



BSI Standards Publication

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

National foreword

This British Standard is the UK implementation of ISO 17506:2022+A1:2025. It supersedes BS ISO 17506:2022, which is withdrawn.

The start and finish of text introduced or altered by amendment is indicated in the text by tags. Tags indicating changes to ISO text carry the number of the ISO amendment. For example, text altered by ISO amendment 1 is indicated by A1 A1.

The UK participation in its preparation was entrusted to Technical Committee AMT/4, Industrial data and manufacturing interfaces.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient’s own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2025
Published by BSI Standards Limited 2025

ISBN 978 0 539 31931 6

ICS 25.040.40

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 April 2022.

Amendments/corrigenda issued since publication

Date	Text affected
28 February 2025	Implementation of ISO amendment 1:2025

INTERNATIONAL STANDARD

ISO
17506
+A1

First edition
2022-03

Amendment 1
2025-01

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*



Reference number
ISO 17506:2022(E)
Documents
Document Sharing Hub
© ISO 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

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.

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 **[A1]** conformance tool **[A1]**. 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 .zae 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 .zae archive. Otherwise, the **<scene>** element will be the starting point for application loading the .zae archive. If neither of these conditions is met, then the behaviour is undefined.

The URIs in the .zae 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 .zae 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 URIs, 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 .

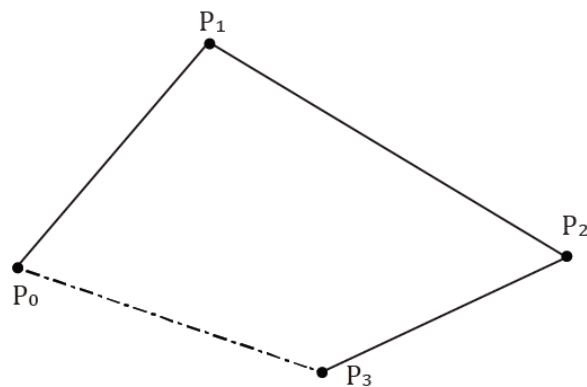


Figure 1 — Linear spline

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:

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

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

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

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

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.

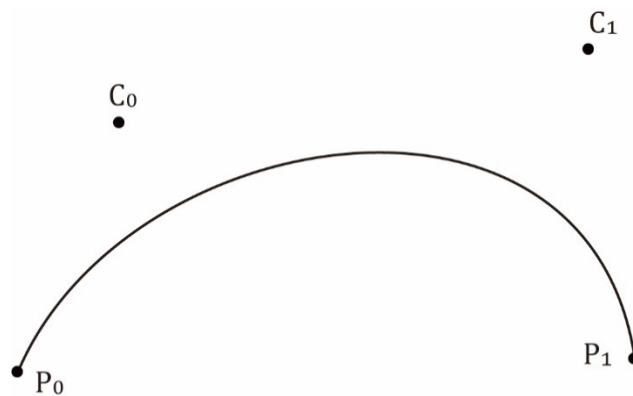


Figure 2 — Bézier Curve

`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:

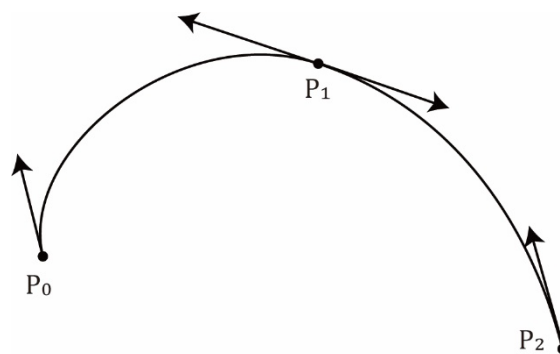


Figure 3 — Hermite Spline

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:

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

$$T_0 = 3(P_1 - C_1)$$

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:

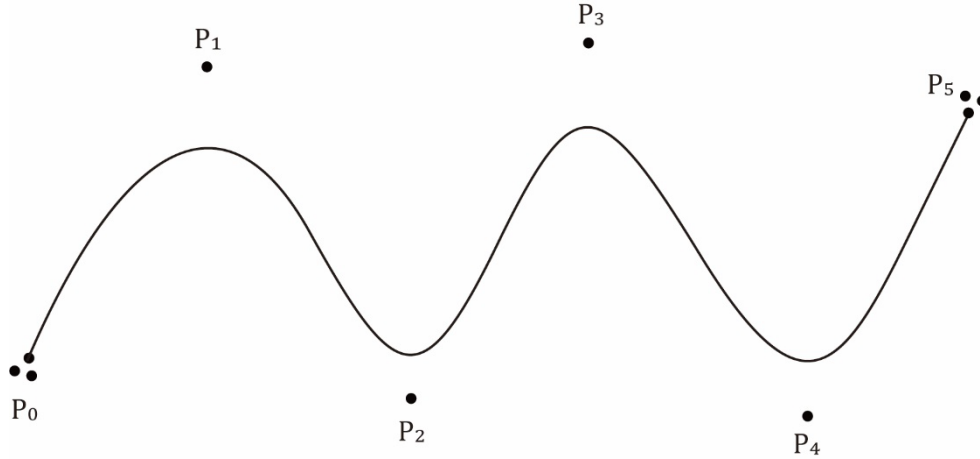


Figure 4 — Example of B-splines

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.