0 0.0 3.0
  0.0 0.0 1.0 4.0
  0.0 0.0 0.0 1.0
</matrix>
```

5.5.51 mesh

Category: Geometry

5.5.51.1 General

Describes basic geometric meshes using vertex and primitive information.

5.5.51.2 Concepts

Meshes embody a general form of geometric description that primarily includes vertex and primitive information.

Vertex information is the set of attributes associated with a point on the surface of the mesh. Each vertex includes data for attributes such as:

- Vertex position
- Vertex color
- Vertex normal
- Vertex texture coordinate

The mesh also includes a description of how the vertices are organized to form the geometric shape of the mesh. The mesh vertices are collated into geometric primitives such as polygons, triangles, or lines.

5.5.51.3 Attributes

The **<mesh>** element has no attributes.

5.5.51.4 Related elements

The **<mesh>** element relates to the following elements:

Parent elements	geometry
Child elements	See the following subclause.
Other	None

5.5.51.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<Source>	Provides the bulk of the mesh's vertex data. See main entry.	N/A	1 or more
<vertices>	Describes the mesh-vertex attributes and establishes their topological identity. See main entry.	N/A	1
primitive_elements	Geometric primitives, which assemble values from the inputs into vertex attribute data. Can be any combination of the following in any order:		
	<lines> Contains line primitives. See main entry.	N/A	0 or more
	<linestrips> Contains line-strip primitives. See main entry.	N/A	0 or more
	<polygons> Contains polygon primitives which may contain holes. See main entry.	N/A	0 or more
	<polylist> Contains polygon primitives that cannot contain holes. See main entry.	N/A	0 or more
	<triangles> Contains triangle primitives. See main entry.	N/A	0 or more
	<trifans> Contains triangle-fan primitives. See main entry.	N/A	0 or more
	<tristrips> Contains triangle-strip primitives. See main entry.	N/A	0 or more
<extra>	See main entry.	N/A	0 or more

5.5.51.6 Details

To describe geometric primitives that are formed from the vertex data, the **<mesh>** element may contain zero or more of the primitive elements **<lines>**, **<linestrips>**, **<polygons>**, **<polylist>**, **<triangles>**, **<trifans>**, and **<tristrips>**.

The **<vertices>** element under **<mesh>** is used to describe mesh-vertices. Polygons, triangles, and so forth index mesh-vertices, not positions directly. Mesh-vertices shall have at least one **<input>** (unshared) element with a semantic attribute whose value is **POSITION**.

For texture coordinates, COLLADA's right-handed coordinate system applies; therefore, an ST texture coordinate of [0,0] maps to the lower-left texel of a texture image, when loaded in a professional 2D texture viewer/editor.

5.5.51.7 Example

The following is an example of an empty **<mesh>** element with the allowed attributes:

```
<mesh>
  <source id="box-Pos"/>
  <vertices id="box-Vtx">
    <input semantic="POSITION" source="#box-Pos">
  </vertices>
</mesh>
```

For sharing index data, i.e. to optimize the index data, and still have distinct set attributes, you can move the **<input>** element from the **<vertices>** element into the primitive element(s) and reuse the offset attribute value of the input with **VERTEX** semantic:

```
<vertices>
  <input semantic="POSITION"/>
  <input semantic="TEXCOORD"/>
  <input semantic="NORMAL"/>
</vertices>
<polygons>
  <input semantic="VERTEX" offset="0"/>
...
```

use the following:

```
<vertices>
  <input semantic="POSITION"/>
</vertices>
<polygons>
  <input semantic="VERTEX" offset="0"/>
  <input semantic="TEXCOORD" offset="0" set="1"/>
  <input semantic="NORMAL" offset="0" set="4"/>
...
```

5.5.52 morph

Category: Controller

5.5.52.1 General

Describes the data required to blend between sets of static meshes.

5.5.52.2 Concepts

A morph is used to blend a base mesh with one or more “morph target” meshes to form a final mesh, which can be used as an input to a skinning operation or can be rendered as is. Some DCC tools refer to morphs as *deformers*. A common use of morphs is to apply facial expressions to a character. Morphs operate only on the sources pointed to by **<vertices>** elements in the base and target meshes. The rest

of the information (such as **<polylist>** and other primitive tags) always comes from the base mesh. Target meshes can contain anything that is legal for a **<mesh>** element, but for the purposes of the morph, only the contents of the **<vertices>** element is used.

A **<morph>** contains weights, which describe how the base mesh and target meshes are blended. In the case of a single target, at a weight of 0, the morph outputs vertices that match the base mesh. As the weight moves towards 1, the output vertices gradually move from their values in the base mesh to the values of the corresponding vertices in the morph target (for example, vertex 5 in the base moves towards vertex 5 in the target, and so on). When the weight reaches 1, the output vertices of the morph will match those in the target.

The **<vertices>** elements in the base mesh and target meshes shall all contain the same number of **<input>** elements with the same semantics in the same order. The same number of vertices shall occur in all the morph's meshes. For good results, there shall be a one-to-one correspondence between the vertices in the base mesh and the vertices in the targets. For example, if the base mesh and morph target are both faces of characters, vertex 5 on the base mesh and vertex 5 on the morph target shall represent roughly the same place on both faces (for example, the corner of the eye).

Anything contained in the **<vertices>** element is blended by the morph. If the **<vertices>** element has **<input>**s with semantics for POSITION, NORMAL, TEXCOORD, or any other numeric values, all are morphed.

The **<targets>** element inside the **<morph>** points to one **<Source>** element that contains a list of the morph-target **<mesh>** elements and another **<Source>** that contains a **<float_array>** of weights.

The **method** attribute of the **<morph>** element specifies the formula used to combine these meshes. There are different methods available to combine morph targets; the two common methods are:

- NORMALIZED

$$(Target1, Target2, \dots) * (w1, w2, \dots) = (1-w1-w2-\dots) * BaseMesh + w1 * Target1 + w2 * Target2 + \dots$$

- RELATIVE

$$(Target1, Target2, \dots) + (w1, w2, \dots) = BaseMesh + w1 * Target1 + w2 * Target2 + \dots$$

5.5.52.3 Attributes

The **<morph>** element has the following attributes:

source	xs:anyURI	Refers to the <geometry> that describes the base mesh. Required. For more information, see "Address syntax" in 5.3 Schema concepts.
method	Enumeration	Which blending technique to use. Valid values are NORMALIZED and RELATIVE. The default is NORMALIZED. Optional.

5.5.52.4 Related elements

The **<morph>** element relates to the following elements:

Parent elements	controller
Child elements	See the following subclause.
Other	None

5.5.52.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<Source>	Data for morph weights and for morph targets. See main entry.	N/A	2 or more
<targets>	Input meshes (morph targets) to be blended. This shall contain at least one child <input> element with a semantic of MORPH_WEIGHT and one with a semantic of MORPH_TARGET . See main entry.	N/A	1
<extra>	See main entry.	N/A	0 or more

5.5.52.6 Details

See the annotated example at COLLADA wiki sites [9].

5.5.52.7 Example

The following is an example of an empty **<morph>** element:

```
<morph source="#the-base-mesh" method="RELATIVE">
  <source id="morph-targets"/>
  <source id="morph-weights"/>
  <targets>
    <input semantic="MORPH_TARGET" source="#morph-targets"/>
    <input semantic="MORPH_WEIGHT" source="#morph-weights"/>
  </targets>
  <extra/>
</morph>
```

5.5.53 Name_array

Category: Data flow

5.5.53.1 General

Stores a homogenous array of symbolic name values.

5.5.53.2 Concepts

The **<Name_array>** element stores name values as data for generic use within the COLLADA schema. The array itself is strongly typed but without semantics. It simply stores a sequence of XML name values.

5.5.53.3 Attributes

The **<Name_array>** element has the following attributes:

count	uint_type	The number of values in the array. Required.
id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.53.4 Related elements

The **<Name_array>** element relates to the following elements:

Parent elements	Source (Core)
Child elements	None.
Other	accessor

5.5.53.5 Details

An **<Name_array>** element contains a list of XML name values (**xs:Name**). These values are a repository of data to **<Source>** elements. An application can specify any application-defined name values.

When **<Name_array>** is used as a source for curve-interpolation descriptions, allows an application to specify the type of curve to be processed; the common profile defines the values **BEZIER**, **LINEAR**, **BSPLINE**, and **HERMITE** for curves.

5.5.53.6 Example

The following is an example of an **<Name_array>** element that provides a sequence of four name values:

```
<Name_array id="names" name="myNames" count="4">
  Node1 Node2 Joint3 WristJoint
</Name_array>
```

The following is an example that supplies interpolation types to a sampler:

```
<source id="translate_X-interpolations">
  <Name_array id="translate_X-interpolations-array" count="2">
    BEZIER BEZIER
  </Name_array>
  <technique_common>
    <accessor source="#translate_X-interpolations-array" count="2" stride="1">
      <param name="INTERPOLATION" type="Name"/>
    </accessor>
  </technique_common>
</source>
<sampler id="translate_X-sampler">
  <input semantic="INTERPOLATION" source="#translate_X-interpolations"/>
</sampler>
```

5.5.54 newparam

Category: Parameters

Profile: External, Effect, CG, COMMON, GLES, GLES2, GLSL

5.5.54.1 General

Creates a new, named parameter object, and assigns it a type and an initial value.

5.5.54.2 Concepts

Parameters are typed data objects that are available to compilers and functions at run time.

In FX, the parameter is created in the FX runtime and can have additional attributes assigned at declaration time.

In Kinematics, a parameter provides access to specific properties of instantiated kinematics objects.

5.5.54.3 Attributes

The **<newparam>** element has the following attribute:

sid	sid_type	Identifier for this parameter (that is, the variable name). Required. For details, see “Address syntax” in 5.3 Schema concepts.

5.5.54.4 Related elements

The **<newparam>** element relates to the following elements:

Parent elements	In FX: effect , profile_CG , profile_COMMON , profile_GLSL , profile_GLES , profile_GLES2 In Kinematics: instance_kinematics_model , instance_articulated_system , instance_kinematics_scene , axis_info , effector_info , kinematics_model/technique_common In core: formula
Child elements	See the following subclauses.
Other	param (reference), setparam

5.5.54.5 Child elements in FX

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><annotate></code>	See main entry. (Not valid in COMMON.)	N/A	0 or more
<code><semantic></code>	See main entry.	None	0 or 1
<code><modifier></code>	See main entry. (Not valid in COMMON.)	None	0 or 1
<code>parameter_type_element</code>	The parameter's type. Shall be exactly one element from the appropriate group, described in "Parameter-Type Elements" in 5.11 Types: - CG: <code>cg_param_group</code> - GLSL: <code>glsl_value_group</code> - In <code><effect></code>: <code>fx_newparam_group</code> - GLS: <code>gles_param_group</code> - GLS2: <code>gles2_value_group</code> - In COMMON scope, shall be one of the following: <code><float></code> <code><float2></code> <code><float3></code> <code><float4></code> <code><sampler2D></code>	None	1

5.5.54.6 Child elements in kinematics and in `<formula>/<newparam>`

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code>parameter_type</code>	The parameter's type. Shall be exactly one of the following elements: <code><float></code> <code><int></code> <code><bool></code> <code><SIDREF></code>	None	1

5.5.54.7 Example

The following is an example in FX:

```
<newparam sid="diffuseColor">
  <annotate name="UIWidget"><string>none</string></annotate>
  <semantic>DIFFUSE</semantic>
  <modifier>EXTERN</modifier>
  <float3> 0.30 0.56 0.12 </float>
</newparam>
```

The following is an example in kinematics:

```
<instance_kinematics_model url="#KINEMATICS_MODEL_ARM" sid="model">
  <newparam sid="kinematics.model">
    <SIDREF>model</SIDREF>
  </newparam>
</instance_kinematics_model>
```


5.5.55 node

Category: Scene

5.5.55.1 General

Declares a point of interest in a scene.

5.5.55.2 Concepts

The **<node>** element embodies the hierarchical relationship of elements in a scene by declaring a point of interest in a scene. A node denotes one point on a branch of the scene graph. The **<node>** element is essentially the root of a subgraph of the entire scene graph.

Within the scene graph abstraction, there are arcs and nodes. *Nodes* are points of information within the graph. *Arcs* connect nodes to other nodes. Nodes are further distinguished as interior (branch) nodes and exterior (leaf) nodes. COLLADA uses the term node to denote interior nodes. Arcs are also called *paths*.

5.5.55.3 Attributes

The **<node>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.
sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.
type	Enumeration	The type of the <node> element. Valid values are JOINT or NODE. The default is NODE. Optional.
layer	list_of_names_type	The names of the layers to which this node belongs. For example, a value of foreground glowing" indicates that this node belongs to both the layer named foreground and the layer named glowing. The default is empty, indicating that the node doesn't belong to any layer. Optional.

5.5.55.4 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	Allows the node to express asset management information. See main entry.	N/A	0 or 1
transformation_elements	Any combination of the following transformation elements: • <lookat> • <matrix> • <rotate> • <scale> • <skew> • <translate> See main entries.	None	0 or more
<instance_camera>	Allows the node to instantiate a camera object. See main entry.	N/A	0 or more
<instance_controller>	Allows the node to instantiate a controller object. See main entry.	N/A	0 or more
<instance_geometry>	Allows the node to instantiate a geometry object. See main entry.	N/A	0 or more
<instance_light>	Allows the node to instantiate a light object. See main entry.	N/A	0 or more
<instance_node>	Allows the node to instantiate a hierarchy of other nodes. See main entry.	N/A	0 or more
<node>	Allows the node to recursively define hierarchy. See main entry.	N/A	0 or more
<extra>	Allows the node to recursively define hierarchy. See main entry.	N/A	0 or more

5.5.55.5 Related elements

The **<node>** element relates to the following elements:

Parent elements	library_nodes , node , visual_scene
Child elements	See the following subclause.
Other	instance_node

5.5.55.6 Details

The **<node>** elements form the basis of the scene graph topology. As such they can have a wide range of child elements, including **<node>** elements themselves.

The **<node>** element represents a context in which the child transformation elements are composed in the order that they occur. All the other child elements are affected equally by the accumulated transformations in the scope of the **<node>** element.

The transformation elements transform the coordinate system of the **<node>** element. Mathematically, this means that the transformation elements are converted to matrices and postmultiplied in the order in which they are specified within the **<node>** to compose the coordinate system.

5.5.55.7 Example

The following example shows a simple outline of a **<visual_scene>** element with two **<node>** elements. The names of the two nodes are “earth” and “sky” respectively:

```
<visual_scene>
  <node name="earth">
  </node>
  <node name="sky">
  </node>
</visual_scene>
```

5.5.56 optics

Category: Camera

5.5.56.1 General

Represents the apparatus on a camera that projects the image onto the image sensor.

5.5.56.2 Concepts

Optics are composed of one or more optical elements. Optical elements are usually categorized by how they alter the path of light:

- Reflective elements – for example, mirrors (for example, the concave primary mirror in a Newtonian telescope, or a chrome ball, used to capture environment maps);
- Refractive elements – lenses, prisms.

A particular camera optics might have a complex combination of the above. For example, a Schmidt telescope contains both a concave lens and a concave primary mirror and lenses in the eyepiece.

A variable focal-length “zoom lens” might, in reality, contain more than 10 lenses and a variable aperture (iris).

The commonly used “perspective” camera model in computer graphics is a simple approximation of a “zoom lens” with an infinitely small aperture and the field-of-view specified directly (instead of its related value, the focal length).

5.5.56.3 Attributes

The **<optics>** element has no attributes.

5.5.56.4 Related elements

The **<optics>** element relates to the following elements:

Parent elements	camera
Child elements	See the following subclause.
Other	None

5.5.56.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<technique_common>	Specifies optics information for the common profile that all COLLADA implementations shall support. See "The common profile" clause for usage information and the following subclause for child element details. Also see main entry.	N/A	1
<technique> (Core)	Each <technique> specifies optics information for a specific profile as designated by the <technique> 's profile attribute. See main entry.	N/A	0 or more
<extra>	See main entry.	N/A	0 or more

5.5.56.6 Child elements for <optics> / <technique_common>

Name/example	Description	Default	Occurrences
<orthographic> or <perspective>	The projection types. See main entries.	N/A	1

5.5.56.7 Details

The COMMON profile defines the optics types **<perspective>** and **<orthographic>**. All other **<optics>** types shall be specified within a profile-specific **<technique>**.

5.5.56.8 Example

The following is an example of a **<camera>** element that describes a perspective view of the scene with a 45-degree field of view:

```
<camera name="eyepoint">
  <optics>
    <technique_common>
      <perspective>
        <yfov>45.0</yfov>
        <aspect_ratio>1.33333</aspect_ratio>
        <znear>0.1</znear>
        <zfar>32767.0</zfar>
```

```

        </perspective>
    </technique_common>
</optics>
</camera>

```

5.5.57 orthographic

Category: Camera

5.5.57.1 General

Describes the field of view of an orthographic camera.

5.5.57.2 Concepts

Orthographic projection describes a way of drawing a 3D scene on a 2D surface. In an orthographic projection, the apparent size of an object does not depend on its distance from the camera.

Compare to **<perspective>**.

5.5.57.3 Attributes

The **<orthographic>** element has no attributes.

5.5.57.4 Related elements

The **<orthographic>** element relates to the following elements:

Parent elements	optics / technique_common
Child elements	See the following subclause.
Other	None

5.5.57.5 Child elements

Child elements shall appear in the following order if present:

The **<orthographic>** element shall contain one of:

- A single **<xmag>** element;
- A single **<ymag>** element;
- Both an **<xmag>** and a **<ymag>** element;
- The **<aspect_ratio>** element and either **<xmag>** or **<ymag>**.

These describe the field of view of the camera. If the **<aspect_ratio>** element is not present, the aspect ratio is to be calculated from the **<xmag>** or **<ymag>** elements and the current viewport.

Name/example	Description	Default	Occurrences
<code><xmag sid="..."></code>	Contains a floating-point number describing the horizontal (X) magnification of the view. The <code>sid</code> attribute is optional.	None	See "Note"
<code><yimag sid="..."></code>	Contains a floating-point number describing the vertical (Y) magnification of the view. The <code>sid</code> attribute is optional.	None	See "Note"
<code><aspect_ratio sid="..."></code>	Contains a floating-point number describing the aspect ratio of the field of view. The <code>sid</code> attribute is optional.	None	See "Note"
<code><znear sid="..."></code>	Contains a floating-point number that describes the distance to the near clipping plane. The <code>sid</code> attribute is optional.	None	1
<code><zfar sid="..."></code>	Contains a floating-point number that describes the distance to the far clipping plane. The <code>sid</code> attribute is optional.	None	1

5.5.57.6 Details

The X and Y magnifications are simple scale factors, applied to the X and Y components of the orthographic viewport. As such, if your default orthographic viewport is $[-1,1],[-1,1]$ as in OpenGL and DirectX, your COLLADA orthographic viewport becomes $[[xmag,ymag],[-ymag,ymag]]$. This gives an orthographic width of $xmag/2$ and an orthographic height of $ymag/2$.

The centre screen pixel is assumed to be (0,0) in screen coordinates.

5.5.57.7 Example

The following is an example of an `<orthographic>` element specifying a standard view (no magnification and a standard aspect ratio):

```
<orthographic>
  <xmag sid="animated_zoom">1.0</xmag>
  <aspect_ratio>0.1</aspect_ratio>
  <znear>0.1</znear>
  <zfar>1000.0</zfar>
</orthographic>
```

5.5.58 param (data flow)

Category: Data flow

5.5.58.1 General

Declares parametric information for its parent element.

NOTE For `<param>` in other elements, see "`<param>` (reference)".

5.5.58.2 Concepts

A functional or grammatical format requires a means for users to specify parametric information. This information represents function parameter (argument) data.

Material shader programs may contain code representing vertex or pixel programs. These programs require parameters as part of their state information.

The basic declaration of a parameter describes the name, data type, and value data of the parameter. That parameter name identifies it to the function or program. The parameter type indicates the encoding of its value. The **<param>** element contains information of type `xs:string`, which is the parameter's actual value.

5.5.58.3 Attributes

The **<param>** element has the following attributes:

name	xs:token	The text string name of this element. Optional.
sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.
type	xs:NMTOKEN	The type of the value data. This text string shall be understood by the application. Required.
semantic	xs:NMTOKEN	The user-defined meaning of the parameter. Optional.

5.5.58.4 Related elements

The **<param>** element relates to the following elements:

Parent elements	accessor, bind_material
Child elements	None
Other	None

5.5.58.5 Details

The **<param>** element describes parameters for generic data flow.

5.5.58.6 Example

The following is an example of two **<param>** elements that describe the output of an **<accessor>**:

```
<accessor source="#values" count="3" stride="3">
  <param name="A" type="int" />
  <param name="B" type="int" />
</accessor>
```

5.5.59 param (reference)

Category: Parameters

Profile: External, COMMON, CG, GLES2, GLSL

5.5.59.1 General

References a predefined parameter.

NOTE For this element in `<accessor>` or `<bind_material>`, see “`<param>` (data flow)”.

5.5.59.2 Concepts

Parameters are typed data objects that are created in the runtime and are available to compilers and functions at run time.

In FX, this references a predefined parameter in shader binding declarations.

In kinematics, this references a predefined parameter in binding declarations. They provide access to specific properties of kinematics objects.

5.5.59.3 Attributes

See the “Details” subclause.

5.5.59.4 Related elements

The `<param>` element relates to the following elements:

Parent elements	See the “Details” subclause.
Child elements	None
Other	<code>modifier</code> , <code>newparam</code> , <code>setparam</code> , <code>usertype</code>

5.5.59.5 Details

`<param>` refers to the SID of an existing parameter that was created using `<newparam>`. The method of referring to the SID varies depending on the `<param>`’s parent elements.

For details about SIDs, see “Address syntax” in 5.3 Schema concepts.

5.5.59.5.1 In elements of type `fx_common_color_or_texture_type`, `common_float_or_param_type`, and `<bind_uniform>`

In the shader attribute elements (`<ambient>`, `<diffuse>`, and so on) and in `<bind_uniform>`.

The `<param>` element has the following attribute:

Ref	<code>sidref_type</code>	Required. A path to the SID of an existing parameter.

The `<param>` element relates to the following elements:

Parent elements	<code>ambient</code> (FX), <code>diffuse</code> , <code>emission</code> , <code>reflective</code> , <code>specular</code> , <code>transparent</code> , <code>index_of_refraction</code> , <code>reflectivity</code> , <code>shininess</code> , <code>transparency</code> , <code>bind_uniform</code>

The `<param>` element does not contain any information, i.e., `<param></param>` is always empty.

5.5.59.5.2 In render targets and kinematics

The **<param>** element has the following attribute:

ref	xs:token	Required. Refers to the ID of an existing parameter.

The **<param>** element relates to the following elements:

Parent elements	color_target , depth_target , stencil_target , bind (kinematics), bind_kinematics_model

The **<param>** element does not contain any information, i.e., **<param></param>** is always empty.

5.5.59.5.3 In <texture*>

The **<param>** element has no attributes.

The **<param>** element relates to the following elements:

Parent elements	texture1D , texture2D , texture3D , textureCUBE , textureRECT , textureDEPTH

The **<param>** element contains information of type **sidref_type**, which represents the SID of an existing parameter.

5.5.59.5.4 In <bind_material> and <accessor>

The **<param>** element has the following attributes:

name	xs:token	Optional. The text string name of this element.
sid	sid_type	Optional.
semantic	xs:NMTOKEN	Optional. The user-defined meaning of this parameter.
type	xs:NMTOKEN	Required. The type of the value data. This text string shall be understood by the application. When <param> is a child of the <accessor> element, this attribute restricted to the set of array types: int , float , Name , bool , IDREF , and SIDREF .

The **<param>** element relates to the following elements:

Parent elements	bind_material , accessor

The **<param>** element contains information of type **xs:string**, which represents the SID of an existing parameter.

5.5.59.6 Example

The following is an example in a shader:

```
<shader stage="VERTEX">
  <sources entry="main"><import ref="ThinFilm2"/></sources>
  <compiler_platform="PC" target="ARBVP1"/>
  <bind_uniform symbol="lightpos">
    <param ref="LightPos_03"/>
  </bind_uniform>
</shader>
```

The following is an example in kinematics:

```
<instance_articulated_system sid="system" url="#MOTION">
  <bind symbol="motion.kinematics.model">
    <param ref="kinematics.model"/>
  </bind>
</instance_articulated_system>
```

5.5.60 perspective

Category: Camera

5.5.60.1 General

Describes the field of view of a perspective camera.

5.5.60.2 Concepts

Perspective embodies the appearance of objects relative to each other as determined by their distance from a viewer. Computer graphics techniques apply a perspective projection in order to render 3D objects onto 2D surfaces to create properly proportioned images on display monitors.

Compare to **<orthographic>**.

5.5.60.3 Attributes

The **<perspective>** element has no attributes.

5.5.60.4 Child elements

Child elements shall appear in the following order if present.

The **<perspective>** element shall contain one of:

- A single **<xfov>** element;
- A single **<yfov>** element;
- Both an **<xfov>** and a **<yfov>** element;
- The **<aspect_ratio>** element and either **<xfov>** or **<yfov>**.

These describe the field of view of the camera. In the first two cases, the application can calculate the camera aspect ratio based on the viewport aspect ratio.

Name/example	Description	Default	Occurrences
<code><xfov sid="..."></code>	Contains a floating-point number describing the horizontal field of view in degrees. The <code>sid</code> attribute is optional.	None	See "Note"
<code><yfov sid="..."></code>	Contains a floating-point number describing the vertical field of view in degrees. The <code>sid</code> attribute is optional.	None	See "Note"
<code><aspect_ratio sid="..."></code>	Contains a floating-point number describing the aspect ratio of the field of view. The <code>sid</code> attribute is optional.	None	See "Note"
<code><znear sid="..."></code>	Contains a floating-point number that describes the distance to the near clipping plane. The <code>sid</code> attribute is optional.	None	1
<code><zfar sid="..."></code>	Contains a floating-point number that describes the distance to the far clipping plane. The <code>sid</code> attribute is optional.	None	1

5.5.60.5 Related elements

The `<perspective>` element relates to the following elements:

Parent elements	<code>optics / technique_common</code>
Child elements	See the following subclause.
Other	None

5.5.60.6 Details

If the `<aspect_ratio>` element is not specified, it is calculated from the `<xfov>` or `<yfov>` elements and the current viewport. The aspect ratio is defined as the ratio of the field of view's width over its height; therefore, the aspect ratio can be derived from, or be used to derive, the field of view parameters:

$$\text{aspect\_ratio} = \text{xfov} / \text{yfov}.$$

The centre screen pixel is assumed to be (0,0) in screen coordinates.

The distances to the clipping planes are specified in the current units as defined by `<asset>/<unit>` in the scope for this element.

5.5.60.7 Example

The following is an example of a `<perspective>` element specifying a horizontal field-of-view of 90 degrees that also may be targeted by an animation:

```
<perspective>
  <xfov sid="animated_zoom">90.0</xfov>
  <aspect_ratio>1.333</aspect_ratio>
  <znear>0.1</znear>
  <zfar>1000.0</zfar>
</perspective>
```

5.5.61 point

Category: Lighting

5.5.61.1 General

Describes a point light source.

5.5.61.2 Concepts

The **<point>** element declares the parameters required to describe a point light source. A point light source radiates light in all directions from a known location in space. The intensity of a point light source is attenuated as the distance to the light source increases.

The position of the light is defined by the transform of the node in which it is instantiated.

5.5.61.3 Attributes

The **<point>** element has no attributes.

5.5.61.4 Related elements

The **<point>** element relates to the following elements:

Parent elements	light / technique_common
Child elements	See the following subclause.
Other	None

5.5.61.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<color>	Contains three floating-point numbers specifying the color of the light. See main entry.	None	1
<constant_attenuation sid="...">	See "Details." The sid attribute is optional.	1.0	0 or 1
<linear_attenuation sid="...">	See "Details." The sid attribute is optional.	0.0	0 or 1
<quadratic_attenuation sid="...">	Contains a floating-point number that describes the distance to the near clipping plane. The sid attribute is optional.	None	1

5.5.61.6 Details

The **<constant_attenuation>**, **<linear_attenuation>**, and **<quadratic_attenuation>** are used to calculate the total attenuation of this light given a distance. The formula used is

$$A = \text{constant\_attenuation} + (\text{Dist} * \text{linear\_attenuation}) + ((\text{Dist}^2) * \text{quadratic\_attenuation})$$

5.5.61.7 Example

The following is an example of a **<point>** element:

```
<light id="blue">
  <technique_common>
    <point>
      <color>0.1 0.1 0.5</color>
      <linear_attenuation>0.3</linear_attenuation>
    </point>
  </technique_common>
</light>
```

5.5.62 polygons

Category: Geometry

5.5.62.1 General

Provides the information needed for a mesh to bind vertex attributes together and then organize those vertices into individual polygons.

5.5.62.2 Concepts

The **<polygons>** element declares the binding of geometric primitives and vertex attributes for a **<mesh>** element.

Polygons are not the preferred way of storing data. Use **<triangles>** or **<polylist>** for the most efficient representation of geometry. Use **<polygons>** only if holes are needed, and even then, only for the specific portions with holes.

The vertex array information is supplied in distinct attribute arrays of the **<mesh>** element that are then indexed by the **<polygons>** element.

The polygons described can contain arbitrary numbers of vertices. Ideally, they would describe convex shapes, but they also may be concave or even self-intersecting. The polygons may also contain holes. Polygon primitives that contain four or more vertices may be non-planar as well.

Many operations need an exact orientation of a surface point. The normal vector partially defines this orientation, but it still leaves the “rotation” about the normal itself ambiguous. One way to “lock down” this extra rotation is to also specify the surface tangent at the same point.

Assuming that the type of the coordinate system is known (for example, right-handed), this fully specifies the orientation of the surface, meaning that we can define a 3x3 matrix to transform between object-space and surface space.

The tangent and the normal specify two axes of the surface coordinate system (two columns of the matrix) and the third one, called binormal may be computed as the cross-product of the tangent and the normal.

COLLADA supports two different types of tangents, because they have different applications and different logical placements in a document:

- Texture-space tangents: specified with the **TEXTANGENT** and **TEXBINORMAL** semantics and the set attribute on the **<input>** (shared) elements;

- Standard (geometric) tangents: specified with the TANGENT and BINORMAL semantics on the **<input>** (shared) elements

5.5.62.3 Attributes

The **<polygons>** element has the following attributes:

count	<code>uint_type</code>	The number of polygon primitives. Required.
material	<code>xs:NCName</code>	Declares a symbol for a material. This symbol is bound to a material at the time of instantiation; see <instance_geometry> and <bind_material> . Optional. If not specified, then the lighting and shading results are application defined.
name	<code>xs:token</code>	Optional.

5.5.62.4 Related elements

The **<polygons>** element relates to the following elements:

Parent elements	mesh , convex_mesh
Child elements	See the following subclause.
Other	None

5.5.62.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (shared)	When at least one input is present, one input shall specify <code>semantic="VERTEX"</code> . See main entry.	None	0 or more
<p>	Contains a list of <code>uint_type</code> values that specifies the vertex attributes (indices) for an individual polygon. See "Details."	None	0 or more
<ph>	Describes a polygon that contains one or more holes. See the following subclause.	0	0 or more
<extra>	See main entry.	N/A	0 or more

5.5.62.6 polygons / ph child element

The **<ph>** element has no attributes.

Child elements of **<p>** shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<p>	Contains a list of <code>uint_type</code> values that specifies the vertex attributes (indices) for an individual polygon. See "Details."	None	1
<h>	Contains a list of <code>uint_type</code> values that specifies the indices of a hole in the polygon specified by <p> . See "Details."	None	1 or more

5.5.62.7 Details

The indices in a **<p>** ("primitive") (or **<h>**) element refer to different inputs depending on their order. The first index in a **<p>** element refers to all inputs with an offset of 0. The second index refers to all inputs with an offset of 1. Each vertex of the polygon is made up of one index into each input. After each input is used, the next index again refers to the inputs with offset of 0 and begins a new vertex.

The winding order of vertices produced is counter clockwise and describes the front side of each polygon.

If the primitives are assembled without vertex normals then the application may generate per-primitive normals to enable lighting.

5.5.62.8 Example

The following is an example of a **<polygons>** element that describes a single square. The **<polygons>** element contains two **<Source>** elements that contain the position and normal data, according to the **<input>** (shared) element semantics. The **<p>** element index values indicate the order in which the input values are used:

```
<mesh>
  <source id="position" />
  <source id="normal" />
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <polygons count="1" material="Bricks">
    <input semantic="VERTEX" source="#verts" offset="0"/>
    <input semantic="NORMAL" source="#normal" offset="1"/>
    <p>0 0 2 1 3 2 1 3</p>
  </polygons>
</mesh>
```

Here's a simple example of how to specify geometric tangents.

NOTE Because the normal and tangent inputs both have an offset of 1, they share an entry in the **<p>** element.

```
<mesh>
  <source id="position" />
  <source id="normal" />
  <source id="tangent" />
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <polygons count="1" material="Bricks">
```

```

    <input semantic="VERTEX" source="#verts" offset="0"/>
    <input semantic="NORMAL" source="#normal" offset="1"/>
    <input semantic="TANGENT" source="#tangent" offset="1"/>
    <p>0 0 2 1 3 2 1 3</p>
  </polygons>
</mesh>

```

Here's a simple example of how to specify texture space tangents.

NOTE that the texture space tangents are associated with the specific set of texture coordinates by the set attribute and not the offset or the order of the inputs.

```

<mesh>
  <source id="position"/>
  <source id="normal"/>
  <source id="tex-coord"/>
  <source id="tex-tangent"/>
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <polygons count="1" material="Bricks">
    <input semantic="VERTEX" source="#verts" offset="0"/>
    <input semantic="NORMAL" source="#normal" offset="1"/>
    <input semantic="TEXCOORD" source="#tex-coord" offset="2" set="0"/>
    <input semantic="TEXTANGENT" source="#tex-tangent" offset="3" set="0"/>
    <p>0 0 0 1 2 1 2 0 3 2 1 2 1 3 3 3</p>
  </polygons>
</mesh>

```

5.5.63 polylist

Category: Geometry

5.5.63.1 General

Provides the information needed for a mesh to bind vertex attributes together and then organize those vertices into individual polygons.

5.5.63.2 Concepts

The **<polylist>** element declares the binding of geometric primitives and vertex attributes for a **<mesh>** element.

The vertex array information is supplied in distinct attribute arrays of the **<mesh>** element that are then indexed by the **<polylist>** element.

The polygons described in **<polylist>** can contain an arbitrary number of vertices. Polylist primitives that contain four or more vertices may be nonplanar as well.

Many operations need an exact orientation of a surface point. The normal vector partially defines this orientation, but it still leaves the "rotation" about the normal itself ambiguous. One way to "lock down" this extra rotation is to also specify the surface tangent at the same point.

Assuming that the type of the coordinate system is known (for example, right-handed), this fully specifies the orientation of the surface, meaning that we can define a 3x3 matrix to transform between object-space and surface space.

The tangent and the normal specify two axes of the surface coordinate system (two columns of the matrix) and the third one, called binormal may be computed as the cross-product of the tangent and the normal.

COLLADA supports two different types of tangents, because they have different applications and different logical placements in a document:

- Texture-space tangents: specified with the **TEXTANGENT** and **TEXBINORMAL** semantics and the set attribute on the **<input>** (shared) elements;
- Standard (geometric) tangents: specified with the **TANGENT** and **BINORMAL** semantics on the **<input>** (shared) elements.

5.5.63.3 Attributes

The **<polylist>** element has the following attributes:

name	<code>xs:token</code>	The text string name of this element. Optional.
count	<code>uint_type</code>	The number of polygon primitives. Required.
material	<code>xs:NCName</code>	Declares a symbol for a material. This symbol is bound to a material at the time of instantiation; see <instance_geometry> and <bind_material> . Optional. If not specified, then the lighting and shading results are application defined.

5.5.63.4 Related elements

The **<polylist>** element relates to the following elements:

Parent elements	mesh
Child elements	See the following subclause.
Other	None

5.5.63.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (shared)	When at least one input is present, one input shall specify semantic="VERTEX" . See main entry.	None	0 or more
<vcount>	Contains a list of integers, each specifying the number of vertices for one polygon described by the <polylist> element.	None	0 or 1
<p>	Contains a list of integers that specify the vertex attributes (indices) for an individual polylist. ("p" stands for "primitive".)	None	0 or 1
<extra>	See main entry.	N/A	0 or more

5.5.63.6 Details

The winding order of vertices produced is counter clockwise and describes the front side of each polygon.

If the primitives are assembled without vertex normals then the application may generate per-primitive normals to enable lighting.

5.5.63.7 Example

The following is an example of a `<polylist>` element that describes two quadrilaterals and a triangle. The `<polylist>` element contains two `<Source>` elements that contain the position and normal data, according to the `<input>` (shared) element semantics. The `<p>` element index values indicate the order in which the input values are used:

```
<mesh>
  <source id="position" />
  <source id="normal" />
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <polylist count="3" material="Bricks">
    <input semantic="VERTEX" source="#verts" offset="0" />
    <input semantic="NORMAL" source="#normal" offset="1" />
    <vcount>4 4 3</vcount>
    <p>0 0 2 1 3 2 1 3 4 4 6 5 7 6 5 7 8 8 10 9 9 10</p>
  </polylist>
</mesh>
```

5.5.64 rotate

Category: Transform

5.5.64.1 General

Specifies how to rotate an object around an axis.

5.5.64.2 Concepts

Rotations change the orientation of objects in a coordinate system without any translation. Computer graphics techniques apply a rotational transformation in order to orient or otherwise move values with respect to a coordinate system. Conversely, rotation can mean the translation of the coordinate axes about the local origin.

This element contains an angle and a mathematical vector that represents the axis of rotation.

5.5.64.3 Attributes

The `<rotate>` element has the following attribute:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.

5.5.64.4 Related elements

The **<rotate>** element relates to the following elements:

Parent elements	In core: node In Physics: technique_common / mass_frame in rigid_body and instance_rigid_body , shape , ref_attachment , attachment In Kinematics: frame_object , frame_origin , frame_tcp , frame_tip , link
Child elements	None
Other	None

5.5.64.5 Details

The **<rotate>** element contains a list of four floating-point values, similar to rotations in the OpenGL® and RenderMan®² specification [13]. These values are organized into a column vector [X, Y, Z] specifying the axis of rotation followed by an angle in degrees.

For more information about how transformation elements are applied, see **<node>**.

5.5.64.6 Example

The following is an example of a **<rotate>** element forming a rotation of 90 degrees about the y axis:

```
<rotate>
  0.0 1.0 0.0 90.0
</rotate>
```

5.5.65 sampler

Category: Animation

5.5.65.1 General

Declares an interpolation sampling function for an animation.

5.5.65.2 Concepts

Animation function curves are represented by 1D **<sampler>** elements in COLLADA. The sampler defines sampling points and how to interpolate between them. When used to compute values for an animation channel, the sampling points are the animation key frames.

Sampling points (key frames) are input data sources to the sampler, as are interpolation type symbolic names. Animation channels direct the output data values of the sampler to their targets.

5.5.65.2.1 Animation curves (<animation>/<sampler>)

Animations use curves to define how animated parameters evolve over time. The definition of the curves is similar to the definitions for the **<geometry>/<spline>**, except that there is a special one-

² RenderMan is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of this product.

dimensional axis that contains the keys for the animation. The keys define how a given parameter, or a set of parameters, evolves with time throughout the animation.

Keys are often TIME values, but they can be any other variable. For example, it is possible to associate the rotation of a wheel of a train with the position of the train on the track, so, by moving the train forward or backward, the wheel and other mechanisms can automatically move.

Animations are limited to monotonic curves in the key axis, i.e. animation keys need to be sorted in increasing order of **INPUT** and cannot be duplicated. This implies that animation curves cannot be closed.

The keys are stored in the **<Source>** array, and they replace the first axis of all the **POSITION** inputs of the **<geometry>/<spline>**. Several parameters can be animated with different curves with the same key values. Those parameters are given by the **OUTPUT** array.

In short:

```
POSITION[i].X = INPUT[i]
POSITION[i].Y = OUTPUT[i]
```

And for n curves, the point i of the curve j is:

```
POSITION[j][i] = INPUT[j][i]
POSITION[j][i+1] = OUTPUT[j][i]
```

5.5.65.3 Attributes

The **<sampler>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
pre_behavior	Enumeration	Optional. Indicates what the sampled value shall be before the first key. Valid values are: • UNDEFINED: Default value. The before and after behaviours are not defined. • CONSTANT: The value for the first (behavior_before) or last (behavior_after) is returned • GRADIENT: The value follows the line given by the last two keys in the sample. (Same as LINEAR in Maya³.) • CYCLE: The key is mapped in the [first_key , last_key] interval so that the animation cycles. • OSCILLATE: The key is mapped in the [first_key , last_key] interval so that the animation oscillates. • CYCLE_RELATIVE: The animation continues indefinitely. See "Details" for more information.
post_behavior	Enumeration	Optional. Indicates what the sampled value shall be after the last key. Valid values are the same as for pre_behavior.

³ Maya is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of this product.

5.5.65.4 Related elements

The **<sampler>** element relates to the following elements:

Parent elements	animation
Child elements	See the following subclause.
Other	None

5.5.65.5 Child elements

Name/example	Description	Default	Occurrences
<input> (unshared)	At least one <input> (unshared) element shall have a semantic attribute whose value is INTERPOLATION. See main entry.	None	1 or more

5.5.65.6 Details

Sampling points are described by the **<input>** elements, which refer to **<Source>** elements. The semantic attribute of the **<input>** element can be one of, but is not limited to, INPUT, INTERPOLATION, IN_TANGENT, OUT_TANGENT, or OUTPUT.

COLLADA recognizes the following interpolation types: LINEAR, BEZIER, CARDINAL, HERMITE, BSPLINE, and STEP. These symbolic names are held in a **<Source>** element that contains a **<Name_array>** that stores them. These values are fed into the sampler by the INTERPOLATION **<input>** element.

To be complete, a **<sampler>** element shall contain an **<input>** element with a semantic attribute of INTERPOLATION. COLLADA does not specify a default interpolation type. If an interpolation type is not specified, the resulting **<sampler>** behaviour is application defined.

For more information, see “Curve Interpolation” in 5.4 Programming guideline.

5.5.65.6.1 STEP interpolation

Animation curves allow an additional type of interpolation: STEP. This says that the value remains constant to the value of the first point of the segment, until the next segment, as in the following curve (Figure 11):

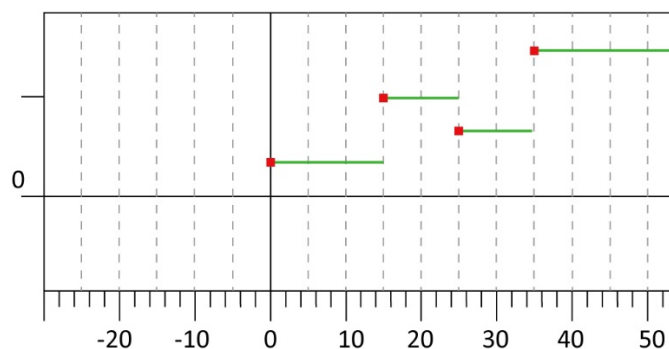


Figure 11 — STEP Interpolation

The COLLADA code for this would be:

```
<animation>
  <source id="time_axis" >
    <float_array count="4"... >
      ... <technique_common><accessor>
        <param name="TIME">
          ...</accessor></technique_common>
      ...</source>
  <source id="positions" >
    <float_array count="4" ...>
      <technique_common>... <accessor>
        <param name="name_of_parameter_animated" type="float" ...
          ...</accessor></technique_common>
      ...</source>
  <source id="interpolations" >
    <Name_array count="4"> STEP STEP STEP STEP </Name_array>
    <!-- last one ignored -->
    <technique_common>... ... <accessor>
      <param name="INTERPOLATION" type="Name" ...
        ...</accessor></technique_common>
    ...</source>
  <sampler>
    <input semantic="INPUT" source = "#time_axis" />
    <input semantic="OUTPUT" source="#positions" />
    <input semantic="INTERPOLATION" source="#interpolations" />
```

5.5.65.6.2 Linear animation curves

The LINEAR interpolation is similar to STEP, but the parameter's value is interpolated linearly between the key values.

5.5.65.6.3 Bézier and Hermite animation curves

BEZIER and HERMITE interpolations are similar to the description given for **<spline>** except that there is no POSITION **<input>** semantic, but rather INPUT and OUTPUT semantics. The INPUT and OUTPUT semantics are always 1D parameters. As explained already, if OUTPUT has more than one dimension, then several parameters are interpolated independently using the same key values. The IN_TANGENT and OUT_TANGENT semantics have one key value, and then one value for each parameter.

The same formulae for cubic Bézier and Hermite interpolation already defined for **<spline>** are to be used, with the following geometry vector, for parameter j , segment $[i]$:

For Bézier:

- P_0 is (INPUT $[i]$, OUTPUT $[j][i]$);
- C_0 (or T_0) is (OUT_TANGENT $[0][i]$, OUT_TANGENT $[j][i]$);
- C_1 (or T_1) is (IN_TANGENT $[0][i+1]$, IN_TANGENT $[j][i+1]$);
- P_1 is (INPUT $[i+1]$, OUTPUT $[j][i+1]$).

5.5.65.6.4 Special case: 1D tangent values

Some exporters have been exporting curves with a degenerate form of tangent. This is not supported by the COLLADA specification, and the degenerate cases shall disappear with updates to the affected exporters. The following is provided for informational purposes only.

In this special case of 1D tangent data, the `OUT_TANGENT` and `IN_TANGENT` do not include the key values, and therefore have the same dimension as the `OUTPUT` array.

The missing key values are provided as a linear interpolation of the keys provided by the `INPUT` segment. The geometry vector values are provided the same way as for a regular animation curve:

- P_0 is $(\text{INPUT}[i], \text{OUTPUT}[j][i])$
- C_0 is $(\text{INPUT}[i]/3 + \text{INPUT}[i+1] * 2/3, \text{OUT\_TANGENT}[j][i])$
- C_1 is $(\text{INPUT}[i] * 2/3 + \text{INPUT}[i+1]/3, \text{IN\_TANGENT}[j][i+1])$
- C_1 is $(\text{INPUT}[i+1], \text{OUTPUT}[j][i+1])$

5.5.65.6.5 B-Spline and cardinal animation curves

The same principles discussed previously apply to `BSPLINE` and `CARDINAL` curves. The `POSITION` is given by combining the `INPUT` and `OUTPUT`. The same formulae defined previously apply to these animation curves.

5.5.65.6.6 Behavior before and after

The two optional attributes `pre_behavior` and `post_behavior` indicate what the sampled value shall be before the first key and after the last key. The following Figure 12, Figure 13, Figure 14, Figure 15, Figure 16 and pseudocodes provide examples of the different behaviour options.

The behaviour for `CONSTANT` is:

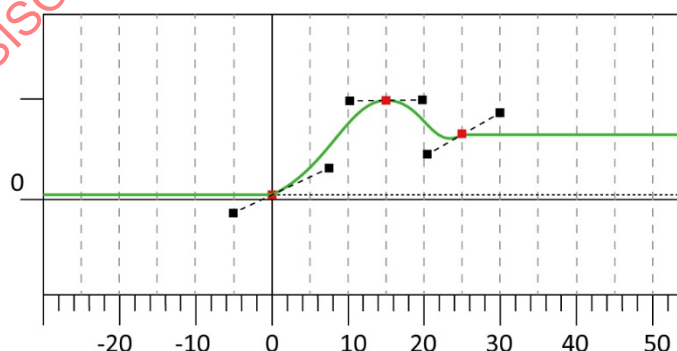


Figure 12 — `CONSTANT` behaviour

The behaviour for GRADIENT is:

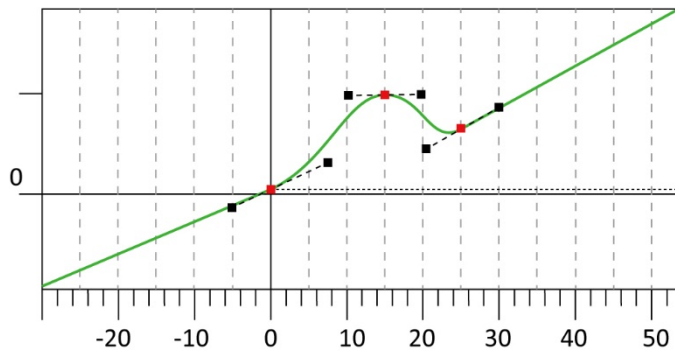


Figure 13 — GRADIENT behaviour

```
++ pseudo code for post_behavior: key > last_key > first_key
if the INTERPOLATION provides tangent values:
    tangent = tangent[last_key]
else
    tangent = value[last_key]-value[penultimate_key] / (last_key -
penultimate_key)
return value[last_key] + tangent * (key-last_key)
++ pseudo code for pre_behavior: key < first_key < last_key
if the INTERPOLATION provides tangent values:
    tangent = tangent[first_key]
else
    tangent = value[second_key]-value[first_key] / (second_key- first_key)
return value[first_key] + tangent * (key-first_key)
```

The behaviour for CYCLE is:

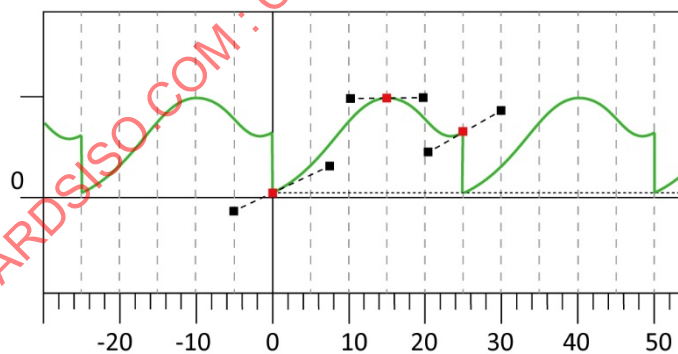


Figure 14 — CYCLE behaviour

```
++ pseudo code for post_behavior: key > last_key > first_key
repeat = (int) (key - first_key) / (last_key - first_key)
new_key = key - (last_key - first_key) * repeat;
return value[new_key]
++ pseudo code for pre_behavior: key < first_key < last_key
repeat = (int) (first_key - key) / (last_key - first_key)
new_key = key + (last_key - first_key) * (repeat + 1);
return value[new_key]
```

The behaviour for OSCILLATE is:

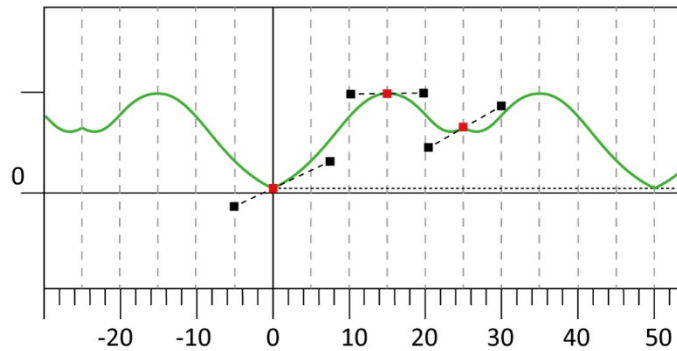


Figure 15 — OSCILLATE behaviour

```

++ pseudo code for post_behavior: key > last_key > first_key
repeat = (int) (key - first_key) / (last_key - first_key)
if (repeat is even) // same as CYCLE
    new_key = key - (last_key - first_key) * repeat;
else // play animation backward
    new_key = first_key + last_key - (key - (last_key - first_key) * repeat);
return value[new_key]
++ pseudo code for pre_behavior: key < first_key < last_key
repeat = (int) (first_key - key) / (last_key - first_key)
if (repeat is odd) // same as CYCLE
    new_key = key + (last_key - first_key) * (repeat + 1);
else // play animation backward
    new_key = first_key + last_key - (key + (last_key - first_key) * (repeat + 1));
return value[new_key]

```

The behaviour for CYCLE_RELATIVE is:

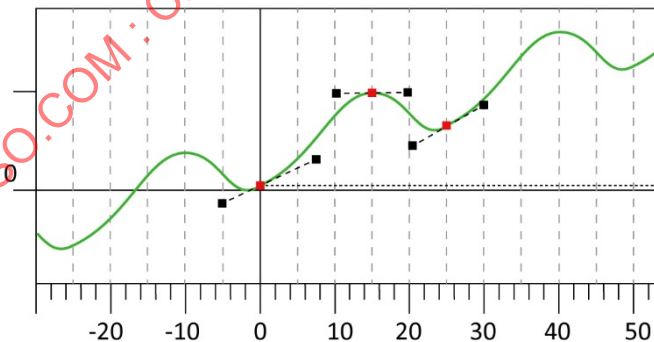


Figure 16 — CYCLE_RELATIVE behaviour

```

++ pseudo code for post_behavior: key > last_key > first_key
repeat = (int) (key - first_key) / (last_key - first_key)
new_key = key - (last_key - first_key) * repeat;
return value[new_key] + (value[last_key] - value[first_key])*repeat
++ pseudo code for pre_behavior: key < first_key < last_key
repeat = (int) (key - first_key) / (last_key - first_key)
new_key = key + (last_key - first_key) * (repeat + 1);
return value[new_key] - (value[last_key] - value[first_key])*(repeat+1)

```

5.5.65.7 Example

The following is an example of a **<sampler>** element that evaluates the y-axis values of a key-frame source element whose id is "group1_translate-anim-outputY". The INTERPOLATION inputs are shown in their **<Source>** element for added clarity:

```

<animation id="group1_translate-anim">
  <source id="group1_translate-anim-inputY">
    ...
  </source>
  <source id="group1_translate-anim-outputY">
    ...
  </source>
  <source id="group1_translate-anim-interpY">
    <Name_array count="3" id="group1_translate-anim-interpY-array">
      BEZIER BEZIER BEZIER
    </Name_array>
    <technique_common>
      <accessor count="3" source="#group1_translate-anim-interpY-array">
        <param name="Y" type="Name"/>
      </accessor>
    </technique_common>
  </source>
  <sampler id="group1_translate-anim-samplerY">
    <input semantic="INPUT" source="#group1_translate-anim-inputY"/>
    <input semantic="OUTPUT" source="#group1_translate-anim-outputY"/>
    <input semantic="IN_TANGENT" source="#group1_translate-anim-intanY"/>
    <input semantic="OUT_TANGENT" source="#group1_translate-anim-outtanY"/>
    <input semantic="INTERPOLATION" source="#group1_translate-anim-interpY"/>
  </sampler>
  <channel source="#group1_translate-anim-samplerY"
    target="group1/translate.Y"/>
</animation>

```

5.5.66 scale

Category: Transform

5.5.66.1 General

Specifies how to change an object's size.

5.5.66.2 Concepts

Scaling changes the size of objects in a coordinate system without any rotation or translation. Computer graphics techniques apply a scale transformation to change the size or proportions of values with respect to a coordinate system axis.

This element contains a mathematical vector that represents the relative proportions of the x, y, and z axes of a coordinate system.

5.5.66.3 Attributes

The **<scale>** element has the following attribute:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.

5.5.66.4 Related elements

The **<scale>** element relates to the following elements:

Parent elements	node
Child elements	None
Other	None

5.5.66.5 Details

The **<scale>** element contains a list of three floating-point values. These values are organized into a column vector suitable for matrix composition.

A zero scale value results in a projection onto the plane that is perpendicular to that axis. For example, if $z=0$ then all points are on the (x,y) plane. A negative scale value results in a scale with reflection on the associated axis.

For more information about how transformation elements are applied, see **<node>**.

5.5.66.6 Example

The following is an example of a **<scale>** element that describes a uniform increase in size of an object (or coordinate system) by a factor of two:

```
<scale>
  2.0 2.0 2.0
</scale>
```

5.5.67 scene

Category: Scene

5.5.67.1 General

Embodies the entire set of information that can be visualized from the contents of a COLLADA resource.

5.5.67.2 Concepts

Each COLLADA document can contain, at most, one **<Scene>** element.

The **<Scene>** element declares the base of the scene hierarchy or scene graph. The scene contains elements that provide much of the visual and transformational information content as created by the authoring tools.

The hierarchical structure of the scene is organized into a scene graph. A scene graph is a directed acyclic graph (DAG) or tree data structure that contains nodes of visual information and related data. The structure of the scene graph contributes to optimal processing and rendering of the data and is therefore widely used in the computer graphics domain.

5.5.67.3 Attributes

The **<Scene>** element has no attributes.

5.5.67.4 Related elements

The **<Scene>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.5.67.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<instance_physics_scene>	See main entry in Physics.	N/A	0 or more
<instance_visual_scene>	See main entry.	N/A	0 or 1
<instance_kinematics_scene>	See main entry in Kinematics.	N/A	0 or 1
<extra>	See main entry.	N/A	0 or more

5.5.67.6 Details

There is at most one **<Scene>** element declared under the **<COLLADA>** document (root) element. The scene graph is built from the **<visual_scene>** elements instantiated under **<Scene>**. The instantiated **<physics_scene>** elements describe any physics being applied to the scene.

5.5.67.7 Example

The following example shows a simple **<Scene>** element that instantiates a visual scene with the id 'world':

```
<COLLADA>
  <scene>
    <instance_visual_scene url="#world"/>
  </scene>
</COLLADA>
```

5.5.68 setparam

Category: Parameters

FX Profile: **External**, **Effect**, **CG**, **GLS2**

5.5.68.1 General

Assigns a new value to a previously defined parameter.

5.5.68.2 Concepts

Parameters can be defined at run time as **<newparam>** or can be discovered as global parameters in source code or precompiled binaries at compile/link time. Each **<setparam>** is a speculative call, saying in effect:

- Search for a symbol called "X". If you find one in the current scope, attempt to assign a value of this data type to it. If you do not find the symbol or cannot assign the value, ignore and continue loading.

In FX, under advanced language profiles, **<setparam>** can be used to assign concrete array sizes to previously unsized arrays using the **<array length="N"/>** element as well as connect instances of **<usertype>** parameters to abstract interface typed parameters.

Outside of FX, **<setparam>** can assign values only from the pool of common COLLADA data types.

5.5.68.3 Attributes

The **<setparam>** element has the following attributes:

ref	xs:token	References the ID of the predefined parameter that will have its value set. Required.

5.5.68.4 Related elements

The **<setparam>** element relates to the following elements:

Parent elements	In FX: instance_effect , usertype In Kinematics: instance_articulated_system , instance_kinematics_scene , axis_info , effector_info , instance_kinematics_model , In core: instance_formula
Child elements	See the following subclause.
Other	newparam , param (reference)

5.5.68.5 Child elements

Name/example	Description	Default	Occurrences
parameter_type_element	See “Parameter-Type Elements” for parameter-type elements valid in the appropriate scope: • CG: cg_param_group • GLES2: gles2_value_group • In <instance_effect>: fx_setparam_group • Kinematics and <instance_formula>, shall be one of the following: • <float> • <int> • <bool> • <SIDREF> • <connect_param>: See main entry in Kinematics.	N/A	1

5.5.68.6 Details

FX Runtime loaders are free to report failed **<setparam>** attempts but shall not abort loading an effect on failure.

5.5.68.7 Example

The following is an example in FX:

```
<setparam ref="light_Direction">
  <float3> 0.0 1.0 0.0 </float3>
</setparam>
```

The following is an example in kinematics:

```
<instance_articulated_system url="#MOTION_SYSTEM" sid="model">
  <setparam ref="motion.model.elbow.x.locked">
    <bool>true</bool>
  </setparam>
</instance_articulated_system>
```

5.5.69 SIDREF_array

Category: Data flow

5.5.69.1 General

Declares the storage for a homogenous array of scoped-identifier reference values.

5.5.69.2 Concepts

The **<SIDREF_array>** element stores values that reference scoped identifiers (SIDs) within the instance document.

5.5.69.3 Attributes

The **<SIDREF_array>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of this element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.
count	uint_type	The number of values in the array. Required.

5.5.69.4 Related elements

The **<SIDREF_array>** element relates to the following elements:

Parent elements	Source (Core)
Child elements	None
Other	accessor

5.5.69.5 Details

An **<SIDREF_array>** element contains a list of COLLADA scoped-identifier address values (sidref_type). These values are a repository of data for **<Source>** elements.

For details about scoped identifiers, see “Address syntax” in 5.3 Schema concepts.

5.5.69.6 Example

The following is an example of an **<SIDREF_array>** element:

```
<source id="ring.brep.lib.geo.brep.geom-curves2d">
  <SIDREF_array count="12" id="ring.brep.lib.geo.brep.geom-curves2e-array">
    curve2d-1 curve2d-2 curve2d-3 curve2d-4
    curve2d-5 curve2d-6 curve2d-7 curve2d-8
    curve2d-9 curve2d-10 curve2d-11 curve2d-12
  </SIDREF_array>
</source>
```

5.5.70 skeleton

Category: Controller

5.5.70.1 General

Indicates where a skin controller is to start searching for the joint nodes that it needs.

5.5.70.2 Concepts

As a scene graph increases in complexity, the same object might have to appear in the scene more than once. To save space, the actual data representation of an object can be stored once and referenced in multiple places. However, the scene might require that the object be transformed in various ways each time it appears. In the case of a skin controller, the object's transformation is derived from a set of external nodes.

There may be occasions where multiple instances of the same skin controller need to reference separate instances of a set of nodes. This is the case when each controller needs to be animated independently because, to animate a skin controller, you shall animate the nodes that influence it.

There may also be occasions where instances of different skin controllers might need to reference the same set of nodes, for example when attaching clothing or armour to a character. This allows the transformation of both controllers from the manipulation of a single set of nodes.

5.5.70.3 Attributes

The **<skeleton>** element has no attributes.

5.5.70.4 Related elements

The **<skeleton>** element relates to the following elements:

Parent elements	instance_controller
Child elements	None
Other	None

5.5.70.5 Details

This element contains a URI of type `xs:anyURI`.

5.5.70.6 Example

The following example shows how the **<skeleton>** element is used to bind two controller instances that refer to the same locally defined **<controller>** element, identified as “skin”, to different instances of a skeleton:

```
<library_controllers>
  <controller id="skin">
    <skin source="#base_mesh">
      <source id="Joints">
        <Name_array count="4"> Root Spine1 Spine2 Head </Name_array>
        ...
      </source>
      <source id="Weights"/>
      <source id="Inv_bind_mats"/>
      <joints>
        <input source="#Joints" semantic="JOINT"/>
      </joints>
      <vertex_weights/>
    </skin>
  </controller>
</library_controllers>
<library_nodes>
  <node id="Skeleton1" sid="Root">
    <node sid="Spine1">
      <node sid="Spine2">
        <node sid="Head"/>
      </node>
    </node>
  </node>
</library_nodes>
```



```

<node id="skel01">
  <instance_node url="#Skeleton1"/>
</node>
<node id="skel02">
  <instance_node url="#Skeleton1"/>
</node>
<node>
  <instance_controller url="#skin">
    <skeleton>#skel01</skeleton>
  </instance_controller>
</node>
<node>
  <instance_controller url="#skin">
    <skeleton>#skel02</skeleton>
  </instance_controller>
</node>

```

5.5.71 skew

Category: Transform

5.5.71.1 General

Specifies how to deform an object along one axis.

5.5.71.2 Concepts

Skew (shear) deforms an object along one axis of a coordinate system. It translates values along the affected axis in a direction that is parallel to that axis. Computer graphics techniques apply a skew or shear transformation to deform objects or to correct distortion in images.

This element contains an angle and two mathematical vectors that represent the axis of rotation and the axis of translation.

5.5.71.3 Attributes

The **<skew>** element has the following attribute:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.

5.5.71.4 Related elements

The **<skew>** element relates to the following elements:

Parent elements	node
Child elements	None
Other	None

5.5.71.5 Details

As in the RenderMan® specification [13], the **<skew>** element contains a list of seven floating-point values. These values are organized into an angle in degrees followed by two column vectors specifying the axes of rotation and translation.

For more information about how transformation elements are applied, see **<node>**.

5.5.71.6 Example

The following is an example of a **<skew>** element forming a displacement of points along the x axis due to a rotation of 45 degrees around the y axis:

```
<skew>
  45.0 0.0 1.0 0.0 1.0 0.0 0.0
</skew>
```

5.5.72 skin

Category: Controller

5.5.72.1 General

Contains vertex and primitive information sufficient to describe blend-weight skinning.

5.5.72.2 Concepts

For character skinning, an animation engine drives the joints (skeleton) of a skinned character. A skin mesh describes the associations between the joints and the mesh vertices forming the skin topology. The joints influence the transformation of skin mesh vertices according to a controlling algorithm.

A common skinning algorithm blends the influences of neighbouring joints according to weighted values.

The classical skinning algorithm transforms points of a geometry (for example vertices of a mesh) with matrices of nodes (sometimes called joints) and averages the result using scalar weights. The affected geometry is called the skin, the combination of a transform (node) and its corresponding weight is called an influence, and the set of influencing nodes (usually a hierarchy) is called a skeleton.

“Skinning” involves two steps:

- Pre-processing, known as “binding the skeleton to the skin”.
- Running the skinning algorithm to modify the shape of the skin as the pose of the skeleton changes. The results of the pre-processing, or “skinning information” consists of the following:
 - bind-shape: also called “default shape”. This is the shape of the skin when it was bound to the skeleton. This includes positions (required) for each corresponding **<mesh>** vertex and may optionally include additional vertex attributes.
 - influences: a variable-length lists of node + weight pairs for each **<mesh>** vertex.
 - bind-pose: the transforms of all influences at the time of binding. This per-node information is usually represented by a “bind-matrix”, which is the local-to-world matrix of a node at the time of binding.

In the skinning algorithm, all transformations are done relative to the bind-pose. This relative transform is usually pre-computed for each node in the skeleton and is stored as a skinning matrix.

To derive the new (“skinned”) position of a vertex, the skinning matrix of each influencing node transforms the bind-shape position of the vertex and the result is averaged using the blending weights.

The easiest way to derive the skinning matrix is to multiply the current local-to-world matrix of a node by the inverse of the node’s bind-matrix. This effectively cancels out the bind-pose transform of each node and allows us to work in the common object space of the skin.

The binding process usually involves:

- Storing the current shape of the skin as the bind-shape;
- Computing and storing the bind-matrices;
- Generating default blending weights, usually with some fall-off function: the farther a joint is from a given vertex, the less it influences it. Also, if a weight is 0, the influence can be omitted.

After that, the artist is allowed to hand-modify the weights, usually by “painting” them on the mesh.

5.5.72.3 Attributes

The **<skin>** element has the following attribute:

source	xs:anyURI	A URI reference to the base mesh (a static mesh or a morphed mesh). This also provides the bind-shape of the skinned mesh. Required.

5.5.72.4 Related elements

The **<skin>** element relates to the following elements:

Parent elements	controller
Child elements	See the following subclause.
Other	None

5.5.72.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<bind_shape_matrix>	Provides extra information about the position and orientation of the base mesh before binding. Contains sixteen floating-point numbers representing a four-by-four matrix in column-major order; it is written in row-major order in the COLLADA document for human readability. If <bind_shape_matrix> is not specified, then an identity matrix may be used as the <bind_shape_matrix> . This element has no attributes.	None	0 or 1
<Source>	Provides most of the data required for skinning the given base mesh. See main entry.	N/A	3 or more
<joints>	Aggregates the per-joint information needed for this skin. See main entry.	N/A	1
<vertex_weights>	Describes a per-vertex combination of joints and weights used in this skin. An index of -1 into the array of joints refers to the bind shape. Weights shall be normalized before use. See main entry.	N/A	1
<extra>	See main entry.	N/A	0 or more

5.5.72.6 Details

The skinning calculation for each vertex v in a bind shape is

$$outv = \sum_{i=0}^n \{ ((v * BSM) * IBM_i * JM_i) * JW \}$$

where

- n is the number of joints that influence vertex v
- BSM is the bind shape matrix
- IBM_i is the inverse bind matrix of joint i
- JM_i is the joint matrix of joint i
- JW is the joint weight/influence of joint i on vertex v

Common optimizations include:

- $(v * BSM)$ is calculated and stored at load time.

Definitions related to skinning in COLLADA:

- Bind shape (or base mesh) : The vertices of the mesh referred to by the source attribute of the **<skin>** element.

- Joints: Nodes specified by SID in the **<Source>** referred to by the **<input>** (unshared) element with semantic="JOINT". The SIDs are typically stored in a **<Name_array>** where one name represents one SID (node). Upon instantiation of a skin controller, the **<skeleton>** elements define where to start the SID lookup. The joint matrices can be obtained at runtime from these nodes.
- Weights: Values in the **<Source>** referred to by the **<input>** (unshared) element with semantic="WEIGHT". Typically stored in a **<float_array>** and taken one floating-point number at a time. The **<vertex_weights>** element describes the combination of joints and weights used by the skin.
- Inverse bind matrix: Values in the **<Source>** element referred to by the **<input>** (unshared) element with semantic="INV_BIND_MATRIX". Typically stored in a **<float_array>** taken 16 floating-point numbers at a time. The **<joints>** element associates the joints to their inverse bind matrices.
- Bind shape matrix: A single matrix that represents the transform of the bind shape before skinning.

5.5.72.7 Example

The following is an example of a **<skin>** element with the allowed attributes:

```
<controller id="skin">
  <skin source="#base_mesh">
    <source id="Joints">
      <Name_array count="4"> Root Spine1 Spine2 Head </Name_array>
      ...
    </source>
    <source id="Weights">
      <float_array count="4"> 0.0 0.33 0.66 1.0 </float_array>
      ...
    </source>
    <source id="Inv_bind_mats">
      <float_array count="64"> ... </float_array>
      ...
    </source>
    <joints>
      <input semantic="JOINT" source="#Joints"/>
      <input semantic="INV_BIND_MATRIX" source="#Inv_bind_mats"/>
    </joints>
    <vertex_weights count="4">
      <input semantic="JOINT" source="#Joints"/>
      <input semantic="WEIGHT" source="#Weights"/>
      <vcount>3 2 2 3</vcount>
      <v>
        -1 0 0 1 1 2
        -1 3 1 4
        -1 3 2 4
        -1 0 3 1 2 2
      </v>
    </vertex_weights>
  </skin>
</controller>
```

5.5.73 source (Core)

Category: Data flow

5.5.73.1 General

Declares a data repository that provides values according to the semantics of an **<input>** element that refers to it.

NOTE For **<source>** in **<sampler*>** elements, see those elements.

5.5.73.2 Concepts

A data source is a well-known source of information that can be accessed through an established communication channel.

The data source provides access methods to the information. These access methods implement various techniques according to the representation of the information. The information may be stored locally as an array of data or a program that generates the data.

5.5.73.3 Attributes

The **<source>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Required.
name	xs:token	The text string name of this element. Optional.

5.5.73.4 Related elements

The **<source>** element relates to the following elements:

Parent elements	In core: animation, mesh, morph, skin, spline In Physics: convex_mesh In B-Rep: brep, nurbs, nurbs_surface
Child elements	See the following subclause.
Other	accessor

5.5.73.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
array_element	A data array element. Can be one of: • <bool_array> • <float_array> • <IDREF_array> • <int_array> • <Name_array> • <SIDREF_array> • <Token_array> See main entries.	None	0 or 1
<technique_common>	Specifies source information for the common profile that all COLLADA implementations shall support. See "The common profile" clause for usage information and the following subclause for child element details.	N/A	0 or 1
<technique> (Core)	Each <technique> specifies source information for a specific profile as designated by the <technique> 's profile attribute. See main entry.	N/A	0 or more

5.5.73.6 Child elements of **<source>** / **<technique_common>**

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	1

5.5.73.7 Example

The following is an example of a **<source>** element that contains an array of floating-point values that compose a single RGB color:

```
<source id="color_source" name="Colors">
  <float_array id="values" count="3">
    0.8 0.8 0.8
  </float_array>
  <technique_common>
    <accessor source="#values" count="1" stride="3">
      <param name="R" type="float"/>
      <param name="G" type="float"/>
      <param name="B" type="float"/>
    </accessor>
  </technique_common>
</source>
```

5.5.74 spline

Category: **Geometry**

5.5.74.1 General

Describes a multi-segment spline with control vertex (CV) and segment information.

5.5.74.2 Concepts

The organization of **<spline>** is very similar to that of **<mesh>**. A **<spline>** contains **<Source>** elements that provide the attributes and a **<control_vertices>** element to “assemble” the attribute streams. Information about each segment is stored with the information about its preceding control vertex.

5.5.74.3 Attributes

The **<spline>** element has the following attribute:

closed	xs:boolean	Whether there is a segment connecting the first and last control vertices. The default is false, indicating that the spline is open. Optional.

5.5.74.4 Related elements

The **<spline>** element relates to the following elements:

Parent elements	geometry
Child elements	See the following subclause.
Other	None

5.5.74.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<Source>	Provides the values for the CVs and segments of the spline. See main entry.	N/A	1 or more
<control_vertices>	Describes the CVs of the spline. See main entry.	N/A	1
<extra>	See main entry.	N/A	0 or more

5.5.74.6 Details

For more information, see:

- <control_vertices>
- “Curve Interpolation” in 5.4 Programming guideline.

5.5.74.7 Example

The following is an example of an empty <spline> element with the allowed attributes:

```
<spline closed="true">
  <source id="CVs-Pos" />
  <source id="CVs-Interp" />
  <source id="CVs-LinSteps" />
  <control_vertices>
    <input semantic="POSITION" source="#CVs-Pos"/>
    <input semantic="INTERPOLATION" source="#CVs-Interp"/>
    <input semantic="LINEAR STEPS" source="#CVs-LinSteps"/>
  </control_vertices>
</spline>
```

5.5.75 spot

Category: Lighting

5.5.75.1 General

Describes a spot light source.

5.5.75.2 Concepts

A spot light source radiates light in one direction in a cone shape from a known location in space. The intensity of the light is attenuated as the radiation angle increases away from the direction of the light source. The intensity of a spot light source is also attenuated as the distance to the light source increases.

The light's default direction vector in local coordinates is [0,0,-1], pointing down the negative z axis. The actual direction of the light is defined by the transform of the node in which the light is instantiated.

5.5.75.3 Attributes

The **<spot>** element has no attributes.

5.5.75.4 Related elements

The **<spot>** element relates to the following elements:

Parent elements	light / technique_common
Child elements	See the following subclause.
Other	None

5.5.75.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<color>	Contains three floating-point spot numbers specifying the color of the light. See main entry.	None	1
<constant_attenuation sid="...">	The sid attribute is optional.	1.0	0 or 1
<linear_attenuation sid="...">	The sid attribute is optional.	0.0	0 or 1
<quadratic_attenuatio n sid="...">	The sid attribute is optional.	0.0	0 or 1
<falloff_angle sid="...">	The sid attribute is optional.	180.0	0 or 1
<falloff_exponent sid="...">	The sid attribute is optional.	0.0	0 or 1

5.5.75.6 Details

The **<constant_attenuation>**, **<linear_attenuation>**, and **<quadratic_attenuation>** are used to calculate the total attenuation of this light given a distance. The equation used is

$$A = \text{constant\_attenuation} + (\text{Dist} * \text{linear\_attenuation}) + ((\text{Dist}^2) * \text{quadratic\_attenuation})$$

The **<falloff_angle>** and **<falloff_exponent>** are used to specify the amount of attenuation based on the direction of the light.

5.5.75.7 Example

The following is an example of a **<spot>** element:

```
<light id="blue">
  <technique_common>
    <spot>
      <color>0.1 0.1 0.5</color>
      <linear_attenuation>0.3</linear_attenuation>
    </spot>
  </technique_common>
</light>
```

5.5.76 targets

Category: Controller

5.5.76.1 General

Declares morph targets, their weights, and any user-defined attributes associated with them.

5.5.76.2 Concepts

The **<targets>** element declares the morph targets and the morph weights. The **<input>** elements define the set of meshes to be blended, and the array of weights used to blend between them. They can also be used to specify additional information to be associated with the morph targets.

5.5.76.3 Attributes

The **<targets>** element has no attributes.

5.5.76.4 Related elements

The **<targets>** element relates to the following elements:

Parent elements	morph
Child elements	See the following subclause.
Other	None

5.5.76.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (unshared)	Shall occur once with semantic="MORPH_TARGET" and once with semantic="MORPH_WEIGHT" . See main entry.	None	2 or more
<extra>	See main entry.	N/A	0 or more

5.5.76.6 Example

The following is an example of a complete **<targets>** element:

```
<targets>
  <input source="#morph-targets" semantic="MORPH_TARGET">
  <input source="#morph-weights" semantic="MORPH_WEIGHT">
</targets>
```

5.5.77 technique (Core)

Category: Extensibility

5.5.77.1 General

Declares the information used to process some portion of the content.

For **<technique>** in **<profile_*>** elements, see "**<technique>** (FX)."

5.5.77.2 Concepts

A technique describes information needed by a specific platform or program. The platform or program is specified with the `profile` attribute. Each technique conforms to an associated profile. Two things define the context for a technique: its profile and its parent element in the instance document.

Techniques generally act as a "switch". If more than one is present for a particular portion of content on import, one or the other is picked, but usually not both. Selection shall be based on which profile the importing application can support.

Techniques contain application data and programs, making them assets that can be managed as a unit.

5.5.77.3 Attributes

The **<technique>** element has the following attribute:

profile	xs:NMTOKEN	The type of profile. This is a vendor-defined character string that indicates the platform or capability target for the technique. Required.
xmlns	xs:anyURI	This XML Schema namespace attribute identifies an additional schema to use for validating the content of this instance document. Optional.

5.5.77.4 Related elements

The **<technique>** element relates to the following elements:

Parent elements	extra , Source (Core), light , optics , imager , force_field , physics_material , physics_scene , rigid_body , rigid_constraint , instance_rigid_body , bind_material , motion , kinematics , kinematics_model
Child elements	See "Details"
Other	None

5.5.77.5 Details

The **<technique>** element can contain any well-formed XML data. Any data that can be, will be validated against the COLLADA schema. It is also possible to specify another schema to use for validating the data. Anything else will also be considered legal, but cannot be validated.

5.5.77.6 Example

The following is an example of the different things that can be done in a **<technique>**

```
<technique profile="Max" xmlns:max="some/max/schema">
  <param name="wow" sid="animated" type="string">a validated string parameter
  from the COLLADA schema.</param>
  <max:someElement>defined in the Max schema and validated.</max:someElement>
  <uhoh>something well-formed and legal, but that can't be validated because
  there is no schema for it!</uhoh>
</technique>
```

Unqualified element names are not recommended because they cannot be validated, and they could become invalid if future COLLADA schema use that element name.

The following example shows roughly equivalent operations for the platform or profile named "OTHER" and for all other platforms (the **<technique_common>** information):

```
<channel source="#YFOVSampler" target="Camera01/YFOV"/>
...
<camera id="#Camera01">
  <optics>
    <technique_common>
      <perspective>
        <yfov sid="YFOV">45.0</yfov>
        <aspect_ratio>1.33333</aspect_ratio>
        <znear>1.0</znear>
        <zfar>1000.0</zfar>
      </perspective>
    </technique_common>
    <technique profile="OTHER">
      <param sid="YFOV" type="float">45.0</param>
      <otherStuff type="MySpecialCamera">DATA</otherStuff>
    </technique>
  </optics>
</camera>
```

5.5.78 technique_common

Category: Extensibility

5.5.78.1 General

Specifies information for a specific element for the common profile that all COLLADA implementations shall support.

5.5.78.2 Concepts

Specifies technique information that consuming applications can use if no technique specific to the application is provided in the COLLADA document.

In other words, if an element has **<technique>** child elements for one or more specific profiles, applications reading the COLLADA document shall use the technique most appropriate for the application. If none of the specific **<technique>**s is appropriate, the application shall use the element's **<technique_common>** instead, if one is specified.

Each element's **<technique_common>** attributes and children are unique. Refer to each parent element for details.

5.5.78.3 Attributes

See main entries for each parent element.

5.5.78.4 Related elements

The **<technique_common>** element relates to the following elements:

Parent elements	bind_material, instance_rigid_body, light, optics, physics_material, physics_scene, rigid_body, rigid_constraint, Source (Core), motion, kinematics, kinematics_model
Child elements	See main entries for each parent element.
Other	technique

5.5.78.5 Remarks

For additional information about the common profile and customized profiles, see "The common profile" clause.

5.5.78.6 Example

See parent elements.

5.5.79 Token_array

Category: Data flow

5.5.79.1 General

Stores a homogenous array of xs:token string values

5.5.79.2 Concepts

The **<Token_array>** element stores xs:token string values as data for generic use within the COLLADA schema. The array itself is strongly typed but without semantics. It simply stores a sequence of XML xs:token values.

5.5.79.3 Attributes

The **<Token_array>** element has the following attributes:

count	uint_type	The number of values in the array. Required.
id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.79.4 Related elements

The **<Token_array>** element relates to the following elements:

Parent elements	Source (Core)
Child elements	None.
Other	accessor

5.5.79.5 Details

A **<Token_array>** element contains a list of XML name values (*xs:token*). These values are a repository of data to **<Source>** elements. An application can specify any application-defined name values.

5.5.80 translate

Category: Transform

5.5.80.1 General

Changes the position of an object in a local coordinate system.

5.5.80.2 Concepts

This element contains a mathematical vector that represents the distance along the x, y, and z axes.

Computer graphics techniques apply a translation transformation to position or move values with respect to a coordinate system.

5.5.80.3 Attributes

The **<translate>** element has the following attribute:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.

5.5.80.4 Related elements

The **<translate>** element relates to the following elements:

Parent elements	In core: node , In Physics: shape , technique_common / mass_frame in rigid_body and instance_rigid_body , ref_attachment , attachment In Kinematics: frame_object , frame_origin , frame_tcp , frame_tip , link
Child elements	None
Other	None

5.5.80.5 Details

The **<translate>** element contains a list of three floating-point values. These values are organized into a column vector suitable for a matrix composition.

For more information about how transformation elements are applied, see **<node>**.

5.5.80.6 Example

The following is an example of a **<translate>** element forming a displacement of 10 units along the x axis:

```
<translate>
  10.0 0.0 0.0
</translate>
```

5.5.81 triangles

Category: Geometry

5.5.81.1 General

Provides the information needed to for a mesh to bind vertex attributes together and then organize those vertices into individual triangles.

5.5.81.2 Concepts

The **<triangles>** element declares the binding of geometric primitives and vertex attributes for a **<mesh>** element.

The vertex array information is supplied in distinct attribute arrays that are then indexed by the **<triangles>** element.

Each triangle described by the mesh has three vertices. The first triangle is formed from the first, second, and third vertices. The second triangle is formed from the fourth, fifth, and sixth vertices, and so on.

5.5.81.3 Attributes

The **<triangles>** element has the following attributes:

name	xs:token	The text string name of this element. Optional.
count	uint_type	The number of triangle primitives. Required.
material	xs:NCName	Declares a symbol for a material. This symbol is bound to a material at the time of instantiation; see <instance_geometry> and <bind_material> . Optional. If not specified, then the lighting and shading results are application defined.

5.5.81.4 Related elements

The **<triangles>** element relates to the following elements:

Parent elements	mesh , convex_mesh
Child elements	See the following subclause.
Other	None

5.5.81.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (shared)	When at least one input is present, one input shall specify semantic="VERTEX" . See main entry.	None	0 or more
<p>	("p" stands for primitive.) Contains indices that describe the vertex attributes for a number of triangles. The indices reference into the parent's <Source> elements that are referenced by the <input> elements. This element has no attributes. See "Details."	None	0 or 1
<extra>	See main entry.	N/A	0 or more

5.5.81.6 Details

The indices in a **<p>** element refer to different inputs depending on their order. The first index in a **<p>** element refers to all inputs with an **offset** of 0. The second index refers to all inputs with an **offset** of 1. Each vertex of the triangle is made up of one index into each input. After each input is used, the next index again refers to the inputs with **offset** of 0 and begins a new vertex.

The winding order of vertices produced is counter clockwise and describes the front side of each triangle.

If the primitives are assembled without vertex normals then the application may generate per-primitive normals to enable lighting.

5.5.81.7 Example

The following is an example of a **<triangles>** element that describes two triangles. There are two **<Source>** elements that contain the position and normal data, according to the **<input>** element semantics. The **<p>** element index values indicate the order in which the input values are used:

```
<mesh>
  <source id="position"/>
  <source id="normal"/>
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <triangles count="2" material="Bricks">
    <input semantic="VERTEX" source="#verts" offset="0"/>
    <input semantic="NORMAL" source="#normal" offset="1"/>
    <p>
      0 0 1 3 2 1
      0 0 2 1 3 2
    </p>
  </triangles>
</mesh>
```

5.5.82 trifans

Category: Geometry

5.5.82.1 General

Provides the information needed for a mesh to bind vertex attributes together and then organize those vertices into connected triangles.

5.5.82.2 Concepts

The **<trifans>** element declares the binding of geometric primitives and vertex attributes for a **<mesh>** element.

The vertex array information is supplied in distinct attribute arrays of the **<mesh>** element that are then indexed by the **<trifans>** element.

Each triangle described by the mesh has three vertices. The first triangle is formed from the first, second, and third vertices. Each subsequent triangle is formed from the current vertex, reusing the first and the previous vertices.

5.5.82.3 Attributes

The **<trifans>** element has the following attributes:

name	xs:token	The text string name of this element. Optional.
count	uint_type	The number of triangle-fan primitives. Required.
material	xs:NCName	Declares a symbol for a material. This symbol is bound to a material at the time of instantiation; see <instance_geometry> and <bind_material> . Optional. If not specified, then the lighting and shading results are application defined.

5.5.82.4 Related elements

The **<trifans>** element relates to the following elements:

Occurrences	Number of elements defined in the schema
Parent elements	mesh, convex_mesh
Child elements	See the following subclause.
Other	None

5.5.82.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (shared)	When at least one input is present, one input shall specify semantic="VERTEX" . See main entry.	None	0 or more
<p>	("p" stands for primitive.) Contains indices that describe the vertex attributes for an arbitrary number of connected triangles. The indices reference into the parent's <Source> elements that are referenced by the <input> elements. This element has no attributes. See "Details."	None	0 or more
<extra>	See main entry.	N/A	0 or more

5.5.82.6 Details

A **<trifans>** element can contain a sequence of **<p>** elements.

The indices in a **<p>** element refer to different inputs depending on their order. The first index in a **<p>** element refers to all inputs with an **offset** of 0. The second index refers to all inputs with an **offset** of 1. Each vertex of the triangle is made up of one index into each input. After each input is used, the next index again refers to the inputs with **offset** of 0 and begins a new vertex.

The winding order of vertices produced is counter clockwise and describes the front side of each triangle.

If the primitives are assembled without vertex normals then the application may generate per-primitive normals to enable lighting.

5.5.82.7 Example

The following is an example of a **<trifans>** element that describes two triangles. There are two **<Source>** elements that contain the position and normal data, according to the **<input>** element semantics. The **<p>** element index values indicate the order in which the input values are used:

```
<mesh>
  <source id="position"/>
  <source id="normal"/>
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
```

```

<trifans count="1" material="Bricks">
  <input semantic="VERTEX" source="#verts" offset="0"/>
  <input semantic="NORMAL" source="#normal" offset="1"/>
  <p>0 0 1 3 2 1 3 2</p>
</trifans>
</mesh>

```

5.5.83 tristrips

Category: Geometry

5.5.83.1 General

Provides the information needed for a mesh to bind vertex attributes together and then organize those vertices into connected triangles.

5.5.83.2 Concepts

The **<tristrips>** element declares the binding of geometric primitives and vertex attributes for a **<mesh>** element.

The vertex array information is supplied in distinct attribute arrays of the **<mesh>** element that are then indexed by the **<tristrips>** element.

Each triangle described by the mesh has three vertices. The first triangle is formed from the first, second, and third vertices. Each subsequent triangle is formed from the current vertex, reusing the previous two vertices.

5.5.83.3 Attributes

The **<tristrips>** element has the following attributes:

name	xs:token	The text string name of this element. Optional.
count	uint_type	The number of triangle-strip primitives. Required.
material	xs:NCName	Declares a symbol for a material. This symbol is bound to a material at the time of instantiation; see <instance_geometry> and <bind_material> . Optional. If not specified, then the lighting and shading results are application defined.

5.5.83.4 Related elements

The **<tristrips>** element relates to the following elements:

Occurrences	Number of elements defined in the schema
Parent elements	mesh , convex_mesh
Child elements	See the following subclause.
Other	None

5.5.83.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (shared)	When at least one input is present, one input shall specify semantic="VERTEX". See main entry.	None	0 or more
<p>	("p" stands for primitive.) Contains indices that describe the vertex attributes for an arbitrary number of connected triangles. The indices reference into the parent's <Source> elements that are referenced by the <input> elements. This element has no attributes. See "Details."	None	0 or more
<extra>	See main entry.	N/A	0 or more

5.5.83.6 Details

A **<tristrips>** element can contain a sequence of **<p>** elements.

The indices in a **<p>** element refer to different inputs depending on their order. The first index in a **<p>** element refers to all inputs with an **offset** of 0. The second index refers to all inputs with an **offset** of 1. Each vertex of the triangle is made up of one index into each input. After each input is used, the next index again refers to the inputs with **offset** of 0 and begins a new vertex.

The winding order of vertices produced is counter clockwise for the first, (third, fifth, etc.) triangle and clockwise for the second (fourth, sixth, etc.) and describes the front side of each triangle.

If the primitives are assembled without vertex normals then the application may generate per-primitive normals to enable lighting.

5.5.83.7 Example

The following is an example of a **<tristrips>** element that describes two triangles. There are two **<Source>** elements that contain the position and normal data, according to the **<input>** element semantics. The **<p>** element index values indicate the order in which the input values are used:

```
<mesh>
  <source id="position"/>
  <source id="normals"/>
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <tristrips count="1" material="Bricks">
    <input semantic="VERTEX" source="#verts" offset="0"/>
    <input semantic="NORMAL" source="#normals" offset="1"/>
    <p>0 0 1 3 2 1 3 2</p>
  </tristrips>
</mesh>
```

5.5.84 vertex_weights

Category: Controller

5.5.84.1 General

Describes the combination of joints and weights used by a skin.

5.5.84.2 Concepts

The **<vertex_weights>** element associates a set of joint-weight pairs with each vertex in the base mesh.

5.5.84.3 Attributes

The **<vertex_weights>** element has the following attribute:

count	uint_type	The number of vertices in the base mesh. Required.

5.5.84.4 Related elements

The **<vertex_weights>** element relates to the following elements:

Parent elements	skin
Child elements	See the following subclause.
Other	None

5.5.84.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (shared)	One of the <input> elements, as a child of <vertex_weights> , shall specify semantic="JOINT". The <input> elements describe the joints and the attributes to be associated with them. See main entry.	None	2 or more
<vcount>	Contains a list of integers, each specifying the number of bones associated with one of the influences defined by <vertex_weights> . This element has no attributes.	None	0 or 1
<v>	Contains a list of indices that describe which bones and attributes are associated with each vertex. An index of -1 into the array of joints refers to the bind shape. Weights shall be normalized before use. This element has no attributes.	None	0 or 1
<extra>	See main entry.	N/A	0 or more

5.5.84.6 Example

The following is an example of an empty **<vertex_weights>** element:

```
<skin>
  <vertex_weights count="">
    <input semantic="JOINT"/>
    <input/>
    <vcount/>
    <v/>
    <extra/>
  </vertex_weights>
</skin>
```

The following is an example of a more complete **<vertex_weights>** element. Note that the **<vcount>** element says that the first vertex has 3 bones, the second has 2, etc. Also, the **<v>** element says that the first vertex is weighted with weights[0] towards the bind shape, weights[1] towards bone 0, and weights[2] towards bone 1:

```
<skin>
  <source id="joints"/>
  <source id="weights"/>
  <vertex_weights count="4">
    <input semantic="JOINT" source="#joints"/>
    <input semantic="WEIGHT" source="#weights"/>
    <vcount>3 2 2 3</vcount>
    <v>
      -1 0 0 1 1 2
      -1 3 1 4
      -1 3 2 4
      -1 0 3 1 2 2
    </v>
  </vertex_weights>
</skin>
```

5.5.85 vertices

Category: Geometry

5.5.85.1 General

Declares the attributes and identity of mesh-vertices.

5.5.85.2 Concepts

The **<vertices>** element describes mesh-vertices in a mesh. The mesh-vertices represent the position (identity) of the vertices comprising the mesh and other vertex attributes that are invariant to tessellation.

5.5.85.3 Attributes

The **<vertices>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Required.
name	xs:token	The text string name of this element. Optional.

5.5.85.4 Related elements

The **<vertices>** element relates to the following elements:

Parent elements	mesh , convex_mesh , brep
Child elements	See the following subclause.
Other	None

5.5.85.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input> (unshared)	One input shall specify semantic="POSITION" to establish the topological identity of each vertex in the mesh. See main entry.	None	1 or more
<extra>	See main entry.	N/A	0 or more

5.5.85.6 Details

In the **<brep>** element, the **<vertices>** element specifies only the boundaries of edges or of single points without an edge. Any additional data, such as color, texture, and so on, is not evaluated for B-rep.

5.5.85.7 Example

The following is an example of a **<vertices>** element that describes the vertices of a mesh:

```
<mesh>
  <source id="position"/>
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
</mesh>
```

5.5.86 visual_scene

Category: Scene

5.5.86.1 General

Embodies the entire set of information that can be visualized from the contents of a COLLADA resource.

5.5.86.2 Concepts

The hierarchical structure of the **visual_scene** is organized into a scene graph. A scene graph is a directed acyclic graph (DAG) or tree data structure that contains nodes of visual information and related data. The structure of the scene graph contributes to optimal processing and rendering of the data and is therefore widely used in the computer graphics domain.

5.5.86.3 Attributes

The **<visual_scene>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.5.86.4 Related elements

The **<visual_scene>** element relates to the following elements:

Parent elements	library_visual_scenes
Child elements	See the following subclause.
Other	instance_visual_scene

5.5.86.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry.	N/A	0 or 1
<node>	See main entry.	N/A	1 or more
<evaluate_scene>	The <evaluate_scene> element declares information specifying how to evaluate this visual_scene . See main entry.	N/A	0 or more
<extra>	See main entry.	N/A	0 or more

5.5.86.6 Details

The **<visual_scene>** element forms the root of the scene graph topology.

There might be multiple **<visual_scene>** elements declared within a **<library_visual_scenes>** element. The **<instance_visual_scene>** element in the **<Scene>** element, which is declared under the **<COLLADA>** document (root) element, declares which **<visual_scene>** element is to be used for the document.

5.5.86.7 Example

The following example shows a simple outline of a COLLADA resource containing a **<visual_scene>** element with no child elements. The name of the scene is "world":

```
<?xml version="1.0" encoding="utf-8"?>
<COLLADA xmlns="http://www.collada.org/2008/03/COLLADASchema" version="1.5.0">
  <library_visual_scenes>
    <visual_scene id="world">
      <node id="root"/>
    </visual_scene>
  </library_visual_scenes>
</COLLADA>
```

```

</library_visual_scenes>
<scene>
  <instance_visual_scene url="#world"/>
</scene>
</COLLADA>

```

COLLADA supports layering and visibility. Each node has a layer attribute that is a list of `xs:NCName`. The node belongs to each layer that it lists there. Then, in the visual scene, there is an **<evaluate_scene>** that describes how a scene is to be rendered. This is also where one would use full-screen effects.

The following document fragment shows how this works. This solution works for layers. It might not be ideal for “visibility,” but you can achieve the same results with it:

```

<visual_scene>
  <node id="Node1" layer="visible"/>
  <node id="Node2" layer="visible"/>
  <node id="Node3" layer="notvisible"/>
  <node id="camera"><instance_camera url="#cam01"/></node>
  <evaluate_scene>
    <render camera_node="#camera">
      <layer>visible</layer>
    </render>
  </evaluate_scene>
</visual_scene>

```

5.6 Physics Reference

5.6.1 General

This clause covers the elements that compose COLLADA Physics.

5.6.2 Elements by category

5.6.2.1 General

This subclause lists elements in alphabetical order. The following tables list elements by category, for ease in finding related elements.

5.6.2.2 Analytical shape

Elements	Description
box	Declares an axis-aligned box that is centred at its local origin.
capsule	Declares a capsule that is centred on its local origin and aligned with, the local y axis.
convex_mesh	Contains or refers to information that describes basic geometric meshes.
cylinder	Declares a cylinder capsule that is centred on its local origin and aligned with, the local y axis.
plane	Declares an infinite plane primitive.
shape	Describes components of a <rigid_body> .
sphere	Declares a sphere primitive that is centred on its local origin.

5.6.2.3 Physics material

Elements	Description
instance_physics_material	Let's a shape specify its surface properties using a previously defined <physics_material> element.
library_physics_materials	Provides a library in which to place <physics_material> elements.
physics_material	Defines the physical properties of an object using a technique/profile with parameters.

5.6.2.4 Physics model

Elements	Description
attachment	Defines an attachment to a rigid body or a node.
ref_attachment	Defines an attachment to a rigid body or a node to be used as a reference frame.
instance_physics_model	Embeds a physics model inside another physics model or instantiates a physics model within a physics scene.
instance_rigid_body	Provides a means to interface with a particular <rigid_body> of a <physics_model> that has been instantiated with <instance_physics_model> .
instance_rigid_constraint	Provides the interface to a constraint that gets created by instantiating a physics model that has a constraint.
library_physics_models	Provides a library in which to place <physics_model> elements.
physics_model	Allows for building complex combinations of rigid bodies and constraints that may be instantiated multiple times.
rigid_body	Describes simulated bodies that do not deform.
rigid_constraint	Connects components, such as <rigid_body> , into complex physics models with moveable parts.

5.6.2.5 Physics scene

Elements	Description
force_field	Provides a general container for force fields.
instance_force_field	Declares the instantiation of a <force_field> .
instance_physics_scene	Declares the instantiation of a <physics_scene> .
library_force_fields	Provides a library in which to place <force_field> elements.
library_physics_scenes	Provides a library in which to place <physics_scene> elements.
physics_scene	Specifies an environment in which physical objects are instantiated and simulated.

5.6.2.6 General

In a 3D application, the data and the processes used for physical simulation of a virtual world are often different than what is used for rendering. COLLADA Physics enables content creators to describe these physics scenes.

COLLADA Physics currently supports basic rigid body dynamics. A rigid body is a nondeformable object with shape (geometry) and mass properties that interacts with other rigid bodies according to Newton's basic laws of physics. Physically based systems usually do not have the same notion of hierarchy or parent-child relationships used by animated articulated models. Instead, rigid bodies can be connected to each other or to the world by constraints.

A constraint can specify how one body can move in relation to the other. For example, a car wheel can be constrained to a chassis, so that it rotates only along the x axis, and does not translate or rotate along other axes. In general, a rigid constraint has many parameters that can limit the various angular and linear degrees of freedom. Related rigid bodies and constraints are then logically grouped into a physics model to define complex physical systems, such as a car or a character's "rag doll." Finally, physics models, which are defined in a library of physics models, are instantiated in a physics scene similarly to how visual geometry is instantiated in a visual scene.

The simulated models in a physics scene are visualized by having their instantiated rigid bodies directly control the placement of nodes in the visual scene. The node could display different geometry than what is used by the physics scene, or even be a bone used for rendering a skinned mesh. A rigid body can be dynamic, which means that its behaviour is completely determined by the physics simulation. Alternatively, it can be kinematic, meaning that its position and orientation are controlled by an animation, but it still influences other dynamic bodies in the simulation. Animation can also indirectly influence a physical simulation by targeting the animation data to other physics parameters, such as the preferred position and orientation between two constrained bodies.

5.6.2.7 Physical units

COLLADA does not impose a particular linear scale. Data can be stored in inches, feet, meters, or miles. This applies to physics as well. Consequently, the scales for velocities, forces, and mass properties depend on the specified units for the file. For example, if distances and lengths are specified in meters, mass in kilograms, and time in seconds, then forces are in Newtons. If distances are in inches, then velocity is in inches per second. Density is specified as units of mass per one cubic unit. For example, using pounds and feet implies that density is the number of pounds per cubic foot of matter. One pitfall of allowing different scales is that, when using metric standards with meter as the unit of distance and kg as the unit of mass, density is now per cubic meter, which is different than the standard metric density definition of kilograms per cubic decimetre (litre).

If needed, units shall be taken from the "base" of the COLLADA document. COLLADA uses the following units of measurement:

Measurement	Unit
Time	seconds (standard units)
Angle	degrees (standard units) used for constraint/joint limits
Mass	kilograms (standard units)
Distance	meters (default units). The <asset> element includes the <unit> element, by which the measure of distance can be redefined for the corresponding asset.

5.6.2.8 Geometry types

Physics simulation typically uses some sort of mesh representation to describe the geometry of a rigid body. Because COLLADA already has a way of describing geometry for visual data, the physics representation uses the same system. In fact, meshes used by rigid bodies can also be referred to by visual nodes. The COLLADA schema for meshes may seem complex for physics purposes, which require only basic vertex position and triangle information.

Physics engines can depend on meshes being convex for proper collision. COLLADA provides an explicit element, **<convex_mesh>**, to indicate that a mesh is convex or that the convex hull shall be generated for the given mesh.

In addition to general meshes and convex hulls, physics engines often also use analytical shapes (boxes, spheres, capsules) for collision volumes. This helps the physics simulation to better represent certain round surfaces, improve performance, and reduce memory usage. Therefore, COLLADA adds a handful of primitive geometry types intended for use in physics, in particular, **<box>**, **<sphere>**, **<capsule>**, and **<cylinder>**.

These primitives have child elements, such as radius, height, or extents, to specify the size of the geometry. Each of these is axis-aligned and centred at the origin. Geometry elements are children of shape elements that specify their position and orientation within the rigid body. The shape also has child elements to specify surface properties. A rigid body includes one or more shapes listed.

- **<box>**
- **<capsule>**
- **<convex_mesh>**
- **<cylinder>**
- **<plane>**
- **<sphere>**

5.6.3 attachment

Category: Physics model

5.6.3.1 General

Defines an attachment frame, to a rigid body or a node, within a rigid constraint.

5.6.3.2 Concepts

A **<rigid_constraint>** attaches (and limits the motion between) two rigid bodies together. **<attachment>** refers to the second rigid body, and **<ref_attachment>** to the first. For example, in the case of a hinge constraint between a door and a wall, one of them is the reference attachment (in this case, the wall), and the other is the attachment (the door).

The **<attachment>** also defines the local coordinate frame for that end of the connection, relative to the rigid body (or node), using **<translate>** and **<rotate>** elements. For example, you attach the hinge (rigid constraint) to the middle of the edge of the door (rigid body), relative to the door's local origin.

5.6.3.3 Attributes

The **<attachment>** element has the following attribute:

rigid_body	xs:anyURI	A scoped-identifier reference to a <rigid_body> or <node> . This shall refer to the SID of a <rigid_body> either in <attachment> or in <ref_attachment> ; they cannot both be <node> s. Required.

5.6.3.4 Related elements

The **<attachment>** element relates to the following elements:

Parent elements	rigid_constraint
Child elements	See the following subclause.
Other	ref_attachment

5.6.3.5 Child elements

Child elements **<translate>** and **<rotate>** can appear in any order, and **<extra>** shall be last if present:

Name/example	Description	Default	Occurrences
<translate>	Changes the position of the attachment point. See main entry in Core.	N/A	0 or more
<rotate>	Changes the position of the attachment point. See main entry in Core.	N/A	0 or more
<extra>	See main entry in Core.	N/A	0 or more

5.6.3.6 Example

```
<attachment rigid_body="../../SomeRigidBody">
  <translate/>
  <rotate/>
  <extra/>
</attachment>
```

For a more complete example, see **<rigid_constraint>**.

5.6.4 box

Category: Analytical shape

5.6.4.1 General

Declares an axis-aligned box that is centred around its local origin.

5.6.4.2 Concepts

Box is one of the geometric primitives in COLLADA physics that enables more efficient collision detection than using the equivalent mesh with eight vertices. See the "Geometry types" in 5.6.2.8.

5.6.4.3 Attributes

The **<box>** element has no attributes.

5.6.4.4 Related elements

The **<box>** element relates to the following elements:

Parent elements	shape
Child elements	See the following subclause.
Other	None

5.6.4.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<half_extents>	Contains 3 floating-point values that represent the extents of the box. The dimensions of the box are double the half extents. This element has no attributes.	None	1
<extra>	See main entry in Core.	N/A	0 or more

5.6.4.6 Example

A 5x2x2 box is represented as:

```
<shape>
  <box>
    <half_extents> 2.5 1.0 1.0 </half_extents>
  </box>
</shape>
```

5.6.5 capsule

Category: Analytical shape

5.6.5.1 General

Declares a capsule primitive that is centred on the local origin and aligned along the *y* axis.

5.6.5.2 Concepts

The capsule is a geometric primitive added specifically for physics, where it is commonly used for collision detection. See the “Geometry types” in 5.6.2.8.

A capsule is a cylinder with rounded caps. More formally, it can be described as the convex hull generated by two spheres, or the Makowski summation of a sphere and a line segment (line swept sphere). While spherical capsules are the most common, COLLADA generalizes to allow ellipsoid endpoints.

5.6.5.3 Attributes

The **<capsule>** element has no attributes.

5.6.5.4 Related elements

The **<capsule>** element relates to the following elements:

Parent elements	shape
Child elements	See the following subclause.
Other	None

5.6.5.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<height>	Contains a floating-point value that represents the length of the line segment connecting the centres of the capping hemispheres (ellipsoids). This element has no attributes.	None	1
<radius>	Contains three floating-point values that represent the x, y, and z radii of the capsule (it may be elliptical). This element has no attributes.	None	1
<extra>	See main entry in Core.	N/A	0 or more

5.6.5.6 Example

```
<capsule>
  <height> 4.0 </height>
  <radius> 1.0 2.0 3.0 </radius>
</capsule>
```

A spherical capsule can be constructed by setting all three radii to be equal:

```
<capsule>
  <height> 2.0 </height>
  <radius> 1.0 1.0 1.0 </radius>
</capsule>
```

5.6.6 convex_mesh

Category: Analytical shape

5.6.6.1 General

Contains or refers to information sufficient to describe basic geometric meshes.

5.6.6.2 Concepts

The definition of **<convex_mesh>** is identical to **<mesh>** except that, instead of a complete description (**<Source>**, **<vertices>**, **<polygons>**, and so on), it may simply point to another **<geometry>** to derive its shape. The latter case means that the convex hull of that **<geometry>** shall be computed and is indicated by the optional **convex_hull_of** attribute.

This is very useful because it allows for reusing a **<mesh>** (that is used for rendering) for physics to minimize the document size and to maintain a link to the original **<mesh>**.

The minimal way to describe a **<convex_mesh>** is to specify its vertices (via a **<vertices>** element and its corresponding source) and let the importer compute the convex hull of that point cloud.

5.6.6.3 Attributes

The **<convex_mesh>** element has the following attribute:

convex_hull_of	xs:anyURI	A URI string of a <geometry> . If specified, compute the convex hull of the specified mesh; in this case, your application shall ignore <Source> and <vertices> , if specified. Optional.

5.6.6.4 Related elements

The **<convex_mesh>** element relates to the following elements:

Parent elements	geometry
Child elements	See the following subclause.
Other	None

5.6.6.5 Child elements

No child elements are required. However, if any child elements appear, they shall appear in the following order and with the specified number of occurrences:

Name/example	Description	Default	Occurrences
<Source>	Provides the bulk of the mesh's vertex data. Required if convex_hull_of is not specified. See main entry in Core.	N/A	1 or more
<vertices>	Describes the mesh-vertex attributes and establishes their topological identity. Required if convex_hull_of is not specified. See main entry in Core.	N/A	1

Name/example	Description		Default	Occurrences
primitive_elements	Primitive elements can be any combination of the following:			
	<lines>	Contains line primitives. See main entry in Core.	N/A	0 or more
	<linestrips>	Contains line-strip primitives. See main entry in Core.	N/A	0 or more
	<polygons>	Contains polygon primitives which may contain holes. See main entry in Core.	N/A	0 or more
	<polylist>	Contains polygon primitives that cannot contain holes. See main entry in Core.	N/A	0 or more
	<triangles>	Contains triangle primitives. See main entry in Core.	N/A	0 or more
	<trifans>	Contains triangle-fan primitives. See main entry in Core.	N/A	0 or more
	<tristrips>	Contains triangle-strip primitives. See main entry in Core.	N/A	0 or more
<extra>	See main entry in Core.		N/A	0 or more

5.6.6.6 Details

If the attribute `convex_hull_of` is not used, specify child elements **<Source>** and **<vertices>** to define a valid **<convex_mesh>**.

5.6.6.7 Example

```

<geometry id="myConvexMesh">
  <convex_mesh>
    <source>...</source>
    <vertices>...</vertices>
    <polygons>...</polygons>
  </convex_mesh>
</geometry>
or:
<geometry id="myArbitraryMesh">
  <mesh>
    ...
  </mesh>
</geometry>
<geometry id="myConvexMesh">
  <convex_mesh convex_hull_of="#myArbitraryMesh"/>
</geometry>

```

5.6.7 cylinder

Category: Analytical shape

5.6.7.1 General

Declares a cylinder primitive that is centred around its local origin and aligned along its local *y* axis.

NOTE For this element in **<surface>**, see “**<cylinder>** (B-rep)” in 5.9 B-Rep Reference.

5.6.7.2 Concepts

Geometric primitives, also called analytical shapes, are mostly useful for collision shapes for physics. See the “Geometry types” in 5.6.2.8.

5.6.7.3 Attribute

The **<cylinder>** element has no attributes.

5.6.7.4 Related elements

The **<cylinder>** element relates to the following elements:

Parent elements	shape
Child elements	See the following subclause.
Other	None

5.6.7.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<height>	Contains a floating-point value that represents the length of the cylinder along the y axis. This element has no attributes.	None	1
<radius>	Contains two floating-point values that represent the radii of the cylinder (it may be elliptical). This element has no attributes.	None	1
<extra>	See main entry in Core.	N/A	0 or more

5.6.7.6 Example

```
<cylinder>
  <height> 2.0 </height>
  <radius> 1.0 1.0 </radius>
</cylinder>
```

5.6.8 force_field

Category: Physics scene

5.6.8.1 General

Provides a general container for force fields.

5.6.8.2 Concepts

Force fields affect physical objects, such as rigid bodies, and may be instantiated under a **physics_scene** or an instance of **physics_model**.

5.6.8.3 Attributes

The **<force_field>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	Optional.

5.6.8.4 Related elements

The **<force_field>** element relates to the following elements:

Parent elements	library_force_fields
Child elements	See the following subclause.
Other	instance_force_field

5.6.8.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<technique> (Core)	See main entry in Core.	N/A	1 or more
<extra>	See main entry in Core.	N/A	0 or more

5.6.8.6 Details

Currently there is no COMMON technique/profile for **<force_field>**. The **<technique>** element can contain any well-formed XML data.

5.6.8.7 Example

```
<library_force_fields>
  <force_field>
    <technique profile="SomePhysicsProfile">
      <program url="#SomeWayToDescribeAForceField">
        <param> ... </param>
        <param> ... </param>
      </program>
    </technique>
  </force_field>
</library_force_fields>
```

5.6.9 instance_force_field

Category: Physics scene

5.6.9.1 General

Instantiates an object described by a **<force_field>** element.

5.6.9.2 Concepts

For details about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

5.6.9.3 Attributes

The **<instance_force_field>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URL of the location of the <force_field> element to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the “#” character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.6.9.4 Related elements

The **<instance_force_field>** element relates to the following elements:

Parent elements	instance_physics_model , physics_scene
Child elements	See the following subclause.
Other	force_field

5.6.9.5 Child elements

Name/example	Description	Default	Occurrences
<extra>	See main entry in Core.	N/A	0 or more

5.6.9.6 Details

This element has no standard specification. Usage is application dependent.

5.6.9.7 Example

```
<instance_force_field url="#my_force_field">
</instance_force_field>
```

5.6.10 instance_physics_material

Category: Physics material

5.6.10.1 General

Let's a shape specify its surface properties using a previously defined **<physics_material>** element.

5.6.10.2 Concepts

For efficiency, some physics engines reference a palette of physics materials instead of storing these properties with each shape.

For general information about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

5.6.10.3 Attributes

The `<instance_physics_material>` element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URL of the location of the <code><physics_material></code> element to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the “#” character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.6.10.4 Related elements

The `<instance_physics_material>` element relates to the following elements:

Parent elements	<code>rigid_body / technique_common</code> , <code>instance_rigid_body / technique_common</code> , <code>shape</code>
Child elements	See the following subclause.
Other	<code>physics_material</code>

5.6.10.5 Child elements

Name/example	Description	Default	Occurrences
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.6.10.6 Example

```
<shape>
  <sphere> <radius> 1 </radius> </sphere>
  <instance_physics_material url="#my_force_field" />
</shape>
```

5.6.11 instance_physics_model

Category: Physics model

5.6.11.1 General

Embeds a physics model inside another physics model or instantiates a physics model within a physics scene.

5.6.11.2 Concepts

This element is used for two purposes: to hierarchically embed a physics model inside another physics model during its definition, and to instantiate a physics model within a physics scene. It is possible to override parameters of the contained rigid bodies and constraints in both usages.

When instantiating a physics model inside a physics scene, at a minimum, the rigid bodies that are included in the physics model can be linked with transform nodes in the visual scene to let the physics animate meshes that are being displayed. Similarly, if a rigid_body is kinematic instead of dynamic, an application could take transform information from a targeted node that is being influenced by animation so that the rigid_body is moved within its physics environment.

Additionally, it is possible to specify a parent attribute for the instantiated physics model. This parent will dictate the initial position and orientation of the physics models (and correspondingly, of its rigid bodies). The parent (or grandparent, etc.) can also be targeted by some animation controller, to combine key-frame kinematics of nondynamic rigid bodies with physical simulation.

For details about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

5.6.11.3 Attributes

The **<instance_physics_model>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. This allows for targeting elements of the <instance_physics_model> instance for animation . Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	Which <physics_model> to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the “#” character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.
parent	xs:anyURI	Points to the id of a node in the visual scene. This allows a physics model to be instantiated under a specific transform node, which will dictate the initial position and orientation, and could be animated to influence kinematic rigid bodies. Optional. By default, the physics model is instantiated under the world, rather than a specific transform node. This parameter is only meaningful when the parent element of the current <physics_model> is a <physics_scene> .

5.6.11.4 Related elements

The `<instance_physics_model>` element relates to the following elements:

Parent elements	<code>physics_scene</code> , <code>physics_model</code>
Child elements	See the following subclause.
Other	<code>physics_model</code>

5.6.11.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><instance_force_field></code>	Instantiates a <code><force_field></code> element to influence this physics model. See main entry.	N/A	0 or more
<code><instance_rigid_body target="#SomeNode"></code>	Instantiates a <code><rigid_body></code> element and allows for overriding some or all its properties. The target attribute defines the <code><node></code> element that has its transforms overwritten by this rigid-body instance. See main entry.	N/A	0 or more
<code><instance_rigid_constraint></code>	Instantiates a <code><rigid_constraint></code> element to override some of its properties. This element does not have a target attribute because its <code><rigid_constraint></code> children define which <code><node></code> elements are targeted. See main entry.	N/A	0 or more
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.6.12 instance_physics_scene

Category: Physics scene

5.6.12.1 General

Instantiates an object described by a `<physics_scene>` element.

5.6.12.2 Concepts

For details about instance elements in COLLADA, see "Instantiation and external referencing" in 5.3 Schema concepts.

5.6.12.3 Attributes

The `<instance_physics_scene>` element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URL of the location of the <code><physics_scene></code> element to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the “#” character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.6.12.4 Related elements

The `<instance_physics_scene>` element relates to the following elements:

Parent elements	Scene
Child elements	See the following subclause.
Other	physics_scene

5.6.12.5 Child elements

Name/example	Description	Default	Occurrences
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.6.12.6 Details

In terms of instantiation, the `<Scene>` is the starting point of a COLLADA file. A COLLADA physics scene is instantiated in the `<Scene>` element using the `<instance_physics_scene>` element.

5.6.12.7 Example

```
<scene>
  <instance_physics_scene url="my_physics_scene" />
</scene>
```

5.6.13 instance_rigid_body

Category: Physics model

5.6.13.1 General

Provides a means to interface with a particular `<rigid_body>` of a `<physics_model>` that has been instantiated with `<instance_physics_model>`.

5.6.13.2 Concepts

In an application that uses both physics and rich graphics, rigid bodies ultimately set the transforms of a **<node>** in the **<Scene>**. If there isn't already a skeleton for an instantiated physics model, the **<instance_rigid_body>** element can be useful to connect a specific rigid body instance with a node in the visual scene.

The **<instance_rigid_body>** element is used for three purposes:

- To specify the linkage to a **<node>** element;
- To optionally override parameters of a **<rigid_body>** in a specific instance;
- To specify the initial state (linear and angular velocity) of a **<rigid_body>** instance.

For details about instance elements in COLLADA, see "Instantiation and external referencing" in 5.3 Schema concepts.

5.6.13.3 Attributes

The **<instance_rigid_body>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. This allows for targeting elements of the <rigid_body> instance for animation . Optional. For details, see "Address syntax" in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
body	sidref_type	A reference to the SID of the <rigid_body> to instantiate. Required.
target	xs:anyURI	Which <node> is influenced by this <rigid_body> instance. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the "#" character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.6.13.4 Related elements

The **<instance_rigid_body>** element relates to the following elements:

Parent elements	instance_physics_model
Child elements	See the following subclause.
Other	rigid_body

5.6.13.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><technique_common></code>	Specifies the rigid-body information for the common profile that all COLLADA implementations shall support. See “The common profile” clause for usage information and the following subclause for child element details.	N/A	1
<code><technique></code> (Core)	Each <code><technique></code> specifies rigid-body information for a specific profile as designated by the <code><technique></code> ’s profile attribute. See main entry in Core.	N/A	0 or more
<code><extra></code>	User-defined, multirepresentable data that adds information to the <code><instance_rigid_body></code> (as opposed to switching base data, like the <code><technique></code> element does). See main entry in Core.	N/A	0 or more

5.6.13.6 Child elements for `<instance_rigid_body>` / `<technique_common>`

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><angular_velocity></code>	Contains a 3D vector (three floating-point values) that specifies the initial spin or angular velocity of the rigid_body instance. This vector is also known as the axis of rotation, with a magnitude equal to the rate of rotation in radians per second. The direction of spin follows the handedness of the coordinate system. For example, in a right-handed system, a spin vector pointed toward the viewer would correspond to an object that is spinning counter-clockwise from the viewer’s perspective. This element has no attributes.	0 0 0	0 or 1
<code><velocity></code>	Contains a 3D vector (three floating-point values) that specifies the initial linear velocity of the rigid_body instance. This element has no attributes.	0 0 0	0 or 1
<code><dynamic sid="..." false</dynamic></code>	Contains a Boolean that specifies whether the rigid_body is movable. The <code>sid</code> attribute is optional.	true	0 or 1
<code><mass sid="..." 0.5</mass></code>	Contains a floating-point value that specifies the total mass of the rigid_body. The <code>sid</code> attribute is optional.	Derived from density x total shape volume	0 or 1

Name/example	Description	Default	Occurrences
<code><mass_frame></code> <code> <translate> ...</code> <code> </translate></code> <code> <rotate> ...</code> <code> </rotate></code> <code></mass_frame></code>	Defines the centre and orientation of mass of the rigid-body relative to the local origin of the “root” shape. This makes the off-diagonal elements of the inertia tensor (products of inertia) all 0 and allows us to just store the diagonal elements (moments of inertia). The <code><translate></code> and <code><rotate></code> child elements can each appear 0 or more times, although at least one of the two shall be present. See main entries in Core.	“identity” (centre of mass is at the local origin and the principal axes are the local axes).	0 or 1
<code><inertia</code> <code> sid="..."> 1 1 1</code> <code></inertia></code>	Contains three floating-point numbers, which are the diagonal elements of the inertia tensor (moments of inertia), represented in the local frame of the centre of mass. See preceding. The <code>sid</code> attribute is optional.	Derived from mass, shape volume and centre of mass.	0 or 1
<code><physics_material></code> <code> or</code> <code><instance_physics_material></code>	Defines or references a <code>physics_material</code> for the <code>rigid_body</code> . See main entries.	N/A	0 or 1
<code><shape></code>	See main entry.	N/A	0 or more

5.6.13.7 Example

```

<physics_scene id="ColladaPhysicsScene">
  <instance_physics_model sid="firstCatapultAndRockInstance"
    url="#catapultAndRockModel" parent="#catapult1">
    <!--Override attributes of a rigid_body within this physics_model -->
    <!--and specify the initial velocity of the rigid_body -->
    <instance_rigid_body body="/rock/rock" target="#rockNode">
      <technique_common>
        <velocity>0 -1 0</velocity> <!--optional overrides -->
        <mass>10</mass> <!--heavier -->
      </technique_common>
    </instance_rigid_body>
    <!--This instance only assigns the rigid_body to its node. -->
    <!--It does no overriding-->
    <instance_rigid_body body="./catapult/base" target="#baseNode">
      <technique_common/>
    </instance_rigid_body>
  </instance_physics_model>
</physics_scene>

```

5.6.14 instance_rigid_constraint

Category: Physics model

5.6.14.1 General

Provides the interface to a constraint that is created by instantiating a physics model that has a constraint.

5.6.14.2 Concepts

A rigid constraint between two rigid bodies has a few properties than can be adjusted at runtime to add additional variety to the simulation. For example, animation data from an articulated model can be used to update the attachment frame alignments on the corresponding joints. If these joints have a maximum torque, this will provide an elementary physically based character motion system.

For general information about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

5.6.14.3 Attributes

The **<instance_rigid_constraint>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. This allows for targeting elements of the <rigid_constraint> instance for animation. Optional. For details, see “Address syntax” in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
constraint	sidref_type	A reference to the SID of the <rigid_constraint> to instantiate. Required.

5.6.14.4 Related elements

The **<instance_rigid_constraint>** element relates to the following elements:

Parent elements	instance_physics_model
Child elements	See the following subclause.
Other	rigid_constraint

5.6.14.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<extra>	User-defined, multirepresentable data that adds information to the <instance_rigid_constraint> . See main entry in Core.	N/A	0 or more

5.6.14.6 Details

When elements are explicitly included, the **<instance_physics_model>** mirrors the corresponding **<physics_model>**. The two rigid-body instances that an **<instance_rigid_constraint>** connects are the ones that correspond to the rigid bodies referenced by the corresponding **<rigid_constraint>**.

5.6.14.7 Example

```
<instance_physics_model>
  <instance_rigid_constraint sid="my_joint">
    <extra> <maximum_torque> 100 </maximum_torque> </extra>
  </instance_rigid_constraint>
```

5.6.15 library_force_fields

Category: Physics scene

5.6.15.1 General

Provides a library in which to place **<force_field>** elements.

5.6.15.2 Concepts

Some applications use force fields to influence the motion of rigid bodies.

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.6.15.3 Attributes

The **<library_force_fields>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <library_force_fields> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.6.15.4 Related elements

The **<library_force_fields>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.6.15.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<force_field>	See main entry.	N/A	1 or more
<extra>	See main entry in Core.	N/A	0 or more

5.6.15.6 Example

The following is an example of a **<library_force_fields>** element:

```
<library_force_fields>
  <force_field>
    <technique profile="AGEIA"/>
  </force_field>
</library_force_fields>
```

5.6.16 library_physics_materials

Category: Physics material

5.6.16.1 General

Provides a library in which to place **<physics_material>** elements.

5.6.16.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.6.16.3 Attributes

The **<library_physics_materials>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <library_physics_materials> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.6.16.4 Related elements

The **<library_physics_materials>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.6.16.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<physics_material>	See main entry.	N/A	1 or more
<extra>	See main entry in Core.	N/A	0 or more

5.6.16.6 Example

The following is an example of a **<library_physics_materials>** element:

```
<library_physics_materials>
  <physics_material id="phymat1">
    ...
  </physics_material>
  <physics_material id="phymat2">
    ...
  </physics_material>
</library_physics_materials>
```

5.6.17 library_physics_models

Category: Physics model

5.6.17.1 General

Provides a library in which to place **<physics_model>** elements.

5.6.17.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.6.17.3 Attributes

The **<library_physics_models>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.6.17.4 Related elements

The **<library_physics_models>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.6.17.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<physics_model>	See main entry.	N/A	1 or more
<extra>	See main entry in Core.	N/A	0 or more

5.6.17.6 Example

The following is an example of a **<library_physics_models>** element:

```
<library_physics_models>
  <physics_model id="phymod1">
    ...
  </physics_model>
  <physics_model id="phymod2">
    ...
  </physics_model>
</library_physics_models>
```

5.6.18 library_physics_scenes

Category: Physics scene

5.6.18.1 General

Provides a library in which to place **<physics_scene>** elements.

5.6.18.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.6.18.3 Attributes

The **<library_physics_scenes>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.6.18.4 Related elements

The `<library_physics_scenes>` element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.6.18.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><asset></code>	See main entry in Core.	N/A	0 or 1
<code><physics_scene></code>	See main entry.	N/A	1 or more
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.6.18.6 Example

The following is an example of a `<library_physics_scenes>` element:

```
<library_physics_scenes>
  <physics_scene id="physcel1">
    ...
  </physics_scene>
  <physics_scene id="physce2">
    ...
  </physics_scene>
</library_physics_scenes>
```

5.6.19 physics_material

Category: Physics material

5.6.19.1 General

Defines the physical properties of an object.

5.6.19.2 Concepts

This element uses a technique/profile with parameters to define an object's physical properties. The COMMON profile defines the built-in names, such as `static_friction`.

Physics materials can be used inline within a shape or can be stored under a `<library_physics_materials>` element and instantiated by a shape using `<instance_physics_material>`.

5.6.19.3 Attributes

The **<physics_material>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.6.19.4 Related elements

The **<physics_material>** element relates to the following elements:

Parent elements	library_physics_materials , shape , instance_rigid_body / technique_common , rigid_body / technique_common
Child elements	See the following subclause.
Other	instance_physics_material

5.6.19.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<technique_common>	Specifies physics-material information for the common profile that all COLLADA implementations shall support. See the following subclause.	N/A	1
<technique> (Core)	Each <technique> specifies physics-material information for a specific profile as designated by the <technique> 's profile attribute. See main entry in Core.	N/A	0 or more
<extra>	See main entry in Core.	N/A	0 or more

5.6.19.6 Child elements for **<physics_material>** / **<technique_common>**

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<dynamic_friction sid="..."> 0.23 </dynamic_friction>	Contains a floating-point number that specifies the dynamic friction coefficient. The sid attribute is optional.	0	0 or 1
<restitution sid="..."> 0.2 </restitution>	Contains a floating-point number that is the proportion of the kinetic energy preserved in the impact (typically ranges from 0.0 to 1.0). Also known as "bounciness" or "elasticity." The sid attribute is optional.	0	0 or 1
<static_friction sid="..."> 0.23 </static_friction>	Contains a floating-point number that specifies the static friction coefficient. The sid attribute is optional.	0	0 or 1

5.6.19.7 Example

```
<rigid_body id="bouncy_ball">
  <shape>
    <sphere> <radius> 1 </radius> </sphere>
    <instance_physics_material url="#my_physics_material" />
    <physics_material id="bouncy_material">
      <technique_common>
        <dynamic_friction> 0.12 </dynamic_friction>
        <restitution> 0.05 </restitution>
        <static_friction> 0.23 </static_friction>
      </technique_common>
    </physics_material>
  </shape>
</rigid_body>
```

5.6.20 physics_model

Category: Physics model

5.6.20.1 General

Allows for building complex combinations of rigid bodies and constraints that may be instantiated multiple times.

5.6.20.2 Concepts

Visual-scene graph-node hierarchies have a natural grouping by using the root nodes. Rigid bodies do not have an articulated hierarchy. Instead, a simulation considers them all to be at the same level. The **<physics_model>** element provides a logical grouping mechanism for a collection of rigid bodies and constraints. Physics models might be as simple as a single rigid body, or as complex as a character with bones (rigid bodies) that have joints (constraints) linking them. Unlike **<node>**, a **<physics_model>** does not have transform children to specify a position and orientation.

Each child element defined inside a physics model has a `sid` attribute instead of an `id`. The `sid` is used to access and override components of a physics-model at the point of instantiation.

To use a **<physics_model>** in a simulation, it shall be instantiated in a **<physics_scene>** by using an **<instance_physics_model>** element.

There is a mechanism for a physics model to contain other previously defined physics models similar to how nodes can reference and reuse other nodes within a visual scene. For example, a house physics model could contain several instantiated physics models, such as walls made from bricks. This element defines the structure of such a model; the **<instance_physics_model>** element instantiates a **<physics_model>** and can override many of the **<physics_model>** parameters. The **<instance_physics_model>** element has child elements that indicate its position and orientation within the parent **<physics_model>**.

5.6.20.3 Attributes

The `<physics_model>` element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.6.20.4 Related elements

The `<physics_model>` element relates to the following elements:

Parent elements	<code>library_physics_models</code>
Child elements	See the following subclause.
Other	<code>instance_physics_model</code>

5.6.20.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><asset></code>	See main entry in Core.	N/A	0 or 1
<code><rigid_body></code>	Defines a <code><rigid_body></code> element and sets its nondefault properties. See main entry.	N/A	0 or more
<code><rigid_constraint></code>	Defines a <code><rigid_constraint></code> element and allows for overriding some or all of its properties. See main entry.	N/A	0 or more
<code><instance_physics_model></code>	Instantiates a physics model from the given url, and assigns a <code>sid</code> to it, to distinguish it from other child elements. See main entry.	N/A	0 or more
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.6.20.6 Example

```
<library_physics_models>
  <!-- Defines a catapult physics model that can be reused and/or -->
  <!-- modified in other physics models or in a physics_scene. -->
  <physics_model id="catapultModel">
    <!-- This is the base of the catapult, defined inline. -->
    <rigid_body sid="base">
      <technique_common>
        <dynamic>false</dynamic>
        <instance_physics_material url="#catapultBasePhysicsMaterial"/>
        <shape>
          <instance_geometry url="#catapultBaseConvexMesh"/>
          <!-- Local position of base relative to the catapult model. -->
          <translate> 0 -1 0 </translate>
        </shape>
      </technique_common>
    </rigid_body>
  </physics_model>
</library_physics_models>
```

```

<!-- The top (or arm) of the catapult is defined similarly. -->
<rigid_body sid="top">
  <technique_common>
    <dynamic>true</dynamic>
    <shape>
      <instance_geometry url="#catapultTopConvexMesh"/>
      <translate> 0 3 0 </translate>
    </shape>
  </technique_common>
</rigid_body>
<!-- Define the angular spring that drives the catapult movement.
      Optionally, a url could have been provided to copy a rigid
      constraint from some other physics model. -->
<rigid_constraint sid="spring_constraint">
<ref_attachment rigid_body="./base">
  <translate sid="translate">-2. 1. 0</translate>
</ref_attachment>
<attachment rigid_body="./top">
  <translate sid="translate">1.23205 -1.86603 0</translate>
  <rotate sid="rotateZ">0 0 1 -30.</rotate>
</attachment>
<technique_common>
  <limits>
    <swing_cone_and_twist>
      <min> -180.0 0.0 0.0 </min>
      <max> 180.0 0.0 0.0 </max>
    </swing_cone_and_twist>
  </limits>
  <spring>
    <angular>
      <stiffness>500</stiffness>
      <damping>0.3</damping>
      <target_value>90</target_value>
    </angular>
  </spring>
</technique_common>
</rigid_constraint>
<extra> <skeleton url="#my_catapult_nodes"/> </extra>
</physics_model>
<!-- This physics model combines the two previously defined models. -->
<physics_model id="catapultAndRockModel">
<!-- This rock is taken from a library of predefined physics models. -->
  <instance_physics_model sid="rock"
    url="http://feelingsoftware.com/models/rocks.dae#rockModels/bigRock">
    <!-- Placement of rock on catapult in catapultAndRockModel space -->
    <translate> 0 4 0 </translate>
  </instance_physics_model>
  <instance_physics_model sid="catapult" url="#catapultModel"/>
</physics_model>
</library_physics_models>

```

5.6.21 physics_scene

Category: Physics scene

5.6.21.1 General

Specifies an environment in which physical objects are instantiated and simulated.

5.6.21.2 Concepts

COLLADA allows for multiple simulations to run independently for the following main reasons:

- Multiple simulations may need different global settings and they might even run on different physics engines or on different hardware.
- By providing such a high-level grouping mechanism, we can minimize interactions to improve performance. For example, rigid bodies in one physics scene are known not collide with rigid bodies of other physics scenes, so no collision tests need to be done between them.
- It allows for supporting multiple levels of detail (LOD)

The **<physics_scene>** element may contain techniques, extra elements, and a list of **<instance_physics_model>** elements.

The “active” **<physics_scene>**s (ones that are simulated) are indicated by instantiating them under the main **<Scene>**.

5.6.21.3 Attributes

The **<physics_scene>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.6.21.4 Related elements

The **<physics_scene>** element relates to the following elements:

Parent elements	library_physics_scenes
Child elements	See the following subclause.
Other	instance_physics_scene

5.6.21.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<instance_force_field>	Instantiates a <force_field> element to influence this physics scene. See main entry.	N/A	0 or more
<instance_physics_model>	Instantiates a <physics_model> element and allows for overriding some or all of its children. See main entry.	N/A	0 or more

Name/example	Description	Default	Occurrences
<technique_common>	Specifies physics-scene information for the common profile that all COLLADA implementations shall support. See "The common profile" clause for usage information and the following subclause for child element details.	N/A	1
<technique> (Core)	Each <technique> specifies physics-scene information for a specific profile as designated by the <technique> 's profile attribute. See main entry in Core.	N/A	0 or more
<extra>	See main entry in Core.	N/A	0 or more

5.6.21.6 Child elements for **<physics_scene>** / **<technique_common>**

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<gravity sid=" " >	A vector representation of the scene's gravity force field. It is given as a denormalized direction vector of three floating-point values that indicate both the magnitude and direction of acceleration caused by the field. The sid attribute is optional.	N/A	0 or 1
<time_step sid=" " >	The integration time step, measured in seconds, of the physics scene. This value is engine-specific. If omitted, the physics engine's default is used. Contains a floating-point number. The sid attribute is optional.	N/A	0 or 1

5.6.21.7 Example

```

<library_physics_scenes>
<!-- regular physics scene. -->
  <physics_scene id="ColladaPhysicsScene">
    <instance_physics_model sid="firstCatapultAndRockInstance">
      url="#catapultAndRockModel" parent="#catapult1">
    <!-- Instance of physics model, with overrides.
    The current transform matrix will dictate the initial position and
    orientation of the physics model in world space. -->
    <instance_rigid_body body="./rock/rock" target="#rockNode">
      <technique_common>
        <velocity>0 -1 0</velocity> <!-- optional overrides -->
        <mass>10</mass>
    <!-- heavier -->
      </technique_common>
    </instance_rigid_body>
    <instance_rigid_body body="./catapult/top" target="#catapultTopNode"/>
    <instance_rigid_body body="./catapult/base" target="#baseNode"/>
  </instance_physics_model>
  <technique_common>
    <gravity>0 -9.8 0</gravity>
    <time_step>3.e-002</time_step>
  </technique_common>
</physics_scene>

```



```

</library_physics_scenes>
<!-- A scene where an "army" of two physically simulated catapults is
instantiated -->
<library_visual_scenes>
  <visual_scene id="battlefield">
    <node id="catapult1">
      <translate sid="translate">0 -0.9 0</translate>
      <node id="rockNode">
        <instance_geometry url="#someRockVisualGeometry"/>
      </node>
      <node id="catapultTopNode">
        <instance_geometry url="#someVisualCatapultTopGeometry"/>
      </node>
      <node id="catapultBaseNode">
        <instance_geometry url="#someVisualCatapultBaseGeometry"/>
      </node>
    </node>
  <!-- Can replicate a physics model by instantiating one of its parent nodes -->
  <node id="catapult2">
    <translate/>
  <!-- Position the second catapult somewhere else -->
  <rotate/>
  <instance_node url="#catapultNode1"/> <!-- replicate physics model &
visuals -->
  </node>
</visual_scene>
</library_visual_scenes>
<scene>
  <!-- Indicates that the physics scene is applicable to this visual scene -->
  <instance_physics_scene url="#ColladaPhysicsScene"/>
  <instance_visual_scene url="#battlefield"/>
</scene>

```

5.6.22 plane

Category: Analytical shape (Physics)

Surfaces (B-Rep)

5.6.22.1 General

Defines an infinite plane.

5.6.22.2 Concepts

In Physics, a plane is often used as a static collision object to prevent dynamic rigid bodies from going beyond a boundary or falling forever. See the “Geometry types” in 5.6.2.8.

In B-Rep, a plane is another type of surface. See **<surface>** in 5.9 B-Rep Reference for an explanation of the plane’s coordinate system in a b-rep.

5.6.22.3 Attributes

The **<plane>** element has no attributes.

5.6.22.4 Related elements

The **<plane>** element relates to the following elements:

Parent elements	shape, surface (B-Rep)
Child elements	See the following subclause.
Other	None

5.6.22.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<equation>	Contains four floating-point values that represent the coefficients for the plane's equation: $Ax + By + Cz + D = 0$ This element has no attributes.	None	1
<extra>	See main entry in Core.	N/A	0 or more

5.6.22.6 Example

```
<rigid_body>
  <dynamic>false</dynamic>
  <shape>
    <plane>
      <!-- Plane equation: Ax + By + Cz + D = 0 -->
      <!-- A, B, C, D coefficients (normal & D) -->
      <equation> 0.0  1.0  0.0  0.0 </equation> <!-- The X-Z plane (ground) -->
    </plane>
  </shape>
</rigid_body>
```

5.6.22.7 ref_attachment

Category: Physics model

5.6.22.8 General

Defines an attachment frame of reference, to a rigid body or a node, within a rigid constraint.

5.6.22.9 Concepts

A **<rigid_constraint>** attaches (and limits the motion between) two rigid bodies together. **<ref_attachment>** refers to the first rigid body, and **<attachment>** to the second. For example, in the case of a hinge constraint between a door and a wall, one of them is the reference attachment (in this case, the wall), and the other is the attachment (the door).

The **<ref_attachment>** also defines the local coordinate frame for that end of the connection, relative to the rigid body (or node), using **<translate>** and **<rotate>** elements. For example, you attach the hinge (rigid constraint) to the middle of the edge of the wall (rigid body), relative to the wall's local origin.

5.6.22.10 Attributes

The **<ref_attachment>** element has the following attribute:

rigid_body	xs:anyURI	A scoped-identifier reference to a <rigid_body> or <node> . This shall refer to the SID of a <rigid_body> either in <attachment> or in <ref_attachment> ; they cannot both be <node> s. Required.

5.6.22.11 Related elements

The **<ref_attachment>** element relates to the following elements:

Parent elements	rigid_constraint
Child elements	See the following subclause.
Other	attachment

5.6.22.12 Child elements

Child elements **<translate>** and **<rotate>** can appear in any order, and **<extra>** shall be last if present:

Name/example	Description	Default	Occurrences
<translate>	The position of the transform indicates the attachment point on the corresponding <rigid_body> . See main entry in Core.	N/A	0 or more
<rotate>	The orientation of the transform indicates the alignment of the joint frame for that <rigid_body> . See main entry in Core.	N/A	0 or more
<extra>	See main entry in Core.	N/A	0 or more

5.6.22.13 Example

```
<ref_attachment rigid_body="./SomeRigidBody">
  <translate>
  <rotate>
  <extra>
</ref_attachment>
```

For a more complete example, see **<rigid_constraint>**.

5.6.23 rigid_body

Category: Physics model

5.6.23.1 General

Describes simulated bodies that do not deform.

5.6.23.2 Concepts

Rigid bodies may or may not be connected by constraints (hinge, ball-joint, and so on). Rigid bodies, constraints, and so on are encapsulated in **<physics_model>** elements to allow the instantiation of complex models.

Rigid bodies consist of parameters and a collection of shapes for collision detection. Each shape may be rotated and/or translated to allow for building complex collision shapes ("bounding shape"). These shapes are described by one or more **<shape>** elements.

5.6.23.3 Attributes

The **<rigid_body>** element has the following attributes:

sid	sid_type	A text string containing the scoped identifier of the <rigid_body> element. This value shall be unique among its sibling elements. Associates each rigid body with a visual <node> when a <physics_model> is instantiated. Required. For details, see "Address syntax" in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.

5.6.23.4 Related elements

The **<rigid_body>** element relates to the following elements:

Parent elements	physics_model
Child elements	See the following subclause.
Other	instance_rigid_body

5.6.23.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<technique_common>	Specifies rigid-body information for the common profile that every COLLADA implementation shall support. See "The common profile" clause for usage information and the following subclause for child element details.	N/A	1
<technique> (Core)	Each <technique> specifies rigid-body information for a specific profile as designated by the <technique> 's profile attribute. See main entry in Core.	N/A	0 or more
<extra>	User-defined, multi-representable data that adds information to the <rigid_body> (as opposed to switching base-data, like the <technique> element does). See main entry in Core.	N/A	0 or more

5.6.23.6 Child elements for <rigid_body> / <technique_common>

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><dynamic sid="..."> false </dynamic></code>	Contains a Boolean that specifies whether the rigid_body is movable. The <code>sid</code> attribute is optional.	None	0 or 1
<code><mass sid="..."> 0.5</mass></code>	Contains a floating-point value that specifies the total mass of the rigid_body. The <code>sid</code> attribute is optional.	Derived from density x total shape volume	0 or 1
<code><mass_frame> <translate> ... </translate> <rotate> ... </rotate> </mass_frame></code>	Specifies the centre of mass and the alignment of the principal axes of the rigid_body using COLLADA's mechanism for expressing transformations. Because there can be multiple <code><translate></code> and <code><rotate></code> entries in any order, the affine parts of the resulting transformation shall be extracted. This final translation and rotation correspond to the centre of mass and the principal axes, respectively. The <code><translate></code> and <code><rotate></code> child elements can each appear 0 or more times, although at least one of the two shall be present. See their main entries in Core.	"identity" (centre of mass is at the local origin and the principal axes are the local axes).	0 or 1
<code><inertia sid="..."> 1 1 1 </inertia></code>	Contains three floating-point numbers, which are the diagonal elements of the inertia tensor matrix for the rigid body as aligned with its principal axes. With this alignment the off-diagonal elements of the tensor matrix are zero. The <code>sid</code> attribute is optional.	Derived from mass, shape volume and centre of mass.	0 or 1
<code><physics_material> or <instance_physics _material></code>	Defines or references a physics_material for the rigid_body. See main entries.	N/A	0 or 1
<code><shape></code>	See main entry.	N/A	1 or more

5.6.23.7 How the various mass properties are specified

There are four possible cases :

- Rigid body's mass not provided: Uses shape volumes and masses to compute rigid-body mass, inertia, and mass frame. Any supplied inertia or frame transformation would be overwritten during this calculation. See **<shape>** for more information on specification of mass or density at that level;
- Rigid body's mass provided but inertia not provided: As when mass is not specified, it uses shape volumes and masses to compute inertia and mass frame. Additionally, it scales the derived inertia tensor by the ratio of the provided mass to the mass predicted by the volume integration;

- Rigid body's mass provided, and inertia provided: It expects that mass frame is also provided or else uses the identity, which would be saying that the body's centre of gravity is the local origin and it happens to be lined up along the principle axes;
- Both the rigid-body and its shapes may specify either mass or density. If neither is defined, density will default to 1.0 and mass will be computed using the total volume of the shapes.

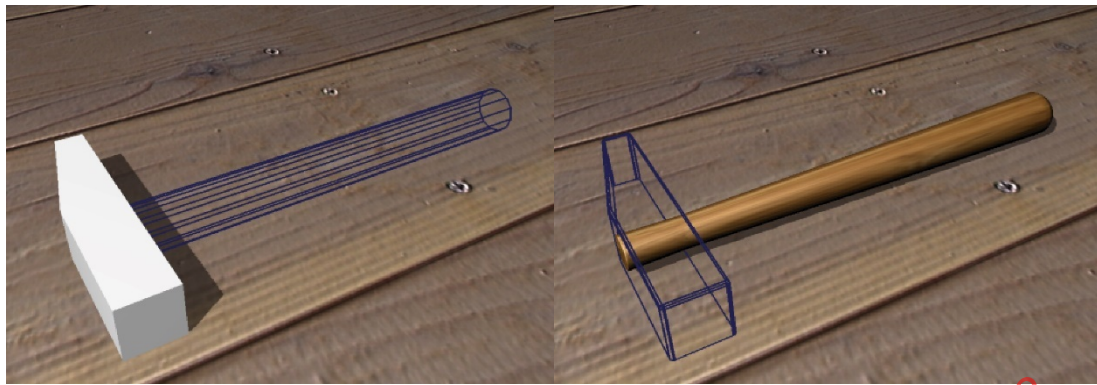
5.6.23.8 Example

The following is a compound rigid-body. Note the difference between the shapes meant for physics (cylinder primitive and simple convex hull) and the ones for rendering (See with Figure 17) (textured, tapered handle and bevelled head):

```
<library_geometries>
  <geometry id="hammerHeadForPhysics">
    <mesh>
...
    </mesh>
  </geometry>

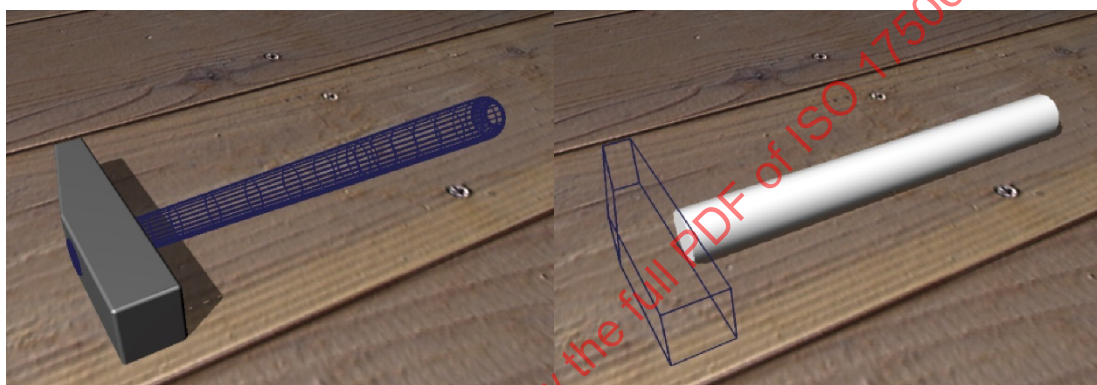
  <geometry id="hammerHandleToRender">
    <mesh>
    </mesh>
  </geometry>

  <geometry id="hammerHeadToRender">
    <mesh>
...
    </mesh>
  </geometry>
</library_geometries>
<library_physics_models>
  <physics_model id="HammerPhysicsModel">
    <rigid_body id="HammerHandleRigidBody">
      <technique_common>
        <mass> 0.25 </mass>
        <mass_frame> ... </mass_frame>
        <inertia> ... </inertia>
        <shape>
          <instance_physics_material url="#WoodPhysMtl"/>
<!-- This geometry is small and not used elsewhere, so it is inlined -->
          <cylinder>
            <height> 8.0 </height>
            <radius> 0.5 0.5 </radius>
          </cylinder>
        </shape>
        <shape>
          <mass> 1.0 </mass>
          <!-- This geometry is referenced rather than inlined -->
          <instance_physics_material url="#SteelPhysMtl"/>
          <instance_geometry url="#hammerHeadForPhysics"/>
          <translate> 0.0 4.0 0.0 </translate>
        </shape>
      </technique_common>
    </rigid_body>
  </physics_model>
</library_rigid_bodies>
```



a) hammerHeadForPhysics

b) hammerHandleToRender



c) hammerHeadToRender

d) HammerHandleRigidBody

Figure 17 — Examples of rigid body

5.6.24 rigid_constraint

Category: Physics model

5.6.24.1 General

Constrains rigid bodies to each other or to the world.

5.6.24.2 Concepts

Constraints are configurable in how they limit freedom of linear and angular relative movement. A collection of rigid bodies and constraints can compose complex physics models with moveable parts.

Building interesting physical models generally means attaching some of the rigid bodies together, using springs, ball joints, or other types of rigid constraints.

COLLADA supports constraints that link two rigid bodies or a rigid body and a coordinate frame in the scene hierarchy (for example, world space). Instead of defining a large combination of constraint primitive elements, COLLADA offers one very flexible element, the general six-degrees-of-freedom (DOF) constraint. Simpler constraints (for example, linear or angular spring, ball joint, hinge) may be expressed in terms of this general constraint.

A constraint is specified by:

- Two attachment frames defined using a translation and orientation relative to a rigid body's local space or to a coordinate frame in the scene hierarchy. To remain consistent with the rest of COLLADA, this is expressed using standard **<translate>** and **<rotate>** elements;
- Its degrees-of-freedom (DOF). A DOF specifies the variability along a given axis of translation or axis of rotation, expressed in the space of the attachment frame. For example, a door hinge typically has one degree of freedom, along a given axis of rotation. In contrast, a slider joint has one degree of freedom along a single axis of translation.

Degrees-of-freedom and limits are specified by the very flexible **<limits>** element.

5.6.24.3 Attributes

The **<rigid_constraint>** element has the following attributes:

sid	sid_type	A text string containing the scoped identifier of the <rigid_constraint> element. This value shall be unique within the scope of the parent element. Required. For details, see "Address syntax" in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.

5.6.24.4 Related elements

The **<rigid_constraint>** element relates to the following elements:

Parent elements	physics_model
Child elements	See the following subclause.
Other	instance_rigid_constraint

5.6.24.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><ref_attachment></code>	Defines the attachment frame of reference (to a rigid_body or a node) within a rigid constraint. See main entry.	N/A	1
<code><attachment></code>	Defines an attachment frame (to a rigid body or a node) within a rigid constraint. See main entry.	N/A	1
<code><technique_common></code>	Specifies rigid-constraint information for the common profile that all COLLADA implementations shall support. See “The common profile” clause for usage information and the following subclause for child element details.	N/A	1
<code><technique></code> (Core)	Each <code><technique></code> specifies rigid-constraint information for a specific profile as designated by the <code><technique></code> 's profile attribute. See main entry in Core.	N/A	0 or more
<code><extra></code>	User-defined, multi-representable data. See main entry in Core.	N/A	0 or more

5.6.24.6 Child elements for `<rigid_constraint>` / `<technique_common>`



Figure 18 — Example of swing cone and twist

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><enabled sid="...">true</enabled></code>	Contains a Boolean. If false, the <code><constraint></code> doesn't exert any force or influence on the rigid bodies. The <code>sid</code> attribute is optional.	True	0 or 1
<code><interpenetrate sid="...">true</interpenetrate></code>	Contains a Boolean. If true, the attached rigid bodies may interpenetrate. The <code>sid</code> attribute is optional.	False	0 or 1

Name/example	Description	Default	Occurrences
Two constraints with “swing-cone and twist”-type angular limits: <pre><limits> <swing_cone_and_twist> <min sid="..."> -15.0 -15.0 -INF </min> <max sid="..."> 15.0 15.0 INF </max> </swing_cone_and_twist> <linear> <min sid="..."> 0 0 0 </min> <max sid="..."> 0 0 0 </max> </linear> </limits></pre>	The <limits> element provides a flexible way to specify the constraint limits (degrees of freedom and ranges). This element has no attributes. This element may contain the optional child elements <swing_cone_and_twist>, <linear>, or both, which shall appear in the order shown if both are used. If these limit descriptions are not sufficient, use a custom <technique>. The <linear> element describes linear (translational) limits along each axis. The <swing_cone_and_twist> element describes the angular limits along each rotation axis in degrees. The <min> and <max> elements are optional but shall appear in the order shown if used. Their sid attributes are optional. They each contain three floating-point values representing x, y, and z limits. The values INF and -INF, corresponding to +/- infinity, can also be used to indicate that there is no limit along that axis. Limits are expressed in the space of ref_attachment. In <swing_code_and_twist>, the x and y limits describe a “swing cone” and the z limits describe the “twist angle” range (see Figure 18).	linear: min: 0.0 0.0 0.0 max: 0.0 0.0 0.0 swing_cone_and_twist: min: 0.0 0.0 0.0 max: 0.0 0.0 0.0 This corresponds to a completely fixed rigid constraint, that is, the two rigid bodies do not move relative to each other. (No rotation or translation allowed.)	0 or 1
Example 1: <pre><spring> <linear> <stiffness sid="...">5.4544 </stiffness> <damping sid="...">0.4132 </damping> <target_value sid="...">3 </target_value> </linear> </spring></pre> Example 2: <pre><spring> <angular> <stiffness>5.4544 </stiffness> <damping>0.4132 </damping> <target_value>90 </target_value> </angular> </spring></pre>	Spring is based on either distance (<linear>) or angle (<angular>), or both; if both are specified, <angular> shall appear first. Each can have three optional child elements, which shall appear in the order shown if used. They each contain a single floating-point value. Their sid attributes are optional. The <stiffness> (also called spring coefficient) has units of force/distance for <linear> or force/angle in degrees for <angular>. Spring is expressed in the space of ref_attachment.	stiffness: 1.0 damping: 0.0 target_value: 0.0 This corresponds to an “infinitely rigid” constraint, that is, no spring.	0 or 1

5.6.24.7 Examples

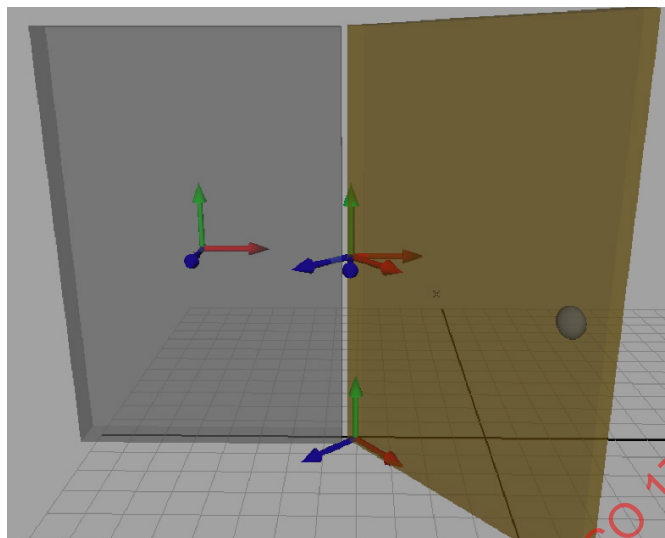


Figure 19 — Example a door with hinge

This example demonstrates a door with a hinge in Figure 19. The wall rigid body (in gray, on the right) has its local space frame in its centre. The door has its local space on the floor and rotated 45 degrees on the y axis. The hinge constraint is limited to rotate ± 90 degrees on its y axis. Each attachment frame has its translate/rotate transforms defined in terms of the rigid body's local space.

```
<library_physics_models>
  <physics_model>
    <rigid_body sid="doorRigidBody">
      <technique_common>...</technique_common>
    </rigid_body>
    <rigid_body sid="wallRigidBody">
      <technique_common>...</technique_common>
    </rigid_body>
    <rigid_constraint sid="rigidHingeConstraint">
      <ref_attachment rigid_body="#wallRigidBody">
        <translate sid="translate">5 0 0</translate>
      </ref_attachment>
      <attachment rigid_body="#doorRigidBody">
        <translate sid="translate">0 8 0</translate>
        <rotate sid="rotateX">0 1 0 -45.0</rotate>
      </attachment>
    </rigid_constraint>
  </physics_model>
<!--Adding sid attributes here allows us to target the limits from animations -->
  <technique_common>
    <limits>
      <swing_cone_and_twist>
        <min sid="swing_min">0 90 0</min>
        <max sid="swing_max">0 -90 0</max>
      </swing_cone_and_twist>
    </limits>
  </technique_common>
</library_physics_models>
```

5.6.25 shape

Category: Physics model

5.6.25.1 General

Describes components of a **<rigid_body>**.

5.6.25.2 Concepts

Rigid-bodies may contain a single shape or a collection of shapes for collision detection. Each shape may be rotated and/or translated to allow for building complex collision shapes (a *bounding shape*).

These shapes are described by **<shape>** elements, each of which may contain:

- A **<physics_material>** definition or instance that describes the surface properties for restitution and friction;
- Physical properties (mass or density);
- Transforms (**<rotate>**, **<translate>**);
- An instance or an inlined definition of a **<geometry>**.

The mass properties at the shape level are irrelevant in physics simulation. It is only the mass properties at the rigid-body level that matter. The shape element can specify mass or density so that the parent rigid body can derive, if necessary, the body's mass properties by integrating over its child shapes. COLLADA is designed to be an interchange format. Typically, it is easier for content creators to assign a weight to individual things within a group than to properly assign a mass, centre of mass, and moment of inertia to group of geometries. The calculation of the mass properties is usually done during the content export pipeline after art creation and before instantiating within the physics engine.

Shapes may be hollow, meaning that the mass is not distributed through the whole volume, and is instead at the surface. In this case, density, if specified, indicates mass per unit-of-length square. Hollow versus non-hollow (solid) affects only the calculation of mass, inertia, and centre of mass.

5.6.25.3 Attributes

The **<shape>** element has no attributes.

5.6.25.4 Related elements

The **<shape>** element relates to the following elements:

Parent elements	rigid_body / technique_common , instance_rigid_body / technique_common
Child elements	See the following subclause.
Other	None

5.6.25.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><hollow sid="..."> true</hollow></code>	Contains a Boolean. If true, the mass is distributed along the surface of the shape. The <code>sid</code> attribute is optional.	None	0 or 1
<code><mass sid="..."> 0.5 </mass></code>	Contains a floating-point number specifying the mass of the shape. The <code>sid</code> attribute is optional.	Derived from density x shape volume	0 or 1
<code><density sid="..."> 0.5 </density></code>	Contains a floating-point number specifying the density of the shape. The <code>sid</code> attribute is optional.	Derived from mass / shape volume	0 or 1
inline definition or instance: <code><physics_material></code> or <code><instance_physics_material></code>	The <code><physics_material></code> used for this shape.	From the geometry that is instantiated or defined by the <code><shape></code> .	0 or 1
geometry of the shape	This can be either of the following: - An inline definition using one of the following elements: <code><plane></code>, <code><box></code>, <code><sphere></code>, <code><cylinder></code>, or <code><capsule></code>. - A geometry instance using the <code><instance_geometry></code> element, which references other geometry types (<code><mesh></code>, <code><convex_mesh></code>, <code><spline></code>, and so on).	N/A	1
<code><rotate></code> , <code><translate></code>	Transformation for the shape. Any combination of these elements in any order. See main entry in Core and see <code><node></code> for additional information.	No transforms	0 or more
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.6.25.6 Details

See also the `<rigid_body>` element.

5.6.25.7 Example

```
<physics_model>
  <rigid_body sid="HammerHandleRigidBody">
    <technique_common>
      <shape>
        <mass> 0.25 </mass>
        <instance_physics_material url="#WoodPhysMtl"/>
        <instance_geometry url="#hammerHandleForPhysics"/>
      </shape>
    </technique_common>
  </rigid_body>
</physics_model>
```

5.6.26 sphere

Category: Analytical shape (Physics)
Surfaces (B-Rep)

5.6.26.1 General

Describes a sphere that is centred around its local origin.

5.6.26.2 Concepts

In physics, geometric primitives, or analytical shapes, such as spheres are mostly useful for collision shapes. See the “Geometry types” in 5.6.2.8.

In B-rep, a sphere is a type of surface that is defined by its radius and is positioned in space by a coordinate system, the origin of which is the centre of the sphere.

5.6.26.3 Attributes

The **<sphere>** element has no attributes.

5.6.26.4 Related elements

The **<sphere>** element relates to the following elements:

Parent elements	shape , surface (B-rep)
Child elements	See the following subclause.
Other	None

5.6.26.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<radius>	Contains a floating-point value that specifies the radius of the sphere. This element has no attributes.	None	1
<extra>	See main entry in Core.	N/A	0 or more

5.6.26.6 Example

```
<shape>
  <sphere>
    <radius> 1.0 </radius>
  </sphere>
</shape>
```

5.7 Getting started with COLLADA FX

5.7.1 General

COLLADA FX enables authors to describe how to apply color to a visual scene. It is a flexible abstraction for describing material properties across many platforms and application programming interfaces (APIs).

The FX elements of the COLLADA schema allow the description of:

- Single and multipass effects, which are abstract material definitions (for example, plastic)
- Effect parameterizations (using `<newparam>`)
- Effect metadata
- Binding to the scene graph
- Multiple techniques
- Inline and external source code or binary

Multiple application programming interfaces (APIs) and shader languages are supported through `<profile_*>` elements, allowing each effect to be described for multiple platforms. Each platform can be fully described, with platform-specific data types, render states, and capabilities.

Within each platform, each effect can be described using many techniques. A technique is a user-labeled description of a style of rendering (for example, “daytime,” “nighttime,” “magic,” “superhero_mode”), a different level of detail, or a method of calculation (for example, “approximate,” “accurate,” “high_LOD,” “low_LOD”).

At a higher level, a material system allows predefined effects to be specialized into a specific instance by providing values to parameters other than the default values in the effect definition. This allows a single effect to be used as a basis for many different materials.

Finally, materials are put to use when they are bound to one or more points in the scene graph. The `<bind_material>` element in the scene graph’s geometry may instantiate one or more materials and connect them to segments of the geometry. Within the material instance, further specialization of effects may occur by binding to resources in the scene graph, such as lights and cameras, or pairing texture coordinates.

5.7.2 Using profiles for platform-specific effects

5.7.2.1 General

The `<profile_*>` elements allow each effect to be described for multiple platforms.

5.7.2.2 About profiles

The `<profile_*>` elements encapsulate all platform-specific values and declarations for effects in a particular profile. They define the clear interface between concrete, platform-specific data types and the abstract COLLADA data types used in the rest of the document.

COLLADA FX supports the following profiles:

- **<profile_COMMON>**: Encapsulate all the values and declarations for a platform-independent fixed-function shader. All platforms are required to support **<profile_COMMON>**. Effects in this profile are designed to be used as the reliable fallback when no other profile is recognized by the current effects runtime.
- **<profile_CG>**: Encapsulates effects for use with the Cg high-level language.
- **<profile_GLES>**: Encapsulates effects for use with OpenGL ES [7].
- **<profile_GLES2>**: Encapsulates effects for use with OpenGL ES 2.0 [8].
- **<profile_GLSL>**: Encapsulates effects for use with OpenGL Shading Language [16].
- **<profile_BRIDGE>**: Provides the ability to bridge COLLADA FX effects with external effect-file formats such as NVIDIA⁴ CgFX [14] and Microsoft FX.

NOTES:

- Unlike techniques **<profile_COMMON>** does not need to be specified if other profiles are specified.
- **<profile_COMMON>** is a common effect definition that may not be desirable under all circumstances while **<technique_common>** is part of a different extensibility mechanism.

5.7.2.3 FX element attributes and structures in profiles

The specific attributes and child elements of each FX element may vary depending on the profile scope in which it is used. The profile scopes can be grouped as follows:

- External to effects (elements required to attach data to effects or apply effects for usage scenario)
- Effect (elements are valid in effect scope outside specific profiles)
- Common profile
- Cg profile
- OpenGL ES (GLES) profile
- GLSL profile
- OpenGL ES 2.0 (GLES2) profile

⁴ NVIDIA is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of this product(s).

Table 11 shows which elements are valid in which scope:

Table 11 — Valid elements in profiles

Element	External	Effect	Common	Cg	GLSL	GLSL	GLSL2
<alpha>	–	–	–	–	YES	–	–
<ambient> (FX)	–	–	YES	–	–	–	–
<annotate>	YES	YES	YES	YES	YES	YES	YES
<argument>	–	–	–	–	YES	–	–
<array>	–	–	–	YES	–	YES	YES
<binary>	–	–	–	–	–	–	YES
<bind> (FX)	YES	–	–	–	–	–	–
<bind_attribute>	–	–	–	–	–	YES	YES
<bind_material>	YES	–	–	–	–	–	–
<bind_uniform>	–	–	–	YES	–	YES	YES
<bind_vertex_input>	YES	–	–	–	–	–	–
<blinn>	–	–	YES	–	–	–	–
<code>	–	–	–	YES	–	YES	YES
<color_clear>	–	–	–	YES	YES	YES	YES
<color_target>	–	–	–	YES	YES	YES	YES
<compiler>	–	–	–	YES	–	–	YES
<constant> (FX)	–	–	YES	–	–	–	–
<constant> (combiner)	–	–	–	–	YES	–	–
<create_2d>	YES	–	–	–	–	–	–
<create_3d>	YES	–	–	–	–	–	–
<create_cube>	YES	–	–	–	–	–	–
<depth_clear>	–	–	–	YES	YES	YES	YES
<depth_target>	–	–	–	YES	YES	YES	YES
<diffuse>	–	–	YES	–	–	–	–
<draw>	–	–	–	YES	YES	YES	YES
<effect>	–	YES	–	–	–	–	–
<emission>	–	–	YES	–	–	–	–
<evaluate>	–	–	–	YES	YES	YES	YES
<format>	YES	–	–	–	–	–	–
<image>	YES	–	–	–	–	–	–
<include>	–	–	–	YES	–	YES	YES
<index_of_refraction>	–	–	YES	–	–	–	–
<init_from>	YES	–	–	–	–	–	–
<instance_effect>	YES	–	–	–	–	–	–

Element	External	Effect	Common	Cg	GLSL	GLSL	GLSL2
<instance_image>	YES	YES	YES	YES	YES	YES	YES
<instance_material> (geometry)	YES	-	-	-	-	-	-
<instance_material> (rendering)	YES	-	-	-	-	-	-
<lambert>	-	-	YES	-	-	-	-
<library_effects>	YES	-	-	-	-	-	-
<library_images>	YES	-	-	-	-	-	-
<library_materials>	YES	-	-	-	-	-	-
<linker>	-	-	-	-	-	-	YES
<material>	YES	-	-	-	-	-	-
<modifier>	YES	YES	YES	YES	YES	YES	YES
<newparam>	-	YES	YES	YES	YES	YES	YES
<param> (reference)	-	-	YES	YES	-	YES	YES
<pass>	-	-	-	YES	YES	YES	YES
<phong>	-	-	YES	-	-	-	-
<profile_BRIDGE>	YES	-	-	-	-	-	-
<profile_CG>	-	-	-	YES	-	-	-
<profile_COMMON>	-	-	YES	-	-	-	-
<profile_GLES>	-	-	-	-	YES	-	-
<profile_GLES2>	-	-	-	-	-	-	YES
<profile_GLSL>	-	-	-	-	-	YES	-
<program>	-	-	-	YES	-	YES	YES
<reflective>	-	-	YES	-	-	-	-
<reflectivity>	-	-	YES	-	-	-	-
<render>	YES	-	-	-	-	-	-
<RGB>	-	-	-	-	YES	-	-
<sampler1D>	-	-	-	YES	-	YES	-
<sampler2D>	YES	YES	YES	YES	YES	YES	YES
<sampler3D>	YES	YES	-	YES	-	YES	YES
<samplerCUBE>	YES	YES	YES	YES	-	YES	YES
<samplerDEPTH>	YES	YES	YES	YES	-	YES	-
<samplerRECT>	YES	YES	YES	YES	-	YES	-
<sampler_image>	YES	-	-	-	-	-	-
<semantic>	YES	YES	YES	YES	YES	YES	YES
<setparam>	YES	-	-	YES	-	-	YES
<shader>	-	-	-	YES	-	YES	YES
<shininess>	-	-	YES	-	-	-	-

Element	External	Effect	Common	Cg	GLSL	GLSL	GLSL2
<specular>	–	–	YES	–	–	–	–
<states>	–	–	–	YES	YES	YES	YES
<stencil_clear>	–	–	–	YES	YES	YES	YES
<stencil_target>	–	–	–	YES	YES	YES	YES
<technique> (FX)	–	–	YES	YES	YES	YES	YES
<technique_hint>	YES	–	–	–	–	–	–
<texcombiner>	–	–	–	–	YES	–	–
<texenv>	–	–	–	–	YES	–	–
<texture_pipeline>	–	–	–	–	YES	–	–
<transparency>	–	–	YES	–	–	–	–
<transparent>	–	–	YES	–	–	–	–
<usertype>	–	–	–	YES	–	–	YES

5.7.3 About parameters in COLLADA FX

For general information about parameters in COLLADA, see “About Parameters in COLLADA” in 5.4 Programming guideline.

In COLLADA FX, a **<newparam>** element declares a bindable parameter within the given scope. Parameter’s types do not have to strictly match each other to be successfully bound. The types shall be compatible, however, through simple (and sensible as defined by the application) conversion or promotion, such as integer to float_type, or float3_type to float4_type, or Boolean to int_type. COLLADA FX specifies no rules for casting types, so to author a file that is safe for use with a maximum number of applications would be to have proper type matches rather than to expect casting.

In **<effect>** scope, parameters are available to all platforms, but parameters declared inside a specific **<profile_*>** (see “Profiles”) block are available only to shaders that are also inside that profile. Parameters declared outside of this barrier may require casting when used inside a **<profile_*>** (see “Profiles”) block.

COLLADA provides the following elements for working with parameters in effects:

- **<newparam>** — Creates a parameter;
- **<setparam>** — Changes or sets the type and value of a parameter;
- **<modifier>** — Specifies the volatility or linkage of parameters, such as **UNIFORM** or **SHARED**, among others;
- **<array>** — In **<newparam>** or **<setparam>** defines the parameter to be an array;
- **<usertype>** — In **<newparam>** or **<setparam>**, defines the parameter to be a structure;
- **<annotate>** — Represents an object of the form symbol=value for use in parameters and in other places within FX;
- **<param>** (reference) — Refers to an existing parameter created by **<newparam>**.

Locating a Parameter in `<bind>` and `<bind_vertex_input>` The `<bind>` and `<bind_vertex_input>` elements bind the target to a parameter in an `<effect>`. The search string that identifies the parameter in the `<effect>` is specified by the semantic attribute. When locating the parameter in the `<effect>`, search in the following order:

- Find a COLLADA FX parameter by semantic;
- If the profile contains shading language code, find a parameter within the shader by semantic if the language supports semantics;
- Find a COLLADA FX parameter by sid;
- If the profile contains shading language code, find a parameter within the shader by name.

5.7.4 Shaders

COLLADA provides several elements that describe shaders as follows.

- `<blinn>`
- `<constant>`
- `<lambert>`
- `<phong>`

and several elements that describe aspects of shaders as follows.

- `<bind_uniform>`
- `<code>`
- `<compiler>`
- `<include>`
- `<shader>`

5.7.5 Rendering

5.7.5.1 Determining transparency (opacity)

If either `<transparent>` or `<transparency>` exists then transparency rendering is activated, the renderer needs to turn on alpha blending mode, and the following formulae define how to combine the two values. Use these formulae to get the correct results based on the opaque setting of `<transparent>`, where `fb` is the frame buffer, i.e. the image behind what is being rendered, and `mat` is the material color before the transparency calculation.

- In **A_ONE** opaque mode:

```
result.r = fb.r * (1.0f - transparent.a * transparency) + mat.r *
(transparent.a * transparency)
result.g = fb.g * (1.0f - transparent.a * transparency) + mat.g *
(transparent.a * transparency)
result.b = fb.b * (1.0f - transparent.a * transparency) + mat.b *
(transparent.a * transparency)
result.a = fb.a * (1.0f - transparent.a * transparency) + mat.a *
(transparent.a * transparency)
```

- In **RGB_ZERO** opaque mode:

```
result.r = fb.r * (transparent.r * transparency) + mat.r *
(1.0f - transparent.r * transparency)
result.g = fb.g * (transparent.g * transparency) + mat.g *
(1.0f - transparent.g * transparency)
result.b = fb.b * (transparent.b * transparency) + mat.b *
(1.0f - transparent.b * transparency)
result.a = fb.a * (luminance(transparent.rgb) * transparency) + mat.a *
(1.0f - luminance(transparent.rgb) * transparency)
```

- In **A_ZERO** opaque mode:

```
result.r = fb.r * (transparent.a * transparency) + mat.r * (1.0f -
transparent.a * transparency)
result.g = fb.g * (transparent.a * transparency) + mat.g * (1.0f -
transparent.a * transparency)
result.b = fb.b * (transparent.a * transparency) + mat.b * (1.0f -
transparent.a * transparency)
result.a = fb.a * (transparent.a * transparency) + mat.a * (1.0f -
transparent.a * transparency)
```

- In **RGB_ONE** opaque mode:

```
result.r = fb.r * (1.0f - transparent.r * transparency) + mat.r *
(transparent.r * transparency)
result.g = fb.g * (1.0f - transparent.g * transparency) + mat.g *
(transparent.g * transparency)
result.b = fb.b * (1.0f - transparent.b * transparency) + mat.b *
(transparent.b * transparency)
result.a = fb.a * (1.0f - luminance(transparent.rgb) * transparency) +
mat.a * (luminance(transparent.rgb) * transparency)
```

where **luminance** is the function, based on the ISO/CIE color standards [2] (see ITU-R Recommendation BT.709-4), that averages the color channels into one value:

```
luminance = (color.r * 0.212671) +
             (color.g * 0.715160) +
             (color.b * 0.072169)
```

The interaction between **<transparent>** and **<transparency>** is as follows:

- If **<transparent>** does not exist then it has no effect on the formula's result, and the opaque mode is the default opaque mode. This is equivalent to:

```
transparent = <color> 1.0 1.0 1.0 1.0 </color>
```

- If **<transparency>** does not exist, then it has no effect on the formula's result. This is equivalent to a factor that is 1.0:

```
transparency = <float> 1.0 <float>
```

- If both **<transparent>** and **<transparency>** exist, then both are honoured.

In the following example, the colors are used as specified but the RGB values are ignored for transparency calculations because A_ONE specifies that the transparency information comes from the alpha channel, not the RGB channels:

```
<transparent opaque=A_ONE><color>1 0 0.5 0</color></transparent>
```

5.7.6 Texturing

5.7.6.1 Texture mapping in <profile_COMMON>

This subclause provides an introduction to samplers and images.

To use an image as a texture, use the element relationships as follows:

```
texture->sampler->image
```

From the smallest part to the largest:

- An **<image>** element forms a single cohesive structure designed for storing image data. It may also contain 3D hardware concepts, such as MIP mapping, cubemaps, and volume slices because many image formats today can store this additional information. An **<image>** is embedded or referenced file data. It might be a format of traditional 2D planes, such as BMP, or it might be a complicated 3D format, such as DDS or OpenEXR, consisting of multiple image planes;
- A **<sampler*>** contains instructions on how to read data at a specific 1D, 2D, or 3D coordinate from an **<image>**. It references the **<image>** and specifies what operations to perform to sample the data at a given coordinate. A sampler's instructions include information on how to map the coordinate onto the image, such as wrap or mirror. The instructions also include filtering modes to instruct how one or more texels nearby coordinate are combined to produce the final output color;
- A **profile_COMMON's <texture>**'s responsibility is to bind geometry's texture coordinate set (array) to a **<sampler*>** so that the sampler can fetch the correct colors. The texcoord attribute on the **<texture>** is actually a semantic name. It is expected that the **<instance_geometry>**'s **<instance_material>** **<bind_vertex_input>** makes the connection between the **<texture>**'s texcoord attribute and the mesh's texture coordinate array.

Some DCC applications also specify **<extra>** information that modifies the TEXCOORDs before they are plugged into the sampler, such as `offsetU`, `offsetV`, `rotateUV`, or `noise`.

The following is an example of texturing using **<instance_material>** and related elements to instantiate a material with an **<image>** supplied through a **<sampler2D>** parameter:

```
...
<image id="image_id">
  <init_from>image_file.dds</init_from>
</image>
...
<effect id="effect_id">
  ...
  <profile_COMMON>
    <technique sid="technique_sid">
      <newparam sid="sampler2D_param_id">
        <sampler2D>
          <instance image url="#surface_param_id"/>
          ...
        </sampler2D>
      </newparam>
      <lambert>
        <diffuse>
          <texture texture="sampler2D_param_id" texcoord="myUVs"/>
        </diffuse>
      </lambert>
    </profile_COMMON>
  </effect>
  ...
</material id="material_id">
  <instance_effect url="#effect_id"/>
</material>
...
<geometry id="geometry_id">
  ...
  <input semantic="TEXCOORD" source="#..." offset=".." />
  <triangles material="material_symbol" count="...">
  ...
</geometry>
...
<scene>
  ...
  <instance_geometry url="#geometry_id">
    <bind_material>
      <technique_common>
        <instance_material symbol="material_symbol" target="#material_id">
          <bind_vertex_input semantic="myUVs" input_semantic="TEXCOORD" />
        </instance_material>
      </technique_common>
    </bind_material>
  </instance_geometry>
  ...
</scene>
```

5.8 FX reference

5.8.1 General

This clause covers the elements that compose COLLADA FX.

5.8.2 Elements by category

5.8.2.1 General

This clause lists elements in alphabetical order. The following tables list elements by category, for ease in finding related elements.

5.8.2.2 Effects

Element	Description
annotate	Adds a strongly typed annotation remark to the parent object.
bind_vertex_input	Binds geometry vertex inputs to effect vertex inputs upon instantiation.
effect	Provides a self-contained description of a COLLADA effect.
instance_effect	Instantiates a COLLADA effect.
library_effects	Provides a library in which to place <effect> assets.
technique (FX)	Holds a description of the textures, samplers, shaders, parameters, and passes necessary for rendering this effect using one method.
technique_hint	Adds a hint for a platform of which technique to use in this effect.

5.8.2.3 Materials

Element	Description
bind (FX)	Binds values to uniform inputs of a shader or binds values to effect parameters upon instantiation.
bind_material	Binds a specific material to a piece of geometry, binding varying and uniform parameters at the same time.
instance_material (geometry)	Instantiates a COLLADA material resource.
library_materials	Provides a library in which to place <material> assets.
material	Defines the formulas necessary for the visual appearance of geometry and screen-space image processing

5.8.2.4 Parameters

Element	Description
array	Creates a parameter of a one-dimensional array type.
modifier	Provides additional information about the volatility or linkage of a <newparam> declaration.
newparam	Creates a new, named parameter object and assigns it a type and an initial value. See 5.5 Core elements reference.
param (reference)	References a predefined parameter. See 5.5 Core elements reference.
sampler_image	Instantiates an image targeted for samplers.
sampler_states	Allows users to modify an effect's sampler state from a material.
semantic	Provides metadata that describes the purpose of a parameter declaration.
setparam	Assigns a new value to a previously defined parameter. See main entry in 5.5 Core elements reference.
usertype	Creates an instance of a structured class for a parameter.

5.8.2.5 Profiles

Element	Description
profile_BRIDGE	Provides support for referencing effect profiles written with external standards.
profile_CG	Declares a platform-specific representation of an effect written in the NVIDIA® Cg language. [14]
profile_COMMON	Opens a block of platform-independent declarations for the common, fixed-function shader.
profile_GLES	Declares platform-specific data types and <technique> s for OpenGL ES.
profile_GLES2	Declares platform-specific data types and <technique> s for OpenGL ES 2.0.
profile_GLSL	Declares platform-specific data types and <technique> s for OpenGL Shading Language.

5.8.2.6 Rendering

Element	Description
blinn	Produces a shaded surface with a Blinn BRDF approximation.
color_clear	Specifies whether a render target image is to be cleared, and which value to use.
color_target	Specifies which <image> will receive the color information from the output of this pass.
fx_common_color_or_texture_type contains: ambient (FX), diffuse, emission, reflective specular, transparent	A type that describes color attributes of fixed-function shader elements inside <profile_COMMON> effects.

Element	Description
fx_common_float_or_param_type contains: index_of_refraction reflectivity, shininess transparency	A type that describes the scalar attributes of fixed-function shader elements inside <profile_COMMON> effects. See main entry.
constant	Produces a constantly shaded surface that is independent of lighting.
depth_clear	Specifies whether a render target image is to be cleared, and which value to use.
depth_target	Specifies which <image> will receive the depth information from the output of this pass.
draw	Instructs the FX Runtime what kind of geometry to submit.
evaluate	Contains evaluation elements for a rendering pass.
instance_material (rendering)	Instantiates a COLLADA material resource for a screen effect.
lamBERT	Produces a diffuse shaded surface that is independent of lighting.
pass	Provides a static declaration of all the render states, shaders, and settings for one rendering pipeline.
phong	Produces a shaded surface where the specular reflection is shaded according to the Phong BRDF approximation.
render	Describes one effect pass to evaluate a scene.
states	Contains all rendering states to set up for the parent pass.
stencil_clear	Specifies whether a render target image is to be cleared, and which value to use.
stencil_target	Specifies which <image> will receive the stencil information from the output of this pass.

5.8.2.7 Shaders

Element	Description
binary	Identifies or provides a shader in binary form.
bind_attribute	Binds semantics to vertex attribute inputs of a shader.
bind_uniform	Binds values to uniform inputs of a shader or binds values to effect parameters upon instantiation.
code	Provides an inline block of source code.
compiler	Contains command-line or runtime-invocation options for a shader compiler.
include	Imports source code or precompiled binary shaders into the FX Runtime by referencing an external resource.
linker	Contains command-line or runtime-invocation options for shader linkers to combine shaders into programs.
program	Links multiple shaders together to produce a pipeline for geometry processing.
shader	Declares and prepares a shader for execution in the rendering pipeline of a <pass> .
sources	Concatenates the source code for a shader from one or more sources.

5.8.2.8 Texturing

Element	Description
alpha	Defines the alpha portion of a <texture_pipeline> command for combiner-mode texturing.
argument	Defines an argument of the RGB or alpha component of a texture-unit combiner-style texturing command.
create_2d	Assists in the manual creation of a 2D <image> asset.
create_3d	Assists in the manual creation of a 3D <image> asset.
create_cube	Initializes a cube <image> asset.
format	Describes the formatting or memory layout expected of an <image> asset.
image	Declares the storage for the graphical representation of an object.
init_from	Initializes an entire image or portions of an image from referenced or embedded data.
instance_image	Instantiates an image to use in a shader.
library_images	Provides a library in which to place <image> assets.
RGB	Defines the RGB portion of a <texture_pipeline> command for combiner-mode texturing.
fx_sampler_common	A type that describes the sampling states of the <sampler*> elements.
sampler1D	Declares a one-dimensional texture sampler.
sampler2D	Declares a two-dimensional texture sampler.
sampler3D	Declares a three-dimensional texture sampler.
samplerCUBE	Declares a texture sampler for cube maps.
samplerDEPTH	Declares a texture sampler for depth maps.
samplerRECT	Declares a RECT texture sampler.
texcombiner	Defines a <texture_pipeline> command for combiner-mode texturing.
texenv	Defines a <texture_pipeline> command for simple, noncombiner-mode texturing.
texture_pipeline	Defines a set of texturing commands that will be converted into multitexturing operations using glTexEnv in regular and combiner mode.

5.8.3 About COLLADA FX

See 5.7 Getting started with COLLADA FX.

5.8.4 alpha

Category: Texturing

Profile: **GL ES**

5.8.4.1 General

Defines the alpha portion of a **<texture_pipeline>** command for combiner-mode texturing.

5.8.4.2 Concepts

See **<texcombiner>** for details about assignments and overall concepts.

5.8.4.3 Attributes

The **<alpha>** element has the following attributes:

operator	Enumeration	Infers the use of <code>glTexEnv(TEXTURE_ENV, COMBINE_ALPHA, operator)</code> . Optional. See <texcombiner> for details. Valid values are: REPLACE MODULATE ADD ADD_SIGNED INTERPOLATE SUBTRACT
scale	float_type	Infers the use of <code>glTexEnv(TEXTURE_ENV, ALPHA_SCALE, scale)</code> . Optional. See <texcombiner> for details.

5.8.4.4 Related elements

The **<alpha>** element relates to the following elements:

Parent elements	texcombiner
Child elements	See the following subclause.
Other	None

5.8.4.5 Child elements

Name/example	Description	Default	Occurrences
<argument>	Sets up the arguments required for the given operator to be executed. See main entry.	N/A	1 to 3

5.8.4.6 Details

See **<texcombiner>** for details.

5.8.4.7 Example

See **<texture_pipeline>**.

5.8.5 annotate

Category: Effects

Profile: All and external

5.8.5.1 General

Adds a strongly typed annotation remark to the parent object.

5.8.5.2 Concepts

Annotations represent objects of the form *symbol* = *value*, where *symbol* is a user-defined identifier, specified with the `name` attribute, and *value* is a strongly typed value, specified as a child element. Annotations communicate metadata from the Effect Runtime to the application only and are not interpreted by the COLLADA document.

5.8.5.3 Attributes

The `<annotate>` element has the following attribute:

name	xs:token	The text string name of this element that represents the <i>symbol</i> in an object of the form <i>symbol</i> = <i>value</i> . Required.

5.8.5.4 Related elements

The `<annotate>` element relates to the following elements:

Parent elements	effect, technique (FX), pass, newparam
Child elements	See the following subclause.
Other	None

5.8.5.5 Child elements

Name/example	Description	Default	Occurrences
value_element	A strongly typed value that represents the <i>value</i> in an object of the form <i>symbol</i> = <i>value</i> . Consists of a COLLADA type element that contains a value of that type. Valid type elements are: bool, bool2, bool3, bool4, int, int2, int3, int4, float, float2, float3, float4, float2x2, float3x3, float4x4, string See 5.11 Types.	N/A	1

5.8.5.6 Details

There is currently no standard set of annotations.

5.8.5.7 Example

```
<annotate name="UIWidget"> <string> slider </string> </annotate>
<annotate name="UIMinValue"> <float> 0.0 </float> </annotate>
<annotate name="UIMaxValue"> <float> 255.0 </float> </annotate>
```

5.8.6 argument

Category: Texturing

Profile: GLES

5.8.6.1 General

Defines an argument of the RGB or alpha component of a texture-unit combiner-style texturing command.

5.8.6.2 Concepts

See `<texture_pipeline>` for more details about assignments and bigger picture.

This element is context-sensitive based on its parent element.

5.8.6.3 Attributes

The `<argument>` element has the following attributes.

source	Enumeration	Optional. Identifies where the source data for the argument comes from: When the parent is <code><RGB></code> , this infers a call to <code>glTexEnv(TEXTURE_ENV, SRC##idx##_RGB, sourcevalue)</code> . When the parent is <code><alpha></code> , this infers a call to <code>glTexEnv(TEXTURE_ENV, SRC##idx##_ALPHA, sourcevalue)</code> . Valid values are TEXTURE CONSTANT PRIMARY PREVIOUS. There is no default.
operand	Enumeration	Optional. Provides details about how to read the value from the source: When the parent is <code><RGB></code> , this infers a call to <code>glTexEnv(TEXTURE_ENV, OPERAND##idx##_RGB, sourcevalue)</code> and valid values are: SRC_COLOR ONE_MINUS_SRC_COLOR SRC_ALPHA ONE_MINUS_SRC_ALPHA; the default is SRC_COLOR. When the parent is <code><alpha></code> , this infers a call to <code>glTexEnv(TEXTURE_ENV, OPERAND##idx##_ALPHA, sourcevalue)</code> and valid values are: SRC_ALPHA ONE_MINUS_SRC_ALPHA; the default is SRC_ALPHA.
sampler	xs:NCName	Optional. The name of a sampler <code><newparam></code> from which to read the source. Used only when <code>source="TEXTURE"</code> . Acceptable values depend upon which version of OpenGL ES the shader is designed for: • GLES 1.0, all arguments within a <code><texenv></code> element shall refer to the same <code><newparam></code> because there is no combiner crossbar. • GLES 1.1, the texture combiner crossbar is available, so this attribute can refer to any <code><newparam></code>.
"##" is concatenate, idx represents the index in which the argument appeared inside its parent command (<code><texenv></code> or <code><texcombiner></code>) sourcevalue is a placeholder for a value.		

5.8.6.4 Related elements

The `<argument>` element relates to the following elements:

Parent elements	RGB, alpha
Child elements	None
Other	None

5.8.6.5 Details

<argument> sets up the arguments required for the given operator to be executed.

5.8.6.6 Example

See **<texture_pipeline>**.

5.8.7 array

Category: Parameters

Profile: CG, GLES2, GLSL

5.8.7.1 General

Creates a parameter of a one-dimensional array type.

5.8.7.2 Concepts

Array type parameters pass sequences of elements to shaders. Array types are sequences of a single data type. To create a multidimensional array declare it as an array of array types.

Arrays can be either unsized or sized declarations, with an unsized array requiring a concrete size (and data) to be set using **<setparam>** before it can be used as a parameter for a shader.

5.8.7.3 Attributes

The **<array>** element has the following attribute:

length	xs:positiveInteger	Required. The number of elements in the array.
resizable	xs:boolean	Optional. Valid only in CG scope. If true, the array can be resized when changed because it is connected to a Cg unsized array. The default is false.

5.8.7.4 Related elements

The **<array>** element relates to the following elements:

Parent elements	in Core: newparam (in some cases), setparam (in some cases) In FX: create_2d , create_3d , create_cube
Child elements	None for create2d , create3d , createcube . For others, see the following subclause.
Other	None

5.8.7.5 Child elements

Name/example	Description	Default	Occurrences
parameter_element	See “Parameter-Type Elements” in 5.11.3 for parameter-type elements valid in the appropriate scope: • CG: cg_param_group • GLSL: glsl_value_group • GLES2: gles2_value_group Each parameter-type group also includes the <array> element as a child. Use one additional child array element to declare each additional dimension in the array. See Note in “Details” about limitations on parameter-type combinations.	N/A	0 or more

5.8.7.6 Details

Note that on child-element conventions although the schema cannot enforce these conventions, they shall be followed for an array to be valid:

- All parameter-type child elements shall be of a consistent value type;
- There shall be either one parameter-type child, in which case that parameter type sets the type for the array and its value is used to initialize the entire array, or the number of parameter-type elements shall match the length attribute.

After creation, array elements can be addressed directly in **<setparam>** declarations using the normal C/C++ syntax for array indexing, for example, “numbers[3]” would be used to access the fourth element of the array in the example.

5.8.7.7 Example

```
<newparam sid="numbers">
  <array length="4">
    <float>1.0</float>
    <float>2.0</float>
    <float>3.0</float>
    <float>4.0</float>
  </array>
</newparam>
<setparam ref="numbers[2]">
  <float>2.5</float>
</setparam>
```

An example of a rectangular array, size 2x3:

```
<array length="2">
  <array length="3"><float>1</float><float>1</float><float>1</float></array>
  <array length="3"><float>1</float><float>1</float><float>1</float></array>
</array>
```

An example of a jagged array:

```
<array length="2">
  <array length="2"><float>1</float><float>1</float></array>
  <array length="3"><float>1</float><float>1</float><float>1</float></array>
</array>
```


5.8.8 binary

Category: Shaders

Profile: CG, GLES2

5.8.8.1 General

Identifies or provides a shader in binary form.

5.8.8.2 Concepts

This is for platforms that may benefit from, or require, a precompiled shader, although the GLES2 profile is designed so that the source code should still be present so that the binaries can be regenerated.

The binary is typically created by offline compilers to work with the GLES2 API function `glShaderBinary` or other binary extensions. The **<binary>** element occurs at one or more levels inside the **<program>** element where a profile for GLES2 describes the shader information, compiler settings, and linker settings so that the binary can be regenerated.

5.8.8.3 Attributes

The **<binary>** element has no attributes.

5.8.8.4 Related elements

The **<binary>** element relates to the following elements:

Parent elements	compiler, linker
Child elements	See the following subclause.
Other	None

5.8.8.5 Child elements

Exactly one of the following child elements shall occur:

Name/example	Description	Default	Occurrences
<ref>	Contains the URI (<code>xs:anyURI</code>) of a file that contains the binary. This element has no attributes.	None	1
<hex format="">	Contains the binary code as a sequence of hexadecimal-encoded binary octets. The optional format attribute specifies an <code>xs:token</code> that describes the format of the binary; typically, this value is the appropriate file extension.	None	1

5.8.8.6 Details

The binary can either come from an external file via **<ref>** or be embedded in the instance document using **<hex>**. **<ref>** requires the use of a file extension or embedded information to indicate the formatting of the data. **<hex>**, on the other hand, uses the format attribute to convey additional formatting information because no file extension is available.

5.8.8.7 Example

```
<binary><ref>file://c:/test/vertexShader.bin</ref></binary>
<binary><hex format="COMPANY_PLATFORM">0123456789ABCDEF</hex></binary>
```

5.8.9 bind (FX)

Category: Materials

Profile: **External**

5.8.9.1 General

Binds values to uniform inputs of a shader or binds values to effect parameters upon instantiation.

5.8.9.2 Concepts

Shaders with uniform parameters can have values bound to their inputs at compile time, and need values assigned to the uniform parameters at execution time. These values can be literal values, constant parameters or uniform parameters. In the case of constant values, these declarations of parameters for the shader can be used by the compiler to produce optimized shaders for that specific declaration.

<bind> is also used to map predefined parameters to uniform inputs at run time, allowing the FX Runtime to automatically assign values to a shader from its pool of predefined parameters.

5.8.9.3 Attributes

The **<bind>** element has the following attributes:

semantic	xs:NCName	Which effect parameter to bind. Required.
target	sidref_type	A reference to the SID of the value to bind to the specified semantic. This text string is a path-name following a simple syntax described in the "Addressing Syntax" in 5.3.3. Required.

5.8.9.4 Related elements

The **<bind>** element relates to the following elements:

Parent elements	instance_material (geometry), instance_material (rendering)
Child elements	None
Other	None

5.8.9.5 Details

Some FX Runtime compilers require that every uniform input is bound before compilation can happen, while other FX Runtimes can “semicompile” shaders into nonexecutable object code that can be inspected for unbound inputs.

The **<bind>** and **<bind_vertex_input>** elements bind the target to a parameter in an **<effect>**. The search string that identifies the parameter in the **<effect>** is specified by the semantic attribute. When locating the parameter in the **<effect>**, search in the following order:

- Find a COLLADA FX parameter by semantic;
- If the profile contains shading language code, find a parameter within the shader by semantic;
- Find a COLLADA FX parameter by SID;
- If the profile contains shading language code, find a parameter within the shader by name.

5.8.9.6 Example

```
<instance_material symbol="RedMat" target="#RedCGEffect">
  <bind semantic="LIGHTPOS0" target="LightNode/translate"/>
</instance_material>
```

5.8.10 bind_attribute

Category: Shaders

Profile: GLES2, GLSL

5.8.10.1 General

Binds semantics to vertex attribute inputs of a shader.

5.8.10.2 Concepts

Shaders with vertex attribute variables might not use variable names that match up to semantic names of the geometry vertex inputs. This element allows users to add an alternative semantic name to the shader's vertex attribute so that it can be more easily identified and attached to geometry by the runtime system.

5.8.10.3 Attributes

The **<bind_attribute>** element has the following attributes:

symbol	xs:token	The identifier for a vertex attribute variable in the shader (a formal function parameter or in-scope global). Required.

5.8.10.4 Related elements

The **<bind_attribute>** element relates to the following elements:

Parent elements	program
Child elements	semantic
Other	None

5.8.10.5 Child elements

Name/example	Description	Default	Occurrences
<semantic>	An <code>xs:token</code> containing an alternative name to the attribute variable for semantic binding to geometry vertex inputs. See main entry.	None	1

5.8.10.6 Details

If no **<bind_attribute>** exists for a particular variable, the FX runtime shall attempt to bind to the attribute's name directly.

5.8.10.7 Example

```
<program>
  <shader stage="VERTEX"/>
  ...
  <bind_attribute symbol="pos">
    <semantic> POSITION </semantic>
  </bind_attribute>
  <bind_attribute symbol="diffusecol">
    <semantic> COLOR </semantic>
  </bind_attribute>
</program>
```

5.8.11 bind_material

Category: Materials

Profile: **External**

5.8.11.1 General

Binds a specific material to a piece of geometry, binding varying and uniform parameters at the same time.

5.8.11.2 Concepts

When a piece of geometry is declared, it can request that the geometry have a particular material, for example,

```
<polygons name="leftarm" count="2445" material="bluePaint">
```

This abstract symbol needs to be bound to a particular material instance. The application does the instantiation when processing the **<instance_geometry>** elements within the **<bind_material>** elements. The application scans the geometry for material attributes and binds actual material objects to them as indicated by the **<instance_material>** (geometry) symbol attributes. See “Example” in 5.8.11.7.

While a material is bound, shader parameters might also need to be resolved. For example, if an effect requires two light source positions as inputs but the scene contains eight unique light sources, which two light sources will be used on the material? If an effect requires one set of texture coordinates on an object, but the geometry defined two sets of texcoords, which set will be used for this effect? **<bind_material>** is the mechanism for disambiguating inputs in the scene graph.

Inputs are bound to the scene graph by naming the semantic attached to the parameters and connecting them by COLLADA URL syntax to individual elements of nodes in the scene graph, right down to the individual elements of vectors.

5.8.11.3 Attributes

The **<bind_material>** element has no attributes.

5.8.11.4 Related elements

The **<bind_material>** element relates to the following elements:

Parent elements	instance_geometry, instance_controller
Child elements	See the following subclause.
Other	None

5.8.11.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<param> (Core)	In <bind_material> these are added to be targets for animation. These objects can then be bound to input parameters in the normal manner without requiring the animation targeting system to parse the internal layout of an <effect> . See main entry in Core.	None	0 or more
<technique_common>	Specifies material binding information for the common profile that all COLLADA implementations shall support. See “The common profile” in 5.3.5 for usage information and the following subclause for child element details.	N/A	1
<technique> (Core)	Each <technique> specifies material binding information for a specific profile as designated by the <technique> ’s profile attribute. See main entry in Core.	None	0 or more
<extra>	See main entry in Core.	N/A	0 or more

5.8.11.6 Child elements for <bind_material> / <technique_common>

Name/example	Description	Default	Occurrences
<instance_material> (geometry)	See main entry.	N/A	1 or more

5.8.11.7 Example

```

<instance_geometry url="#BeechTree">
  <bind_material>
    <param sid="windAmount" semantic="WINDSPEED" type="float3_type"/>
    <technique_common>
      <instance_material symbol="leaf" target="MidsummerLeaf01"/>
      <instance_material symbol="RedMat" target="beechBark">
        <bind semantic="LIGHTPOS0" target="LightNode/translate"/>
        <bind semantic="TEXCOORD0" target="BeechTree/texcoord2"/>
      </instance_material>
    </technique_common>
  </bind_material>
</instance_geometry>

```

The following example shows <bind_material> binding a material with a geometry. The connection between the <material> id attribute and the material attribute of a <polygons> element is established by the <instance_material> (materials) element:

```

...
<material id="MyMaterial"> ... </material>
...
<geometry>
  ...
  <polygons name="leftarm" count="2445" material="bluePaint">
    ...
  </polygons>
</geometry>
...
<scene>
  ...
  <instance_geometry ...>
    <bind_material>
      <technique_common>
        <instance_material symbol="bluePaint" target="MyMaterial">
          ...
        </instance_material>
      </technique_common>
    </bind_material>
  </instance_geometry>
  ...
</scene>

```

5.8.12 bind_uniform

Category: Shaders

Profile: CG, GLES2, GLSL

5.8.12.1 General

Binds values to uniform inputs of a shader or binds values to effect parameters upon instantiation.

5.8.12.2 Concepts

Shaders with uniform parameters can have values bound to their inputs at compile time, and need values assigned to the uniform parameters at execution time. These values can be literal values, constant parameters or uniform parameters. In the case of constant values, these declarations of parameters for the shader can be used by the compiler to produce optimized shaders for that specific declaration.

<bind_uniform> is also used to map predefined parameters to uniform inputs at run time, allowing the FX Runtime to automatically assign values to a shader from its pool of predefined parameters.

5.8.12.3 Attributes

The **<bind_uniform>** element has the following attributes:

symbol	xs:NCName	The identifier for a uniform input parameter to the shader (a formal function parameter or in-scope global) that will be bound to an external resource. Required.

5.8.12.4 Related elements

The **<bind_uniform>** element relates to the following elements:

Parent elements	shader (in CG), program (in GLSL2 and GLSL)
Child elements	See the following subclause.
Other	None

5.8.12.5 Child elements

Exactly one of the child elements **<param>** or a value type shall appear. They are mutually exclusive.

Name/example	Description	Default	Occurrences
<param> (reference)	See main entry.	None	See "Note"
parameter_type_element	See "Parameter-Type Elements" in 5.11.3 for parameter-type elements valid in CG, GLSL2, or GLSL scope, depending on context: • CG: cg_param_group • GLSL: glsl_value_group • GLSL2: gles2_value_group	N/A	See "Note"

5.8.12.6 Details

Some FX Runtime compilers require that every uniform input is bound before compilation can happen, while other FX Runtimes can "semicompile" shaders into nonexecutable object code that can be inspected for unbound inputs.

5.8.12.7 Example

```
<shader stage="VERTEX">
  <sources entry="main"><import ref="fooShader"/>
  <compiler platform="PC" target="GLSLF"/>
  <bind_uniform symbol="diffusecol">
    <float3> 0.30 .52 0.05 </float3>
  </bind_uniform>
  <bind_uniform symbol="lightpos">
    <param ref="OverheadLightPos_03">
  </bind_uniform>
</shader>
```

5.8.13 bind_vertex_input

Category: Materials

Profile: **External**

5.8.13.1 General

Binds geometry vertex inputs to effect vertex inputs upon instantiation.

5.8.13.2 Concepts

This element is useful, for example, in binding a vertex-program parameter to a **<Source>**. The vertex program needs data already gathered from sources. This data comes from the **<input>** elements under the collation elements such as **<polygons>** or **<triangles>**. Inputs access the data in **<Source>**s and guarantee that it corresponds with the polygon vertex “fetch”. To reference the **<input>**s for binding, use **<bind_vertex_input>**.

5.8.13.3 Attributes

The **<bind_vertex_input>** element has the following attributes:

semantic	xs:NCName	Which effect parameter to bind. Required.
input_semantic	xs:NCName	Which input semantic to bind. Required.
input_set	uint_type	Which input set to bind. Optional.

5.8.13.4 Related elements

The **<bind_vertex_input>** element relates to the following elements:

Parent elements	instance_material (geometry)
Child elements	None
Other	input

5.8.13.5 Details

The **<bind_vertex_input>** element binds geometry vertex streams (identified as **<input>** elements within geometry elements) to material effect vertex stream semantics. Although applications commonly perform automatic binding of vertex streams with identical semantic identifiers, there are frequently mismatches in a semantic identifier's meaning. Use **<bind_vertex_input>** to remove these ambiguities, which are most commonly caused by:

- Generalizations; for example, `TEXCOORD0` vs. `DIFFUSE-TEXCOORD`;
- Spelling differences; for example, `COLOR` vs. `COLOUR`;
- Abbreviations;
- Verbosity;
- Synonyms.

The **<bind>** and **<bind_vertex_input>** elements bind the target to a parameter in an **<effect>**. The search string that identifies the parameter in the **<effect>** is specified by the semantic attribute. When locating the parameter in the **<effect>**, search in the following order:

- Find a COLLADA FX parameter by semantic;
- If the profile contains shading language code, find a parameter within the shader by semantic;
- Find a COLLADA FX parameter by SID;
- If the profile contains shading language code, find a parameter within the shader by name.

5.8.13.6 Example

The following example applies a wet-feathers material to a duck model. The duck model may have normal map texture coordinates, which it calls **TEXCOORD0** (semantic=**TEXCOORD** and set=0), and base color texture coordinates, which it calls **TEXCOORD1**.

There are circumstances where semantic names for texture coordinates (or other geometry streams) do not match up. For example, the wet-feathers material may have normal map texture coordinates called **TEXCOORD1** and base color texture coordinates called **TEXCOORD0**. In this case, the meanings of these identical names have been swapped so, to bind these mismatched objects, swap them using **<bind_vertex_input>**.

Note that the semantic attribute refers to the semantic in the material effect while the attributes prefixed with `input_` refer to the geometry vertex **<bind_vertex_input>** streams, which are identified by the combination of a semantic name and a set number.

```
<instance_geometry url="#duck">
  <bind_material>
    <technique_common>
      <instance_material symbol="region1" target="#wet-feathers">
        <bind_vertex_input semantic="TEXCOORD1"
          input_semantic="TEXCOORD" input_set="0"/>
        <bind_vertex_input semantic="TEXCOORD0"
          input_semantic="TEXCOORD" input_set="1"/>
      </instance_material>
    </technique_common>
  </bind_material>
</instance_geometry>
```

```

    </instance_material>
  </technique_common>
</bind_material>
</instance_geometry>

```

5.8.14 blinn

Category: Rendering

Profile: **COMMON**

5.8.14.1 General

Produces a shaded surface with a Blinn BRDF approximation.

5.8.14.2 Concepts

Used inside a **<profile_COMMON>** effect, **<blinn>** declares a fixed-function pipeline that produces a shaded surface according to the Blinn-Torrance-Sparrow lighting model or a close approximation.

This formula is complex and detailed via the ACM, so it is not detailed here. Refer to [11].

5.8.14.2.1 Maximizing compatibility:

To maximize application compatibility, it is suggested that developers use the Blinn-Torrance-Sparrow for **<shininess>** in the range of 0 to 1. For **<shininess>** greater than 1.0, the COLLADA author was probably using an application that followed the Blinn-Phong lighting model, so it is recommended to support both Blinn formulas according to whichever range the shininess resides in.

5.8.14.2.2 The Blinn-Phong formula

The Blinn-Phong formula is:

$$color = \langle emission \rangle + \langle ambient \rangle * al + \langle diffuse \rangle * \max(N * L, 0) + (specular) * \max(H * N, 0)^{\langle shininess \rangle}$$

where:

al is a constant amount of ambient light contribution coming from the scene. In the COMMON profile, this is the sum of all the **<light><technique_common><ambient>** values in the **<visual_scene>**

- *N* is normal vector (normalized);
- *L* is light vector (normalized);
- *I* is eye vector (normalized);
- *H* is half-angle vector, calculated as halfway between the unit eye and light vectors, using the formula $H = \text{normalize}(I + L)$.

5.8.14.3 Attributes

The **<blinn>** element has no attributes.

5.8.14.4 Related elements

The **<blinn>** element relates to the following elements:

Parent elements	technique (FX) in profile_COMMON
Child elements	See the following subclause.
Other	None

5.8.14.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<emission>	Declares the amount of light emitted from the surface of this object. See fx_common_color_or_texture_type .	N/A	0 or 1
<ambient> (FX)	Declares the amount of ambient light emitted from the surface of this object. See fx_common_color_or_texture_type .	N/A	0 or 1
<diffuse>	Declares the amount of light diffusely reflected from the surface of this object. See fx_common_color_or_texture_type .	N/A	0 or 1
<specular>	Declares the color of light specular reflected from the surface of this object. See fx_common_color_or_texture_type .	N/A	0 or 1
<shininess>	Declares the specular or roughness of the specular reflection lobe. fx_common_float_or_param_type ; see main entry.	N/A	0 or 1
<reflective>	Declares the color of a perfect mirror reflection. See fx_common_color_or_texture_type .	N/A	0 or 1
<reflectivity>	Declares the amount of perfect mirror reflection to be added to the reflected light as a value between 0.0 and 1.0. fx_common_float_or_param_type ; see main entry.	N/A	0 or 1
<transparent>	Declares the color of perfectly refracted light. See fx_common_color_or_texture_type and “Determining transparency (opacity)” in 5.7 Getting started with COLLADA FX.	N/A	0 or 1
<transparency>	Declares the amount of perfectly refracted light added to the reflected color as a scalar value between 0.0 and 1.0. fx_common_float_or_param_type ; see main entry and “Determining transparency (opacity)” in 5.7 Getting started with COLLADA FX.	N/A	0 or 1
<index_of_refraction>	Declares the index of refraction for perfectly refracted light as a single scalar index. fx_common_float_or_param_type ; see main entry.	N/A	0 or 1

5.8.14.6 Example

This is an example of a dark red effect with a shiny, pinpoint specular highlight.

```
<library_effects>
  <effect id="blinn1-fx">
    <profile_COMMON>
      <technique sid="common">
        <blinn>
          <emission>
            <color>0 0 0 1.0</color>
          </emission>
          <ambient>
            <color>0 0 0 1.0</color>
          </ambient>
          <diffuse>
            <color>0.500000 0.002000 0 1.0</color>
          </diffuse>
          <specular>
            <color>0.500000 0.500000 0.500000 1.0</color>
          </specular>
          <shininess>
            <float>0.107420</float>
          </shininess>
          <reflective>
            <color>0 0 0 1.0</color>
          </reflective>
          <reflectivity>
            <float>0</float>
          </reflectivity>
          <transparent opaque="RGB_ZERO">
            <color>0 0 0 1.0</color>
          </transparent>
          <transparency>
            <float>1.000000</float>
          </transparency>
          <index_of_refraction>
            <float>0</float>
          </index_of_refraction>
        </blinn>
      </technique>
    </profile_COMMON>
  </effect>
```

5.8.15 code

Category: Shaders

Profile: CG, GLES2, GLSL

5.8.15.1 General

Provides an inline block of source code.

5.8.15.2 Concepts

Source code can be inlined into the **<effect>** declaration to be used to compile shaders.

5.8.15.3 Attributes

The **<code>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. An identifier for the source code to allow the block to be locally referenced by other elements. Optional. For details, see “Address syntax” in 5.3 Schema concepts.

5.8.15.4 Related elements

The **<code>** element relates to the following elements:

Parent elements	profile_CG, profile_GLSL, profile_GLES2
Child elements	None
Other	None

5.8.15.5 Details

Inlined source code, included as `xs:string`, shall escape all XML identifier characters, for example, converting “<” to “<”.

5.8.15.6 Example

```
<code sid="lighting_code">
matrix4x4 mat : MODELVIEWMATRIX;
float4 lighting_fn( varying float3 pos : POSITION,
    ...
</code>
```

5.8.16 color_clear

Category: Rendering

Profile: CG, GLES, GLES2, GLSL

5.8.16.1 General

Specifies whether a render target image is to be cleared, and which value to use.

5.8.16.2 Concepts

Before drawing, render target images may need to be reset to a blank canvas or default. The **<color_clear>** declarations specify which value to use. If no clearing statement is included, the target image is unchanged as rendering begins.

5.8.16.3 Attributes

The **<color_clear>** element has no attributes in GLES scope.

It has the following attribute in CG, GLES2, and GLSL scope:

index	xs:nonNegativeInteger	Which of the multiple render targets is being set. The default is 0. Optional.

5.8.16.4 Related elements

The **<color_clear>** element relates to the following elements:

Parent elements	evaluate
Child elements	None
Other	None

5.8.16.5 Details

This element contains four floating-point values representing the red, green, blue, and alpha channels.

When this element exists inside a pass, it is a cue to the runtime that a particular backbuffer or render-target resource shall be cleared. This means that all existing image data in the resource shall be replaced with the color provided. This element puts the resource into a fresh and known state so that other operations that use this resource execute as expected.

The index attribute identifies the API's render-target resource index that you want to clear. An index of 0 identifies the primary resource. The primary resource may be the backbuffer or the override provided with an appropriate **<*_target>** element (**<color_target>**, **<depth_target>**, or **<stencil_target>**).

Current platforms have fairly restrictive rules for setting up multiple render targets (MRTs). For example, MRTs on most Direct3D⁵ 9 class platforms can have only four color buffers, which shall all be the same size and pixel format, one depth buffer, and one stencil buffer active for all color buffers. The COLLADA FX declaration is designed to be looser in its restrictions, so an FX runtime shall validate that a particular MRT declaration in a **<pass>** is possible before attempting to apply it, and flag it as an error if it fails.

5.8.16.6 Example

```
<color_clear index="0">0.0 0.0 0.0 0.0</color_clear>
```

5.8.17 color_target

Category: Rendering

Profile: CG, GLES, GLES2, GLSL

5.8.17.1 General

Specifies which **<image>** will receive the color information from the output of this pass.

⁵ Direct3D is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of this product(s).

5.8.17.2 Concepts

Multiple Render Targets (MRTs) allow fragment shaders to output more than one value per pass, or to redirect the standard depth and stencil units to read from and write to arbitrary off-screen buffers. These elements tell the FX Runtime which previously defined images to use or from which parameter to fetch the image.

5.8.17.3 Attributes

The **<color_target>** element has no attributes in GLES scope.

It has the following attributes in CG, GLES2, and GLSL scope:

index	xs:nonNegativeInteger	Indexes one of the Multiple Render Targets. The default is 0. Optional.
slice	xs:nonNegativeInteger	Indexes a sub image inside a target <surface> , including a single MIP-map level, a unique cube face, or a layer of a 3D texture. The default is 0. Optional.
mip	xs:nonNegativeInteger	The default is 0. Optional.
face	Enumeration	Valid values are POSITIVE_X, NEGATIVE_X, POSITIVE_Y, NEGATIVE_Y, POSITIVE_Z, and NEGATIVE_Z. The default is POSITIVE_X. Optional.

5.8.17.4 Related elements

The **<color_target>** element relates to the following elements:

Parent elements	evaluate
Child elements	See the following subclause.
Other	newparam, image

5.8.17.5 Child elements

Exactly one of the child elements **<param>** or **<instance_image>** shall appear. They are mutually exclusive.

Name/example	Description	Default	Occurrences
<param> (reference)	References a sampler parameter to determine which image to use. See main entry.	None	0 or 1
<instance_image>	Directly instantiates a renderable image. See main entry.	None	0 or 1

5.8.17.6 Details

Current platforms have fairly restrictive rules for setting up MRTs; for example, most Direct3D® 9 class hardware supports only four color buffers, which shall be all of the same size and pixel format, and only one depth buffer and one stencil buffer active for all color buffers. The COLLADA FX declaration is designed to be looser in its restrictions, so an FX runtime shall validate that a particular MRT declaration in a **<pass>** is possible before attempting to apply it, and flag it as an error if it fails.

This element contains either a **<param>** that references a **<newparam>** containing a **<sampler_*>** or an **<instance_image>** to link directly to an image. If no **<color_target>** is specified, the FX runtime uses the default backbuffer set for its platform.

5.8.17.7 Example

```
<newparam sid="surfaceTex">
  <sampler2D><instance_image url="renderTarget1"/></sampler2D>
</newparam>
<technique>
  <pass>
    <evaluate>
      <color_target index="0">
        <param ref="surfaceTex"/>
      </color_target>
    </evaluate>
  </pass>
</technique>
```

5.8.18 compiler

Category: Shaders

Profile: CG, GLES2

5.8.18.1 General

Contains command-line or runtime-invocation options for a shader compiler.

5.8.18.2 Concepts

The shader compiler accepts shader program source code (see **<sources>**) and compiles it into machine-executable object code. The shader compiler accepts command-line options that configure it to perform specific operations.

5.8.18.3 Attributes

The **<compiler>** element has the following attributes.

platform	xs:string	Required. The subplatform name to distinguish between multiple compiler settings.
target	xs:string	Optional. Target binary profile. For example, arbvpl, arbfpl, glslv, glslf, hlslv, hlslf, vs_3_0, ps_3_0.
options	xs:string	Optional. Compiler options.

5.8.18.4 Related elements

The **<compiler>** element relates to the following elements:

Parent elements	shader
Child elements	See the following subclause.
Other	sources, linker

5.8.18.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<binary>	See main entry.	None	0 or 1

5.8.18.6 Details

This element contains text that is the compilation options given to the tool as a text string. It can optionally contain a binary representation of the compiled results.

5.8.18.7 Example

```
<compiler platform="PC" target="arbvpl" options="-debug"/>
```

5.8.19 constant (FX)

Category: Rendering

Profile: **COMMON**

5.8.19.1 General

Produces a constantly shaded surface that is independent of lighting.

NOTE For the <constant> related to texture combiners, see <texenv> and <texcombiner>.

5.8.19.2 Concepts

Used inside a <profile_COMMON> effect, declares a fixed-function pipeline that produces a constantly shaded surface that is independent of lighting.

The reflected color is calculated as:

$$color = \langle emission \rangle + \langle ambient \rangle * al$$

where:

al is A constant amount of ambient light contribution coming from the scene. In the COMMON profile, this is the sum of all the <light><technique_common><ambient><color> values in the <visual_scene>.

5.8.19.3 Attributes

The <constant> element has no attributes.

5.8.19.4 Related elements

The **<constant>** element relates to the following elements:

Parent elements	technique (FX) in profile_COMMON
Child elements	See the following subclause.
Other	None

5.8.19.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<emission>	Declares the amount of light emitted from the surface of this object. See fx_common_color_or_texture_type .	N/A	0 or 1
<reflective>	Declares the color of a perfect mirror reflection. See fx_common_color_or_texture_type .	N/A	0 or 1
<reflectivity>	Declares the amount of perfect mirror reflection to be added to the reflected light as a value between 0.0 and 1.0. See fx_common_float_or_param_type .	N/A	0 or 1
<transparent>	Declares the color of perfectly refracted light. See fx_common_color_or_texture_type and "Determining transparency (opacity)" in 5.7 Getting started with COLLADA FX.	N/A	0 or 1
<transparency>	Declares the amount of perfectly refracted light added to the reflected color as a scalar value between 0.0 and 1.0. See fx_common_float_or_param_type and "Determining transparency (opacity)" in 5.7 Getting started with COLLADA FX.	N/A	0 or 1
<index_of_refraction>	Declares the index of refraction for perfectly refracted light as a single scalar index. See fx_common_float_or_param_type .	N/A	0 or 1

5.8.19.6 Example

The following example shows a cube that is fully transparent (invisible):

```
<effect id="surfaceShader1-fx">
  <profile_COMMON>
    <technique sid="common">
      <constant>
        <emission>
          <color>0 0 0 1.0</color>
        </emission>
        <reflective>
          <color>1.000000 1.000000 1.000000 1.0</color>
        </reflective>
        <reflectivity>
```

```

        <float>0.000000</float>
    </reflectivity>
    <transparent opaque="RGB_ZERO">
        <color>1.000000 1.000000 1.000000 1.0</color>
    </transparent>
    <transparency>
        <float>1.000000</float>
    </transparency>
    <index_of_refraction>
        <float>0</float>
    </index_of_refraction>
</constant>
</technique>
</profile_COMMON>
</effect>

```

In the preceding example, to change the cube to opaque black from transparent, change this:

```
<transparent opaque="RGB_ZERO"
```

to this:

```
<transparent opaque="A_ONE"
```

The following example simply sets the constant to red:

```

<profile_COMMON>
  <technique sid="T1">
    <constant>
      <emission><color>1.0 0.0 0.0 1.0</color></emission>
    </constant>
  </technique>
</profile_COMMON>

```

This example takes the color from a parameter:

```

<profile_COMMON>
  <newparam sid="myColor">
    <float4> 0.2 0.56 0.35 1</float4>
  </newparam>
  <technique sid="T1">
    <constant>
      <emission><param ref="myColor"/></emission>
    </constant>
  </technique>
</profile_COMMON>

```

NOTE Rasterizer and ray tracers can produce vastly different results depending on how or whether they choose to support features such as refraction.

5.8.20 create_2d

Category: Texturing

Profile: **External**

5.8.20.1 General

Assists in the manual creation of a 2D **<image>** asset.

5.8.20.2 Concepts

Users can define the image dimensions and structure before filling it in. This element is often used for render targets since they are typically not initialized with data. This element provides far more control over image creation than `<init_from>`, although it is not necessarily a more desirable approach. It describes the 2D structure that the user would like to generate and then describes the data that shall be loaded into each portion of that structure.

5.8.20.3 Attributes

The `<create_2d>` element has no attributes.

5.8.20.4 Related elements

The `<create_2d>` element relates to the following elements:

Parent elements	image
Child elements	See the following subclause.
Other	None

5.8.20.5 Child elements

Child elements shall be in the following order if specified:

Name/example	Description	Default	Occurrences
<code><size_exact width="512" height="512"></code>	Specifies that the surface shall be sized to these exact dimensions. The two <code>xs:unsignedInt</code> attributes are required. Either <code><size_exact></code> or <code><size_ratio></code> , but not both, shall be specified.	None	0 or 1
<code><size_ratio width="1.0" height="1.0"></code>	Specifies that the image size shall be relative to the size of the viewport. For example, 1,1 is the same size as the viewport; 0.5,0.5 is 1/4 the size of the viewport and half as long in either direction. The two <code>float_type</code> attributes are required. Either <code><size_exact></code> or <code><size_ratio></code> , but not both, shall be specified.	None	0 or 1
<code><mips levels="8" auto_generate ="true"></code>	MIP information. Either <code><mips></code> or <code><unnormalized></code> , but not both, shall be specified. Both arguments are required: • The levels attribute is an <code>xs:unsignedInt</code> , where 1 is no MIPs and 0 is maximum levels, defined as the following in both OpenGL and DirectX: $1 + \text{floor}(\log_2(\max(w, h, d)))$ • Use <code>auto_generate</code> attribute to initialize higher MIP levels that shall be automatically generated by the application or its graphics API.	None	0 or 1
<code><unnormalized></code>	Unnormalized addressing of texels. (0-W, 0-H). Either <code><mips></code> or <code><unnormalized></code> , but not both, shall be specified because the addressing is not uniform per level. This is equivalent to OpenGL textureRECT extension. This element has no attributes. There is no equivalent feature in DirectX.	None	0 or 1

Name/example	Description	Default	Occurrences
<array length="32">	Specifies the length of the 2D array. The length attribute is a required <code>xs:positiveInteger</code> .	None	0 or 1
<format>	Specifies an image's pixel or compression format. If not present, the format is assumed to be R8G8B8A8 linear. See main entry.	None	0 or 1
<init_from>	Specifies which 2D image to initialize and which MIP level to initialize. See main entry.	None	0 or more

5.8.20.6 Details

<create_2d> allows the custom initialization of a 2D texture. Initializes a custom 2D image by specifying its size, viewport ratio, MIP levels, normalization, pixel format, and data sources. It also supports arrays of 2D images.

A 2D image's dimensions are specified by either the **<size_exact>** or **<size_ratio>** elements. One or the other shall be specified.

The 2D image type is created by taking into consideration whether either **<unnormalized>** or an **<array>** element exists. If neither exists, it is a regular 2D image where the sampling coordinates are normalized, and it is not an array.

One or more **<init_from>**s initialize each portion of the image, although this is not necessary and the image might remain empty to allow other operations, such as render, to provide the data.

5.8.20.7 Example

The following example shows the custom initialization of a 2D image from a source file. The initialization includes the specification of 6 MIP levels and that the application shall generate them.

```
<library_images>
  <image name="Noise">
    <create_2d>
      <size_exact width="128" height="128"/>
      <mips_levels="6" auto_generate="true"/>
      <format>
        <exact>R8G8B8A8</exact>
      </format>
      <init_from>
        <ref>./images/Noise2D.tga</ref>
      </init_from>
    </create_2d>
  </image>
</library_images>
```

5.8.21 create_3d

Category: Texturing

Profile: **External**

5.8.21.1 General

Assists in the manual creation of a 3D **<image>** asset.

5.8.21.2 Concepts

Users can define the image dimensions and structure before filling it in. This element provides far more control over image creation than **<init_from>**, although it is not necessarily a more desirable approach. It describes the 3D structure that the user would like to generate and then describes the data that shall be loaded into each portion of that structure.

5.8.21.3 Attributes

The **<create_3d>** element has no attributes.

5.8.21.4 Related elements

The **<create_3d>** element relates to the following elements:

Parent elements	image
Child elements	See the following subclause.
Other	None

5.8.21.5 Child elements

Name/example	Description	Default	Occurrences
<size width="256" height="256" depth="256">	Specifies that the surface shall be sized to these exact dimensions. The three <code>xs:unsignedInt</code> attributes are required.	None	1
<mips levels="7" auto_generate ="true">	MIP information. Both attributes are required: - The <code>levels</code> attribute is an <code>xs:unsignedInt</code>, where 1 is no MIPS and 0 is maximum levels, which is defined as the following in both OpenGL and DirectX: $1 + \text{floor}(\log_2(\max(w, h, d)))$ - Use <code>auto_generate</code> attribute to initialize higher MIP levels that shall be automatically generated by the application or its graphics API.	None	1
<array length="8">	Specifies the length of the 3D array. The length attribute is a required <code>xs:positiveInteger</code> . NOTE Currently, few APIs support 3D arrays.	None	0 or 1
<format>	Specifies an image's pixel or compression format. If not present, the format is assumed to be R8G8B8A8 linear. See main entry.	None	0 or 1
<init_from>	Specifies which 3D image to initialize and which MIP level to initialize. See main entry.	None	0 or more

5.8.21.6 Details

Initializes a custom 3D image (a volumetric image) by specifying its size, MIP level, pixel format, and data sources. It also supports arrays of 3D images.

A 3D image's dimensions are specified by the **<size>** element.

The 3D image type is created by taking into consideration whether an **<array>** element exists. If not, it is a regular 3D image that is not an array.

One or more **<init_from>**s initialize each portion of the image, although this is not necessary and the image might remain empty to allow other operations, such as render, to provide the data.

5.8.21.7 Example

The following examples shows the initialization of a 3D noise cube by loading separate 2D images into the depth slices of the 3D image. The format specifies an 8-bit source.

```
<library_images>
  <image name="Noise3D">
    <create_3d>
      <size width="128" height="128" depth="128"/>
      <format>
        <exact>A8</exact>
      </format>
      <init_from depth="0">
        <ref>./images/Noise2D_slice0.tga</ref>
      </init_from>
      <init_from depth="1">
        <ref>./images/Noise2D_slice1.tga</ref>
      </init_from>
      <init_from depth="2">
        <ref>./images/Noise2D_slice2.tga</ref>
      </init_from>
      ...
      <init_from depth="127">
        <ref>./images/Noise2D_slice127.tga</ref>
      </init_from>
    </create_3d>
  </image>
</library_images>
```

5.8.22 create_cube

Category: Texturing

Profile: **External**

5.8.22.1 General

Initializes a cube **<image>** asset.

5.8.22.2 Concepts

Users can define the image dimensions and structure before filling it in. Initializes the six faces of a cube by specifying its size, MIP level, pixel format, and data sources. It also supports arrays of images on each of the cube faces.

This element is to assist in the creation of a cube-shaped **<image>** asset. This provides far more control over image creation than **<init_from>**, although it is not necessarily a more desirable approach. It describes the cube-shaped structure that the user would like to generate and then describes the data to load into each portion of that structure.

5.8.22.3 Attributes

The **<create_cube>** element has no attributes.

5.8.22.4 Related elements

The **<create_cube>** element relates to the following elements:

Parent elements	image
Child elements	See the following subclause.
Other	None

5.8.22.5 Child elements

Name/example	Description	Default	Occurrences
<size width="256">	Specifies that the cube surfaces shall be sized to these exact dimensions. The <code>xs:unsignedInt</code> width attribute is required.	None	1
<mips levels="7" auto_generate ="true">	MIP information. Both attributes are required: - The levels attribute is an <code>xs:unsignedInt</code>, where 1 is no MIPS and 0 is maximum levels, which is defined as the following in both OpenGL and DirectX: $1 + \text{floor}(\log_2(\max(w, h, d)))$ - Use <code>auto_generate</code> attribute to initialize higher MIP levels that shall be automatically generated by the application or its graphics API.	None	1
<array length=16>	Specifies the length of the cube array. The required length attribute is an <code>xs:positiveInteger</code> . NOTE Currently, few APIs support 3D arrays.	None	0 or 1
<format>	Specifies an image's pixel or compression format. If not present, the format is assumed to be R8G8B8A8 linear. See main entry.	None	0 or 1
<init_from>	Specifies which cube image to initialize, which MIP level to initialize, and which cube face within the MIP that is to be initialized. See main entry.	N/A	0 or more

5.8.22.6 Details

A cube image's dimensions are specified by the **<size>** element.

The cube image type is created by taking into consideration whether an **<array>** element exists. If not, it is a regular cube image that is not an array.

One or more **<init_from>**s initialize each portion of the image, although this is not necessary and the image might remain empty to allow other operations, such as render, to provide the data.

5.8.22.7 Example

The following example shows the initialization of the six faces of a cube map:

```
<library_images>
  <image name="SkyCube">
    <create_cube>
      <size width="128"/>
      <mips levels="0" auto_generate="false"/>
      <format>
        <exact>R8G8B8A8</exact>
      </format>
      <init_from face="POSITIVE_X">
        <ref>./images/sky_x_pos.tga</ref>
      </init_from>
      <init_from face="NEGATIVE_X">
        <ref>./images/sky_x_neg.tga</ref>
      </init_from>
      <init_from face="POSITIVE_Y">
        <ref>./images/sky_y_pos.tga</ref>
      </init_from>
      <init_from face="NEGATIVE_Y">
        <ref>./images/sky_y_neg.tga</ref>
      </init_from>
      <init_from face="POSITIVE_Z">
        <ref>./images/sky_z_pos.tga</ref>
      </init_from>
      <init_from face="NEGATIVE_Z">
        <ref>./images/sky_z_neg.tga</ref>
      </init_from>
    </create_cube>
  </image>
</library_images>
```

5.8.23 depth_clear

Category: Rendering

Profile: CG, GLES, GLES2, GLSL

5.8.23.1 General

Specifies whether a render target image is to be cleared, and which value to use.

5.8.23.2 Concepts

Before drawing, render target images may need resetting to a blank canvas or to a default. These **<depth_clear>** declarations specify which value to use. If no clearing statement is included, the target image is unchanged as rendering begins.

5.8.23.3 Attributes

The **<depth_clear>** element has no attributes in GLES scope.

It has the following attribute in CG, GLES2, and GLSL scope:

index	xs:nonNegativeInteger	Which of the multiple render targets (MRTs) is being set. The default is 0. Optional.

5.8.23.4 Related elements

The **<depth_clear>** element relates to the following elements:

Parent elements	evaluate
Child elements	None
Other	None

5.8.23.5 Details

This element contains a single floating-point value that is used to clear a resource.

When this element exists inside a pass, it is a cue to the runtime that a particular backbuffer or render-target resource shall be cleared. This means that all existing image data in the resource shall be replaced with the floating-point value provided. This puts the resource into a fresh and known state so that other operations with this resource execute as expected.

The index attribute identifies the resource that you want to clear. An index of 0 identifies the primary resource. The primary resource may be the backbuffer or the override provided with an appropriate **<*_target>** element (**<color_target>**, **<depth_target>**, or **<stencil_target>**)

Direct3D® 9 class platforms have fairly restrictive rules for setting up MRTs; for example, MRTs can have only four color buffers, which shall be all of the same size and pixel format, and only one depth buffer and one stencil buffer active for all color buffers. The COLLADA FX declaration is designed to be looser in its restrictions, so an FX runtime shall validate that a particular MRT declaration in a **<evaluate>** is possible before attempting to apply it, and flag it as an error if it fails.

5.8.23.6 Example

```
<depth_clear index="0">0.0</depth_clear>
```

5.8.24 depth_target

Category: Rendering

Profile: CG, GLES, GLES2, GLSL

5.8.24.1 General

Specifies which **<image>** will receive the depth information from the output of this pass.

5.8.24.2 Concepts

Multiple Render Targets (MRTs) allow fragment shaders to output more than one value per pass, or to redirect the standard depth and stencil units to read from and write to arbitrary off-screen buffers. These elements tell the FX Runtime which previously defined image to use or which parameter to use to locate the image.

5.8.24.3 Attributes

The **<depth_target>** element has no attributes in **GL ES** scope.

It has the following attributes in **CG**, **GL ES2**, and **GLSL** scope:

index	xs:nonNegativeInteger	Indexes one of the Multiple Render Targets (MRTs). The default is 0. Optional.
slice	xs:nonNegativeInteger	Indexes a subimage inside a target <surface> , including a single MIP-map level, a unique cube face, or layer of a 3D texture. The default is 0. Optional.
mip	xs:nonNegativeInteger	The default is 0. Optional.
face	Enumeration	Valid values are <code>POSITIVE_X</code> , <code>NEGATIVE_X</code> , <code>POSITIVE_Y</code> , <code>NEGATIVE_Y</code> , <code>POSITIVE_Z</code> , and <code>NEGATIVE_Z</code> . The default is <code>POSITIVE_X</code> . Optional.

5.8.24.4 Related elements

The **<depth_target>** element relates to the following elements:

Parent elements	evaluate
Child elements	See the following subclause
Other	None

5.8.24.5 Child elements

Exactly one of the child elements **<param>** or **<instance_image>** shall appear. They are mutually exclusive.

Name/example	Description	Default	Occurrences
<param> (reference)	References a sampler parameter to determine which image to use. See main entry.	None	0 or 1
<instance_image>	Directly instantiates a renderable image. See main entry.	None	0 or 1

5.8.24.6 Details

Current platforms have fairly restrictive rules for setting up MRTs; for example, only four color buffers, which shall be all of the same size and pixel format, and only one depth buffer and one stencil buffer active for all color buffers. The COLLADA FX declaration is designed to be looser in its restrictions, so an FX runtime shall validate that a particular MRT declaration in a **<pass>** is possible before attempting to apply it, and flag it as an error if it fails.

If no **<depth_target>** is specified, the FX Runtime uses the default depthbuffer set for its platform.

5.8.24.7 Example

```
<newparam sid="surfaceTex">
  <sampler2D><instance_image url="renderTarget1"/></sampler2D>
</newparam>
<technique>
  <pass>
    <evaluate>
      <depth_target>
        <param ref="surfaceTex"/>
      </depth_target>
    </evaluate>
    <depth_clear>0.0</depth_clear>
  </pass>
</technique>
```

5.8.25 draw

Category: Rendering

Profile: CG, GLES, GLES2, GLSL

5.8.25.1 General

Instructs the FX Runtime what kind of geometry to submit.

5.8.25.2 Concepts

When executing multipass techniques, each pass may require different types of geometry to be submitted. One pass may require a model to be submitted, another pass may need a full screen quad to exercise a fragment shader over each pixel in an off-screen buffer, while another pass may need only front-facing polygons. **<draw>** declares a user-defined string that can be used as a semantic describing to the FX Runtime what geometry is expected for this pass.

5.8.25.3 Attributes

The **<draw>** element has no attributes.

5.8.25.4 Related elements

The **<draw>** element relates to the following elements:

Parent elements	evaluate
Child elements	None
Other	None

5.8.25.5 Details

The **<draw>** element contains an `xs:string`. The following list includes common strings to use in **<draw>**:

- **GEOMETRY**: The geometry associated with this **<instance_geometry>** or **<instance_material>** (geometry).

- **SCENE_GEOMETRY:** Draw the entire scene's geometry but with this effect, not the effects or materials already associated with the geometry. This is for techniques such as shadow-buffer generation, where you might be interested only in extracting the Z value from the light. This is without regard to ordering on the assumption that Z-buffer handles order.
- **SCENE_IMAGE:** Draw the entire scene into my targets. Use the appropriate effects or materials for each object. This is for effects that need an accurate image of the scene to work on for effects such as postprocessing blurs. This is without regard to ordering on the assumption that depth-buffer handles order.
- **FULL_SCREEN_QUAD:** Positions are 0,0 to 1,1 and the UVs match.
- **FULL_SCREEN_QUAD_PLUS_HALF_PIXEL:** Positions are 0,0 to 1,1 and the UVs are off by plus $\frac{1}{2}$ of a pixel's UV size.

5.8.25.6 Example

```
<!-- Draw the scene to the MyRenderTarget image -->
<pass>
  <evaluate>
    <color_target><param ref="MyRenderTarget" /></color_target>
    <draw>SCENE_IMAGE</draw>
  </evaluate>
</pass>
<!-- The engine shall submit a screen-aligned quad despite whatever geometry
the material might be attached to... draw it to the screen-->
<pass>
  <evaluate>
    <draw>FULL_SCREEN_QUAD</draw>
  </evaluate>
</pass>
<!-- Draw the geometry that the material is attached to ... draw it to the
screen -->
<pass>
  <evaluate>
    <draw>GEOMETRY</draw>
  </evaluate>
</pass>
```

5.8.26 effect

Category: Effects

Profile: **Effect**

5.8.26.1 General

Provides a self-contained description of a COLLADA effect.

5.8.26.2 Concepts

An effect defines the formulas necessary for the visual appearance of geometry and screen-space image processing.

Programmable pipelines allow stages of the 3D pipeline to be programmed using high-level languages. These shaders often require very specific data to be passed to them and require the rest of the 3D pipeline to be set up in a particular way in order to function. Shader Effects is a way of describing not only shaders, but also the environment in which they will execute. The environment requires description of images, samplers, shaders, input and output parameters, uniform parameters, and render-state settings.

Additionally, some algorithms require several passes to render the effect. This is supported by breaking pipeline descriptions into an ordered collection of **<pass>** objects. These are grouped into **<technique>**s that describe one of several ways of generating an effect.

Elements inside the **<effect>** declaration assume the use of an underlying library of code that handles the creation, use, and management of shaders, source code, parameters, etc. It shall be referred to this underlying library as the “FX Runtime”.

Parameters declared inside the **<effect>** element but outside of any **<profile_*>** element are said to be in “**<effect>** scope”. Parameters inside **<effect>** scope can be drawn only from a constrained list of basic data types and, after declaration, are available to **<shader>**s and declarations across all profiles. **<effect>** scope provides a handy way to parameterize many profiles and techniques with a single parameter.

5.8.26.3 Attributes

The **<effect>** element has the following attributes:

id	xs:ID	Global identifier for this object. Required.
name	xs:token	Pretty-print name for this effect. Optional.

5.8.26.4 Related elements

The **<effect>** element relates to the following elements:

Parent elements	library_effects
Child elements	See the following subclause.
Other	instance_effect

5.8.26.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<annotate>	See main entry.	N/A	0 or more
newparam	See main entry.	N/A	0 or more
profile	At least one profile shall appear, but any number of any of the following profiles can be included: • <profile_BRIDGE> • <profile_CG> • <profile_GLES> • <profile_GLES2> • <profile_GLSL> • <profile_COMMON> See main entries.	N/A	1 or more
<extra>	See main entry in Core.	N/A	0 or more

5.8.26.6 Example

```
<effect id="fx-simple-uid71243231231" name="simple_diffuse_with_one_light">
  <profile_CG>
    <!-- see profile_CG example -->
  </profile_CG>
</effect>
```

5.8.27 evaluate

Category: Rendering

Profile: CG, GLES, GLES2, GLSL

5.8.27.1 General

Contains evaluation elements for a rendering pass.

5.8.27.2 Concepts

This element contains actions related to drawing: what to draw and where to draw it to. Its children describe important details of how to use or invoke the pass and what data is required from the scene. This primarily includes command-oriented information that will result in data change, as opposed to other groupings that collect state information or shader information but do not directly manipulate drawable surfaces.

5.8.27.3 Attributes

The **<evaluate>** element has no attributes.

5.8.27.4 Related elements

The **<evaluate>** element relates to the following elements:

Parent elements	pass
Child elements	See the following subclauses.
Other	None

5.8.27.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<color_target>	See main entry.	None	0 or 1
<depth_target>	See main entry.	None	0 or 1
<stencil_target>	See main entry.	None	0 or 1
<color_clear>	See main entry.	None	0 or 1
<depth_clear>	See main entry.	None	0 or 1
<stencil_clear>	See main entry.	None	0 or 1
<draw>	See main entry.	None	0 or 1

5.8.27.6 Details

This element is primarily for organizational purposes. It logically groups elements that are required to evaluate or invoke a pass and separates them from other groups, such as API state information or shader program creation information.

5.8.27.7 Example

```
<newparam sid="renderTex">
  <sampler2D><instance image url="renderTarget1"/></sampler2D>
</newparam>
<technique>
  <pass>
    <states>
      ...
    </states>
    <program>
      ...
    </program>
    <evaluate>
      <color_target>
        <param ref="renderTex"/>
      </color_target>
      <draw>SCENE_GEOMETRY</draw>
    </evaluate>
  </pass>
</technique>
```


5.8.28 format

Category: Texturing

Profile: **External**

5.8.28.1 General

Describes the formatting or memory layout expected of an **<image>** asset.

5.8.28.2 Concepts

An **<image>** asset can lay out its color information in many different ways. The **format** element helps to describe how its texel information is encoded. This element supports an exact encoding and a generic hinting fallback mechanism.

5.8.28.3 Attributes

The **<format>** element has no attributes.

5.8.28.4 Related elements

The **<format>** element relates to the following elements:

Parent elements	create_2d , create_3d , create_cube
Child elements	See the following subclause.
Other	None

5.8.28.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<hint channels=... range=... precision=... space=... >	If this element or a higher precedence element is not present then use a common format R8G8B8A8 with linear color gradient, not sRGB. The element contains no data. The attributes are: • channels: Required enumeration. See “Details.” • range: Required enumeration. See “Details.” • precision: Optional enumeration. The default is DEFAULT. See “Details.” • space: Optional xs:token.	None	1
<exact>	Contains a string representing the profile- and platform-specific texel format that the author would like this surface to use. If this element is not specified, or if it is specified but the application cannot process the specified format, then the application uses the hint. This element has no attributes.	None	0 or 1

5.8.28.6 Details

Digital-content creation (DCC) tools will likely write either nothing or only **<hint>**.

Game-engine tools will likely add **<format>** for design cases such as DirectX 4CC texture codes, i.e. it is very specific to be exact, to minimize memory space, or to maximize performance or quality, e.g. DXT3 or DXT5, or other API-specific formatting codes such as GL_RGBA. More-generic codes are also possible. Direct3D® 10 provides very generic and well formulated enumerations that are recommended in combination without a prefix such as R8G8B8A8.

Applications creating the image shall use the following:

- If format does not exist, assume the common format R8G8B8A8 with linear color gradient, not sRGB;
- If **<exact>** exists and its string is understood, use it;
- Otherwise, if **<hint>** exists, use features and characteristics described there to select an appropriate format for your API.

The `channels` attribute describes the per-textel layout of the format. The length of the enumeration string indicates how many channels there are, and each letter represents the name of a channel. There are typically 1 to 4 channels. Valid enumeration values are:

- RGB – RGB color map;
- RGBA – RGB color with alpha map. Often used for color plus transparency or other things packed into channel A, such as specular power;
- RGBE – RGB color with shared exponent for HDR;
- L – Luminance map, often used for light mapping;
- LA – Luminance with alpha map, often used for light mapping;
- D – Depth map, often used for displacement, parallax, relief, or shadow mapping.

The `range` attribute describes the range of texel channel values. Each channel represents a range of values. Some example ranges are signed or unsigned integers, or are within a clamped range such as 0.0f to 1.0f, or are a high dynamic range via floating point. Valid enumeration values are:

- SNORM – Format represents a decimal value that remains within the -1 to 1 range. Implementation could be fixed-point or floating point;
- UNORM – Format represents a decimal value that remains within the 0 to 1 range. Implementation could be fixed point or floating point;
- SINT – Format represents signed integer numbers. For example, 8 bits is -128 to 127;
- UINT – Format represents unsigned integer numbers. For example, 8 bits is 0 to 255;
- FLOAT – Format shall support full floating-point ranges. High precision is expected to be 32 bits. Mid precision may be 16 to 32 bits. Low precision is expected to be 16 bits.

The **precision** attribute identifies the precision of the texel channel value. Each channel of the texel has a precision. Typically, channels have the same precision. An exact format may lower the precision of an individual channel but applying a higher precision by linking the channels may still convey the same information. Valid enumeration values are:

- **DEFAULT** – Designer does not care as long as it provides “reasonable” precision and performance;
- **LOW** – For integers, this typically represents 8 bits. For floating points, typically 16 bits;
- **MID** – For integers, this typically represents 8 to 24 bits. For floating points, typically 16 to 32 bits;
- **HIGH** – For integers, this typically represents 16 to 32 bits. For floating points, typically 24 to 32 bits;
- **MAX** – Typically 32 bits or 64 bits if available. 64 bits has been separated into its own category beyond HIGH because it typically has significant performance impact and is beyond what non-CAD software considers high precision.

5.8.28.7 Example

```
<library_images>
  <image name="Noise2D">
    <create_2d>
      <size_exact width="128" height="128"/>
      <mips_levels="0" auto_generate="true"/>
      <format>
        <hint channels="RGBA" range="UNORM" precision="LOW"/>
      </format>
      <init_from >
        <ref>./images/Noise2D.tga</ref>
      </init_from>
    </create_2d >
  </image>
  <image name="empty3D">
    <create_3d>
      <size_exact width="128" height="128" depth="128"/>
      <mips_levels="0" auto_generate="true"/>
      <format>
        <exact>R8G8B8A8</exact>
      </format>
    </create_3d>
  </image>
</library_images>
```

5.8.29 fx_common_color_or_texture_type

Category: Rendering

Profile: **COMMON**

5.8.29.1 General

A type that describes color attributes of fixed-function shader elements inside **<profile_COMMON>** effects.

5.8.29.2 Concepts

This type describes the attributes and related elements of the following elements.

- **<ambient>** (FX)
- **<diffuse>**
- **<emission>**
- **<reflective>**
- **<specular>**
- **<transparent>**

5.8.29.3 Attributes

Only **<transparent>** has an attribute; other elements of type **fx_common_color_or_texture_type** have no attributes.

opaque	Enumeration	Specifies from which channel to take transparency information. Optional. Valid values are: • A_ONE (the default): Takes the transparency information from the color's alpha channel, where the value 1.0 is opaque. • RGB_ZERO: Takes the transparency information from the color's red, green, and blue channels, where the value 0.0 is opaque, with each channel modulated independently. • A_ZERO: Takes the transparency information from the color's alpha channel, where the value 0.0 is opaque. • RGB_ONE: Takes the transparency information from the color's red, green, and blue channels, where the value 1.0 is opaque, with each channel modulated independently. For additional information, see "Determining transparency (opacity)" in 5.7 Getting started with COLLADA FX.

5.8.29.4 Related elements

Elements of type **fx_common_color_or_texture_type** relate to the following elements:

Parent elements	constant (FX), lambert , phong , blinn
Child elements	See the following subclause.
Other	None

5.8.29.5 Child elements

Exactly one of the child elements **<color>**, **<param>**, or **<texture>** shall appear. They are mutually exclusive.

Name/example	Description	Default	Occurrences
<color>	The value is a literal color, specified by four floating-point numbers in RGBA order. See main entry.	N/A	See “Note”
<param> (reference)	The value is specified by a reference to a previously defined parameter in the current scope that can be cast directly to a <float4> . See main entry.	N/A	See “Note”
<texture texture="myParam" texcoord="myUVs" <extra.../> </texture>	The value is specified by a reference to a previously defined <sampler2D> object. The texcoord attribute provides a semantic token, which will be referenced within <bind_material> to bind an array of texcoords from a <geometry> instance to the sampler. Both attributes are required and are of type xs:NCName . The <extra> child element can appear 0 or more times. See its main entry in Core.	N/A	See “Note”

5.8.29.6 Details

The schema does not specify default colors for **<ambient>**, **<diffuse>** and other child elements of the shaders **<blinn>**, **<constant>**, **<lambert>**, and **<phong>**. If any child element is unspecified, apply the specified shader formula without that portion. This provides equivalent results to explicitly specifying black for that child element. For example, the formula for **<phong>** without **<diffuse>** would be:

$$color = < emission > + < ambient > * a + < specular > * \max(R \bullet I, 0)^{< shininess >}$$

For a discussion on the behaviour of **<transparent>** or **<transparency>** in determining transparency, see “Determining transparency (opacity)” in the 5.7 Getting started with COLLADA FX.

5.8.30 fx_common_float_or_param_type

Category: Rendering

Profile: **COMMON**

5.8.30.1 General

A type that describes the scalar attributes of fixed-function shader elements inside **<profile_COMMON>** effects.

5.8.30.2 Concepts

This type describes the attributes and related elements of the following elements:

- **<index_of_refraction>**
- **<reflectivity>**

- **<shininess>**
- **<transparency>**

5.8.30.3 Attributes

Elements of type **fx_common_float_or_param_type** have no attributes.

5.8.30.4 Related elements

Elements of type **fx_common_float_or_param_type** relate to the following elements:

Parent elements	constant (FX), lambert, phong, blinn
Child elements	See the following subclause.
Other	None

5.8.30.5 Child elements

Exactly one of the child elements **<float>** or **<param>** shall appear. They are mutually exclusive.

Name/example	Description	Default	Occurrences
<float sid="...">	The value is represented by a literal floating-point scalar, for example: <float> 3.14 </float> The sid attribute is optional.	None	See "Note"
<param> (reference)	The value is represented by a reference to a previously defined parameter that can be directly cast to a floating-point scalar. See main entry.	None	See "Note"

5.8.30.6 Details

For a discussion on the behaviour of **<transparent>** and **<transparency>** in determining transparency, see "Determining transparency (opacity)" in the 5.7 Getting started with COLLADA FX.

5.8.31 fx_sampler_common

Category: Texturing

Profile: External, Effect, CG, COMMON, GLSL, GLES, GLES2

5.8.31.1 General

A type that describes the sampling states of the **<sampler*>** elements.

5.8.31.2 Concepts

This type describes the attributes and related elements of the following elements.

- `<sampler1D>`
- `<sampler2D>`
- `<sampler3D>`
- `<samplerCUBE>`
- `<samplerDEPTH>`
- `<samplerRECT>`
- `<sampler_states>`

The schema type that inherits from this provides the final details of how these states will be used for sampling.

5.8.31.3 Attributes

Elements of this type have no attributes.

5.8.31.4 Related elements

The `<sampler*>` elements relate to the following elements:

Parent elements	In core: <code>newparam</code> , <code>setparam</code> In FX: <code>array</code> .
Child elements	See the following subclauses.
Other	None

5.8.31.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><instance_image></code>	Instantiates a default image from which the sampler is to consume without material <code><setparam></code> . See main entry.	N/A	0 or 1
<code><texcoord semantic=... ></code>	Valid only in GLES <code><newparam>/<sampler2D></code> . Includes a semantic attribute that provides a semantic name for the texcoord channel that the texture unit shall use to read from in the mesh. The channel (array) is mapped here using <code><bind_material></code> . In shader-based programming, <code><texcoord></code> s can be calculated in the shader, but for fixed-function APIs such as OpenGL ES 1.x, the texture coordinates shall come parameterized with the mesh. This element contains no data.	None	0 or 1

Name/example	Description	Default	Occurrences
<code><instance_image></code>	Instantiates a default image from which the sampler is to consume without material <code><setparam></code> . See main entry.	N/A	0 or 1
<code><wrap_s></code>	Controls texture repeating and clamping of the S coordinate. Enumeration; see "Details."	WRAP	0 or 1
<code><wrap_t></code>	Controls texture repeating and clamping of the T coordinate. Enumeration; see "Details."	WRAP	0 or 1
<code><wrap_p></code>	Controls texture repeating and clamping of the P coordinate. Enumeration; see "Details." Not valid in GLES <code><sampler2D></code> .	WRAP	0 or 1
<code><minfilter></code>	Texture minimization. Enumerated type; see "Details." Applying a texture to a primitive implies a mapping from texture image space to framebuffer image space. In general, this mapping involves a reconstruction of the sampled texture image, followed by a homogeneous warping implied by the mapping to framebuffer space, then a filtering, followed finally by a resampling of the filtered, warped, reconstructed image before applying it to a fragment.	LINEAR	0 or 1
<code><magfilter></code>	Texture magnification. Enumerated type; see "Details." When gamma indicates magnification, this value determines how the texture value is obtained.	LINEAR	0 or 1
<code><mipfilter></code>	MIPmap filter. Enumerated type; see "Details."	LINEAR	0 or 1
<code><border_color></code>	When reading past the edge of the texture address space based on the wrap modes involving clamps, this color takes over. Type <code>fx_color_common</code> (four floating-point numbers in RGBA order). Not valid in GLES <code><sampler2D></code> .	None	0 or 1
<code><mip_max_level></code>	An <code>xs:unsignedByte</code> , which is the maximum number of progressive levels that the sampler will evaluate.	0	0 or 1
<code><mip_min_level></code>	An <code>xs:unsignedByte</code> , which is the minimum progressive levels to begin to evaluate. Not valid in GLES <code><sampler2D></code> .	0	0 or 1
<code><mip_bias></code>	A <code>float_type</code> , which biases the gamma (level of detail parameter) that is used by the sampler to evaluate the MIPmap chain.	0.0	0 or 1
<code><max_anisotropy></code>	An <code>xs:unsignedInt</code> , which is the number of samples that can be used during anisotropic filtering. Not valid in GLES <code><sampler2D></code> .	1	0 or 1
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.8.31.6 Details

For more details about all `<sampler*>` child elements, refer to the OpenGL specification.

The following wrap modes affect the interpretation of s , t , and p texture coordinates outside the [0.0 to 1.0] range based on the usage of a particular **<sampler*>** setup. To assist in understanding, the following table describes the wrap mode enumerations and maps them to OpenGL symbols:

Wrap Mode	OpenGL symbol	Description
WRAP	GL_REPEAT	Ignores the integer part of texture coordinates, using only the fractional part. Tiles the texture at every integer junction. For example, for u values between 0 and 3, the texture is repeated three times; no mirroring is performed.
MIRROR	GL_MIRRORED_REPEAT	First mirrors the texture coordinate. The mirrored coordinate is then clamped as described for CLAMP_TO_EDGE. Flips the texture at every integer junction. For u values between 0 and 1, for example, the texture is addressed normally; between 1 and 2, the texture is flipped (mirrored); between 2 and 3, the texture is normal again; and so on.
CLAMP	GL_CLAMP_TO_EDGE	Clamps texture coordinates at all MIPmap levels such that the texture filter never samples a border texel. NOTE GL_CLAMP takes any texels beyond the sampling border and substitutes those texels with the border color. So, CLAMP_TO_EDGE is more appropriate. This also works much better with OpenGL ES where the GL_CLAMP symbol was removed from the OpenGL ES specification. Texture coordinates reaching or exceeding the range [0.0, 1.0] are set just within 0.0 or 1.0 so that the border is not sampled.
BORDER	GL_CLAMP_TO_BORDER	Clamps texture coordinates at all MIPmaps such that the texture filter always samples border texels for fragments whose corresponding texture coordinate is sufficiently far outside the range [0, 1]. Much like CLAMP, except texture coordinates outside the range [0.0, 1.0] are set to the border color.
MIRROR_ONCE		Takes the absolute value of the texture coordinate (thus, mirroring around 0), and then clamps to the maximum value.

In GLES, in **<newparam>/<sampler2D>**, only the following values are valid:

Wrap Mode	OpenGL symbol	Description
REPEAT		
CLAMP		
CLAMP_TO_EDGE		
MIRRORED_REPEAT		Supported by GLES 1.1 and above only.

Applying a texture to a primitive implies a mapping from texture image space to framebuffer image space. In general, this mapping involves a reconstruction of the sampled texture image, followed by a homogeneous warping implied by the mapping to framebuffer space, then a filtering, followed finally by a resampling of the filtered, warped, reconstructed image before applying it to a fragment. The following table shows valid values for the filtering elements:

Enumeration value	Description	Valid in
NONE	No minification.	<mipfilter>
NEAREST	Bilinear	<minfilter>, <mipfilter>, <magfilter>
LINEAR	Trilinear.	<minfilter>, <mipfilter>, <magfilter>
ANISOTROPIC	Compensates for distortion caused by the difference in angle between the polygon and the plane of the screen. Relies on max_anisotropy.	<minfilter>

5.8.32 image

Category: Texturing

Profile: **External**

5.8.32.1 General

Declares the storage for the graphical representation of an object.

5.8.32.2 Concepts

Digital imagery comes in three main forms of data: raster, vector, and hybrid. Raster imagery comprises a sequence of brightness or color values, called picture elements (pixels), that together form the complete picture. Vector imagery uses mathematical formulae for curves, lines, and shapes to describe a picture or drawing. Hybrid imagery combines both raster and vector information, leveraging their respective strengths, to describe the picture.

The <image> element best describes raster image data but can conceivably handle other forms of imagery. Raster imagery data is typically organized in n -dimensional arrays. This array organization can be leveraged by texture look-up functions to access noncolor values such as displacement, normal, or height field values.

5.8.32.3 Attributes

The <image> element has the following attributes:

id	xs:ID	A text string containing the unique identifier of this element. This value shall be unique within the instance document. Optional.
sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.

5.8.32.4 Related elements

The **<image>** element relates to the following elements:

Parent elements	library_images
Child elements	See the following subclause.
Other	instance_image

5.8.32.5 Child elements

Child elements shall appear in the following order if present; no more than one of **<init_from>** or the **<create_*>** elements may occur:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<renderable share=...>	Defines the image as a render target. If this element exists, then the image can be rendered to. This element contains no data. Set the required Boolean attribute share to true if, when instantiated, the render target is to be shared among all instances instead of being cloned.	N/A	0 or 1
<init_from mips_generate=...>	Initializes the image from a URL (for example, a file) or a list of hexadecimal values. Initialize the whole image structure and data from formats such as DDS. Use the Boolean mips_generate attribute to initialize higher MIP levels if data does not exist in the file. See main entry.	N/A	0 or 1
<create_2d>	Initializes a custom 2D image by specifying its size, viewport ratio, MIP levels, normalization, pixel format, and data sources. It also supports arrays of 2D images. See main entry.	N/A	
<create_3d>	Initializes a custom 3D image (a volumetric image) by specifying its size, MIP level, pixel format, and data sources. It also supports arrays of 3D images. See main entry.	N/A	
<create_cube>	Initializes the six faces of a cube by specifying its size, MIP level, pixel format, and data sources. It also supports arrays of images on each of the cube faces. It also supports arrays of cube images. See main entry.	N/A	
<extra>	See main entry in Core.	N/A	0 or more

5.8.32.6 Details

The image asset can be initialized either directly from a file by using **<init_from>** or with a more complex procedural creation by using **<create_*>**. **<create_*>** is particularly important for creating render targets.

For file loading, many applications support only `<init_from>` as the mechanism for file loading and support more complicated structures via the DDS file format. However, some applications, especially those running on platforms where DDS is not available for free, require that their texture be loaded one subimage at a time via the `<create_*>` elements.

`<image>` objects:

- Have a data format describing the size and layout of fields in each pixel;
- Can be sized either in absolute numbers of pixels using `<size_exact>` or as some fractional size of the viewport using `<size_ratio>`;
- Can declare a fixed number of MIP-map levels using `<mips>`.

5.8.32.7 Example

The following is an example of an `<image>` element that refers to an external PNG asset:

```
<library_images>
  <image name="WoodFloor">
    <init_from><ref>./Textures/WoodFloor-01.png</ref></init_from>
  </image>
</library_images>
```

See the `<create_*>` elements for additional examples.

5.8.33 include

Category: Shaders

Profile: CG, GLSL

5.8.33.1 General

Imports source code or precompiled binary shaders into the FX Runtime by referencing an external resource.

5.8.33.2 Attributes

The `<include>` element has the following attributes:

sid	sid_type	Identifier for this source code block or binary shader. Required. For details, see "Address syntax" in 5.3 Schema concepts.
url	xs:anyURI	Location where the resource can be found. Required.

5.8.33.3 Related elements

The `<include>` element relates to the following elements:

Parent elements	profile_CG, profile_GLES2, profile_GLSL
Child elements	None
Other	None

5.8.33.4 Details

The **<include>** element itself contains no data. Instead, it uses the url attribute to reference the data.

5.8.33.5 Example

```
<include sid="ShinyShader" url="file://assets/source/shader.glsl"/>
```

5.8.34 init_from

Category: Texturing

Profile: **External**

5.8.34.1 General

Initializes an entire image or portions of an image from referenced or embedded data.

5.8.34.2 Concepts

The exact usage of this element depends on its parent element:

If the element is a child of **<image>** then it is intended to initialize the complete image. The image's dimensions and structure will match that of the data if possible.

If the element is a child of **<create_*>** which is a child of **<image>** then it's intended to initialize a small portion of the image, one portion of the structure at a time.

Most image assets are initialized from data generated by artists. This data is typically stored in an image file using one of the many popular file formats, such as BMP, JPG, TGA, PSD, DDS, and so on.

Some image data, such as noise textures or lightmaps, is procedurally generated by applications. These can be optionally embedded directly into the COLLADA file.

This element supports the referencing or embedding of the image data and instructions for where within the image structure to load that data.

5.8.34.3 Attributes

As a child of **<image>**, the **<init_from>** element has the following attribute:

mips_generate	xs:boolean	Optional. Initializes higher MIP levels if data does not exist in a file. Defaults to true.

As a child of the **<create_*>** elements, the **<init_from>** element has the following attributes:

array_index	xs:unsignedInt	Optional. Specifies which array element in the image to initialize (fill). The default is 0.
mip_index	xs:unsignedInt	Required. Specifies which MIP level in the image to initialize.
depth	xs:unsignedInt	Required in <create_3d> ; not valid in <create_2d> or <create_cube> . Specifies the slice (depth level) within the MIP that is to be initialized.
face	Enumeration	Required in <create_cube> ; not valid in <create_2d> or <create_3d> . Specifies the cube face within the MIP that is to be initialized. Valid values are: • POSITIVE_X • NEGATIVE_X • POSITIVE_Y • NEGATIVE_Y • POSITIVE_Z • NEGATIVE_Z

5.8.34.4 Related elements

The **<init_from>** element relates to the following elements:

Parent elements	image , create_2d , create_3d , create_cube
Child elements	See the following subclause.
Other	None

5.8.34.5 Child elements

Exactly one of the following child elements shall occur:

Name/example	Description	Default	Occurrences
<ref>	Contains the URL (xs:anyURI) of a file from which to take initialization data. Assumes the characteristics of the file. If it is a complex format such as DDS, this might include cube maps, volumes, MIPs, and so on.	N/A	0 or 1
<hex format=... >	Contains the embedded image data as a sequence of hexadecimal-encoded binary octets. The data typically contains all the necessary information including header info such as data width and height. Use the required format attribute (xs:token) to specify which codec decodes the image's descriptions and data. This is usually its typical file extension, such as BMP, JPG, DDS, TGA.	N/A	0 or 1

5.8.34.6 Details

Most applications simply use the `<init_from>` as a child of `<image>` and load the file as the image where the characteristics of the image itself come mostly from the file. The `mip_generate` attribute assists supporting better sampling behaviour by generating MIPmaps from a MIP level 0 image, although formats such as DDS support the storage of all MIP levels.

The `<init_from>` element with other parents supports procedurally constructing a more-complicated image structure from many files from disk or embedded in the COLLADA document.

5.8.34.7 Example

The following example loads a 3D volume texture from a DDS file located in the subdirectory named `Rag_OpenGL`.

```
<library_images>
  <image name="Noise">
    <init_from>
      <ref>./Rag_OpenGL/NoiseVolume.dds</ref>
    </init_from>
  </image>
</library_images>
```

See `<create_2d>`, `<create_3d>`, and `<create_cube>` for additional examples.

5.8.35 instance_effect

Category: Effects

Profile: **External**

5.8.35.1 General

Instantiates a COLLADA effect.

5.8.35.2 Concepts

An effect defines the formulas necessary for the visual appearance of geometry and screen-space image processing.

For details about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

`<instance_effect>` instantiates an effect definition from the `<library_effects>` and customizes its parameters.

The `url` attribute references the effect.

`<setparam>`s assign values to specific effect and profile parameters that are unique to the instance.

`<technique_hint>`s indicate the desired or last-used technique inside an effect profile. This allows the user to maintain the same look-and-feel of the effect instance as the last time that the user used it. Some runtime render engines may choose new techniques on the fly, but it is important for some effects and for digital-content-creation consistency to maintain the same visual appearance during authoring.

5.8.35.3 Attributes

The **<instance_effect>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
url	xs:anyURI	The URI of the location of the <effect> element to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the "#" character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.

5.8.35.4 Related elements

The **<instance_effect>** element relates to the following elements:

Parent elements	material
Child elements	See the following subclause.
Other	effect

5.8.35.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<technique_hint>	See main entry.	N/A	0 or more
<setparam>	See main entry in Core.	N/A	0 or more
<extra>	See main entry in Core.	N/A	0 or more

5.8.35.6 Example

```
<material id="BlueCarPaint" name="Light blue car paint">
  <instance_effect url="CarPaint">
    <technique_hint profile="CG" platform="PS3" ref="precalc_texture"/>
    <setparam ref="diffuse_color">
      <float3> 0.3 0.25 0.85 </float3>
    </setparam>
  </instance_effect>
</material>
```

5.8.36 instance_image

Category: Texturing

Profile: External, Effect, CG, COMMON, GLES, GLES2, GLSL

5.8.36.1 General

Instantiates an image to use in a shader.

5.8.36.2 Concepts

Typically for use in an effect for shading a geometric surface. However, an image can also be used as a target for rendering. This way, the picture or data inside the image can be updated dynamically with advanced FX shading techniques. An image that is the target for rendering, however, shall contain the **<renderable>** element.

5.8.36.3 Attributes

The **<instance_image>** element has the following attributes:

url	xs:anyURI	Required. The URI of the image asset.
sid	sid_type	Optional. A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. For details, see "Address syntax" in 5.3 Schema concepts.
name	xs:token	Optional. The text string name of this element.

5.8.36.4 Related elements

The **<instance_image>** element relates to the following elements:

Parent elements	<sampler*>, color_target, depth_target, stencil_target
Child elements	See the following subclause.
Other	image, library_images, sampler_image

5.8.36.5 Child elements

Name/example	Description	Default	Occurrences
<extra>	For storage of extra information that is not defined in COLLADA. See main entry in 5.5 Core elements reference.	N/A	0 or more

5.8.36.6 Details

The behaviour of instantiating an image is typically straight-forward, except for images that are not renderable. Renderable images have two behaviour options. If the renderable image is marked as "shared" then the picture or data of that image is shared among all instances. As the image is rendered, all instances will receive the updated data due to sharing. If the renderable image is not shared, then a unique copy of that image is produced for each instance so that rendering to the image instance does not affect other image instances.

5.8.36.7 Example

See `<color_target>`, `<depth_target>`, and `<stencil_target>` for more examples.

```
<newparam sid="surfaceTex">
  <sampler2D>
    <instance_image url="noise1"/>
  </sampler2D>
</newparam>
```

5.8.37 instance_material (geometry)

Category: Materials

Profile: **External**

5.8.37.1 General

Instantiates a COLLADA material resource.

5.8.37.2 Concepts

An effect defines the formulas necessary for the visual appearance of geometry and screen-space image processing. A material instantiates an effect, fills its parameters with values, and selects a technique. A material instance connects the material to geometry or scene items.

For details about instance elements in COLLADA, see “Instantiation and external referencing” in 5.3 Schema concepts.

To use a material, it is instantiated and attached to the geometry. The symbol attribute of `<instance_material>` indicates to which geometry the material is attached and the target attribute references the material that it is instantiating.

In addition to identifying the section of the geometry to attach to (symbol), this element also defines how the vertex stream is remapped and how scene objects are bound to material effect parameters. These are the connections that can be done only very late and that depend on the scene geometry to which it is being connected.

`<bind>` connects a parameter in the material’s effect by semantic to a target in the scene.

`<bind_vertex_input>` connects a vertex shader’s vertex stream semantics (for example, TEXCOORD2) to the geometry’s vertex input stream specified by the input_semantic and input_set attributes.

5.8.37.3 Attributes

The **<instance_material>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.
name	xs:token	The text string name of this element. Optional.
target	xs:anyURI	The URI of the location of the <material> element to instantiate. Required. Can refer to a local instance or external reference. For a local instance, this is a relative URI fragment identifier that begins with the "#" character. The fragment identifier is a XPointer shorthand pointer that consists of the ID of the element to instantiate. For an external reference, this is an absolute or relative URL.
symbol	xs:NCName	Which symbol defined from within the geometry this material binds to. Required.

5.8.37.4 Related elements

The **<instance_material>** element relates to the following elements:

Parent elements	technique_common in bind_material
Child elements	See the following subclause.
Other	material

5.8.37.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<bind> (FX)	See main entry.	N/A	0 or more
<bind_vertex_input>	Binds vertex inputs to effect parameters upon instantiation. See main entry. (Only in <bind_material> .)	N/A	0 or more
<extra>	See main entry in Core.	N/A	0 or more

5.8.37.6 Example

```
<instance_geometry url="#BeechTree">
  <bind_material>
    <param sid="windAmount" semantic="WINDSPEED" type="float3"/>
    <technique_common>
      <instance_material symbol="leaf" target="#MidsummerLeaf01"/>
      <instance_material symbol="bark" target="#MidsummerBark03">
        <bind semantic="LIGHTPOS1" target="/scene/light01/pos"/>
        <bind_vertex_input semantic="TEXCOORD0"
          input_semantic="BeechTree/texcoord2" input_set="2"/>
      </instance_material>
    </technique_common>
  </bind_material>
</instance_geometry>
```

5.8.38 instance_material (rendering)

Category: Rendering

Profile: External

5.8.38.1 General

Instantiates a COLLADA material resource for a screen effect.

5.8.38.2 Concepts

For details about instance elements in COLLADA, see “Instantiation and External Referencing” in 5.3 Schema concepts.

To use a material, it is instantiated. Most instantiated materials are attached to geometry. But, in this case, we are attaching materials to the scene itself for image- or lens-level processing.

The url attribute references the material that it is instantiating.

It also identifies how scene objects are bound to material effect parameters. These are the connections that can be done only very late and that depend on the scene geometry to which it is being connected.

<bind> (FX) connects a parameter in the material’s effect by semantic to a target in the scene.

The **<technique_override>** optionally allows for very specific usage of the material’s technique and pass subelements rather than the typical pattern of using the material’s **<technique_hint>** and rendering each pass. This element is available only when the parent element is a **<render>** element found in **<evaluate_scene>**. This allows added control to invoke the portions of the material as needed to accomplish their scene effect because scene evaluation effect can be much more procedural and complicated to evaluate compared to most geometry surface shaders.

In older versions of COLLADA, the user needed to break the effect up into many small effects and materials to accomplish this, and to manage different parameter tables for each of these broken-up materials. This is no longer necessary with the new control.

5.8.38.3 Attributes

The **<instance_material>** element has the following attribute:

url	xs:anyURI	Location where the material can be found. Required.

5.8.38.4 Related elements

The **<instance_material>** element relates to the following elements:

Parent elements	evaluate_scene/render
Child elements	See the following subclause.
Other	material

5.8.38.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><technique_override ref="" pass="" /></code>	Target specific techniques and passes inside a material rather than having to split the effects techniques and passes into multiple effects. The ref attribute is required and specifies the SID of a <code><technique></code> . The pass attribute is optional and specifies the SID of one pass to execute. If not specified (or empty), then all of the technique's <code><pass></code> es are used.	N/A	0 or 1
<code><bind></code> (FX)	Binds values to effect parameters upon instantiation. See main entry.	N/A	0 or more
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.8.38.6 Details

`<evaluate_scene><render>` allows instantiating materials instead of instantiating effects directly so that the parameter setup for a particular scene's post-processing effect can be used multiple time or across multiple scenes. Effects such as blur could easily be applied multiple times to the same scene or shared among different scenes.

5.8.38.7 Example

```
<visual_scene>
<node>
  <!--a really cool scene here-->
</node>
<evaluate_scene sid="blurredGreen">
  <render sid="greenPass">
    <instance_material url="http://127.0.0.1/foo.dae#greenFilter1"/>
  </render>
  <render sid="blur1">
    <instance_material url="http://127.0.0.1/foo.dae#blur1">
      <technique_override ref="main" pass="vertical"/>
    </instance_material>
  </render>
  <render sid="blur2">
    <instance_material url="http://127.0.0.1/foo.dae#blur1">
      <technique_override ref="main" pass="horizontal"/>
    </instance_material>
  </render>
  <render sid="blur3">
    <instance_material url="http://127.0.0.1/foo.dae#blur1">
      <technique_override ref="main" pass="vertical"/>
    </instance_material>
  </render>
  <render sid="blur4">
    <instance_material url="http://127.0.0.1/foo.dae#blur1">
      <technique_override ref="main" pass="horizontal"/>
    </instance_material>
  </render>
</evaluate_scene>
</visual_scene>
```

5.8.39 lambert

Category: Rendering

Profile: **COMMON**

5.8.39.1 General

Produces a diffuse shaded surface that is independent of lighting.

5.8.39.2 Concepts

Used inside a **<profile_COMMON>** effect, declares a fixed-function pipeline that produces a diffuse shaded surface that is independent of lighting.

The result is based on Lambert's Law, which states that when light hits a rough surface, the light is reflected in all directions equally. The reflected color is calculated simply as:

$$color = \langle emission \rangle + \langle ambient \rangle * al + \langle diffuse \rangle * \max(N * L, 0)$$

where:

- *al* is a constant amount of ambient light contribution coming from the scene. In the COMMON profile, this is the sum of all the **<light><technique_common><ambient><color>** values in the **<visual_scene>**;
- *N* is normal vector;
- *L* is light vector.

5.8.39.3 Attributes

The **<lambert>** element has no attributes.

5.8.39.4 Related elements

The **<lambert>** element relates to the following elements:

Parent elements	technique (FX) in profile_COMMON
Child elements	See the following subclause.
Other	None

5.8.39.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><emission></code>	Declares the amount of light emitted from the surface of this object. See <code>fx_common_color_or_texture_type</code> .	N/A	0 or 1
<code><ambient></code> (FX)	Declares the amount of ambient light reflected from the surface of this object. See <code>fx_common_color_or_texture_type</code> .	N/A	0 or 1
<code><diffuse></code>	Declares the amount of light diffusely reflected from the surface of this object. See <code>fx_common_color_or_texture_type</code> .	N/A	0 or 1
<code><reflective></code>	Declares the color of a perfect mirror reflection. See <code>fx_common_color_or_texture_type</code> .	N/A	0 or 1
<code><reflectivity></code>	Declares the amount of perfect mirror reflection to be added to the reflected light as a value between 0.0 and 1.0. See <code>fx_common_float_or_param_type</code> .	N/A	0 or 1
<code><transparent></code>	Declares the color of perfectly refracted light. See <code>fx_common_color_or_texture_type</code> and "Determining transparency (opacity)" in 5.7 Getting started with COLLADA FX.	N/A	0 or 1
<code><transparency></code>	Declares the amount of perfectly refracted light added to the reflected color as a scalar value between 0.0 and 1.0. See <code>fx_common_float_or_param_type</code> and "Determining transparency (opacity)" in 5.7 Getting started with COLLADA FX.	N/A	0 or 1
<code><index_of_refraction></code>	Declares the index of refraction for perfectly refracted light as a single scalar index. See <code>fx_common_float_or_param_type</code> .	N/A	0 or 1

5.8.39.6 Example

```

<profile_COMMON>
  <newparam sid="myDiffuseColor">
    <float3> 0.2 0.56 0.35 </float3>
  </newparam>
  <technique sid="T1">
    <lambert>
      <emission><color>1.0 0.0 0.0 1.0</color></emission>
      <ambient><color>1.0 0.0 0.0 1.0</color></ambient>
      <diffuse><param ref="myDiffuseColor"/></diffuse>
      <reflective><color>1.0 1.0 1.0 1.0</color></reflective>
      <reflectivity><float>0.5</float></reflectivity>
      <transparent><color>0.0 0.0 1.0 1.0</color></transparent>
      <transparency><float>1.0</float></transparency>
    </lambert>
  </technique>
</profile_COMMON>

```

5.8.40 library_effects

Category: Effects

Profile: **External**

5.8.40.1 General

Provides a library for the storage of **<effect>** assets.

5.8.40.2 Concepts

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.8.40.3 Attributes

The **<library_effects>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <library_effects> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.8.40.4 Related elements

The **<library_effects>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.8.40.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<effect>	See main entry.	N/A	1 or more
<extra>	See main entry in Core.	N/A	0 or more

5.8.40.6 Example

The following is an example of a **<library_effects>** element:

```
<library_effects>
  <effect id="fullscreen_effect1">
    ...
  </effect>
</library_effects>
```

5.8.41 library_images

Category: Texturing

5.8.41.1 General

Provides a library for the storage of **<image>** assets.

5.8.41.2 Concepts

The **<image>** element represents either a picture's data or renderable objects that will receive their data later.

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.8.41.3 Attributes

The **<library_images>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <library_images> element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.8.41.4 Related elements

The **<library_images>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.8.41.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<image>	See main entry.	N/A	1 or more
<extra>	See main entry in Core.	N/A	0 or more

5.8.41.6 Example

The following is an example of a <library_images> element:

```
<library_images>
  <image name="Rose">
    <init_from>../flowers/rose01.jpg</init_from>
  </image>
</library_images>
```

5.8.42 library_materials

Category: Materials

Profile: **External**

5.8.42.1 General

Provides a library for the storage of <material> assets.

5.8.42.2 Concepts

An effect defines the formulas necessary for the visual appearance of geometry and screen-space image processing. A material instantiates an effect, fills its parameters with values, and selects a technique.

As data sets become larger and more complex, they become harder to manipulate within a single container. One approach to managing this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in separate resources as libraries.

5.8.42.3 Attributes

The <library_materials> element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.8.42.4 Related elements

The **<library_materials>** element relates to the following elements:

Parent elements	COLLADA
Child elements	See the following subclause.
Other	None

5.8.42.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<material>	See main entry.	N/A	1 or more
<extra>	See main entry in Core.	N/A	0 or more

5.8.42.6 Example

The following is an example of a **<library_materials>** element:

```
<library_materials>
  <material id="mat1">
    ...
  </material >
  <material id="mat2">
    ...
  </material>
</library_materials>
```

5.8.43 linker

Category: Shaders

Profile: **GLS2**

5.8.43.1 General

Contains command-line or runtime-invocation options for shader linkers to combine shaders into programs.

5.8.43.2 Concepts

Compiling and linking are part of a complicated process of turning high-level, programmer-friendly code into a machine executable problem. The details of this process cannot be described here. It is often specific to the target platform or profile.

Typically, when invoking the linker via the API functions provided by GLSL and GLES2, the API does not require any options. The baseline GLES2 API also does not explicitly support pre-linked binaries. But some platforms provide this extra opportunity for optimization at the cost of managing all necessary shader combinations.

The binary results of the compiler may also be optionally stored here, if the platform supports binaries, rather than having to recompile.

5.8.43.3 Attributes

The **<linker>** element has the following attributes:

platform	xs:string	Required. The sub-platform name to distinguish between multiple linker settings.
target	xs:string	Optional. Target binary profile.
options	xs:string	Optional. Linker options. See your platform provider's documentation for details.

5.8.43.4 Related elements

The **<linker>** element relates to the following elements:

Parent elements	program
Child elements	See the following subclause.
Other	None

5.8.43.5 Child elements

Name/example	Description	Default	Occurrences
<binary>	See main entry.	N/A	0 or more

5.8.43.6 Details

This element contains text that are the linking options given to the tool as a text string. It can optionally contain a binary representation of the compiled and linked results.

5.8.43.7 Example

```
<linker platform="PC" target="assemblyProfile" options="-debug"/>
```

5.8.44 material

Category: Materials

Profile: **External**

5.8.44.1 General

Defines the formulas necessary for the visual appearance of geometry and screen-space image processing.

5.8.44.2 Concepts

A material instantiates an effect, fills its parameters with values, and selects a technique. It describes the appearance of a geometric object or may perform screen-space processing to create camera-lens-like effects such as blurs, blooms, or color filters.

In computer graphics, geometric objects can have many parameters that describe their material properties. These material properties are the parameters for the rendering computations that produce the visual appearance of the object in the final output. Likewise, screen-space processing and compositing may also require many parameters for performing computation.

The specific set of material parameters depend upon the graphics rendering system employed. Fixed function, graphics pipelines require parameters to solve a predefined illumination model, such as Phong illumination. These parameters include terms for ambient, diffuse and specular reflectance, for example.

In programmable graphics pipelines, the programmer defines the set of material parameters. These parameters satisfy the rendering algorithm defined in the vertex and pixel programs.

5.8.44.3 Attributes

The **<material>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
name	xs:token	The text string name of this element. Optional.

5.8.44.4 Related elements

The **<material>** element relates to the following elements:

Parent elements	library_materials
Child elements	See the following subclause.
Other	instance_material (geometry)

5.8.44.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<instance_effect>	See main entry.	N/A	1
<extra>	See main entry in Core.	N/A	0 or more

5.8.44.6 Example

The following is an example of a simple **<material>** element. The material is contained in a material **<library_materials>** element:

```
<library_materials>
  <material id="Blue">
    <instance_effect url="#phongEffect">
      <setparam ref="AMBIENT">
        <float3>0.0 0.0 0.1</float3>
      </setparam>
      <setparam ref="DIFFUSE">
        <float3>0.15 0.15 0.1</float3>
      </setparam>
      <setparam ref="SPECULAR">
        <float3>0.5 0.5 0.5</float3>
      </setparam>
      <setparam ref="SHININESS">
        <float>16.0</float>
      </setparam>
    </instance_effect>
  </material>
</library_materials>
```

5.8.45 modifier

Category: Parameters

Profile: External, Effect, CG, COMMON, GLES, GLES2, GLSL

5.8.45.1 General

Provides additional information about the volatility or linkage of a **<newparam>** declaration.

5.8.45.2 Concepts

Allows COLLADA FX parameter declarations to specify constant, external, or uniform parameters.

5.8.45.3 Attributes

The **<modifier>** element has no attributes.

5.8.45.4 Related elements

The **<modifier>** element relates to the following elements:

Parent elements	newparam
Child elements	None
Other	None

5.8.45.5 Details

Contains a linkage modifier. Not every linkage modifier is supported by every FX runtime. Valid modifiers are CONST, UNIFORM, VARYING, STATIC, VOLATILE, EXTERN and SHARED.

5.8.45.6 Example

```
<newparam sid="diffuseColor">
  <annotate name="UIWidget"><string>none</string></annotate>
  <semantic>DIFFUSE</semantic>
  <modifier>CONST</modifier>
  <float3> 0.30 0.56 0.12 </float>
</newparam>
```

5.8.46 pass

Category: Rendering

Profile: CG, GLES, GLES2, GLSL

5.8.46.1 General

Provides a static declaration of all the render states, shaders, and settings for one rendering pipeline.

5.8.46.2 Concepts

<pass> describes all the render states and shaders for a rendering pipeline, and is the element that the FX Runtime is asked to “apply” to the current graphics state before the program can submit geometry.

A static declaration is one that requires no evaluation by a scripting engine or runtime system in order to be applied to the graphics state. At the time that a **<pass>** is applied, all render state settings and uniform parameters are precalculated and known.

5.8.46.3 Attributes

The **<pass>** element has the following attribute:

sid	sid_type	The optional label for this pass, allowing passes to be specified by name and, if desired, reordered by the application as the technique is evaluated. Optional. For details, see “Address syntax” in 5.3 Schema concepts.

5.8.46.4 Related elements

The **<pass>** element relates to the following elements:

Parent elements	technique (FX) (in profile_CG , profile_GLES , profile_GLSL , profile_GLES2)
Child elements	See the following subclauses.
Other	None

5.8.46.5 Child elements in GLES scope

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<annotate>	See main entry.	None	0 or 1
<states>	See main entry.	None	0 or 1
<evaluate>	See main entry.	None	0 or 1
<extra>	See main entry in Core.	N/A	0 or more

5.8.46.6 Child elements in CG, GLES2, or GLSL scope

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<annotate>	See main entry.	None	0 or 1
<states>	See main entry.	None	0 or 1
<program>	See main entry.	None	0 or 1
<evaluate>	See main entry.	None	0 or 1
<extra>	See main entry in Core.	N/A	0 or more

5.8.46.7 Details

Reordering passes can be useful when a single pass is applied repetitively, for example, a “blur” low-pass convolution may need to be applied to an off-screen texture several times to create the desired effect.

5.8.46.8 Example

The following is an example of a <pass> contained in a <profile_CG>:

```
<pass sid="PixelShaderVersion">
  <states>
    <depth_test_enable value="true"/>
    <depth_func value="LEQUAL"/>
  </states>
  <program>
    <shader stage="VERTEX">
      <sources entry="main">
        <import ref="allFunctions"/>
      </sources>
      <compiler platform="PC" target="GLSLV"/>
    <bind symbol="LightPos">
      <param ref="effectLightPos"/>
    </bind>
  </shader>
  <shader stage="FRAGMENT">
    <sources entry="passThruFS">
      <import ref="allFunctions"/>
    </sources>
  </shader>
</pass>
```



```

    <compiler platform="PC" target="GLSLF"/>
  </shader>
</program>
</pass>

```

5.8.47 phong

Category: Rendering

Profile: **COMMON**

5.8.47.1 General

Produces a shaded surface where the specular reflection is shaded according the Phong BRDF approximation.

5.8.47.2 Concepts

Used inside a **<profile_COMMON>** effect, declares a fixed-function pipeline that produces a specularly shaded surface that reflects ambient, diffuse, and specular reflection, where the specular reflection is shaded according the Phong BRDF approximation.

The **<phong>** shader uses the common Phong shading formula, that is:

$$color = \langle emission \rangle + \langle ambient \rangle * al + \langle diffuse \rangle * \max(N * L, 0) + \langle specular \rangle * \max(R \cdot I, 0)^{\langle shininess \rangle}$$

where:

- *al* is a constant amount of ambient light contribution coming from the scene. In the COMMON profile, this is the sum of all the **<light><technique_common><ambient><color>** values in the **<visual_scene>**;
- *N* is normal vector;
- *L* is light vector;
- *I* is eye vector;
- *R* is perfect reflection vector (reflect (*L* around *N*)).

5.8.47.3 Attributes

The **<phong>** element has no attributes.

5.8.47.4 Related elements

Parent elements	technique (FX) in profile_COMMON
Child elements	See the following subclauses.
Other	None

5.8.47.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><emission></code>	Declares the amount of light emitted from the surface of this object. See <code>fx_common_color_or_texture_type</code> .	N/A	0 or 1
<code><ambient></code> (FX)	Declares the amount of ambient light emitted from the surface of this object. See <code>fx_common_color_or_texture_type</code> .	N/A	0 or 1
<code><diffuse></code>	Declares the amount of light diffusely reflected from the surface of this object. See <code>fx_common_float_or_param_type</code> .	N/A	0 or 1
<code><specular></code>	Declares the color of light specularly reflected from the surface of this object. See <code>fx_common_color_or_texture_type</code> .	N/A	0 or 1
<code><shininess></code>	Declares the specular or roughness of the specular reflection lobe. See <code>fx_common_color_or_texture_type</code> .	N/A	0 or 1
<code><reflective></code>	Declares the color of a perfect mirror reflection. See <code>fx_common_color_or_texture_type</code> .	N/A	0 or 1
<code><reflectivity></code>	Declares the amount of perfect mirror reflection to be added to the reflected light as a value between 0.0 and 1.0. See <code>fx_common_float_or_param_type</code> .	N/A	0 or 1
<code><transparent></code>	Declares the color of perfectly refracted light. See <code>fx_common_color_or_texture_type</code> and “Determining transparency (opacity)” in 5.7 Getting started with COLLADA FX.	N/A	0 or 1
<code><transparency></code>	Declares the amount of perfectly refracted light added to the reflected color as a scalar value between 0.0 and 1.0. See <code>fx_common_float_or_param_type</code> and “Determining transparency (opacity)” in 5.7 Getting started with COLLADA FX.	N/A	0 or 1
<code><index_of_refraction></code>	Declares the index of refraction for perfectly refracted light as a single scalar index. See <code>fx_common_float_or_param_type</code> .	N/A	0 or 1

5.8.47.6 Example

This example has the following properties:

- It is an effect that takes its diffuse color as a parameter. Diffuse is defaulted to (0.2 0.56 0.35) but can be overridden in the material.
- It does not emit any light or absorb any indirect lighting (ambient).
- It has a little white shiny spot. 50 is a moderately high shininess power term, so the shiny spot shall be fairly sharp.

- It is reflective and will reflect the environment at 5% intensity on top of the standard surface color calculations.
- It is not transparent. See “Determining transparency (opacity)” in 5.7 Getting started with COLLADA FX.

```
<profile_COMMON>
  <newparam sid="myDiffuseColor">
    <float3> 0.2 0.56 0.35 </float3>
  </newparam>
  <technique sid="phong1">
    <phong>
      <emission><color>0.0 0.0 0.0 1.0</color></emission>
      <ambient><color>0.0 0.0 0.0 1.0</color></ambient>
      <diffuse><param ref="myDiffuseColor"/></diffuse>
      <specular><color>1.0 1.0 1.0 1.0</color></specular>
      <shininess><float>50.0</float></shininess>
      <reflective><color>1.0 1.0 1.0 1.0</color></reflective>
      <reflectivity><float>0.051</float></reflectivity>
      <transparent><color>0.0 0.0 0.0 1.0</color></transparent>
      <transparency><float>1.0</float></transparency>
    </phong>
  </technique>
</profile_COMMON>
```

5.8.48 profile_BRIDGE

Category: Profiles

Profile: **BRIDGE**

5.8.48.1 General

Provides support for referencing effect profiles written with external standards.

5.8.48.2 Concepts

This element enables users to work with systems that are not currently supported directly by COLLADA, reference existing libraries of effects that were written prior to COLLADA, or use effects written by people who choose not to use COLLADA.

This element enables effect authors to represent COLLADA FX effects with multiple API, platform, and common profiles while still including support for additional representations, APIs, and platforms that are not part of the COLLADA FX schema.

Some example standards that could bridge to FX are Microsoft/HLSL, CgFX (NVIDIA®), and SushiFX (AMD).

This feature:

- Enables the single effect, multiple profiles paradigm to extend the existing COLLADA standard and schema.
- Future-proofs COLLADA FX for shader languages, effects languages, and API without runtime procedural effects building that are currently not supported by COLLADA FX or are introduced between COLLADA schema revisions.

In use, the effect file is imported as a profile. The effect file's parameters become profile-level parameters. The parameter's scoped identifier (SID) and name are the same as the name of the effect-file parameter's name. Similarly, techniques and passes would follow the same rule, where their names in the file become their SIDs for referencing from a **<material>**'s **<setparam>** and **<technique_hint>**, among other potential places in a COLLADA document.

The imported **<profile_BRIDGE>** elements encapsulate all the platform-specific values and declarations for a particular profile. Parameters imported with a **<profile_BRIDGE>** block are not available to other profiles.

The **<profile_BRIDGE>** element defines the clear interface between concrete, platform-specific data types and the abstract COLLADA data types used in the rest of the document. Parameters declared outside of this barrier may require casting when used inside a **<profile_BRIDGE>** block.

For more information, see "Using profiles for platform-specific effects" in 5.7 Getting started with COLLADA FX.

5.8.48.3 Attributes

The **<profile_BRIDGE>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
platform	xs:NCName	The type of platform. This is a vendor-defined character string that indicates the platform or capability target, most likely an OpenGL ES 2.0 platform. It might target a specific piece of hardware or hardware family. Optional.
url	xs:anyURI	The URI of the file to which you are bridging. Required.

5.8.48.4 Related elements

The **<profile_BRIDGE>** element relates to the following elements:

Parent elements	effect
Child elements	See the following subclauses.
Other	None

5.8.48.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	For resource management tracking. See main entry in Core.	N/A	0 or 1
<extra>	A means to store extension data beyond the COLLADA schema specification. See main entry in Core.	N/A	0 or more

5.8.48.6 Example

```
<effect id="uniqueID_12345" name="myEffect">
  <profile_COMMON>
    <constant><emissive><float4>1 1 1 1</float4></emissive></constant>
  </profile_COMMON>
  <profile_BRIDGE platform="DIRECT3D9"
    url="http://www.YourDomain.com/myEffect.fx"/>
</effect>
```

5.8.49 profile_CG

Category: Profiles

Profile: CG

5.8.49.1 General

Declares a platform-specific representation of an effect written in the NVIDIA® Cg language.

5.8.49.2 Concepts

The **<profile_CG>** element is a profile within an effect that encapsulate all the platform-specific values and declarations to achieve for a particular visual appearance. In **<effect>** scope, parameters are available to all platforms, but parameters declared inside a **<profile_CG>** block are available only to shaders that are also inside that profile.

The **<profile_CG>** element defines the clear interface between concrete, platform-specific data types and the abstract COLLADA data types used in the rest of the document. Parameters declared outside of this barrier may require casting when used inside a **<profile_CG>** block.

For more information, see “Using profiles for platform-specific effects” in 5.7 Getting started with COLLADA FX.

5.8.49.3 Attributes

The **<profile_CG>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
platform	xs:NCName	The type of platform. This is a vendor-defined character string that indicates the platform or capability target for the technique. The default is “PC”. Optional.

5.8.49.4 Related elements

The **<profile_CG>** element relates to the following elements:

Parent elements	effect
Child elements	See the following subclauses.
Other	None

5.8.49.5 Child elements

Child elements shall appear in the following order if present, with the following exception:

- `<include>` and `<code>` are interchangeable

Name/example	Description	Default	Occurrences
<code><asset></code>	See main entry in Core.	N/A	0 or 1
<code><code></code>	See main entry.	N/A	0 or more
<code><include></code>	See main entry.	N/A	0 or more
<code><newparam></code>	See main entry.	N/A	0 or more
<code><technique></code> (FX)	See main entry for attributes and description and the following subclause for child element details.	N/A	1 or more
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.8.49.6 Child elements for `<profile_CG>` / `<technique>`

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><asset></code>	See main entry in Core.	N/A	0 or 1
<code><annotate></code>	See main entry.	N/A	0 or more
<code><pass></code>	See main entry.	N/A	1 or more
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.8.49.7 Example

```

<profile_CG>
  <newparam sid="color">
    <float3> 0.5 0.5 0.5 </float3>
  </newparam>
  <newparam sid="lightpos">
    <semantic>LIGHTPOS0</semantic>
    <float3> 0.0 10.0 0.0 </float3>
  </newparam>
  <newparam sid="world">
    <semantic>WORLD</semantic>
    <float4x4> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 </float4x4>
  </newparam>
  <newparam sid="worldIT">
    <semantic>WORLD_INVERSE_TRANSPOSE</semantic>
    <float4x4> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 </float4x4>
  </newparam>
  <newparam sid="worldViewProj">
    <semantic>WORLD_VIEW_PROJECTION</semantic>
    <float4x4> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 </float4x4>
  </newparam>
</code>
void VS (
  in varying float4 pos,
  in varying float3 norm,

```

```

in uniform float3 light_pos,
in uniform float4x4 w: WORLD,
in uniform float4x4 wit: WORLD_INVERSE_TRANSPOSE,
in uniform float4x4 wvp: WORLD_VIEW_PROJECTION,
out varying float4 oPosition : POSITION,
out varying float3 oNormal : TEXCOORD0,
out varying float3 oToLight : TEXCOORD1 )
{ oPosition = mul(wvp, pos);
  oNormal = mul(wit, float4(norm, 1)).xyz;
  oToLight = light_pos - mul(w, pos).xyz;
  return;
}
float3 diffuseFS (
  in uniform float3 flat_color,
  in varying float3 norm : TEXCOORD0,
  in varying float3 to_light : TEXCOORD1 ) : COLOR
{ return flat_color * saturate(NdotL),
  0.0, 1.0);
}
</code>
<technique id="default" sid="default">
  <pass sid="single_pass">
    <program>
      <shader stage="VERTEX">
        <sources entry="VS">
          <import ref="diffuse-code-1"/>
        </sources>
        <compiler platform="PC" target="GLSLV"/>
        <bind_uniform symbol="light_pos">
          <param ref="lightpos"/>
        </bind_uniform>
        <bind_uniform symbol="w">
          <param ref="world"/>
        </bind_uniform>
        <bind_uniform symbol="wit">
          <param ref="worldIT"/>
        </bind_uniform>
        <bind_uniform symbol="wvp">
          <param ref="worldViewProj"/>
        </bind_uniform>
      </shader>
      <shader stage="FRAGMENT">
        <sources entry="diffuseFX">
          <import ref="diffuse-code-1" />
        </sources>
        <compiler platform="PC" target="GLSLV"/>
        <bind_uniform symbol="flat_color">
          <param ref="color"/>
        </bind_uniform>
      </shader>
    </program>
  </pass>
</technique>
</profile_CG>

```

5.8.50 profile_COMMON

Category: Profiles

Profile: **COMMON**

5.8.50.1 General

Opens a block of platform-independent declarations for the common, fixed-function shader.

5.8.50.2 Concepts

The **<profile_COMMON>** elements encapsulate all the values and declarations for a platform-independent fixed-function shader. All platforms are required to support **<profile_COMMON>**. **<profile_COMMON>** effects are designed to be used as the reliable fallback when no other profile is recognized by the current effects runtime.

For more information, see “Using profiles for platform-specific effects” in 5.7 Getting started with COLLADA FX.

5.8.50.3 Attributes

The **<profile_COMMON>** element has the following attribute:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.

5.8.50.4 Related elements

The **<profile_COMMON>** element relates to the following elements:

Parent elements	effect
Child elements	See the following subclauses.
Other	None

5.8.50.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<newparam>	Creates a new parameter from a constrained set of types recognizable by all platforms – <float> , <float2> , <float3> , <float4> , and <sampler2D> , with an additional semantic. See main entry. Example: <pre><newparam sid="mySID"> <semantic> DIFFUSECOLOR </semantic> <float3> 1 2 3 </float3> </newparam></pre>	N/A	0 or more
<technique> (FX)	Declares the only technique for this effect. See main entry for attributes and description and the following subclause for child element details.	N/A	1
<extra>	See main entry in Core.	N/A	0 or more

5.8.50.6 Child elements for `<profile_COMMON>` / `<technique>`

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><asset></code>	See main entry in Core.	N/A	0 or 1
<code>shader_element</code>	One of <code><constant></code> (FX), <code><lambert></code> , <code><phong></code> , or <code><blinn></code> . See main entries.	N/A	0 or more
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.8.50.7 Example

```

<profile_COMMON>
  <newparam sid="myDiffuseColor">
    <float3> 0.2 0.56 0.35 </float3>
  </newparam>
  <technique sid="phong1">
    <phong>
      <emission><color>1.0 0.0 0.0 1.0</color></emission>
      <ambient><color>1.0 0.0 0.0 1.0</color></ambient>
      <diffuse><param ref="myDiffuseColor"/></diffuse>
      <specular><color>1.0 0.0 0.0 1.0</color></specular>
      <shininess><float>50.0</float></shininess>
      <reflective><color>1.0 1.0 1.0 1.0</color></reflective>
      <reflectivity><float>0.5</float></reflectivity>
      <transparent><color>0.0 0.0 1.0 1.0</color></transparent>
      <transparency><float>1.0</float></transparency>
    </phong>
  </technique>
</profile_COMMON>

```

5.8.51 profile_GLES

Category: Profiles

Profile: **GL ES**

5.8.51.1 General

Declares platform-specific data types and `<technique>`s for OpenGL ES.

5.8.51.2 Concepts

The `<profile_GLES>` elements encapsulate all the platform-specific values and declarations for a particular profile. In `<effect>` scope, parameters are available to all platforms, but parameters declared inside a `<profile_GLES>` block are available only to shaders that are also inside that profile.

The `<profile_GLES>` element defines the clear interface between concrete, platform-specific data types and the abstract COLLADA data types used in the rest of the document. Parameters declared outside of this barrier may require casting when used inside a `<profile_GLES>` block.

For more information, see “Using profiles for platform-specific effects” in 5.7 Getting started with COLLADA FX.

5.8.51.3 Attributes

<profile_GLES> has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
platform	xs:NMTOKEN	The type of platform. This is a vendor-defined character string that indicates the platform or capability target for the technique. Optional.

5.8.51.4 Related elements

The <profile_GLES> elements relate to the following elements:

Parent elements	effect
Child elements	See the following subclauses.
Other	None

5.8.51.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<newparam>	Create a new parameter from a constrained set of types recognizable by all platforms – <float>, <float2>, <float3>, <float4>, <surface> and <sampler2D>, with an additional semantic. See main entry. Example: <newparam sid="mySID"> <semantic> DIFFUSECOLOR </semantic> <float3> 1 2 3 </float3> </newparam>	N/A	0 or more
<technique> (FX)	Declares a technique for this effect. See main entry for attributes and description and the following subclause for child element details.	N/A	1 or more
<extra>	See main entry in Core.	N/A	0 or more

5.8.51.6 Child elements for <profile_GLES> / <technique>

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<asset>	See main entry in Core.	N/A	0 or 1
<annotate>	See main entry.	N/A	0 or more
<pass>	See main entry.	N/A	1 or more
<extra>	See main entry in Core.	N/A	0 or more

5.8.51.7 Example

The following example shows terrain rendering to transitions between two different ground textures. It combines gravel texture and grass texture with an alpha transition texture that dictates the per-textel percentages of how they will blend.

```
<profile_GLES>
  <newparam sid="gravel">
    <sampler2D/>
  </newparam>
  <newparam sid="grass">
    <sampler2D/>
  </newparam>
  <newparam sid="transition">
    <sampler2D/>
  </newparam>
  <technique sid="main">
    <pass sid="p0">
      <states>
        <texture_pipeline>
          <value>
            <texcombiner>
              <constant> 0.0f, 0.0f, 0.0f, 1.0f </constant>
              <RGB operator="INTERPOLATE">
                <argument source="TEXTURE" operand="SRC_COLOR" sampler="gravel"/>
                <argument source="TEXTURE" operand="SRC_COLOR" sampler="grass"/>
                <argument source="TEXTURE" operand="SRC_ALPHA"
sampler="transition"/>
              </RGB>
              <alpha operator="INTERPOLATE">
                <argument source="TEXTURE" operand="SRC_ALPHA" sampler="gravel"/>
                <argument source="TEXTURE" operand="SRC_ALPHA" sampler="grass"/>
                <argument source="TEXTURE" operand="SRC_ALPHA"
sampler="transition"/>
              </alpha>
            </texcombiner>
            <texcombiner>
              <RGB operator="MODULATE">
                <argument source="PRIMARY" operand="SRC_COLOR"/>
                <argument source="PREVIOUS" operand="SRC_COLOR"/>
              </RGB>
              <alpha operator="MODULATE">
                <argument source="PRIMARY" operand="SRC_ALPHA"/>
                <argument source="PREVIOUS" operand="SRC_ALPHA"/>
              </alpha>
            </texcombiner>
          </value>
        </texture_pipeline>
      </states>
    </pass>
  </technique>
</profile_GLES>
```

```

        </texcombiner>
    </value>
</texture_pipeline>
</states>
</pass>
</technique>
</profile_GLES>

```

5.8.52 profile_GLES2

Category: Profiles

Profile: **GLES2**

5.8.52.1 General

Declares platform-specific data types and **<technique>**s for OpenGL ES 2.0.

5.8.52.2 Concepts

<profile_GLES2> provides support for OpenGL ES 2.0 (GLES2). This profile's structure is similar to other shader-based profiles, such as Cg and GLSL, but focuses on the scope and details of OpenGL ES 2.0.

5.8.52.3 Attributes

The **<profile_GLES2>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
language	xs:NCName	The shading language that is used. Current valid languages are GLSL-ES and CG. Required.
platforms	list_of_names_type	The type of platform. These are vendor-defined character strings that indicates the platforms or capability targets for the technique. Enables support for multiple OpenGL ES 2.0 platforms. This may target a specific piece of hardware or a hardware family. Optional.

5.8.52.4 Related elements

The **<profile_GLES2>** element relates to the following elements:

Parent elements	effect
Child elements	See the following subclause.
Other	None

5.8.52.5 Child elements

Child elements shall appear in the following order if present, with the following exception:

`<code>` and `<include>` are interchangeable in the order

Name/example	Description	Default	Occurrences
<code><asset></code>	For resource management tracking. See main entry in Core.	N/A	0 or 1
<code><code></code>	An embedded block of source code. See main entry.	N/A	0 or more
<code><include></code>	A block of source code referenced by URL. See main entry.	N/A	0 or more
<code><newparam></code>	Declarations of new parameters to feed the shaders. See main entry.	N/A	0 or more
<code><technique></code> (FX)	A primary or alternative approach to rendering the profile. Typically, LODs. See main entry. See main entry for attributes and description and the following subclause for child element details.	N/A	1 or more
<code><extra></code>	A method for storing extension data beyond the COLLADA schema definition. See main entry in Core.	N/A	0 or 1

5.8.52.6 Child elements for `<profile_GLES2>` / `<technique>`

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><asset></code>	See main entry in Core.	N/A	0 or 1
<code><annotate></code>	See main entry.	N/A	0 or more
<code><pass></code>	See main entry.	N/A	1 or more
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.8.52.7 Details

Some unique characteristics of the GLES2 API are reflected in this profile:

- Unlike Cg API and Direct3D9 shader objects, which are compiled and used directly, GLES2 API shader objects are compiled and then linked by the user into a program object.
- The API supports both source code and binary shaders.
 - Source code is not necessarily required to be GLSL ES due to support for binary shaders.
- GLES2 supports only a limited subset of the OpenGL 2.x API on the PC. It has been stripped down to supply only shader-based rendering and the appropriate render states not controlled by shader source code.

- For more information on GLES2, visit OpenGL ES webpage [8] or the documentation for the specific vendor for the targeted platform.
- One of the most important differences of GLES2 compared to other COLLADA FX profiles is the way in which shaders and programs are put together. Shader source code consists of a list of sources. Segments of source code can be any of the following in any combination or order:
 - Sharable embedded **<code>**
 - Sharable referencing **<include>**
 - Code inline in the list, such as **#define** commands that set up an uber-shader, allowing users to reuse sharable source code segments by specializing uber-shaders with local inlined **#define** commands.

5.8.52.8 Example

For an additional example, refer to Annex B(informative)

Profile GLSL and GLES2 examples.

```

<profile_GLES2 language="GLSL-ES">
  <code sid="diffuseVS">
    attribute vec3 sv_Vertex;
    attribute vec3 sv_Normal;
    uniform mat4 wvp;
    uniform mat4 worldView;
    varying vec3 FragmentNormal;
    void main(void)
    {
      gl_Position = wvp * vec4(sv_Vertex.xyz, 1.0);
      FragmentNormal = mat3(worldView) * sv_Normal.xyz;
    }
  </code>
  <code sid="hemiFS">
    uniform vec4 surfColor;
    uniform vec4 skyColor;
    uniform vec4 groundColor;
    uniform float hemiContrib;
    varying vec3 FragmentNormal;
    void main (void)
    {
      vec3 normal = normalize(FragmentNormal);

      float NdotL = max( 0.0, dot( normal, vec3(0.0, 0.0, 1.0) ) );
      float NdotUp = dot( normal, vec3(0.0, 1.0, 0.0) );

      float mixer = (NdotUp + 1.0) * 0.5;
      vec4 diffuse = NdotL * surfColor;
      vec4 hemiColor = NdotL * mix(groundColor, skyColor, mixer);

      gl_FragColor = diffuse + hemiContrib * hemiColor;
    }
  </code>
  <newparam sid="wvp">
    <semantic>WorldViewProjection</semantic>
    <mat4>1 2 3 4 0 1 0 0 0 0 1 0 0 0 0 1 </mat4>
  </newparam>

```

```

<newparam sid="worldView">
  <semantic>WorldView</semantic>
  <mat4>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 </mat4>
</newparam>
<newparam sid="surfColor">
  <semantic>COLOR</semantic>
  <vec4>0.8 0.8 0.8 0 </vec4>
</newparam>
<newparam sid="skyColor">
  <semantic>COLOR</semantic>
  <vec4>0 0 0.5 0 </vec4>
</newparam>
<newparam sid="groundColor">
  <semantic>COLOR</semantic>
  <vec4>0 0.5 0 0 </vec4>
</newparam>
<newparam sid="hemiContrib">
  <float>1</float>
</newparam>
<technique sid="t0">
  <pass sid="p0">
    <states>
      <depth_test_enable value="true"/>
      <depth_func value="Less"/>
      <cull_face_enable value="true"/>
      <cull_face value="Back"/>
      <front_face value="CCW"/>
    </states>
    <program>
      <shader stage="VERTEX">
        <sources><import ref="diffuseVS"/></sources>
      </shader>
      <shader stage="FRAGMENT">
        <sources> <import ref="hemiFS"/> </sources>
      </shader>
      <bind_uniform symbol="wvp">
        <param ref="wvp"/>
      </bind_uniform>
      <bind_uniform symbol="worldView">
        <param ref="worldView"/>
      </bind_uniform>
      <bind_uniform symbol="surfColor">
        <param ref="surfColor"/>
      </bind_uniform>
      <bind_uniform symbol="skyColor">
        <param ref="skyColor"/>
      </bind_uniform>
      <bind_uniform symbol="groundColor">
        <param ref="groundColor"/>
      </bind_uniform>
      <bind_uniform symbol="hemiContrib">
        <param ref="hemiContrib"/>
      </bind_uniform>
    </program>
    <evaluate/>
  </pass>
</technique>
</profile_GLES2>

```

5.8.53 profile_GLSL

Category: Profiles

Profile: GLSL

5.8.53.1 General

Declares platform-specific data types and **<technique>**s for OpenGL Shading Language.

5.8.53.2 Concepts

The **<profile_GLSL>** elements encapsulate all the platform-specific values and declarations for a particular profile. In **<effect>** scope, parameters are available to all platforms, but parameters declared inside a **<profile_GLSL>** block are available only to shaders that are also inside that profile.

The **<profile_GLSL>** element defines the clear interface between concrete, platform-specific data types and the abstract COLLADA data types used in the rest of the document. Parameters declared outside of this barrier may require casting when used inside a **<profile_GLSL>** block.

For more information, see “Using profiles for platform-specific effects” in 5.7 Getting started with COLLADA FX.

5.8.53.3 Attributes

<profile_GLSL> has the following attributes:

id	xs:ID	A text string containing the unique identifier of the element. This value shall be unique within the instance document. Optional.
platform	xs:NMTOKEN	The type of platform. This is a vendor-defined character string that indicates the platform or capability target for the technique. Optional. The default is “PC”.

5.8.53.4 Related elements

The **<profile_GLSL>** elements relate to the following elements:

Parent elements	effect
Child elements	See the following subclauses.
Other	None

5.8.53.5 Child elements

Child elements shall appear in the following order if present, with the following exception:

- `<include>` and `<code>` are interchangeable.

Name/example	Description	Default	Occurrences
<code><asset></code>	See main entry in Core.	N/A	0 or 1
<code><code></code>	See main entry.	N/A	0 or more
<code><include></code>	See main entry.	N/A	0 or more
<code><newparam></code>	Creates a new parameter from a constrained set of types recognizable by all platforms – <code><float></code> , <code><float2></code> , <code><float3></code> , <code><float4></code> , <code><surface></code> and <code><sampler2D></code> , with an additional semantic. See main entry. Example: <code><newparam sid="mySID"></code> <code><semantic></code> DIFFUSECOLOR <code></semantic></code> <code><float3></code> 1 2 3 <code></float3></code> <code></newparam></code>	N/A	0 or more
<code><technique></code> (FX)	Declares a technique for this effect. See main entry for attributes and description and the following subclause for child element details.	N/A	1 or more
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.8.53.6 Child elements for `<profile_GLSL>` / `<technique>`

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><asset></code>	See main entry in Core.	N/A	0 or 1
<code><annotate></code>	See main entry.	N/A	0 or more
<code><pass></code>	See main entry.	N/A	1 or more
<code><extra></code>	See main entry in Core.	N/A	0 or more

5.8.53.7 Example

See : Annex B

5.8.54 program

Category: Shaders

Profile: CG, GLSL, GLES2

5.8.54.1 General

Links multiple shaders together to produce a pipeline for geometry processing.

5.8.54.2 Concepts

Describes how to create shaders, such as a vertex shader and a fragment shader. Additionally, this describes how to link them to produce a program and bind them to effect parameters for GLES2 and GLSL (for Cg shaders, bind to effect parameters instead of programs).

5.8.54.3 Attributes

The **<program>** element has no attributes.

5.8.54.4 Related elements

The **<program>** element relates to the following elements:

Parent elements	pass
Child elements	See the following subclauses.
Other	None

5.8.54.5 Child elements in CG scope

Within the scope of **<profile_CG>**, child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<shader>	Setup and compilation information for shaders such as vertex and pixel shaders. See main entry.	N/A	0 or more

5.8.54.6 Child elements in GLSL scope

Within the scope of **<profile_GLSL>**, child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<shader>	Setup and compilation information for shaders such as vertex and pixel shaders. See main entry.	N/A	0 or more
<bind_attribute>	Information for binding the shader variables to effect parameters. See main entry.	N/A	0 or more
<bind_uniform>	Binds a uniform shader variable to a parameter or a value. See main entry.	N/A	0 or more

5.8.54.7 Child elements in GLES2 scope

Within the scope of `<profile_GLES2>`, child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><shader></code>	See above.	N/A	0 or more
<code><linker></code>	Information regarding shaders together or capturing the results of linking. See main entry.	N/A	0 or more
<code><bind_attribute></code>	Information for binding the shader variables to effect parameters. See main entry.	N/A	0 or more
<code><bind_uniform></code>	Binds a uniform shader variable to a parameter or a value. See main entry.	N/A	0 or more

5.8.54.8 Example

```
<program>
  <shader stage="VERTEX"> ... </shader>
  <shader stage="FRAGMENT"> ... </shader>
</program>
```

5.8.55 render

Category: Rendering

Profile: **External**

5.8.55.1 General

Describes one effect pass to evaluate a scene.

5.8.55.2 Concepts

This element indicates one pass of rendering for camera lens or screen post-processing. Rendering can be straightforward without a particular material or effect such as layered rendering, or it can add special effects, typically called postprocessing effects, lens effects, or scene effects such as blur, bloom, or depth of field.

Within this element you can also change your camera and the layers of the scene that you are rendering for each pass.

5.8.55.3 Attributes

The `<render>` element has the following attributes:

name	xs:token	The text string name of this element. Optional.
sid	sid_type	A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.
camera_node	xs:anyURI	Refers to a node that contains a camera describing the viewpoint from which to render this compositing step. Optional.

5.8.55.4 Related elements

The **<render>** element relates to the following elements:

Parent elements	evaluate_scene
Child elements	See the following subclauses.
Other	None

5.8.55.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<layer>	Specifies which layer or layers to render in this compositing step while evaluating the scene. Contains layer names of type <code>xs:NCName</code> . This element has no attributes.	None	0 or more
<instance_material> (rendering)	Specifies which effect to render in this compositing step while evaluating the scene. See main entry.	N/A	0 or 1
<extra>	See main entry in Core.	N/A	0 or more

5.8.55.6 Example

See **<visual_scene>**.

5.8.56 RGB

Category: Texturing

Profile: **GL ES**

5.8.56.1 General

Defines the RGB portion of a **<texture_pipeline>** command for combiner-mode texturing operation.

5.8.56.2 Concepts

See **<texcombiner>** for details about assignments and overall concepts.

5.8.56.3 Attributes

The **<RGB>** element has the following attributes:

operator	Enumeration	Infers the use of <code>glTexEnv(TEXTURE_ENV, COMBINE_RGB, operator)</code> . See <texcombiner> for details. Valid values are: REPLACE MODULATE ADD ADD_SIGNED INTERPOLATE SUBTRACT DOT3_RGB DOT3_RGBA
scale	float_type	Infers the use of <code>glTexEnv(TEXTURE_ENV, RGB_SCALE, scale)</code> . See <texcombiner> for details.

5.8.56.4 Related elements

The **<RGB>** element relates to the following elements:

Parent elements	texcombiner
Child elements	See the following subclauses.
Other	None

5.8.56.5 Child elements

Name/example	Description	Default	Occurrences
<argument>	Sets up the arguments required for the given operator to be executed. See main entry.	None	1 to 3

5.8.56.6 Details

See **<texcombiner>** for details.

5.8.56.7 Example

See **<texture_pipeline>**.

5.8.57 sampler1D

Category: Texturing

Profile: **CG, GLSL**

5.8.57.1 General

Declares a one-dimensional texture sampler.

5.8.57.2 Attributes

The **<sampler1D>** element has no attributes.

5.8.57.3 Related elements

The **<sampler1D>** element relates to the following elements:

Parent elements	In core: newparam , setparam In FX: array
Child elements	See fx_sampler_common .
Other	None

5.8.57.4 Details

Use of **<wrap_t>** and **<wrap_p>** has no effect on the results because 1D samplers do not use the *t* and *p* coordinate axes.

5.8.57.5 Example

This example repeats a texture across a surface regardless of any UVs exceeding the 0-to-1 range. It linearly magnifies the texture if it needs to be enlarged. It does trilinear filtering if the texels are smaller than the pixels being rasterized. This reads from a one-dimensional surface, that is, a surface that is *N* by 1 (height=1).

```
<sampler1D>
  <wrap_s>WRAP</wrap_s>
  <minfilter>LINEAR</minfilter>
  <magfilter>LINEAR</magfilter>
</sampler1D>
```

5.8.58 sampler2D

Category: Texturing

Profile: External, Effect, CG, COMMON, GLES2, GLSL

5.8.58.1 General

Declares a two-dimensional texture sampler.

5.8.58.2 Attributes

The **<sampler2D>** element has no attributes.

5.8.58.3 Related elements

The **<sampler2D>** element relates to the following elements:

Parent elements	In core: newparam , setparam In FX: array
Child elements	See fx_sampler_common .
Other	None

5.8.58.4 Details

Use of **<wrap_p>** has no effect on the results because 1D samplers do not use the *p* coordinate axis.

5.8.58.5 Example

This is an example of the most common sampler type. It repeats a texture across a surface regardless of any UVs exceeding the 0-to-1 range. It linearly magnifies the texture if it needs to be enlarged. It does trilinear filtering if the texels are smaller than the pixels being rasterized.

```

<sampler2D>
  <wrap_s>WRAP</wrap_s>
  <wrap_t>WRAP</wrap_t>
  <minfilter>LINEAR</minfilter>
  <magfilter>LINEAR</magfilter>
</sampler2D>

```

5.8.59 sampler3D

Category: Texturing

Profile: External, Effect, CG, GLES2, GLSL

5.8.59.1 General

Declares a three-dimensional texture sampler.

5.8.59.2 Attributes

The **<sampler3D>** element has no attributes.

5.8.59.3 Related elements

The **<sampler3D>** element relates to the following elements:

Parent elements	In core: newparam , setparam In FX: array
Child elements	See fx_sampler_common .
Other	None

5.8.59.4 Example

This example repeats a texture across a surface regardless of any UVs exceeding the 0-to-1 range. It linearly magnifies the texture if it needs to be enlarged. It does trilinear filtering if the texels are smaller than the pixels being rasterized.

This example does this typical sampling operation from a three-dimensional texture, i.e. from a volume. This is common for reading from noise, patterns such as wood, and medical imaging.

```

<sampler3D>
  <wrap_s>WRAP</wrap_s>
  <wrap_t>WRAP</wrap_t>
  <wrap_p>WRAP</wrap_p>
  <minfilter>LINEAR</minfilter>
  <magfilter>LINEAR</magfilter>
</sampler3D>

```

5.8.60 samplerCUBE

Category: Texturing

Profile: External, Effect, CG, COMMON, GLES2, GLSL

5.8.60.1 General

Declares a texture sampler for cube maps.

5.8.60.2 Attributes

The **<samplerCUBE>** element has no attributes.

5.8.60.3 Related elements

The **<samplerCUBE>** element relates to the following elements:

Parent elements	In core: newparam , setparam In FX: array
Child elements	See fx_sampler_common .
Other	None

5.8.60.4 Details

Use of **<wrap_p>** has no effect on the results because 1D samplers do not use the p coordinate axis.

5.8.60.5 Example

This example reads from a cube map surface. The shader passes in a 3D vector that is a normal. That normal points to a location on one of the six sides of a cube map. Samples around the coordinate that it points to are filtered and returned.

```
<samplerCUBE>
  <wrap_s>WRAP</wrap_s>
  <wrap_t>WRAP</wrap_t>
  <minfilter>LINEAR</minfilter>
  <magfilter>LINEAR</magfilter>
</samplerCUBE>
```

5.8.61 samplerDEPTH

Category: Texturing

Profile: External, Effect, CG, COMMON, GLSL

5.8.61.1 General

Declares a texture sampler for depth maps.

5.8.61.2 Attributes

The **<samplerDEPTH>** element has no attributes.

5.8.61.3 Related elements

The **<samplerDEPTH>** element relates to the following elements:

Parent elements	In core: newparam , setparam In FX: array
Child elements	See fx_sampler_common .
Other	None

5.8.61.4 Details

Use of **<wrap_p>** has no effect on the results because 1D samplers do not use the p coordinate axis.

5.8.61.5 Example

This example repeats a texture across a surface regardless of any UVs exceeding the 0-to-1 range. It linearly magnifies the texture if it needs to be enlarged. It does trilinear filtering if the texels are smaller than the pixels being rasterized. If the surface is depth data, it performs percentage closest filtering. This technique provides better results when sampling depth maps for uses such as shadow maps.

```
<samplerDEPTH>
  <wrap_s>WRAP</wrap_s>
  <wrap_t>WRAP</wrap_t>
  <minfilter>LINEAR</minfilter>
  <magfilter>LINEAR</magfilter>
</samplerDEPTH>
```

5.8.62 samplerRECT

Category: Texturing

Profile: External, Effect, CG, COMMON, GLSL

5.8.62.1 General

Declares a RECT texture sampler.

5.8.62.2 Concepts

RECT textures are a OpenGL extension; they are not the same as non-square 2D textures. It is typically used as a render target or screen space processing, not as a general non-square replacement for **<sampler2D>**.

5.8.62.3 Attributes

The **<samplerRECT>** element has no attributes.

5.8.62.4 Related elements

The **<samplerRECT>** element relates to the following elements:

Parent elements	In core: newparam , setparam In FX: array
Child elements	See fx_sampler_common .
Other	None

5.8.62.5 Details

RECT reflects OpenGL RECT samplers. It is not supported in DirectX. RECT is two dimensional. It does not support MIP-mapping. Samples use a float2_type that is in the range [0-to-width, 0-to-height] as opposed to the 2D 0-to-1 range.

5.8.62.6 Example

RECT samplers are very limited. They do not support MIP-mapping, so this trivial example is actually the most common usage:

```
<samplerRECT>
  <instance_image url="myRenderableSurface"/>
</samplerRECT>
```

5.8.63 sampler_image

Category: Parameters

Profile: **External**

5.8.63.1 General

Instantiates an image targeted for samplers.

5.8.63.2 Concepts

This is not a sampler type but is, instead, an element used to modify an existing sampler. The sampler **<newparam>** identified by the parent **<setparam>** receives the instantiated image.

See **<instance_image>** for more details. This derived type has no specific extension but was renamed for clarity in this situation.

5.8.63.3 Attributes

See **<instance_image>**.

5.8.63.4 Related elements

The **<sampler_image>** element relates to the following elements:

Parent elements	<instance_effect>/<setparam>
Child elements	See instance_image
Other	None

5.8.63.5 Example

```
<material id="foo-smiley">
  <instance_effect url="foo">
    <setparam ref="bar">
      <sampler_image url="smiley-1"/>
    </setparam>
  </instance_effect>
</material>
```

5.8.64 sampler_states

Category: Materials

Profile: N/A

5.8.64.1 General

Allows users to modify an effect's sampler state from a material.

5.8.64.2 Concepts

This element is derived from the sampler base type, **fx_sampler_states**. See **fx_sampler_common** for a list of valid states. This includes all elements except **<instance_image>**. A material's **<setparam>** ref attribute points at an effect's **<newparam>** containing a **<sampler*>**. This modifies only the sampling state of the sampler, whereas **<sampler_image>** is used to change the sampler's **<instance_image>**, which is the more common operation.

Most effect authoring tools will not support this feature because it is not common in earlier FX frameworks where it was not possible to modify the sampler state outside of the effect. It is included in COLLADA to be forward-looking, based on flexible game-engine technology and that GL historically carries the sampler states with the texture object.

5.8.64.3 Attributes

The **<sampler_states>** element has no attributes.

5.8.64.4 Related elements

The **<sampler_states>** element relates to the following elements:

Parent elements	setparam
Child elements	See fx_sampler_common .
Other	None

5.8.64.5 Example

```
<material>
  <instance_effect url="simpleTexturing">
    <setparam ref="texParam">
      <sampler_states>
        <wrap_s>WRAP</wrap_s>
        <wrap_t>WRAP</wrap_t>
      </sampler_states>
    </setparam>
    <setparam ref="texParam">
      <sampler_image url="smiley.jpg"/>
    </setparam>
  </instance_effect>
</material>
```

5.8.65 semantic

Category: Parameters

Profile: External, Effect, CG, COMMON, GLES, GLES2, GLSL

5.8.65.1 General

Provides metadata that describes the purpose of a parameter declaration.

5.8.65.2 Concepts

Semantics describe the intention or purpose of a parameter declaration in an effect, using an overloaded concept. Semantics have been used historically to describe three different types of metadata:

- A hardware resource allocated to a parameter, e.g. **TEXCOORD2**, **NORMAL**.
- A value from the scene graph or graphics API that is being represented by this parameter, e.g. **MODELVIEWMATRIX**, **CAMERAPOS**, **VIEWPORTSIZE**.
- A user-defined value that will be set by the application at run time when the effect is being initialized, e.g. **DAMAGE_PERCENT**, **MAGIC_LEVEL**.

Semantics are used by the **<instance_geometry>** declaration inside **<node>** to bind effect parameters to values and data sources that can be found in the scene graph, using the **<bind_material>** mechanism used to disambiguate this mapping.

5.8.65.3 Attributes

The **<semantic>** element has no attributes.

5.8.65.4 Related elements

The **<semantic>** element relates to the following elements:

Parent elements	newparam
Child elements	None
Other	None

5.8.65.5 Details

There is currently no standard set of semantics. This element can contain any `xs:NCName` defined by your application.

See “The common profile” in 5.3 Schema concepts.

5.8.65.6 Example

```
<newparam sid="diffuseColor">
  <annotate name="UIWidget"><string>none</string></annotate>
  <semantic>DIFFUSE</semantic>
  <modifier>EXTERN</modifier>
  <float3> 0.30 0.56 0.12 </float>
</newparam>
```

5.8.66 shader

Category: Shaders

Profile: CG, GLES2, GLSL

5.8.66.1 General

Declares and prepares a shader for execution in the rendering pipeline of a **<pass>**.

5.8.66.2 Concepts

Executable shaders are small functions or programs that execute at a specific stage in the rendering pipeline. Shaders can be built from preloaded, precompiled binaries or dynamically generated at run time from embedded source code. The **<shader>** declaration holds all the settings necessary for compiling a shader and binding values or predefined parameters to the uniform inputs.

COLLADA FX allows declarations of both source code shaders and precompiled binaries, depending on support from the FX Runtime. Precompiled binary shaders already have the target profile specified for them at compile time, but to allow COLLADA readers to validate declarations involving precompiled shaders without having to load and parse the binary headers, profile declarations are still required.

Previously defined parameters, shader source, and binaries are considered merged into the same namespace / symbol table/source code string so that all symbols and functions are available to shader declarations, allowing common functions to be used in several shaders in a **<technique>**, for example, common lighting code. FX Runtimes that use the concept of “translation units” are allowed to name each source code block to break up the namespace.

Shaders with uniform input parameters can bind either previously defined parameters or literal values to these values during shader declaration, allowing compilers to inline literal and constant values.

5.8.66.3 Attributes

The **<shader>** element has the following attributes:

stage	Enumeration	Required. In which pipeline stage this programmable shader is designed to execute. Valid values are: TESSELATION , VERTEX , GEOMETRY , and FRAGMENT .

5.8.66.4 Related elements

The **<shader>** element relates to the following elements:

Parent elements	program
Child elements	See the following subclauses.
Other	None

5.8.66.5 Child elements in CG scope

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<sources>	Concatenates the source code for the shader from one or more sources. See main entry.	N/A	1
<compiler>	Compiler information for one or more platforms. See main entry.	None	0 or more
<bind_uniform>	See main entry.	N/A	0 or more

5.8.66.6 Child elements in GLES2 scope

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<sources>	Concatenates the source code for the shader from one or more sources. See main entry.	N/A	1
<compiler>	Compiler information for one or more platforms. See main entry.	None	0 or more
<extra>	See main entry in Core.	N/A	0 or more

5.8.66.7 Child elements in GLSL scope

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<sources>	Concatenates the source code for the shader from one or more sources. See main entry.	N/A	1
<extra>	See main entry in Core.	N/A	0 or more

5.8.66.8 Example

The following is an example for **<profile CG>**:

```
<shader stage="VERTEX">
  <sources entry="main">
    <import ref="thinFilm2"/>
  </sources>
  <compiler platform="PC" target = "ARBVP1" />
  <bind_uniform symbol="lightpos">
    <param ref="LightPos_03"/>
  </bind_uniform>
</shader>
```

5.8.67 sources

Category: Shaders

Profile: CG, GLES2, GLSL

5.8.67.1 General

Concatenates the source code for a shader from one or more sources.

5.8.67.2 Concepts

Sometimes shader source code cannot be contained in only one included file or one embedded code block. Instead, a user may want to combine common sets of code blocks.

As one example, users could author an uber-shader and bring it into the **<sources>** using an **<import>**, then add one or more **<inline>** blocks above that **<import>** to customize the uber-shader with **#defines**.

As another example, a user writes a pluggable main-function shader that defines a basic formula and relies on function calls for extensibility. The user can then use multiple child elements of the source to combine the function blocks with the main function.

5.8.67.3 Attributes

The **<sources>** element has the following attribute:

entry	xs:token	Required in CG scope; optional in GLES2 scope; not valid in others. Entry-function name for this shader. This identifies the name of the entry point after the child elements are concatenated. In GLES2, the default is "main".

5.8.67.4 Related elements

The **<sources>** element relates to the following elements:

Parent elements	shader
Child elements	See the following subclause.
Other	None

5.8.67.5 Child elements

Child elements can appear in any order, in any combination:

Name/example	Description	Default	Occurrences
<inline>	An <code>xs:string</code> containing code, such as a <code>#define</code> for an imported shader.	None	1 or more
<import ref="">	The <import> element itself contains no data. The required <code>ref</code> attribute contains the SID of a <code> or <include> element at the profile or effect level. The <code>ref</code> attribute is required. For details, see "Address syntax" in 5.3 Schema concepts.	None	0 or more

5.8.67.6 Example

```
<sources entry="main">
  <inline>#define DEBUG 1\n</inline>
  <inline>#define ENVIRONMENT_LOOKUP 1\n</inline>
  <inline>#define PROFILE PHONG\n</inline>
  <import ref="uber"/>
</sources>
```

5.8.68 states

Category: Rendering

Profile: CG, GLES, GLES2, GLSL

5.8.68.1 General

Contains all rendering states to set up for the parent pass.

5.8.68.2 Concepts

Different FX profiles have different sets of render states available for use within the **<pass>** element.

5.8.68.3 Attributes

The **<states>** element has no attributes.

5.8.68.4 Related elements

The **<states>** element relates to the following elements:

Parent elements	pass (in profile_CG , profile_GLES , profile_GLES2 , profile_GLSL)
Child elements	See the following subclause.
Other	None

5.8.68.5 Child elements

Child elements, representing render states, can appear in any combination, in any order.

Each render state – or its child elements if it has any, as shown in the render states table – has the following attributes:

value	type as specified in the following table	Provides a value specific to the render state. Either value or param, but not both, is required unless stated otherwise in the table.
param	sidref_type	Refers to the SID of a parameter whose value is to be used for the render state as an alternative to value. Not valid if value is specified. For details about SIDREFs, see “Address syntax” in 5.3 Schema concepts.
index	type as specified in the following table	Generally, this is a numeric value. Not every render state has this attribute, and its meaning varies depending on the render state; refer to the render states table. Required or Optional is also specified there.

For example:

```
<newparam sid= "someparam"... />
<setparam ref="someparam">1 1 1 0</setparam>
...
<states>
  <fog_color value="0 0 0 0" />
  <fog_enable = "true"/>
  <light_ambient value="1 1 1 0" index="0"/>
  <light_diffuse param="someparam" />
</states>
```

Further descriptions of the following render states are in the OpenGL specification. Refer Khronos Group's OpenGL Registry. [7]

The following Table 12 shows the render states for **<profile_CG>**, **<profile_GLSL>**, **<profile_GLES>**, and **<profile_GLES2>**. Render states are identical in all profiles except for differences noted for the **GLES** and **GLES2** profile.

Table 12 — Render states for profiles

Render states and their child elements	Valid values or types, and index attribute	GL ES	GL ES2
alpha_func func value	NEVER, LESS, LEQUAL, EQUAL, GREATER, NOTEQUAL, GEQUAL, ALWAYS Float value 0.0 – 1.0 inclusive	Yes	No
alpha_test_enable	Boolean	Yes	No
auto_normal_enable	Boolean	No	No
blend_color	float4_type	No	Yes
blend_enable	Boolean	Yes	Yes
blend_equation	FUNC_ADD, FUNC_SUBTRACT, FUNC_REVERSE_SUBTRACT, MIN, MAX	No	Yes
blend_equation_separate rgb alpha	Same as blend_equation values	No	Yes
blend_func src dest	(both src and dest) ZERO, ONE, SRC_COLOR, ONE_MINUS_SRC_COLOR, DEST_COLOR, ONE_MINUS_DEST_COLOR, SRC_ALPHA, ONE_MINUS_SRC_ALPHA, DST_ALPHA, ONE_MINUS_DST_ALPHA, CONSTANT_COLOR, ONE_MINUS_CONSTANT_COLOR, CONSTANT_ALPHA, ONE_MINUS_CONSTANT_ALPHA, SRC_ALPHA_SATURATE	Yes	Yes
blend_func_separate src_rgb dest_rgb src_alpha dest_alpha	Same as blend_func values	No	Yes
clip_plane	float4_type Index attribute specifies which clip plane. Required.	Yes bool4_type	No
clip_plane_enable	Boolean Index attribute specifies which clip plane. Optional.	Yes	No
color_logic_op_enable	Boolean	Yes	No
color_mask	bool4_type	Yes	Yes
color_material face mode	FRONT, BACK, FRONT_AND_BACK EMISSION, AMBIENT, DIFFUSE, SPECULAR, AMBIENT_AND_DIFFUSE	No	No
color_material_enable Enables or disables the use of <color_material> . That is, indicates when runtimes shall perform <code>glEnable (GL_COLOR_MATERIAL)</code> Or <code>glDisable (GL_COLOR_MATERIAL)</code> (or equivalents).	Boolean	Yes	No
cull_face	FRONT, BACK, FRONT_AND_BACK	Yes	Yes
cull_face_enable	Boolean	Yes	Yes
depth_bounds	float2_type	No	No
depth_bounds_enable	Boolean	No	No
depth_clamp_enable	Boolean	No	No

Render states and their child elements	Valid values or types, and index attribute	GL ES	GL ES2
depth_func	NEVER, LESS, LEQUAL, EQUAL, GREATER, NOTEQUAL, GEQUAL, ALWAYS	Yes	Yes
depth_mask	Boolean	Yes	Yes
depth_range	float2_type	Yes	Yes
depth_test_enable	Boolean	Yes	Yes
dither_enable	Boolean	Yes	Yes
fog_color	float4_type	Yes	No
fog_coord_src	FOG_COORDINATE, FRAGMENT_DEPTH	No	No
fog_density	float_type	Yes	No
fog_enable	Boolean	Yes	No
fog_end	float_type	Yes	No
fog_mode	LINEAR, EXP, EXP2	Yes	No
fog_start	float_type	Yes	No
front_face	CW, CCW	Yes	Yes
light_ambient	float4_type Index attribute specifies which light. Required.	Yes	No
light_constant_attenuation	float_type Index attribute specifies which light. Required.	Yes	No
light_diffuse	float4_type Index attribute specifies which light. Required.	Yes	No
light_enable	Boolean Index attribute specifies which light. Required.	Yes	No
light_linear_attenuation	float_type Index attribute specifies which light. Required.	Yes	No
light_model_ambient	float4_type	Yes	NO
light_model_color_control	SINGLE_COLOR, SEPARATE_SPECULAR_COLOR	No	No
light_model_local_viewer_enable	Boolean	No	No
light_model_two_side_enable	Boolean	Yes	No
light_position	float4_type Index attribute specifies which light. Required.	Yes	No
light_quadratic_attenuation	float_type Index attribute specifies which light. Required.	Yes	No
light_specular	float4_type Index attribute specifies which light. Required.	Yes	No
light_spot_cutoff	float_type Index attribute specifies which light. Required.	Yes	No
light_spot_direction	float3_type Index attribute specifies which light. Required.	Yes	No
light_spot_exponent	float_type Index attribute specifies which light. Required.	Yes	No
lighting_enable	Boolean	Yes	no
line_smooth_enable	Boolean	No	No
line_stipple	int2_type	No	No
line_stipple_enable	Boolean	No	No
line_width	float_type	Yes	Yes
logic_op	CLEAR, AND, AND_REVERSE, COPY, AND_INVERTED, NOOP, XOR, OR, NOR, EQUIV, INVERT, OR_REVERSE, COPY_INVERTED, NAND, SET	Yes	No
logic_op_enable	Boolean	No	No
material_ambient	float4_type	Yes	No
material_diffuse	float4_type	Yes	No
material_emission	float4_type	Yes	No

Render states and their child elements	Valid values or types, and index attribute	GLS	GLS2
material_shininess	float_type	Yes	No
material_specular	float4_type	Yes	No
model_view_matrix	float4x4_type	Yes	No
multisample_enable	Boolean	Yes	No
normalize_enable	Boolean	Yes	No
point_distance_attenuation	float3_type	Yes	No
point_fade_threshold_size	float_type	Yes	No
point_size	float_type	Yes	Yes
point_size_enable	Boolean	No	Yes
point_size_max	float_type	Yes	No
point_size_min	float_type	Yes	No
point_smooth_enable	Boolean	No	No
polygon_mode face mode	FRONT, BACK, FRONT_AND_BACK POINT, LINE, FILL	No	No
polygon_offset	float2_type	Yes	Yes
polygon_offset_fill_enable	Boolean	Yes	Yes
polygon_offset_line_enable	Boolean	No	No
polygon_offset_point_enable	Boolean	No	No
polygon_smooth_enable	Boolean	No	No
polygon_stipple_enable	Boolean	No	No
projection_matrix	float4x4_type	Yes	No
rescale_normal_enable	Boolean	Yes	No
sample_alpha_to_coverage_enable	Boolean	Yes	Yes
sample_alpha_to_one_enable	Boolean	Yes	No
sample_coverage value invert	float_type Boolean	No	Yes
sample_coverage_enable	Boolean	Yes	No
scissor	int4_type	Yes	Yes
scissor_test_enable	Boolean	Yes	Yes
shade_model	FLAT, SMOOTH	Yes	No
stencil_func func ref mask	NEVER, LESS, LEQUAL, EQUAL, GREATER, NOTEQUAL, GEQUAL, ALWAYS Unsigned byte Unsigned byte	Yes	Yes
stencil_func_separate front back ref mask	(For front and back) NEVER, LESS, LEQUAL, EQUAL, GREATER, NOTEQUAL, GEQUAL, ALWAYS Unsigned byte Unsigned byte	No	Yes
stencil_mask	int_type	Yes	Yes
stencil_mask_separate face mask	FRONT, BACK, FRONT_AND_BACK Unsigned byte	No	Yes
stencil_op fail zfail	(For fail, zfail, and zpass) KEEP, ZERO, REPLACE, INCR, DECR, INVERT, INCR_WRAP, DECR_WRAP	Yes	Yes

Render states and their child elements	Valid values or types, and index attribute	GL ES	GL ES2
zpass			
stencil_op_separate face fail zfail zpass	FRONT, BACK, FRONT_AND_BACK (For fail, zfail, and zpass:) KEEP, ZERO, REPLACE, INCR, DECR, INVERT, INCR_WRAP, DECT_WRAP	No	Yes
stencil_test_enable	Boolean	Yes	Yes
texture_env_color	float4_type Index attribute specifies which texture unit. Optional.	No (see <texture_p ipeline>)	No
texture_env_mode	xs:string Index attribute specifies which texture unit. Optional.	No (see <texture_p ipeline>)	No
texture_pipeline	String – the name of the <texture_pipeline> parameter.	Yes	No
texture1D	sampler1D type Index attribute specifies which texture unit. Required.	No	No
texture1D_enable	Boolean Index attribute specifies which texture unit. Optional.	No	No
texture2D	sampler2D type Index attribute specifies which texture unit. Required.	No (see <texture_p ipeline>)	No
texture2D_enable	Boolean Index attribute specifies which texture unit. Optional.	No (see <texture_p ipeline>)	No
texture3D	sampler3D type Index attribute specifies which texture unit. Required.	No	No
texture3D_enable	Boolean Index attribute specifies which texture unit. Optional.	No	No
textureCUBE	samplerCUBE type Index attribute specifies which texture unit. Required.	No	No
textureCUBE_enable	Boolean Index attribute specifies which texture unit. Optional.	No	No
textureDEPTH	samplerDEPTH type Index attribute specifies which texture unit. Required.	No	No
textureDEPTH	samplerDEPTH type Index attribute specifies which texture unit. Required.	No	No
textureDEPTH_enable	Boolean Index attribute specifies which texture unit. Optional.	No	No
textureRECT	samplerRECT type Index attribute specifies which texture unit. Required.	No	No
textureRECT_enable	Boolean Index attribute specifies which texture unit. Optional.	No	No

5.8.68.6 Example

```
<states>
  <depth_test_enable value="true"/>
  <depth_func value="Less"/>
  <cull_face_enable value="true"/>
  <cull_face value="Back"/>
  <front_face value="CCW"/>
</states>
```

5.8.69 stencil_clear

Category: Rendering

Profile: CG, GLES, GLES2, GLSL

5.8.69.1 General

Specifies whether a render target image is to be cleared, and which value to use.

5.8.69.2 Concepts

Before drawing a render target image may need resetting to a blank canvas or default. The `<stencil_clear>` declarations specify which value to use. If no clearing statement is included, the target image remains unchanged as rendering begins.

5.8.69.3 Attributes

The `<stencil_clear>` element has the following attribute:

index	xs:nonNegativeInteger	Which of the multiple render targets is being set. The default is 0. Optional.

5.8.69.4 Related elements

The `<stencil_clear>` element relates to the following elements:

Parent elements	<code>evaluate</code>
Child elements	None
Other	None

5.8.69.5 Details

This element contains an `xs:byte` that is the value used to clear a resource.

When this element exists inside a pass, it a cue to the runtime that a particular backbuffer or render-target resource shall be cleared. This means that all existing image data in the resource shall be replaced with the value provided. This puts the resource into a fresh and known state so that other operations with this resource execute as expected.

The index attribute identifies the resource to be cleared. An index of 0 identifies the primary resource. The primary resource may be the backbuffer or the override provided with an appropriate `<*_target>` element (`<color_target>`, `<depth_target>`, or `<stencil_target>`).

Direct3D® 9 class platforms have fairly restrictive rules for setting up MRTs; for example, only four color buffers, which shall be all of the same size and pixel format, and only one depth buffer and one stencil buffer active for all color buffers. The COLLADA FX declaration is designed to be looser in its restrictions, so an FX runtime shall validate that a particular MRT declaration in a `<pass>` is possible before attempting to apply it, and flag it as an error if it fails.

5.8.69.6 Example

```
<stencil_clear index="0">0</stencil_clear>
```

5.8.70 stencil_target

Category: Rendering

Profile: CG, GLES, GLES2, GLSL

5.8.70.1 General

Specifies which `<image>` will receive the stencil information from the output of this pass.

5.8.70.2 Concepts

Multiple Render Targets (MRTs) allow fragment shaders to output more than one value per pass, or to redirect the standard depth and stencil units to read from and write to arbitrary off-screen buffers. These elements tell the FX Runtime which previously defined render targets to use.

5.8.70.3 Attributes

The `<stencil_target>` element has the following attributes:

index	xs:nonNegativeInteger	Indexes one of the Multiple Render Targets. The default is 1. Optional.
slice	xs:nonNegativeInteger	Indexes a subimage inside a target <code><image></code> , including a single MIP-map level, a unique cube face, or a layer of a 3D texture. The default is 0. Optional.
mip	xs:nonNegativeInteger	The MIP level to target. The default is 0. Optional.
face	Enumeration	The cube face to target. Valid values are POSITIVE_X , NEGATIVE_X , POSITIVE_Y , NEGATIVE_Y , POSITIVE_Z , and NEGATIVE_Z . The default is POSITIVE_X . Optional.

5.8.70.4 Related elements

The **<stencil_target>** element relates to the following elements:

Parent elements	evaluate
Child elements	See the following subclause
Other	None

5.8.70.5 Child elements

Exactly one of the following child elements shall occur:

Name/example	Description	Default	Occurrences
<param> (reference)	References a sampler parameter to determine which image to use. See main entry.	None	0 or 1
<instance_image>	Instantiates a renderable image directly. See main entry.	N/A	0 or 1

5.8.70.6 Details

Direct3D® 9 class platforms have fairly restrictive rules for setting up MRTs; for example, only four color buffers, which shall be all of the same size and pixel format, and only one depth buffer and one stencil buffer active for all color buffers. The COLLADA FX declaration is designed to be looser in its restrictions, so an FX runtime shall validate that a particular MRT declaration in a **<pass>** is possible before attempting to apply it, and flag it as an error if it fails.

If no **<stencil_target>** is specified, the FX Runtime will use the default stencil buffer set for its platform.

5.8.70.7 Example

```
<newparam sid="surfaceTex">
  <sampler2D><instance_image url="renderTarget1"/></sampler2D>
</newparam>
<technique>
  <pass>
    <evaluate>
      <stencil_target>
        <param ref="surfaceTex"/>
      </stencil_target>
    </evaluate>
  </pass>
</technique>
```

5.8.71 technique (FX)

Category: Effects

Profile: CG, COMMON, GLES, GLES2, GLSL

5.8.71.1 General

Holds a description of the textures, samplers, shaders, parameters, and passes necessary for rendering this effect using one method.

For **<technique>** in elements other than **<profile_*>**, see “**<technique>** (Core).”

5.8.71.2 Concepts

Techniques hold all the necessary elements required to render an effect. Each effect can contain many techniques, each of which describes a different method for rendering that effect. There are three different scenarios for which techniques are commonly used.

- One technique might describe a high-LOD version while a second technique describes a low-LOD version of the same effect.
- Describe an effect in different ways and use validation tools in the FX Runtime to find the most efficient version of an effect for an unknown device that uses a standard API.
- Describe an effect under different game states, for example, a daytime and a night-time technique, a normal technique, and a “magic-is-enabled” technique.

5.8.71.3 Attributes

The **<technique>** element has the following attributes:

id	xs:ID	Optional. A text string containing the unique identifier of the element. This value shall be unique within the instance document.
sid	sid_type	Required. A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. For details about SIDs, see “Address syntax” in 5.3 Schema concepts.

5.8.71.4 Related elements

The **<technique>** element relates to the following elements:

Parent elements	profile_CG , profile_COMMON , profile_GLSL , profile_GLES , profile_GLES2
Child elements	See the following subclause.
Other	None

5.8.71.5 Child elements

Child elements vary by profile. See the parent element main entries for details. The following list summarizes valid child elements. The child elements shall appear in the following order if present, with the following exceptions: **<blinn>**, **<constant>**, **<lambert>**, **<phong>** are choices that can appear in any order in that position:

Name	profile_CG	profile_COMMON	profile_GLES	profile_GLES2	profile_GLSL	Occurrences
<asset>	yes	yes	yes	yes	yes	0 or 1
<annotate>	yes	-	yes	yes	yes	0 or more
<blinn>	-	yes	-	-	-	1
<constant> (FX)	-	yes	-	-	-	
<lambert>	-	yes	-	-	-	
<phong>	-	yes	-	-	-	
<pass>	yes	-	yes	yes	yes	1 or more
<extra>	yes	yes	yes	yes	yes	0 or more

5.8.71.6 Details

Techniques can be managed as first-class **<asset>**s, allowing tools to automatically generate techniques for effects and track their creation time, freshness, parent-child relationships, and the tools used to generate them.

5.8.71.7 Example

```
<effect id="BumpyDragonSkin">
  <profile_GLSL>
    <technique sid="HighLOD">
      ...
    </technique>
    <technique sid="LowLOD">
      ...
    </technique>
  </profile_GLSL>
</effect>
```

5.8.72 technique_hint

Category: Effects

Profile: **External**

5.8.72.1 General

Adds a hint for a platform of which technique to use in this effect.

5.8.72.2 Concepts

Shader editors require information on which technique to use by default when an effect is instantiated. Subject to validation, the suggested technique shall be used if your FX Runtime recognizes the platform string.

5.8.72.3 Attributes

The **<technique_hint>** element has the following attributes:

platform	xs:Name	Defines a string that specifies for which platform this hint is intended. Optional.
ref	xs:NCName	A reference to the name of the platform. Required.
profile	xs:NCName	A string that specifies for which API profile this hint is intended. It is the name of the profile within the effect that contains the technique. Profiles are constructed by appending this attribute's value to "profile". For example, to select profile_CG , specify profile="CG" . Optional.

5.8.72.4 Related elements

The **<technique_hint>** element relates to the following elements:

Parent elements	instance_effect
Child elements	None
Other	None

5.8.72.5 Example

```
<technique_hint platform="PS3" ref="HighLOD"/>
<technique_hint platform="OpenGLES" ref="twopass"/>
<technique_hint profile="CG" platform="GL" ref="HighLOD"/>
<technique_hint profile="GLES" platform="NOKIA_SW" ref="OneLight"/>
```

5.8.73 texcombiner

Category: Texturing

Profile: **GLES**

5.8.73.1 General

Defines a **<texture_pipeline>** command for combiner-mode texturing.

5.8.73.2 Concepts

This element sets the combiner states for the sampler to which it is assigned. It defines a set of texturing commands that will be converted into multitexturing operations using glTexEnv in regular and combiner mode.

See **<texture_pipeline>** for details about assignments and overall concepts.

5.8.73.3 Attributes

The **<texcombiner>** element has no attributes.

5.8.73.4 Related elements

The **<texcombiner>** element relates to the following elements:

Parent elements	texture_pipeline
Child elements	See the following subclause.
Other	None

5.8.73.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<constant value= ... param= ... > (combiner)	A static or parameter <code>float4_type</code> that may be passed to the OpenGL ES texturing unit. This value is combined with the color sampled from the texture to produce the final color output from that texturing unit. The equations are dependent on the <texture_pipeline> setup. The element contains no data. The arguments are optional; use only one: • value: Specifies a <code>float4_type</code> for <code>glTexEnv (TEXTURE_ENV, TEXTURE_ENV_COLOR, value)</code> . • param: Specifies the SID of a parameter that contains a <code>float4_type</code> value for the <code>glTexEnv</code> function.	N/A	0 or 1
<RGB>	Sets up the RGB component of the texture combiner command. See main entry.	N/A	0 or 1
<alpha>	Sets up the alpha component of the texture combiner command. See main entry.	N/A	0 or 1

5.8.73.6 Details

This is a complex stage command in a **<texture_pipeline>**. It is used for more customized operations than are available via **<texenv>**.

Read about **<texenv>** first, as the following information builds upon basic knowledge.

The **<RGB>** and **<alpha>** children elements are roughly the same; **<alpha>** is simply a subset of **<RGB>**.

While **<texenv>** allows you to specify one operator equation that will be used for the entire state, **<texcombiner>** adds flexibility by allowing the selection of different equations for the **<RGB>** channel and **<alpha>** channel.

The equations specified consist of up to 3 arguments. The arguments are specified in a series (*Arg0*, *Arg1*, *Arg2*). Each channel may specify its own **<argument>**s to the equation. Each **<argument>** specifies a source, operand, and sampler. The **<argument>** source attribute determines where the value for that equation's argument comes from:

- TEXTURE: From the sampler specified in the sampler attribute;

- **CONSTANT:** The **<texcombiner>** schema also allows each stage to have its own **<constant>** that may be used by the operators;
- **PRIMARY:** The incoming fragment color from the material;
- **PREVIOUS:** The incoming color from the previous texture pipeline stage.

The **<argument>** operand attribute determines which part of the value selected by the source will be used in the equation:

- **SRC_COLOR:** The RGB portion of the source;
- **ONE_MINUS_SRC_COLOR:** The per-component inverse (one minus) of SRC_COLOR;
- **SRC_ALPHA:** The alpha portion of the source;
- **ONE_MINUS_SRC_ALPHA:** The inverse (one minus) of SRC_ALPHA.

The following operator equations are available for texcombiners.

- **REPLACE:** $Arg0$
- **MODULATE:** $Arg0 + Arg1$
- **ADD:** $Arg0 + Arg1$
- **ADD_SIGNED:** $Arg0 + Arg1 - 0.5$
- **INTERPOLATE:** $Arg0 \times Arg2 + Arg1 \times (1 - Arg2)$
- **SUBTRACT:** $Arg0 - Arg1$
- **DOT3 (for <RGB> only):**

$$4 \times ((Arg0.r - 0.5) \times (Arg1.r - 0.5) + (Arg0.g - 0.5) \times (Arg1.g - 0.5) + (Arg0.b - 0.5) \times (Arg1.b - 0.5))$$

Lastly, each channel may be scaled. The RGB and alpha results of the equation are multiplied by the scale attribute (if specified) to compute the final values per channel.

The RGB and alpha channels are then placed back together as a four-component color as fed to the next stage as its PREVIOUS source.

For more information about any of these enumerations, refer to the OpenGL and OpenGL ES specifications.

Commands are eventually assigned to OpenGL ES hardware texture units. For this command type, each texture unit shall be changed into texture-combiner mode with the following command:

```
glTexEnv(TEXTURE_ENV, TEXTURE_ENV_MODE, COMBINE)
```

See **<texture_pipeline>** for more details about OpenGL ES hardware texture-unit assignments.

5.8.73.7 Example

See `<texture_pipeline>`.

5.8.74 texenv

Category: Texturing

Profile: GLES

5.8.74.1 General

Defines a `<texture_pipeline>` command for simple, non-combiner-mode texturing.

5.8.74.2 Concepts

This element sets the states for the sampler to which it is assigned.

See `<texture_pipeline>` for details about assignments and overall concepts.

5.8.74.3 Attributes

The `<texenv>` element has the following attributes:

operator	Enumeration	Optional. The operation to execute upon the incoming fragment. Valid values are: REPLACE MODULATE DECAL BLEND ADD
sampler	sidref_type	Optional. A reference to the SID of the sampler. For details, see “Address syntax” in 5.3 Schema concepts.

5.8.74.4 Related elements

The `<texenv>` element relates to the following elements:

Parent elements	<code>texture_pipeline</code>
Child elements	See the following subclause.
Other	None

5.8.74.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<code><constant value= ... param= ... ></code> (combiner)	A static or parameter <code>float4_type</code> that may be passed to the OpenGL ES sampler. This value is combined with the color sampled from the texture to produce the final color output from that sampler. The equations are dependent on the texture pipeline setup. The element contains no data. The arguments are optional; use only one: • <code>value</code>: Specifies a static <code>float4_type</code> for <code>glTexEnv(TEXTURE_ENV, TEXTURE_ENV_COLOR, value)</code>. • <code>param</code>: Specifies the SID of a parameter that contains a <code>float4_type</code> value for the <code>glTexEnv</code> command.	N/A	0 or 1

5.8.74.6 Details

This is a simple stage command in a `<texture_pipeline>`. It is used for very common operations with less setup.

Infers a call to `glTexEnv(TEXTURE_ENV, TEXTURE_ENV_MODE, operator)` for the sampler to which it is assigned.

The equation used by this operation is specified via the operator attribute. The following equations are available.

- **REPLACE**: The output is the value sampled by the `sampler` specified in the `sampler` attribute regardless of the alpha value.
- **MODULATE**: The output is the multiplication of the incoming value and the value sampled by the `sampler` specified in the `sampler` attribute.
- **DECAL**: The output is a blend (based on the alpha) of the color sampled from the `sampler` specified in the `sampler` attribute and the input of the previous stage or material.
- **BLEND**: The output is a blend (based on each color component) of the color sampled from the `sampler` specified in the `sampler` attribute and the input of the previous stage or material.
- **ADD**: The output is the addition of the input of the previous stage or material and the color sampled from the `sampler` specified in the `sampler` attribute.

For more information about any of these enumerations, refer to the OpenGL and OpenGL ES specifications.

5.8.74.7 Example

See `<texture_pipeline>`

5.8.75 texture_pipeline

Category: Texturing

Profile: GLES

5.8.75.1 General

Defines a set of texturing commands that will be converted into multitexturing operations using glTexEnv in regular and combiner mode.

5.8.75.2 Concepts

This element contains an ordered sequence of commands that together define all of the GLES multitexturing states.

Each command is eventually assigned to a texture unit:

- The <texcombiner> defines a texture-unit setup in combiner mode.
- The <texenv> element defines a texture-unit setup in non-combiner mode.

Commands are assigned to texture units in a late binding step based on texture-unit names and usage characteristics of commands.

A pass uses the <texture_pipeline> state to activate a fragment shader.

The ordering of the commands is 1:1 on which hardware texture unit they are assigned to. Depending on whether the texturing crossbar is supported (GL ES 1.1), the named texture-unit objects (<sampler2D>) from each command are assigned into appropriate hardware texture units. On GL ES 1.0, the sampler shall come from the existing unit, so two arguments with source="sampler" would not be valid unless they referenced the same <sampler2D> element (see the <RGB> and <alpha> elements in the "Examples" subclause).

5.8.75.3 Attributes

The <texture_pipeline> element has the following attribute:

sid	sid_type	Optional. A text string value containing the scoped identifier of this element. This value shall be unique within the scope of the parent element. For details, see "Address syntax" in 5.3 Schema concepts.

5.8.75.4 Related elements

The <texture_pipeline> element relates to the following elements:

Parent elements	states
Child elements	See the following subclause.
Other	None

5.8.75.5 Child elements

Child elements can appear in any order:

Name/example	Description	Default	Occurrences
<code><texcombiner></code>	See main entry.	N/A	0 or more
<code><texenv></code>	See main entry.	N/A	0 or more
<code><extra></code>	Contains application-specific additional information, such as OpenGL ES extensions. See main entry in Core.	N/A	0 or more

5.8.75.6 Details

The `<texture_pipeline>` creates a command sequence that describes how the user would like to combine material color, textures, and destination (backbuffer) data. Each stage in the texture pipeline is a command. There are two available command types:

- `<texcombiner>`
- `<texenv>`

The purpose of each stage in the pipeline is to combine existing and/or new input data to produce a new output color. The output color from each stage is then available as input data to the next stage in the pipeline.

Read the main entries for `<texenv>` and `<texcombiner>` to understand how the `<texture_pipeline>` is converted to GL calls.

The stage commands in the API typically translate to `glTexEnv` function calls. The main difference between these APIs and what is in COLLADA is that that a `glTexEnv` call is paired with a particular sampler. The COLLADA design has freed the operation from the sampler so that the author can design the operations more easily and enable the importer or conditioner to assign appropriate indices in a resolve operation. The index of the stage in the texturing pipeline becomes the index of the texture unit in which the `glTexEnv` calls are to be assigned. In OpenGL ES 1.0, the texture referenced by the unit attribute shall be placed into that same texture unit. In OpenGL ES 1.1, the textures can be placed anywhere to utilize the crossbar, although pairing them in the same way as in 1.0 may perform better on some hardware.

NOTE Some `<texture_pipeline>`s may resolve directly under OpenGL ES 1.1 but not under OpenGL ES 1.0 due to support for texturing crossbars. Additionally, some with a large number of textures may not resolve on certain hardware because of hardware limitations on the number of textures.

5.8.75.7 Example

```
<texture_pipeline>
  <value>
    <texcombiner>
      <constant> 0.0f, 0.0f, 0.0f, 1.0f </constant>
      <RGB operator="INTERPOLATE">
        <argument source="TEXTURE" operand="SRC_COLOR" sampler="gravel"/>
        <argument source="TEXTURE" operand="SRC_COLOR" sampler="grass"/>
        <argument source="TEXTURE" operand="SRC_ALPHA" sampler="transition"/>
      </RGB>
      <alpha operator="INTERPOLATE">
        <argument source="TEXTURE" operand="SRC_ALPHA" sampler="gravel"/>
      </alpha>
    </texcombiner>
  </value>
</texture_pipeline>
```

```

        <argument source="TEXTURE" operand="SRC_ALPHA" sampler="grass"/>
        <argument source="TEXTURE" operand="SRC_ALPHA" sampler="transition"/>
    </alpha>
</texcombiner>
<texcombiner>
    <RGB operator="MODULATE">
        <argument source="PRIMARY" operand="SRC_COLOR"/>
        <argument source="PREVIOUS" operand="SRC_COLOR"/>
    </RGB>
    <alpha operator="MODULATE">
        <argument source="PRIMARY" operand="SRC_ALPHA"/>
        <argument source="PREVIOUS" operand="SRC_ALPHA"/>
    </alpha>
</texcombiner>
    <texenv sampler="debug-decal-unit" operator="DECAL"/>
</value>
</texture_pipeline>

```

5.8.76 usertype

Category: Parameters

Profile: CG, GLES2

5.8.76.1 General

Creates an instance of a structured class for a parameter.

5.8.76.2 Concepts

Interface objects declare the abstract interface for a class of objects. Interface objects declare only the function signatures required and make no requirements for specific member data.

User types are concrete instances of these interfaces, structures that contain function declarations that provide implementations for each function declared in the interface along with any necessary member data.

User types can be declared only inside source code or included shaders, and so **<usertype>** declarations can take place only after all source code has been declared for a technique.

5.8.76.3 Attributes

The **<usertype>** element has the following attributes:

typename	xs:token	Required. The identifier for the struct declaration that will be found inside the current source-code translation unit.
source	xs:NCName	Optional in CG scope; not valid for GLES2. References a code or include element that defines the usertype.

5.8.76.4 Related elements

The **<usertype>** element relates to the following elements:

Parent elements	In core: newparam , setparam In FX: array , bind_uniform
Child elements	See the following subclause.
Other	None

5.8.76.5 Child elements

Name/example	Description	Default	Occurrences
<setparam>	Use a series of these to set the members by name. The ref attribute is relative to the parent usertype. See main entry in Core.	N/A	0 or more

5.8.76.6 Details

Elements of a **<usertype>** can be initialized at creation time in **<newparam>** by accessing each leaf node by name using a series of **<setparam>** declarations.

Some usertypes do not have data. They can be used only to implement interface functions.

5.8.76.7 Example

```
<include sid="simple_cg_source" url="simple.cgfx"/>
<newparam sid="lightsource0">
  <usertype typename="spotlight" source="simple_cg_source">
    <float3> 10 12 10 </float3>
    <float3> 0.3 0.3 0.114 </float3>
  </usertype>
</newparam>
<newparam sid="lightsource1">
  <usertype typename="spotlight" source="simple_cg_source">
    <setparam ref="position"><float3> 10 12 10 </float3></setparam>
    <setparam ref="direction"><float3> 0.3 0.3 0.114 </float3></setparam>
  </usertype>
</newparam>
```

5.9 B-Rep Reference

5.9.1 General

This clause covers the elements that compose the boundary representation “B-rep” portion of COLLADA animation.

5.9.2 Elements by category

5.9.2.1 General

This subclause lists elements in alphabetical order. The following tables list elements by category, for ease in finding related elements.

5.9.2.2 Geometry

Element	Description
brep	Describes a boundary representation (B-rep) structure.

5.9.2.3 Curves

Element	Description
circle	Describes a circle in 3D space.
curve	Describes a specific curve.
curves	Contains all curves that are used in the B-rep structure.
ellipse	Describes an ellipse in 3D space.
hyperbola	Describes a hyperbola in 3D space.
line	Describes a single line in 3D space.
nurbs	Describes a NURBS curve in 3D space.
parabola	Describes a parabola in 3D space.
surface_curves	Contains all parametric curves (pcurves) that are used in the B-rep structure.

5.9.2.4 Topology

Element	Description
edges	Describes the edges of a B-rep structure.
faces	Describes the faces of a B-rep structure.
pcurves	Specifies how an edge is represented in a face's parametric space.
shells	Describes the shells of a B-rep structure.
solids	Describes the solids of a B-rep structure.
wires	Describes the wires of a B-rep structure.

5.9.2.5 Surfaces

Element	Description
cone	Describes a conical surface.
cylinder	Describes an unlimited cylindrical surface.
nurbs_surface	Describes a NURBS surface in 3D space.
plane (in Physics)	Describes an infinite planar surface.
sphere (in Physics)	Describes a centred sphere primitive.
surface	Describes a specific surface.
surfaces	Contains all surfaces that are used in the B-rep structure.
swept_surface	Describes a surface by extruding or revolving a curve.
torus	Describes a torus in 3D space.

5.9.2.6 Transformation

Element	Description
orient	Describes the orientation of the object frame.
origin	Describes the origin of the object frame.

5.9.3 About B-rep in COLLADA

5.9.3.1 Overview

The **<brep>** element can be placed under the **<geometry>** element instead of **<mesh>**, **<convex_mesh>** or **<spline>**.

Boundary representation (B-rep) models are composed of two parts: topology and geometry as shown in Figure 20.

The topology specifies different entities (vertices, edges, and so on) that limit the corresponding unbounded geometry. Topological entities are:

- Vertex: The lowest topological entity;
- Edge: Bounded by two vertices;
- Wire: Bounded by, or built by, one or more edges;
- Face: Bounded by one or more wires;
- Shell: Bounded by, or built by, one or more faces;
- Solid: Bounded by one or more shells.

Geometrical entities are:

- Points: In 3D space;
- Curves: Such as lines, circles, or NURBS in 3D space;
- Parametric curves: Such as lines and circles in the parametric space of a circle;
- Surfaces: Such as cylinders, spheres, or planes in 3D space.

In boundary representation, topology and geometry are merged:

- A vertex is represented by a point;
- An edge is represented by a curve;
- A face is represented by a surface.

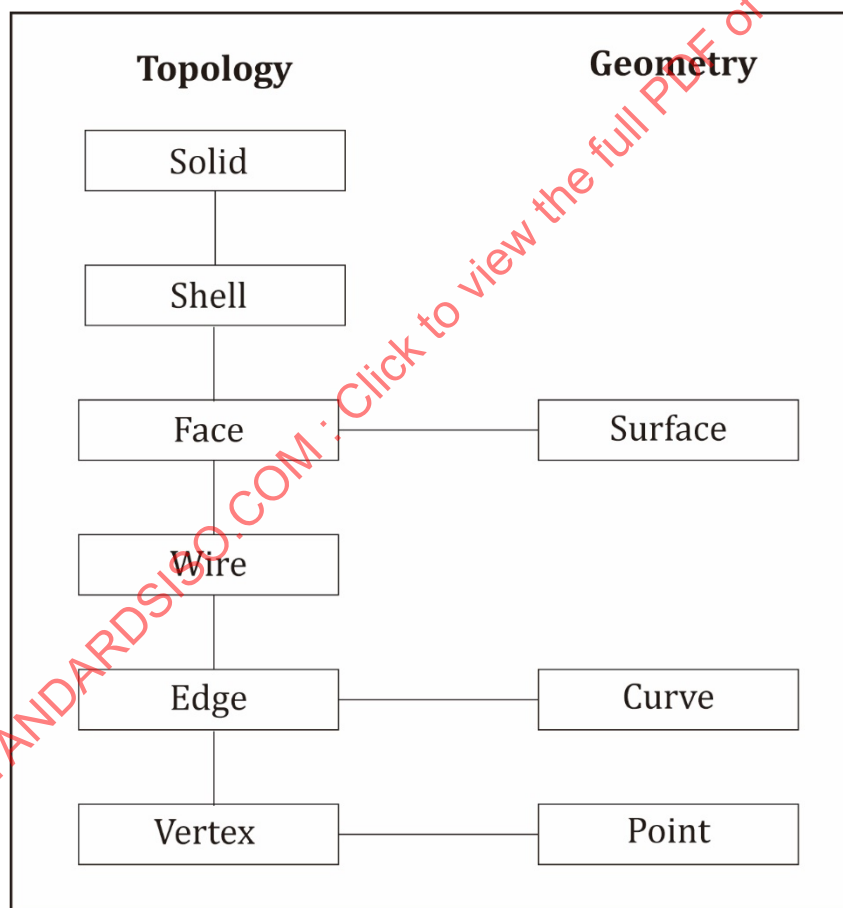


Figure 20 — Topology and geometry of B-Rep

5.9.3.2 Orientation

Every topological entity has a base orientation (example in Figure 21) that is defined by either its geometrical representation or the logical order when building it from its sub-entities:

Entity	Base orientation defined by
Vertex	Vertices have no orientation.
Edge	The order of its vertices. If the start and end vertices are the same (for example, for a circle) the orientation is defined by the base orientation of the curve.
Wire	The orientation of its edges.
Face	The orientation of its surface.
Shell	The orientation of its faces.

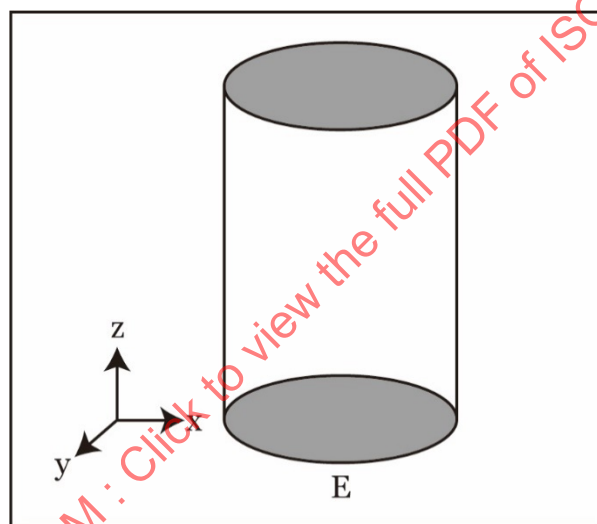


Figure 21 — Orientation

When using a topological entity in a next higher entity, its base orientation can be used (FORWARD) or its orientation can be reversed (REVERSED).

5.9.3.3 Parametric Curves

In general, a B-rep model is described sufficiently with topological entities. But most CAD file formats – such as STEP, IGES, or Parasolid – also describe the geometric representation of edges in the parametric space of the surface that is bounded by these edges. These curves are needed to exactly specify the boundary of a surface, which is helpful for rendering the faces.

5.9.3.3.1 Example of parametric curves

A closed cylinder has an edge E that is represented in 3D space as a circle. This edge limits two faces, the cylindrical one and the plane at the bottom. So, this edge E has two representations as a parametric curve, one on the plane and one on the cylindrical surface. In the parametric space of the plane, the curve is a circle, too. But in the parametric space of the cylinder, the curve is a line. This becomes clearer by imagining the cylinder slit and rolled out as a rectangle, as shown in the following Figure 22:

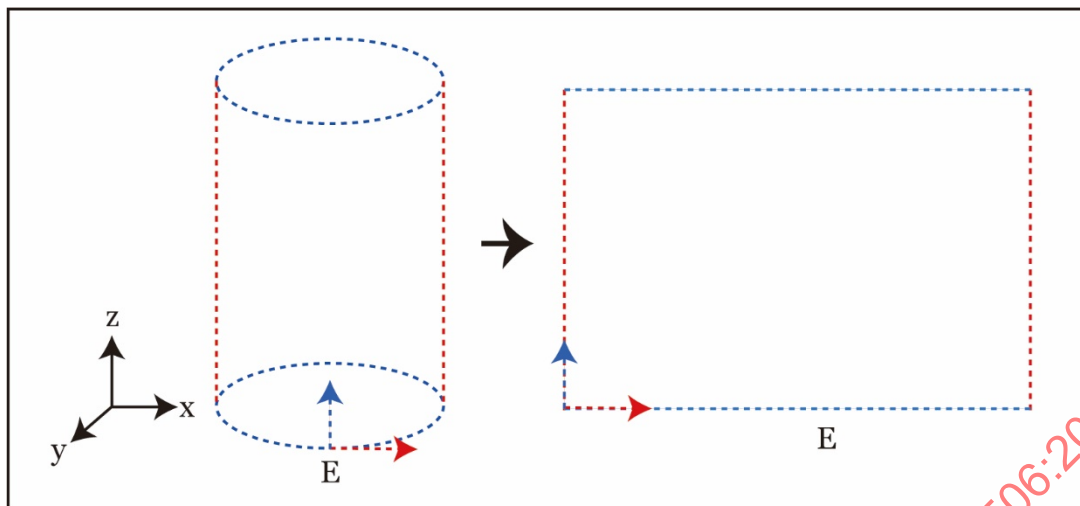


Figure 22 — Example of parametric curves

5.9.3.3.2 Degenerate edges

As seen in the preceding example, edges can have one 3D representation and one or more 2D representations. If an edge has only 2D representations, then it is called a degenerate edge.

An example of a model with degenerate edges is a sphere or a cone. The degenerate edge of a cone is its apex and, for a sphere, its poles.

The Figure 23 try to explain the creation of degenerate edges. The green edges in the parametric space of the sphere and of the cone do not appear in 3D space. They are the poles of the circle and the cone and – in 3D – they are just single points.

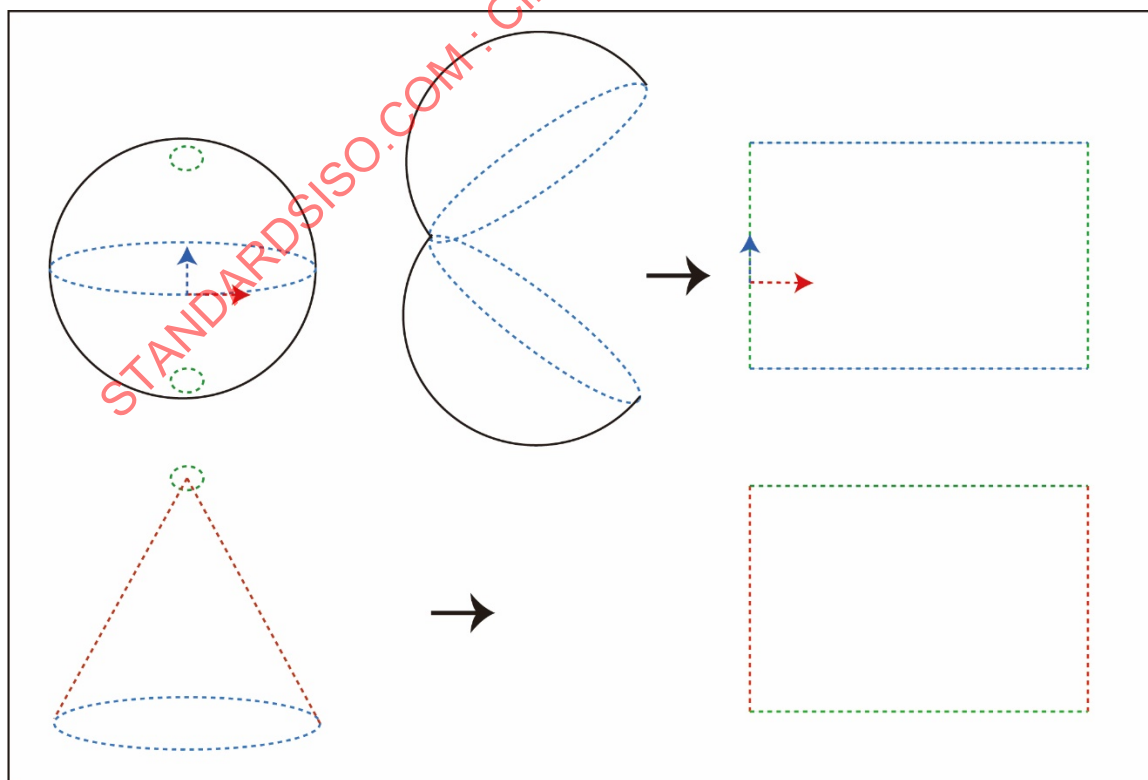


Figure 23 — Example of a degenerate edge

5.9.3.4 Nonmanifold B-Reps

Nonmanifold *B-reps* are B-reps that cannot be manufactured. These are still valid B-reps.

In most cases, there are two ways in which nonmanifold B-reps are created in a construction process:

- They clarify a situation in a view of a construction.
- They result from Boolean operations.

The following are two examples.

The following Figure 24 represents a reinforced concrete beam. This beam consists of two materials, steel surrounded by concrete. In a B-rep, this beam could be constructed as a 3D block, because the only thing that is visible is concrete. But to show where the steel is located within the block, its position is represented as a long cylinder.



Figure 24 — Example of nonmanifold B-Reps

The following Figure 25 shows the union of a cube with a cylinder, which is a cube. But the intersections of the faces at the top and bottom often still exist after this Boolean operation. These intersections are represented as nonmanifold faces because they have two outer wires: one rectangular wire for the cube and one circular wire for the cylinder.

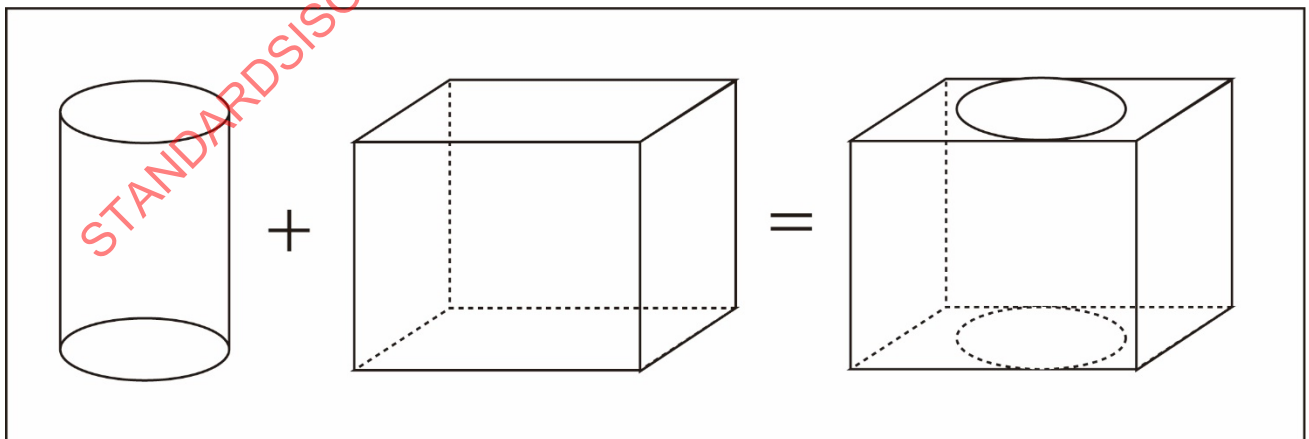


Figure 25 — Example of nonmanifold B-Reps

5.9.4 brep

Category: Geometry

5.9.4.1 General

Describes a boundary representation (B-rep) structure.

5.9.4.2 Concepts

A B-rep can be a single solid or a single face or vertex. The B-rep can be nonmanifold or it can consist of several solids. The **<brep>** element contains the complete topological description of a static structure and the corresponding geometrical descriptions of the vertices.

5.9.4.3 Attributes

The **<brep>** element has no attributes.

5.9.4.4 Related elements

The **<brep>** element relates to the following elements:

Parent elements	geometry (see 5.5 Core elements reference)
Child elements	See the following subclause.
Other	None

5.9.4.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<curves>	Contains all curves used in this B-rep. Curves are required for the B-rep structure. This includes curves that describe the kind of an edge. This element is required if the <edges> element is present. See main entry.	N/A	0 or 1
<surface_curves>	Contains all 2D curves used in this B-rep. This includes surfaces that describe the kind of the face. This element is required if the <faces> element is present. See main entry.	N/A	0 or 1
<surfaces>	Contains all surfaces used in this B-rep. See main entry.	N/A	0 or 1

Name/example	Description	Default	Occurrences
<source>	Contains all the necessary data for the topological entities. Provides vertices, edges, and faces for their geometric entities. At least one <source> element is needed for the vertices. If <edges> is specified, an additional <source> element is needed for accessing the curves in the <curve> element by an SIDREF_array (see main entry in Core). If <faces> is specified, an additional <source> element is needed for accessing the surfaces in the <surface> element by an SIDREF_array. See main entry in Core.	N/A	1 or more
<vertices>	Describes all vertices of the B-rep. Vertices are the base topological entity for all B-rep structures, so this element is required. See main entry in Core.	N/A	1
<edges>	Describes all edges of the B-rep. See main entry.	N/A	0 or 1
<wires>	Describes all wires of the B-rep. See main entry.	N/A	0 or 1
<faces>	Describes all faces of the B-rep. See main entry.	N/A	0 or 1
<pcurves>	Describes all pcurves of the B-rep. See main entry.	N/A	0 or 1
<shells>	Describes all shells of the B-rep. See main entry.	N/A	0 or 1
<solids>	Describes all solids of the B-rep. See main entry.	N/A	0 or 1
<extra>	See main entry in Core.	N/A	0 or more

5.9.4.6 Example

The following is an example of the **<brep>** element:

```

<geometry id="geo">
  <brep>
    <curves/>
    <surface_curves/>
    <surfaces/>
    <source id="geom-points"/>
    <source id="geom-curves"/>
    <source id="geom-curves2d"/>
    <source id="geom-surfaces"/>
    <source id="orientations"/>
    <source id="curve-params"/>
    <source id="materials"/>
    <vertices id="vertices"/>
    <edges count="6" id="edges"/>
    <wires count="6" id="wires"/>
    <faces count="4" id="faces"/>
    <pcurves count="10" id="pcurves"/>
    <shells count="1" id="shells"/>
    <solids count="1" id="solids"/>
  </brep>
</geometry>

```

5.9.5 circle

Category: Curves

5.9.5.1 General

Describes a circle in 3D space.

5.9.5.2 Concepts

The circle is defined with its centre in the origin of the local coordinate system. The circle lies in the (x,y) plane.

5.9.5.3 Attributes

The **<circle>** element has no attributes.

5.9.5.4 Related elements

The **<circle>** element relates to the following elements:

Parent elements	curve
Child elements	See the following subclause.
Other	None

5.9.5.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<radius>	Contains a floating-point number that specifies the radius of the circle. This element has no attributes.	None	1
<extra>	See main entry in Core.	N/A	0 or more

5.9.5.6 Details

A circle is defined by its radius and, as with any conic curve, is positioned in space with a right-handed coordinate system where:

- The origin is the centre of the circle;
- The origin, x direction, and y direction define the plane of the circle.

This coordinate system is the local coordinate system of the circle. The main direction of this coordinate system is the vector that is normal to the plane of the circle.

The axis or main axis of the circle is the axis of which the origin and unit vector are respectively the origin and main direction of the local coordinate system. The main direction of the local coordinate system gives an explicit orientation to the circle (definition of the trigonometric sense), determining the direction in which the parameter increases along the circle.

The circle is parameterized by an angle:

$$P(u) = O + R * \cos(u) * XDir + R * \sin(u) * YDir$$

where:

P is the point of parameter u ;

O , $XDir$, and $YDir$ are respectively the origin, x direction, and y direction of its local coordinate system;

R is the radius of the circle;

The x axis of the local coordinate system therefore defines the origin of the parameter of the circle. The parameter is the angle with this x direction. A circle is a closed and periodic curve. The period is 2π and parameter range is $0 \leq u < 2\pi$

5.9.5.7 Example

The following is an example of the **<circle>** element:

```
<curve sid="curve">
  <circle>
    <radius>3</radius>
  </circle>
  <origin>0 0 10</origin>
</curve>
```

5.9.6 cone

Category: Surfaces

5.9.6.1 General

Describes a conical surface.

5.9.6.2 Concepts

A cone is defined by the half-angle at its apex, and is positioned in space by a coordinate system and a reference radius as follows:

- The z axis of the local coordinate system is the axis of revolution of the cone;
- The plane defined by the origin (0,0,0), the x axis and the y axis of the local coordinate system is the reference plane of the cone. The intersection of the cone with this reference plane is a circle of radius equal to the reference radius.

5.9.6.3 Attributes

The **<cone>** element has no attributes.

5.9.6.4 Related elements

The **<cone>** element relates to the following elements:

Parent elements	surfaces/surface
Child elements	See the following subclause.
Other	None

5.9.6.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<radius>	Contains a floating-point number that specifies the radius of the cone. This element has no attributes.	None	1
<angle>	Contains a floating-point number that specifies the conical surface semi angle]0,PI/2[. This element has no attributes.	None	1
<extra>	See main entry in Core.	N/A	0 or more

5.9.6.6 Details

This coordinate system is the local coordinate system of the cone. The following applies:

- Rotation around its z axis, in the trigonometric sense given by the x axis and the y axis, defines the u parametric direction;
- Its x axis gives the origin for the u parameter;
- Its z axis is the v parametric direction of the cone;
- Its origin is the origin of the v parameter.

The parametric range of the u and v parameters is:

for u, $0 \leq u < 2\pi$;

for v, $-\infty < v < +\infty$;

The parametric formula of the cone is:

$$P(u, v) = O + (R + v * \tan(Ang)) * (\cos(u) * XDir + \sin(u) * YDir) + v * ZDir$$

where:

- *O* is the origin;
- *XDir*, *YDir*, and *ZDir* are the x direction, the y direction, and the z direction;
- *Ang* is the half-angle at the apex of the cone;
- *R* is the reference radius.

5.9.6.7 Example

The following is an example of the **<cone>** element:

```
<cone>
  <radius>1.92574</radius>
  <angle>0.785398</angle>
</cone>
```

5.9.7 curve

Category: Curves

5.9.7.1 General

Describes a specific curve.

5.9.7.2 Concepts

This element defines the attributes of a curve. With **<orient>** and **<origin>**, the surface can be positioned to its correct location.

5.9.7.3 Attributes

The **<curve>** element has the following attributes:

sid	sid_type	A text string value containing the scoped identifier of this element. This shall be unique within the scope of the parent element. Optional. For details, see "Address syntax" in 5.3 Schema concepts.
name	xs:token	The text string name of the element. Optional.

5.9.7.4 Related elements

The **<curve>** element relates to the following elements:

Parent elements	curves, surface_curves, swept_surface
Child elements	See the following subclause.
Other	None

5.9.7.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
curve element	The curve element shall be exactly one of the following: <line> <circle> <ellipse> <parabola> <hyperbola> <nurbs> See main entries.	N/A	1
<orient>	Describes the orientation of the object frame. See main entry.	None	0 or more
<origin>	Describes the origin of the object frame. See main entry.	0 0 0	0 or 1

5.9.7.6 Example

The following is an example of the <curve> element:

```
<curve sid="curve">
  <line>
    <origin>5 0 0</origin>
    <direction>0 0 1</direction>
  </line>
</curve>
```

5.9.8 curves

Category: Curves

5.9.8.1 General

Contains all curves that are used in a B-rep structure.

5.9.8.2 Concepts

This element is a container for all 3D curves used by the edges of this B-rep structure.

5.9.8.3 Attributes

The <curves> element has no attributes.

5.9.8.4 Related elements

The <curves> element relates to the following elements:

Parent elements	brep
Child elements	See the following subclause.
Other	None

5.9.8.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<curve>	Describes a single curve. See main entry.	N/A	1 or more
<extra>	See main entry in Core.	N/A	0 or more

5.9.8.6 Details

This element holds all the curves that are needed for a B-rep structure. This includes curves that describe the kind of an edge and curves that are needed to create an extrusion for a surface.

This element is a container for all 3D curves that are used by the topological entity **<edges>**.

5.9.8.7 Example

The following is an example of the **<curves>** element:

```
<curves>
  <curve sid="curve-1"/>
  <curve sid="curve-2"/>
</curves>
```

5.9.9 cylinder (B-Rep)

Category: Surfaces

5.9.9.1 General

Describes an unlimited cylindrical surface.

NOTE For the **<cylinder>** element in **<shape>**, see "**<cylinder>** (Physics)".

5.9.9.2 Concepts

An unlimited cylinder has a radius but is assumed to extend to an infinite length.

5.9.9.3 Attributes

The **<cylinder>** element has no attributes.

5.9.9.4 Related elements

The **<cylinder>** element relates to the following elements:

Parent elements	surface (B-rep)
Child elements	See the following subclause.
Other	None

5.9.9.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<radius>	Contains two floating-point values that represent the radii of the cylinder (it may be elliptical). This element has no attributes.		1
<extra>	See main entry in Core.	N/A	0 or more

5.9.9.6 Example

The following is an example of the < cylinder > element:

```
<cylinder>
  <radius>5.0</radius>
</cylinder>
```

5.9.10 edges

Category:

Topology

5.9.10.1 General

Describes the edges of a B-rep structure.

5.9.10.2 Concepts

Edges are limited by two vertices and have a curve for a geometric representation. The segment of the curve is also limited by its start and end parameters.

5.9.10.3 Attributes

The < edges > element has the following attributes:

id	xs:ID	A text string containing the unique identifier of this element. This value shall be unique within the instance document. Required.
name	xs:token	The text string name of the element. Optional.
count	xs:unsignedLong	The number of edges. Required.

5.9.10.4 Related elements

The < edges > element relates to the following elements:

Parent elements	brep
Child elements	See the following subclause.
Other	None

5.9.10.5 Child elements

Name/example	Description	Default	Occurrences
<input>	Four <input> elements are needed to define an edge: • One with <code>semantic="CURVE"</code> to reference the corresponding geometric element for the edge. • Two with <code>semantic="VERTEX"</code> to reference the two vertices that limit each edge. • One with <code>semantic="PARAM"</code> to set the parametric values (start and end parameters) of the curve. See main entry in Core.	N/A	4 or more
<p>	References the indices for the inputs; this describes the attributes of all the edges in the B-rep. This element has no attributes.	None	0 or 1
<extra>	See main entry in Core.	N/A	0 or more

5.9.10.6 Details

The edges are always declared FORWARD, so the first vertex is the starting vertex and the second is the ending vertex. (See Figure 26)

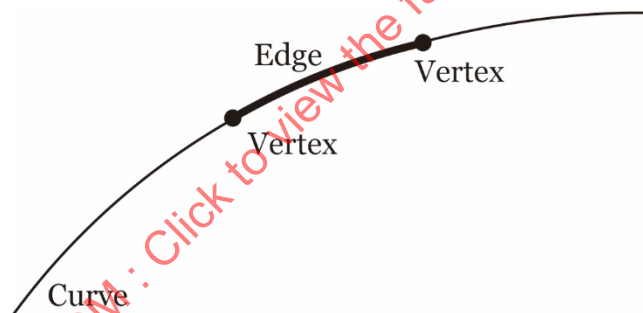


Figure 26 — Starting vertex and ending vertex

5.9.10.7 Example

The following is an example of the **<edges>** element:

```
<edges id="edges" count="6">
  <input semantic="CURVE" source="#geom-curves" offset="0"/>
  <input semantic="VERTEX" source="#vertices" offset="1"/>
  <input semantic="VERTEX" source="#vertices" offset="2"/>
  <input semantic="PARAM" source="#curve-params" offset="4"/>
  <p>
    0 0 1 0
    1 1 1 1
    2 0 0 2
    3 2 2 3
    4 3 3 4
    5 2 3 5
  </p>
</edges>
```

5.9.11 ellipse

Category: Curves

5.9.11.1 General

Describes an ellipse in 3D space.

5.9.11.2 Concepts

An ellipse is defined by its major and minor radii and, as with any conic curve, is positioned in space with a right-handed coordinate system where

- The origin is the centre of the ellipse.
- The x axis defines the major axis.
- The y axis defines the minor axis.

5.9.11.3 Attributes

The **<ellipse>** element has no attributes.

5.9.11.4 Related elements

The **<ellipse>** element relates to the following elements:

Parent elements	curve
Child elements	See the following subclause.
Other	None

5.9.11.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<radius>	Contains two floating-point numbers that specify the radii of the ellipse. This element has no attributes.	None	1
<extra>	See main entry in Core.	N/A	0 or more

5.9.11.6 Details

The ellipse is defined with its centre in the origin of the local coordinate system. The ellipse lies in the (x,y) plane. The first radius is the major radius (in the u direction in parametric space) and the second is the minor radius (in the v direction).

The origin, x direction and y direction of this coordinate system define the plane of the ellipse. The coordinate system is the local coordinate system of the ellipse.

The main direction of this coordinate system is the vector that is normal to the plane of the ellipse. The axis, or main axis, of the ellipse is the axis of which the origin and unit vector are respectively the origin and main direction of the local coordinate system. The main direction of the local coordinate system gives an explicit orientation to the ellipse (definition of the trigonometric sense), determining the direction in which the parameter increases along the ellipse. The ellipse is parameterized by an angle:

$$P(u) = O + MajorRad * \cos(u) * XDir + MinorRad * \sin(u) * YDir$$

where:

- P is the point of parameter u ;
- O , $XDir$ and $YDir$ are respectively the origin, x direction, and y direction of its local coordinate system;
- $MajorRad$ and $MinorRad$ are the major and minor radii of the ellipse.

The x axis of the local coordinate system therefore defines the origin of the parameter of the ellipse. An ellipse is a closed and periodic curve. The period is 2π and the parameter range is $0 \leq u < 2\pi$.

5.9.11.7 Example

The following is an example of the **<ellipse>** element:

```
<curve sid="curve">
  <ellipse>
    <radius>3 5</radius>
  </ellipse>
</curve>
```

5.9.12 faces

Category: Topology

5.9.12.1 General

Describes the faces of a B-rep structure.

5.9.12.2 Concepts

Faces are limited by one or more wires. Generally, a face is bounded by one outer wire and zero or more inner wires. In this case, it is a manifold face. But COLLADA B-rep also supports nonmanifold B-rep, so a face can be bounded by one or more outer wires.

5.9.12.3 Attributes

The **<faces>** element has the following attributes:

id	xs:ID	A text string containing the unique identifier of the <faces> element. This value shall be unique within the instance document. Required.
name	xs:token	The text string name of the element. Optional.
count	xs:unsignedLong	The number of faces. Required.

5.9.12.4 Related elements

The **<faces>** element relates to the following elements:

Parent elements	brep
Child elements	See the following subclause.
Other	None

5.9.12.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<input>	There shall be at least three <input> s: • One with semantic= "SURFACE" to reference the corresponding geometric element for the face. • One with semantic="WIRE" to reference the wires for each face. • One with semantic="ORIENTATION" defines the orientation of the referenced wire within the face. Additionally, another <input> can specify semantic="MATERIAL" to link a single face with a symbolic name that can be bound to a material when instantiating the B-rep. See main entry in Core.	N/A	3 or more
<vcount>	Contains a list of integers describing the number of wires for each face. This element has no attributes.	None	1
<p>	References the indices for the input. This describes all the attributes of all faces in the B-rep. This element has no attributes.	None	0 or 1
<extra>	See main entry in Core.	N/A	0 or more

5.9.12.6 Details

A face is bounded by one or more wires. The wires need an orientation for the faces.

A face is visible if its outer wire has the same orientation as the normal of the corresponding surface. It is visible from both sides.

The outer wire limits the surface.

The inner wires cut holes in the bounded surface (their orientations are opposite to the surface normal).

A symbolic name of a material is bound to each face using the **<input semantic="MATERIAL">** with a token source. Mesh primitives assemble vertex frequency data instead of faces. Faces do not assemble vertex data because a face consists of a surface and one or more wires, and there is exactly one **<faces>** element.

5.9.12.7 Example

The following is an example of the **<faces>** element:

```
<faces id="wires" count="6">
  <input semantic="SURFACE" source="#geom-surfaces" offset="0"/>
  <input semantic="WIRE" source="#wires" offset="1"/>
  <input semantic="ORIENTATION" source="#orientations" offset="2"/>
  <input semantic="MATERIAL" source="#materials" offset="3"/>
  <vcount>1 2 2 1</vcount>
  <p> 0 0 1 0 1 1 0 0 1 2 0 0 2 3 1 0 2 4 1 0 3 5 0 0</p>
</faces>
```

5.9.13 hyperbola

Category: Curves

5.9.13.1 General

Describes a hyperbola in 3D space.

5.9.13.2 Concepts

A hyperbola is defined by its major and minor radii and, as with any conical curve, is positioned in space with a right-handed coordinate system where:

- The origin is the centre of the hyperbola;
- The x axis defines the major axis;
- The y axis defines the minor axis.

5.9.13.3 Attributes

The **<hyperbola>** element has no attributes.

5.9.13.4 Related elements

The **<hyperbola>** element relates to the following elements:

Parent elements	curve
Child elements	See the following subclause.
Other	None

5.9.13.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<radius>	Contains two floating-point numbers that specify the radii of the hyperbola. This element has no attributes.	None	1
<extra>	See main entry in Core.	N/A	0 or more

5.9.13.6 Details

The plane of the hyperbola is defined by the origin, x direction, and y direction of this coordinate system. The coordinate system is the local coordinate system of the hyperbola.

5.9.13.7 Example

The following is an example of the **<hyperbola>** element:

```
<curve sid="curve">
  <hyperbola>
    <radius>3 5</radius>
  </hyperbola>
</curve>
```

5.9.14 line

Category: Curves

5.9.14.1 General

Describes a single line in 3D space.

5.9.14.2 Concepts

A line is defined and positioned in space with an origin and a unit vector representing its direction.

5.9.14.3 Attributes

The **<line>** element has no attributes.

5.9.14.4 Related elements

The **<line>** element relates to the following elements:

Parent elements	curve
Child elements	See the following subclause.
Other	None

5.9.14.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<origin>	Contains three floating-point numbers that describe the origin of the line. See main entry.	None	1
<direction>	Contains three floating-point numbers that describe the direction of the line as a unit vector. This element has no attributes.	None	1
<extra>	See main entry in Core.	N/A	0 or more

5.9.14.6 Example

The following is an example of the **<line>** element:

```
<curve sid="curve">
  <line>
    <origin>5 0 0</origin>
    <direction>0 0 1</direction>
  </line>
</curve>
```

5.9.15 nurbs

Category: Curves

5.9.15.1 General

Describes a NURBS curve in 3D space.

5.9.15.2 Concepts

A NURBS curve is defined by:

- Its degree.
- Its periodic or nonperiodic nature.
- A table of poles (also called control points), with their associated weights if the NURBS curve is rational. The poles of the curve are control points used to deform the curve. If the curve is nonperiodic, the first pole is the start point of the curve, and the last pole is the end point of the curve. The segment that joins the first pole to the second pole is the tangent to the curve at its start point, and the segment that joins the last pole to the second-from-last pole is the tangent to the curve at its end point. If the curve is periodic, these geometric properties are not verified. A weight has an influence on how close the curve passes the corresponding control point (pole). However, it is true that the most prominent need for weight values different from one is to describe conic sections. Use a NURBS curve with inner knots as an example as a curve with more than one segment is more general. It also gives the opportunity to relate the continuity of the curve to the multiplicity of the knots. If the weights of all the poles are equal, the curve has a polynomial equation; it is therefore a nonrational curve.

- A table of knots with their multiplicities. For a NURBS, if the table of knots is an increasing sequence of reals without repetition, then curve have continuity degree of number of control points. If knots vector repeated, in which case the continuity order at that point is reduced by 1 for each repetition. Refer to [12].

5.9.15.3 Attributes

The **<nurbs>** element has the following attributes:

degree	uint_type	Specifies the degree of the NURBS curve. Required.
closed	xs:boolean	Specifies whether a NURBS curve is closed. The default is false. Optional.

5.9.15.4 Related elements

The **<nurbs>** element relates to the following elements:

Parent elements	curve
Child elements	See the following subclause.
Other	None

5.9.15.5 Child elements

Child elements shall appear in the following order if present:

Name/example	Description	Default	Occurrences
<source>	See main entry in Core.	None	1 or more
<control_vertices>	See main entry in Core.	None	1
<extra>	See main entry in Core.	N/A	0 or more

5.9.15.6 Details

The degree value is referred as p . The order of a NURBS segment, denoted as o , is defined by $p + 1$.

NURBS curves are evaluated using three sources of information, denoted by the following **semantic** values on **<source>/<input>**:

- **POLE**: The control vertices (also called poles). The number of control vertices in one segment is referred to as n .
- **WEIGHT**: The weights of the poles. This is the last component of the pole source used to describe control vertices as homogeneous coordinates.
- **KNOT**: The knot vector of the NURBS. A list of nondecreasing floating-point vectors, one vector per segment. The size of one vector, referred to as k , is bound to the relationship:

$$k = (n + p + 1) = (n + o)$$

If the WEIGHT input semantic is present, its size shall match the POLE input size.

The POLE source can be either of type float3_type or float4_type for a 3D position. When using float4, the fourth component is used as the weight applied to the control vertex, making a WEIGHT source useless; if WEIGHT is also specified, it shall be ignored.

The number of values in the KNOT source shall follow this relationship:

$$\text{Count(KNOTS)} = \text{SUM}(n + p + 1)$$

where n is the number of control vertices in the segment, and p is the degree.

5.9.15.7 Example

The following is an example of the **<nurbs>** element:

```
<nurbs degree="5" closed="false">
  <source id="curve.knots">
    <float_array id="curve.knots-array" count="12">
      9.12168 9.12168 9.12168 9.12168 9.12168 9.12168
      20.1173 20.1173 20.1173 20.1173 20.1173 20.1173
    </float_array>
    <technique_common>
      <accessor count="12" source="#curve.knots-array">
        <param name="KNOT" type="float"/>
      </accessor>
    </technique_common>
  </source>
  <source id="curve.points">
    <float_array id="curve.points-array" count="18">
      2.19911 0.838995 1.53577 2.19911 6.98207 -1.82021
      2.19911 0.554555 -6.2853 2.19911 -5.58852 -2.92932
      2.19911 -5.30408 4.89175 2.19911 0.838995 1.53577
    </float_array>
    <technique_common>
      <accessor count="6" source="#curve.points-array" stride="3">
        <param name="X" type="float"/>
        <param name="Y" type="float"/>
        <param name="Z" type="float"/>
      </accessor>
    </technique_common>
  </source>
  <source id="curve.weights">
    <float_array id="curve.weights-array" count="6">
      1 0.2 0.2 0.2 0.2 1
    </float_array>
    <technique_common>
      <accessor count="6" source="#curve.weights-array">
        <param name="WEIGHT" type="float"/>
      </accessor>
    </technique_common>
  </source>
  <control_vertices>
    <input semantic="KNOT" source="#curve.knots"/>
    <input semantic="POLE" source="#curve.points"/>
    <input semantic="WEIGHT" source="#curve.weights"/>
  </control_vertices>
</nurbs>
```

5.9.16 nurbs_surface

Category: Surfaces

5.9.16.1 General

Describes a NURBS surface in 3D space.

5.9.16.2 Concepts

In each parametric direction, a NURBS surface can be:

- Uniform or nonuniform;
- Rational or nonrational;
- Periodic or nonperiodic.

A NURBS surface is defined by:

- Its degrees, in the u and v parametric directions;
- Its periodic characteristic, in the u and v parametric directions;
- A table of poles, also called control points (together with the associated weights if the surface is rational);
- A table of knots.

5.9.16.3 Attributes

The **<nurbs_surface>** element has the following attributes:

degree_u	uint_type	Specifies the degree of the NURBS curve for the u direction. Required.
closed_u	xs:boolean	Specifies whether a NURBS curve is closed for the u direction. The default is false. Optional.
degree_v	uint_type	Specifies the degree of the NURBS curve for the v direction. Required.
closed_v	xs:boolean	Specifies whether a NURBS curve is closed for the v direction. The default is false. Optional.

5.9.16.4 Related elements

The **<nurbs_surface>** element relates to the following elements:

Parent elements	surface (B-rep)
Child elements	See the following subclause.
Other	None