



**COLLADA – Digital Asset Schema
リリース 1.5.0**

仕様書

2008年4月

編集者：Mark Barnes および Ellen Levy Finch, Sony Computer Entertainment Inc.

© 2005-2009 The Khronos Group Inc., Sony Computer Entertainment Inc.

All Rights Reserved.

This specification is protected by copyright laws and contains material proprietary to the Khronos Group, Inc. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast, or otherwise exploited in any manner without the express prior written permission of Khronos Group. You may use this specification for implementing the functionality therein, without altering or removing any trademark, copyright, or other notice from the specification, but the receipt or possession of this specification does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

Khronos Group grants express permission to any current Promoter, Contributor, or Adopter member of Khronos to copy and redistribute UNMODIFIED versions of this specification in any fashion, provided that NO CHARGE is made for the specification and the latest available update of the specification for any version of the API is used whenever possible. Such distributed specification may be reformatted AS LONG AS the contents of the specification are not changed in any way. The specification may be incorporated into a product that is sold as long as such product includes significant independent work developed by the seller. A link to the current version of this specification on the Khronos Group website should be included whenever possible with specification distributions.

Khronos Group makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this specification, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose or noninfringement of any intellectual property. Khronos Group makes no, and expressly disclaims any, warranties, express or implied, regarding the correctness, accuracy, completeness, timeliness, and reliability of the specification. Under no circumstances will the Khronos Group, or any of its Promoters, Contributors, or Members or their respective partners, officers, directors, employees, agents, or representatives be liable for any damages, whether direct, indirect, special, or consequential damages for lost revenues, lost profits, or otherwise, arising from or in connection with these materials.

Khronos is a trademark of The Khronos Group Inc.

COLLADA is a trademark of Sony Computer Entertainment Inc. used by permission by Khronos.

All other trademarks are the property of their respective owners and/or their licensors.

Publication date: May 2009

Khronos Group
14525 SW Millikan Way #45043
Beaverton, Oregon 97005-2343 USA

Sony Computer Entertainment Inc.
2-6-21 Minami-Aoyama, Minato-ku,
Tokyo 107-0062 Japan

Sony Computer Entertainment America
919 E. Hillsdale Blvd.
Foster City, CA 94404, U.S.A.

Sony Computer Entertainment Europe
30 Golden Square
London W1F 9LD, U.K.

目次

このドキュメントについて	ix
対象読者	ix
このドキュメントの内容	ix
表記法	x
リファレンスの表記法と構成	x
参考資料	xi
1 章 設計上の考慮事項	1-1
概要	1-1
前提条件と依存関係	1-1
目標とガイドライン	1-1
2 章 ツールの要件とオプション	2-1
概要	2-1
エクスポータ	2-1
インポータ	2-4
アーカイブへのパッケージ化	2-4
3 章 スキーマのコンセプト	3-1
概要	3-1
XML の概要	3-1
アドレス構文	3-1
インスタンス化と外部参照	3-5
共通プロファイル	3-6
4 章 プログラミングガイド	4-1
概要	4-1
COLLADA のパラメータについて	4-1
曲線補間	4-1
COLLADA のスキン変形 (スキニング)	4-7
5 章 コア要素のリファレンス	5-1
概要	5-1
要素のカテゴリ分類	5-1
accessor	5-4
ambient	5-9
animation	5-10
animation_clip	5-12
asset	5-14
bool_array	5-16
camera	5-17
channel	5-19
COLLADA	5-20
color	5-21
contributor	5-22
controller	5-23
control_vertices	5-25
directional	5-27
evaluate_scene	5-28
extra	5-29
float_array	5-31
formula	5-32
geographic_location	5-34

geometry	5-35
IDREF_array	5-37
imager	5-38
input	5-40
input	5-42
instance_animation	5-44
instance_camera	5-45
instance_controller	5-46
instance_formula	5-49
instance_geometry	5-50
instance_light	5-51
instance_node	5-53
instance_visual_scene	5-55
int_array	5-56
joints	5-57
library_animation_clips	5-58
library_animations	5-59
library_cameras	5-60
library_controllers	5-61
library_formulas	5-62
library_geometries	5-63
library_lights	5-64
library_nodes	5-65
library_visual_scenes	5-66
light	5-67
lines	5-69
linestrips	5-70
Lookat	5-72
matrix	5-73
mesh	5-74
morph	5-77
Name_array	5-79
newparam	5-80
node	5-82
optics	5-84
orthographic	5-85
param	5-87
param	5-88
perspective	5-90
point	5-91
polygons	5-92
polylist	5-95
rotate	5-97
sampler	5-98
scale	5-104
scene	5-105
setparam	5-106
SIDREF_array	5-108
skeleton	5-109
skew	5-110
skin	5-111
source	5-114
spline	5-116
spot	5-117
targets	5-118
technique	5-119
technique_common	5-121

	translate	5-122
	triangles	5-123
	trifans	5-124
	tristrips	5-126
	vertex_weights	5-128
	vertices	5-129
	visual_scene	5-130
6 章	フィジックスリファレンス	6-1
	概要	6-1
	要素のカテゴリ分類	6-1
	Attachment	6-4
	box	6-5
	capsule	6-6
	convex_mesh	6-7
	cylinder	6-9
	force_field	6-10
	instance_force_field	6-11
	instance_physics_material	6-12
	instance_physics_model	6-13
	instance_physics_scene	6-14
	instance_rigid_body	6-15
	instance_rigid_constraint	6-18
	library_force_fields	6-19
	library_physics_materials	6-20
	library_physics_models	6-21
	library_physics_scenes	6-23
	physics_material	6-24
	physics_model	6-25
	physics_scene	6-28
	plane	6-31
	ref_attachment	6-32
	rigid_body	6-33
	rigid_constraint	6-36
	shape	6-40
	sphere	6-42
7 章	FX 入門	7-1
	概要	7-1
	プラットフォーム固有のエフェクトにプロファイルを利用する	7-1
	FX のパラメータについて	7-4
	シェーダ	7-5
	レンダリング	7-6
	テクスチャリング	7-7
8 章	FX リファレンス	8-1
	概要	8-1
	要素のカテゴリ分類	8-1
	COLLADA FX について	8-4
	alpha	8-4
	annotate	8-5
	argument	8-6
	array	8-7
	binary	8-9
	bind	8-10
	bind_attribute	8-11
	bind_material	8-12

bind_uniform	8-15
bind_vertex_input	8-16
blinn	8-18
code	8-21
color_clear	8-22
color_target	8-23
compiler	8-25
constant	8-26
create_2d	8-28
create_3d	8-30
create_cube	8-32
depth_clear	8-34
depth_target	8-35
draw	8-37
effect	8-38
evaluate	8-40
format	8-41
fx_common_color_or_texture_type	8-44
fx_common_float_or_param_type	8-45
fx_sampler_common	8-46
image	8-49
include	8-51
init_from	8-52
instance_effect	8-54
instance_image	8-55
instance_material	8-57
instance_material	8-58
lamBERT	8-60
library_effects	8-62
library_images	8-63
library_materials	8-64
linker	8-65
material	8-66
modifier	8-68
pass	8-69
phong	8-71
profile_BRIDGE	8-73
profile_CG	8-75
profile_COMMON	8-78
profile_GLES	8-79
profile_GLES2	8-82
profile_GLSL	8-85
program	8-87
render	8-89
RGB	8-90
sampler1D	8-91
sampler2D	8-92
sampler3D	8-92
samplerCUBE	8-93
samplerDEPTH	8-94
samplerRECT	8-95
sampler_image	8-96
sampler_states	8-96
semantic	8-97
shader	8-98
sources	8-100
states	8-101

	stencil_clear	8-107
	stencil_target	8-108
	technique	8-110
	technique_hint	8-111
	texcombiner	8-112
	texenv	8-114
	texture_pipeline	8-116
	usertype	8-118
9 章	B-Rep リファレンス	9-1
	概要	9-1
	要素のカテゴリ分類	9-1
	COLLADA の B-rep について	9-2
	brep	9-6
	circle	9-8
	cone	9-10
	curve	9-11
	curves	9-12
	cylinder	9-13
	edges	9-14
	ellipse	9-16
	faces	9-17
	hyperbola	9-19
	line	9-20
	nurbs	9-21
	nurbs_surface	9-23
	orient	9-26
	origin	9-26
	parabola	9-27
	pcurves	9-28
	shells	9-29
	solids	9-31
	surface	9-32
	surfaces	9-34
	surface_curves	9-35
	swept_surface	9-36
	torus	9-38
	wires	9-39
	B-rep の完全な例	9-40
10 章	キネマティックスリファレンス	10-1
	概要	10-1
	要素のカテゴリ分類	10-1
	articulated_system	10-3
	attachment_end	10-4
	attachment_full	10-5
	attachment_start	10-6
	axis_info	10-7
	bind	10-10
	bind_joint_axis	10-11
	bind_kinematics_model	10-12
	connect_param	10-13
	effector_info	10-14
	frame_object、 frame_origin、 frame_tcp、 frame_tip	10-15
	instance_articulated_system	10-16
	instance_joint	10-18
	instance_kinematics_model	10-19

instance_kinematics_scene	10-20
joint	10-22
kinematics	10-23
kinematics_model	10-25
kinematics_scene	10-27
library_articulated_systems	10-28
library_joints	10-29
library_kinematics_models	10-30
library_kinematics_scenes	10-31
link	10-32
motion	10-33
prismatic	10-35
revolute	10-36
11 章 型	11-1
概要	11-1
単純な値の型	11-1
パラメータの型要素	11-2
他の単純型	11-3
値/パラメータ選択型	11-3
付録 A : COLLADA のサンプル	A-1
例 : キューブ	A-1
付録 B : プロファイル GLSL および GLES2 のサンプル	B-1
例 : <profile_GLSL>	B-1
例 : <profile_GLES2>	B-6
用語集	G-1
総合索引	I-1
COLLADA 要素の索引	I-1

このドキュメントについて

本ドキュメントでは、COLLADA スキーマについて解説します。COLLADA は、3D オーサリングアプリケーションが情報を失うことなく自由にデジタルアセットを交換することを可能にする、XML ベースのスキーマを定義して、複数のソフトウェアパッケージを非常に強力なツールチェーンに組み込むことを可能にする、共同設計活動 (COLLABorative Design Activity) です。

本ドキュメントの趣旨は、ソフトウェアディベロッパの方が COLLADA リソースの処理ツールを作成できることを目的に、COLLADA スキーマの詳しい仕様について解説することです。COLLADA は、ビデオゲーム、映画業界、CAD ツールなどで使われる、DCC (デジタルコンテンツ作成) アプリケーション、3D 対話型アプリケーションやツールチェーン、プロトタイピングツール、リアルタイム可視化アプリケーションなどとインポートやエクスポートを行う際に重要です。本ドキュメントでは、COLLADA スキーマの初期設計と仕様、さらに COLLADA エクスポートに最低限必要な条件について扱い、COLLADA の簡単なインスタンス文書の例については「付録 A」で示します。

対象読者

本ドキュメントは、パブリックに公開されているものです。COLLADA スキーマを利用したアプリケーションや、そういったアプリケーションのプラグインを作成したいプログラマの方を対象としています。

本ドキュメントの読者は、以下のような予備知識が必要です。

- XML と XML スキーマの知識
- NVIDIA® の Cg や Pixar の RenderMan® といったシェーディング言語に関する詳細な知識
- コンピュータグラフィックや OpenGL® のようなグラフィックス API に対する一般的な知識と理解

このドキュメントの内容

本ドキュメントは、以下の章から構成されています。

章・節	解説
1 章 設計上の考慮事項	COLLADA の設計に関する問題
2 章 ツールの要件とオプション	ツール作成者向けの COLLADA ツールの要件
3 章 設計上の考慮事項	COLLADA を理解して利用するために必要なスキーマや設計の概要、およびキーコンセプトの記述
4 章 プログラミングガイド	COLLADA を利用したプログラミングに関する詳しい説明
5 章 コア要素のリファレンス	COLLADA スキーマの基本的な要素に関する詳しいリファレンス
6 章 フィジックスリファレンス	COLLADA フィジックス要素の詳しいリファレンス
7 章 FX 入門	COLLADA FX 要素のコンセプトおよび使用上の注意
8 章 FX リファレンス	COLLADA FX 要素の詳しいリファレンス
9 章 B-Rep リファレンス	COLLADA B-Rep 要素の詳しいリファレンス
10 章 キネマティックスリファレンス	COLLADA キネマティックス要素の詳しいリファレンス
11 章 型	簡単な COLLADA 型いくつかの定義
付録 A : COLLADA のサンプル	COLLADA 文書の例
付録 B : プロファイル GLSL および GLES2 のサンプル	COLLADA FX <profile_GLSL>要素の詳しい例
用語集	この文書で使われる用語 (XML 用語を含む) の定義
総合索引	概念や主な用語の索引
COLLADA 要素の索引	専用のリファレンスページのない小さい要素を含む、すべての COLLADA 要素の索引

表記法

本文の意味を明確にするために、以下のような表記法を使用しています。

記法	解説
明朝体	説明文
<code><blue text></code>	XML 要素
Courier New	属性名
Courier New bold	ファイル名
青色	ハイパーリンク
斜体	新しい用語もしくは強調
<i>Italic Courier New</i>	コマンドやコード中の値のプレースホルダ
<i>element1 / element2</i>	<i>element1</i> は親要素、 <i>element2</i> は子要素。詳しくは、 http://www.w3schools.com/xpath/xpath_syntax.asp にある「Xpath Syntax」を参照

リファレンスの表記法と構成

スキーマ・リファレンスの各章では、COLLADA スキーマのシンタックスについて機能ごとに解説します。スキーマの個々の XML 要素は、以下のセクションに分けて解説してあります。

節	解説
概要	要素の名前と目的
コンセプト	要素が定義された背景と意義
属性	要素に適用可能な属性
関連要素	親要素などの関連要素の一覧
子要素	有効な子要素の一覧と説明
詳細	要素を使用する上で関係した情報
例	要素の使い方の紹介

子要素の規則

子要素テーブルには、指定された要素の子要素がすべてリストされます。

- 「主見出し項目を参照してください」というのは、その子要素の主見出し項目が「リファレンス」の章の中にあるので、子要素の使い方、属性、そのまた子要素などの詳細については、その項目を参照してくださいという意味です。
- メインエントリがリファレンス章にない場合、またはローカルな子要素のプロパティがメインエントリとは異なる場合、子要素の情報は、子要素テーブルまたは追加の要素専用サブセクションで指定されます。

以下に例を示します。

名前/例	解説	デフォルト値	出現回数
<code><camera></code>	主見出し項目を参照してください (これは、camera には「リファレンス」の中に主見出し項目があるので、詳細についてはその項目を参照してくださいという意味です)。		1 以上
<code><technique_common></code>	下記サブセクションを参照してください (これは、詳細はこの項目の中の別の表に記載されているという意味です)。	N/A (該当しないという意味)	

名前/例	解説	デフォルト値	出現回数
<code><yfov sid="..."></code>	属性、コンテンツ、関連する子要素などの記述を含む説明 (<code>yfov</code> には主見出し項目がないという意味。詳細は、この項目に記載)。	なし (斜体小文字は割り当てられていないという意味) NONE (NONE という値を意味する)	

子要素の順序

XML では、スキーマ定義の中に、親要素の中で要素が特定の順序で出現することを要求する記法を含めることができます。このリファレンスに、子要素が以下の順序で出現すると記載されている場合、以下のような宣言に対応しています。この宣言では、XML `<sequence>` 要素によって、`<extra>` が `<asset>` の前に出現する必要があることを示しています。

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

また XML には、複数の子要素が任意の順序で出現することができることを示す表記法もあります。このリファレンスに 2 つの子要素が任意の順序で出現できることが記載されている場合、最大値が無制限の XML `<choice>` 要素に対応しています。たとえば、以下の例では、`<image>` と `<newparam>` は、`<extra>` の前で、なおかつ `<asset>` の後に出現する必要がありますが、その間にさえあれば、任意の順序で出現することができます。unbounded 属性は、必要に応じて任意の数の要素を好きなように組合せてよいことを示しています。

```
<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>
```

参考資料

本ドキュメントの参考資料としては、以下のリソースがあります。

- NVIDIA® の Cg や Pixar の RenderMan® といったシェーディング言語に関する詳細な知識
- コンピュータグラフィックや OpenGL® のようなグラフィックス API に対する一般的な知識と理解
- 「*Collada: Sailing the Gulf of 3d Digital Content Creation*」Remi Arnaud および Mark C. Barnes 著。AK Peters, Ltd 発行。2006 年 8 月 30 日。ISBN-13: 978-1568812878
- [Extensible Markup Language \(XML\) 1.0, 2nd Edition](#)
- [XML Schema](#)
- [XML Base](#)
- [XML Path Language](#)
- [XML Pointer Language フレームワーク](#)
- [Extensible 3D \(X3D™\) encodings ISO/IEC FCD 19776-1:200x](#)
- [Softimage® dotXSI™ FTK](#)
- [NVIDIA® Cg Toolkit](#)

- [Pixar's RenderMan®](#)

COLLADA の詳細については、以下を参照してください。

- www.khronos.org/collada
- <http://collada.org>

1章 設計上の考慮事項

概要

COLLADA デジタルアセット交換スキーマの開発は、共同設計活動であり、多くの企業の設計者やソフトウェア技術者が関与しています。この章では、プロジェクトの始動時に設計者によって設定された重要な設計目標、思想、前提条件などについて概説します。

前提条件と依存関係

COLLADA の最初の設計段階で、協力者は以下のような前提条件について議論し、合意に達しました。

- COLLADA は、ゲームエンジンやランタイム配信形式ではありません。COLLADA は、対話型アプリケーション用のオーサリングツールのユーザや、コンテンツ作成パイプラインに有益なものであることが前提です。また、インタラクティブアプリケーションの多くにおいて、COLLADA は、最終的な配信メカニズムではなく、むしろ製作時のパイプラインとして利用されることを想定しています。たとえば、ゲームの多くでは、サイズを最適化されストリーミングに適した専用のバイナリファイルが使われています。
- またアーティストやユーザは、頂点/ピクセル・プログラム（シェーダ）のような高度なレンダリングテクニックを含んでいても、比較的簡単なコンテンツやテストモデルを素早く作成してテストしたいはずです。アーティストやディベロッパにとって、コンテンツのラピッドプロトタイプが重要であり、また人間にとって読みやすいテキストベースの形式や、有効な「空」コンテンツや部分的なコンテンツを作成する機能が不可欠です。

目標とガイドライン

COLLADA デジタルアセット交換スキーマの設計目標は、以下のようなものです。

- デジタルアセットを、専用のバイナリフォーマットから解放して、仕様の明確な、XML ベース、ロイヤリティフリー、オープンスタンダードのフォーマットに移行すること。
- 既存のコンテンツツールチェーンで直接 COLLADA アセットを利用できるようにし、ツールチェーンの統合を容易にするような、標準化された共通言語フォーマットを提供すること。
- なるべく多くのデジタルコンテンツユーザによって採用されること。
- あらゆるデータを COLLAD を介して利用できるように、簡単な統合メカニズムを提供すること。
- 3D アプリケーション間のデータ交換の共通基盤となること。
- ディベロッパ間のデジタルアセットスキーマ設計において、ディベロッパやデジタルコンテンツ制作（DCC）、ハードウェア、ミドルウェアベンダの間の共同作業を促進すること。

以降のセクションでは、COLLADA の目標について解説し、その結果と根拠についても述べておきます。

デジタルアセットの専用バイナリフォーマットからの解放

目標：デジタルアセットを、専用のバイナリフォーマットから解放して、仕様の明確な、XML ベース、ロイヤリティフリー、オープンスタンダードのフォーマットに移行すること。

デジタルアセットは、3D アプリケーションユーザの多くにとって、最も貴重な作品です。

ディベロッパは、大量のコストをかけて制作してきた大量のデジタルアセットを、不明瞭な専用のフォーマットで蓄積しています。このような専用ツールからデータをエクスポートするには、複雑な専用のソ

ソフトウェア開発キットを開発しなくてはならず、かなりの投資が必要です。また、このような投資を行った後でも、そのツールの外でデータを変更してから、再度インポートするというようなことは、依然として不可能です。データが陳腐化することを覚悟の上で、進化し続けるツールとともに、エクスポートをも更新し続ける必要があります。

また、ハードウェア・ベンダは、新たなハードウェアを利用するために、ますます複雑なアセットを必要とするようになってきています。必要なデータはツールの中に存在するかもしれませんが、そのデータをツールからエクスポートする方法がない場合もあります。このようなデータのエクスポート処理は、ディベロッパが高度な機能を利用する上でも、ハードウェア・ベンダが新製品を普及させる上でも障害となるような、複雑な処理だということです。

ミドルウェアやツールのベンダは、ミドルウェアやツールをあらゆるツールチェーンに組み込んで、ディベロッパが利用できるようにする必要がありますが、これはほとんど無理です。ミドルウェア・ベンダが成功するためには、拡張性のある独自のツールチェーンおよびフレームワークを提供して、それをディベロッパに採用させる必要があります。このため、ディベロッパは、同一のプロジェクトで複数の DCC ツールを使うことが困難であるのと同じように、同一のプロジェクトで複数のミドルウェアツールを使うことが不可能になります。

このような目標を達成するためになされた決定には、以下のようなものがあります。

- COLLADA では XML を使う。

XML は、構造化されたコンテンツに対して、きわめて明確に定義されたフレームワークを提供します。文字セット (ASCII、Unicode、シフト JIS) のような問題は、すでに XML 規格がカバーしているので、XML を利用するあらゆるスキーマは、そのまま国際的に利用できるようになります。また、XML は、インスタンス文書の例さえあれば、資料がなくても容易に理解することができますが、このようなことは、他のフォーマットでは稀です。XML パーサやテキストエディタは、プラットフォームを問わず、ほとんどあらゆる言語用のものがあるので、XML 文書は、ほとんどあらゆるアプリケーションから簡単にアクセスすることができます。

- COLLADA では、XML 中でバイナリデータを使わない。

ロードの容易さ、高速化、およびアセットサイズの削減などのために、頂点やアニメーションのデータを、何らかの 2 進表現で保存すべきではないかという議論は何度も交わされました。この主張は、残念ながら、開発チームの大多数のユーザにとって便利であるべきという要求に反します。その上、XML 文書にバイナリデータを保存するとトラブルの原因になりがちであり、Base64 エンコードでないと正しくサポートされません。このエンコードはデータサイズ逆に増やすことになります。COLLADA を完全にテキストベースのフォーマットにしておけば、大多数の選択肢をサポートすることができます。COLLADA には、バイナリデータを外部に保存して、それを COLLADA アセットから参照できるメカニズムが用意されています。

- COLLADA 共通プロファイルには、なるべく多くの共通データが含まれるように拡張を続ける。

標準の共通言語フォーマットを提供する

目標: 既存のコンテンツツールチェーンで直接 COLLADA アセットを利用できるようにし、ツールチェーンの統合を容易にするような、標準化された共通言語フォーマットを提供すること。

共通プロファイルは、この目的を実現するために設定されたものです。共通プロファイルの目標は、ツールが COLLADA アセットを読み込むことができ、共通プロファイルによって示されるデータを使うことができ、任意の DCC ツールをコンテンツ作成に利用できるようにすることです。

COLLADA アセットのツールチェーンへの統合を容易にするためには、スキーマや仕様だけではなく、COLLADA アセットを既存のツールチェーンへ統合するのに役立つように設計された API (COLLADA API) を提供する必要があると思われます。この新しい開発の方向性は、ディベロッパの労力を減らすばかりでなく、さまざまな可能性を新たに開き得るものです。COLLADA API のデザインは、既存のコンテンツツールチェーンで使われている固有のデータ構造との統合が容易にできるものである必要があります。

COLLADA では、ツールチェーンに組み込んで、プロ用のコンテンツ・パイプラインを構築することができるよう、さまざまなツールの開発が行えます。COLLADA は、メンテナンスの難しい一体化したツールチェーンよりも、むしろ、特定の用途に特化したさまざまなツールの設計を容易にします。内部もしくは外部で開発されたツールの再利用を容易にすることは、ディベロッパおよびツールやミドルウェアのメーカーに経済的かつ技術的な利点をもたらすので、デジタルアセット交換用の標準言語としての COLLADA の地位を強化します。

多数のデジタルコンテンツユーザによって採用される

目標：なるべく多くのデジタルコンテンツユーザによって採用されること。

COLLADA が採用されるためには、ディベロッパに役に立つものである必要があります。COLLADA ユーティリティが自分たちの問題に役立つかを評価するディベロッパのために、私たちは正確な情報を提供して、COLLADA ツールの品質を評価できるようにする必要があります。そのようなパラメータとしては、以下のようなものがあります。

- ツールの品質や適合度を評価するための、適合性試験スイートを提供する。
- 多くのディベロッパの役に立つために、ツール供給者が従うべき必要条件の一覧を仕様の中で提供する(これらの目標は、2章 ツールの要件とオプションの中で指定します)。
- ユーザから意見を集めて、必要条件や規格適合性試験スイートに反映させる。
- バグレポートの問題を管理し、実装上の疑問点を公開する。このためには、バグに優先順位をつけて、COLLADA パートナーの間で修正のスケジューリングを行う必要があります。
- アセット交換やアセットマネージメントに対するソリューションを促進する。
- COLLADA のエクスポートやインポートなどのツールを DCC ツールやミドルウェアのベンダに直接サポートさせる。

ディベロッパにとっては、パイプラインであらゆるパッケージを利用できるという利点があります。ツール・ベンダにとっては、より多くのユーザを獲得とする機会を得るといった利点をもたらします。

- 自動ビルド処理の中に、DCC ツールのエクスポートやインポートのタスクを組み込めるような、コマンドライン・インタフェースを提供する。

簡単な統合メカニズムの提供

目標：あらゆるデータを COLLAD を介して利用できる簡単な統合メカニズムを提供すること。

COLLADA は十分な拡張性を持っており、ディベロッパ固有のニーズに対応することが可能。ここから、以下のような目標が導き出されます。

- XML Schema 機能、および簡易コード生成を全面的に利用することにより、拡張処理が容易になるように COLLADA API の設計と今後の COLLADA の改良を図る。
- 拡張が用意なエクスポートやインポートを作成することを、DCC ベンダに奨励する。
- ディベロッパがまだ COMMON プロファイルに用意されていない機能を必要とする場合には、それらの機能をベンダがベンダ独自の拡張としてエクスポートやインポートに追加することを奨励する。

これに相当するのは、たとえばアンドゥスタックのようなツール固有の情報であるとか、あるいは、複雑なシェーダのような、すぐにでも必要であるにもかかわらず、COLLADA への導入がまだ検討中であるようなコンセプトです。

- このような情報を収集して、COMMON プロファイル中の問題を COLLADA の次バージョンで解決するためワーキンググループを導いて行く。

COLLADA アセット・マネジメントを使いやすくする。

- たとえば、DCC ツールで選択したデータの一部を特定のアセットとしてエクスポート可能にする。
- アセット識別を可能にし、正しいメタデータを使用するようにする。
- そのアセット・メタデータの使用を、エクスポートおよびインポートに対して強制する。

共通データ交換の基盤としての役割

目標：3D アプリケーション間のデータ交換の共通基盤となること。

この目標の最も重要な帰結は、COLLADA 共通プロファイルは常に拡張し続ける必要があるということである。COLLADA 共通プロファイルは元々、ポリゴンベースのモデル、マテリアルとシェーダ、若干のアニメーション、および DAG ベースのシーングラフをカバーしていました。将来は、より複雑なデータ形式を、ツール間の情報交換が可能になるような共通の方法でカバーする予定です。

デジタルアセットスキーマ設計の促進

目標：デジタルアセットスキーマ設計において、ディベロッパやデジタルコンテンツ制作（DCC）、ハードウェア、ミドルウェアベンダの間の共同作業を促進すること。

DCC ベンダ、ハードウェアベンダ、ミドルウェアベンダなどの市場セグメントの中や間には、激しい競争が存在します。けれども、彼らはすべて、開発時のデジタルコンテンツの問題を解決するために、コミュニケーションを行う必要があります。彼らが、共通のデジタルアセットフォーマットに関して協力しないと、市場全体のソリューションの最適化に、直接的な影響を及ぼします。

- ハードウェアベンダは、DCC ツールの露呈する機能の欠如に苦しんでいます。
- ミドルウェア・ベンダはツールチェーンの間の互換性の欠如に苦しんでいます。
- DCC ベンダは、ディベロッパを満足させるために必要なサポートや開発作業の量の多さに苦しんでいます。
- ディベロッパは、「使える」ツールチェーンを作成するために必要な投資額の大きさに苦しんでいます。

他の関係者と相談して、営業的もしくは技術的な利点を考慮に入れない限り、共通言語フォーマットの設計を主導することはできません。つまり、関係者全員を満足させるような目標を設定できる者は、この中にはいないのです。けれども、共通フォーマットは関係者全員に受け入れられる必要があります。この共通フォーマットが幅広く採用され、受け入れられるためには、主な関係者全員がその設計に満足する必要があるのです。

このような共同作業を行うための触媒として、ビデオゲーム業界の主導者の地位にある Sony Computer Entertainment（SCE）こそがふさわしかったのです。SCE には、ミドルウェアやディベロッパプログラムにおいて、ツール・ベンダとゲームディベロッパの仲介を行ってきた歴史があったので、この共通フォーマットの問題を、次世代プラットフォーム用のコンテンツの品質や量を向上させたい、という要求とともに、議題に挙げることができました。

この運動を常に SCE が推進しようとは思っておらず、時機がくれば、主導者の役割を他の協力者に譲り渡したいと思っています。

- ただ、主導者の役割をあまり早く譲ってしまうと、SCE が COLLADA を捨てたのではないかという印象を与えるので、パートナーに好ましくない影響を与えるでしょう。
- 役割を譲るのが遅すぎると、SCE の COLLADA に対する制御が強すぎると考える企業の、外部からの関与や長期的な投資を妨げることになるでしょう。

2章 ツールの要件とオプション

概要

COLLADA に完全に準拠したツールはどれも、COLLADA スキーマによって表現されたあらゆるデータの仕様に対応する必要があります。しかしながら、単純にスキーマ仕様に準拠する以上の要件が求められる場合もあります。たとえば、値の解釈を統一する、重要なユーザ設定可能なオプションを提供する、ツールに独自の機能を組み込む方法を詳しく規定する、といったものです。本章の目的は、こういった問題点に優先順位を付けることです。

それぞれの「要件」のセクションでは、個々の対応ツールがかならず実装しなければならないオプションについて詳しく述べてあります。ただし、指定された情報をアプリケーションで利用できない場合は例外です。たとえば、レイヤをサポートしていないツールでは（そういったレイヤ情報をエクスポートするのが通常は必須だと仮定した場合）、レイヤ情報をエクスポートできなくてもかまいません。しかしながら、レイヤをサポートしたツールは、すべてレイヤ情報を適切にエクスポートできなければなりません。

また、「オプション」のセクションでは、かならずしも実装する必要のない（ただし、一部のユーザにとっては重要で高度な）オプション機能などのメカニズムについて述べてあります。

以降で解説する要件はどれも、ツールの品質を保証すると同時に、COLLADA の目的に沿っていることを保証するために、ツールに組み込まれるものです。この重要なデータ解釈やオプションは、クロスプラットフォームのアプリケーション開発パイプラインの相互接続性やカスタマイズ性のニーズを満たすことを目指しています。解釈が曖昧だったり、必要なオプションを省略した場合、COLLADA を利用することで得られる利益や効用が著しく損なってしまう可能性があります。このセクションは、そういった問題が発生してしまうのを最小限に抑えるために記述されています。

このセクションで必要とされている各機能は、COLLADA 規格適合性試験に用意されている複数のテストケースでチェックしてあります。クロスグループが開発中の COLLADA 規格適合性試験スイートは、Maya[®]、XSI[®]、3ds Max[®] のようなアプリケーション用のエクスポート・インポートのテストを自動化するツールのセットです。各テストケースは、ネイティブコンテンツと、ツールの COLLADA インポート/エクスポートプラグインを通過した後のコンテンツを比較します。

エクスポート

範囲

COLLADA エクスポートの役割は、指定されたデータを特定の必須オプションにしたがって書き出すことです。

要件

階層および変換

データ	エクスポートの可/不可
平行移動	平行移動
拡大/縮小	スケーリング
回転	回転
親子関係	親子関係

データ	エクスポートの可/不可
静的オブジェクトのインスタンス化	静的オブジェクトのインスタンス化。そのようなオブジェクトは複数の変換を持つことができます。
アニメーションオブジェクトのインスタンス化	アニメーションオブジェクトのインスタンス化。そのようなオブジェクトは複数の変換を持つことができます。
傾斜	スキュー
透明度/反射率	透明度や反射率のための追加マテリアルパラメータ
テクスチャマッピング方式	テクスチャマッピング方式（円筒形や球形など）
ジオメトリなしの変換	ジオメトリのないもの（ロケータや NULL など）の変換が可能でなければなりません。

マテリアルとテクスチャ

データ	エクスポートの可/不可
RGB テクスチャ	任意の数の RGB テクスチャ
RGBA テクスチャ	任意の数の RGBA テクスチャ
焼付け手続き型テクスチャ座標	焼付け手続き型テクスチャ座標
共通プロファイルのマテリアル	共通プロファイルのマテリアル（PHONG、LAMBERT など）
面単位のマテリアル	プリミティブ単位のマテリアル

頂点属性

データ	エクスポートの可/不可
頂点テクスチャ座標	頂点ごとに任意の数のテクスチャ座標
頂点法線	頂点法線
頂点従法線	頂点従法線
頂点接線	頂点接線
頂点 UV 座標	頂点 UV 座標（テクスチャ座標とは区別されます）
頂点カラー	頂点カラー
カスタム頂点属性	カスタム頂点属性

アニメーション

特に指定がない限り、ユーザが指定したオプションに応じて、以下の種類のアニメーションをすべて、サンプルもしくはキーフレームを利用してエクスポートできなければなりません。

通常、アプリケーションでは、大まかなキーフレームと、複雑な制御や制約（コンストレイント）を利用してアニメーションを表現します。両者は、アニメーションが再生されて最終的な出力を提供する際に、アプリケーションによって組み合わせられます。アプリケーションがエクスポートするアニメーションデータを解析する際、そのアニメーションで利用されている制約やコントローラが全部は実装できず、オリジナルに対して忠実ではないアニメーションが出力されてしまう可能性があります。したがって、個々の変換データを指定された間隔で定期的にエクスポートするオプションが必要となります。サンプルがキーフレームより優先される場合は、ユーザがサンプリングレートを指定できなければなりません。

利用可能なアニメーション用のパラメータは、すべてエクスポートできなければなりません。そのようなパラメータとしては、以下のようなものがあります。

- マテリアルパラメータ
- テクスチャパラメータ
- UV 位置パラメータ
- 光源パラメータ
- カメラパラメータ

- シェーダパラメータ
- 全体環境パラメータ
- メッシュ構築パラメータ
- ノードパラメータ
- ユーザパラメータ

シーンデータ

データ	エクスポートの可/不可
空のノード	空のノード
カメラ	カメラ
スポットライト	スポットライト
平行光源	平行光源
点光源	点光源
環境光	環境光

エクスポートのユーザインタフェースオプション

データ	エクスポートの可/不可
三角形リストのエクスポート	三角形リスト
ポリゴンリストのエクスポート	ポリゴンリスト
前処理済み行列	前処理済み行列
単一の<matrix>要素	各ノードに<matrix>要素を1つだけ含むインスタンス文書（後述の「単一の<matrix>要素オプション」を参照）

単一の<matrix>要素オプション

COLLADA では、各類の変換要素をスタックによって指定した順序で合成する、という形式で「変換」を表現できます。このような表現は、アプリケーション内部で、段階的な複数の変換を利用する際に、変換を正しく保存したり交換したりするのに便利です。ただし、この機能をアプリケーションで実装する際には、前処理済みの単一の<matrix>要素だけを保存する機能をユーザオプションとして提供すべきです。

この要件には、変換スタック内部の特定の要素を対象としているデータ（アニメーションなど）はどれも、代わりに行列を参照する必要がある、という副作用があります。

コマンドライン操作

エクスポートのすべての機能は、コマンドラインインタフェースから実行できる必要があります。この要件の趣旨は、ユーザとのやり取りを必要とするエクスポートを避けることです。もちろん、便利なインタラクティブ・ユーザインタフェースがあるのは良いことですが、そのインタラクティブな機能は（必須ではなく）オプションである必要があります。

オプション

エクスポートは、新規のデータを任意に追加することができます。

シェーダのエクスポート

エクスポートは、シェーダ（Cg、GLSL、HLSL など）をエクスポートしてもかまいません。

インポータ

範囲

COLLADA インポータの役割は、指定された全データを、特定の必須オプションにしたがって読み込むことです。

一般に、インポータは、対応するエクスポートが行う機能すべての完全な逆機能を提供する必要があります。インポータは、「エクスポート」に記載されたあらゆるエクスポート・オプションの逆の機能を提供する必要がある場合があります。このセクションで説明するのは、インポータの要求がエクスポートの逆機能とは異なっていたり、より詳しい説明が必要な場合についてだけです。

要件

COLLADA に準拠したデータは、すべてインポート可能でなければなりません。一部のデータがツールによって認識されない場合であっても、後でエクスポートされるために保持されなければなりません。エクスポートされたデータの中に同期を必要とするものがあるかを外部のツールが認識するためには、`<asset>`要素が使われます。

オプション

インポータに固有のオプションはありません。

アーカイブへのパッケージ化

DCC ツールは、インポートおよびエクスポート時に `.zae` フォーマットをサポートする必要があります。`.zae` フォーマットは、任意の数の `.dae` ファイル (COLLADA 文書)、および、そこから参照される全コンテンツ (テクスチャ) の zip アーカイブです。

このアーカイブには、`manifest.xml` という名前の、XML でコーディングされた `<dae_root>` 要素を含むファイルが含まれている必要があります。この要素は、`.dae` ファイルを指す相対 URI を UTF8 でエンコードしたものです。この URI がフラグメントを含む場合、指定された要素が、アプリケーションが `.zae` アーカイブをロードする際の出発点となります。それ以外の場合には、`<scene>` 要素がアプリケーションが `.zae` アーカイブをロードする際の出発点となります。このどちらの条件も満たされない場合の動作は、未定義です。

`.zae` ファイル中の URI は、URI 標準にしたがってアーカイブのルートからの相対パスを利用することにより、アーカイブ中の他の任意のファイルを参照することができます。

また、アーカイブ自体も他のアーカイブ (zip、rar、kmz、zae) を含むことができます。それ自体が `.zae` アーカイブの中にある、入れ子になったアーカイブ中の文書を参照する URI のパスの中では、入れ子になったアーカイブの名前が使われます。

以下に例を示します。

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

相対 URI を使ってアーカイブ外部のコンテンツを参照することはできませんが、以下のように、絶対 URI を使ってコンテンツを参照することは可能です。

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

3章 スキーマのコンセプト

概要

COLLADA スキーマは、XML (eXtensible Markup Language) のデータベーススキーマです。COLLADA の機能セットを記述するには、XML Schema 言語が利用されています。

COLLADA スキーマを利用した文書、つまり、COLLADA スキーマが有効であると定義した XML 要素を含む文書は、COLLADA インスタンス文書と呼ばれます。COLLADA 文書に割り当てられたファイル拡張子は `.dae` (Digital Asset Exchange の頭字語) です。`.dae` という拡張子をインターネットで検索してみたところ、2003 年の時点では、使用例は見当たりませんでした。

この章では、基本的な XML 用語を簡単に紹介してから、COLLADA 要素が他の COLLADA 要素を参照する方法を説明し、さらに、COLLADA がどう動作するかに関する概念的な情報を提供します。

XML の概要

XML は、ファイルや文書やデータセットのコンテンツ、構造、セマンティックスを記述するための標準的な言語として利用できます。XML 文書は、主に複数の「要素」で構成されます。それぞれの要素は、開始・終了タグで囲まれた情報ブロックです。

```
<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>
```

この例には、4 つの要素 (`<node>`、`<translate>`、`<rotate>`、`<matrix>`) が含まれています。最後の 3 つの要素は、最初の `<node>` 要素の中にネストされています。つまり、要素を任意の深さにネストさせることが可能です。

要素には、その要素の特徴を記述した「属性」を指定することも可能です。たとえば、例の中の `<node>` 要素には、`"here"` という値が設定された `id` 属性が指定されています。そのため、`id` 属性が `"there"` である別の `<node>` 要素と区別できるようになります。この場合だと、属性の名前は `id` で、値は `here` となります。

XML の語彙の詳細に関しては「用語集」を参照してください。

アドレス構文

COLLADA では、インスタンス文書の要素や値を参照するために、以下の 2 種類のメカニズムを利用するようになっています。

- URI 参照方式：要素の `id` 属性を参照します。 `url` 属性や `source` 属性で使われます。詳しくは「URI 参照方式」のセクションで説明します。
- スコープ付き識別子 (SID) 参照方式：要素の `sid` 属性を参照します。 `target` 属性や `ref` 属性、`<SIDREF>` 要素や `<SIDREF_array>` 要素、その他、SID を含むか、もしくは参照している属性や要素で使われます。詳しくは「スコープ付き識別子 (SID) 参照方式」で説明します。

URI 参照方式

多くの要素の `url` 属性や `source` 属性では、インスタンス文書およびその中の要素を `id` 属性値によって特定する URI 参照方式を利用しています。

URI フラグメント識別子

COLLADA 要素の多くには、`id` 属性があります。このような要素は、ユニフォームリソース識別子 (URI) のフラグメント識別子記法を利用して参照できます。XML 文書内の URI フラグメント識別子の構文は、XML 仕様で定義されています。URI フラグメント識別子は、XPointer 構文に準拠している必要があります。COLLADA で URI を使って参照するのは、一意の識別子だけなので、ここで使われる XPointer の構文は *短縮ポインタ* (ショートハンド・ポインタ) 記法と呼ばれます。短縮ポインタで指定するのは、インスタンス文書中の要素の、`id` 属性の値です。

`url` 属性と `source` 属性では、URI フラグメント識別子の前にシャープまたは井桁の記号 (#) が付きません。`target` 属性は URI ではないため、シャープ記号はありません。これらの記法を使うと、たとえば、同一の `<source>` 要素を以下のように参照することができます。

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

たとえば、COLLADA インスタンス文書の中で定義された「Lt-Light」という ID を持つ光源は、`<instance_light url = "#Lt-Light">` という構文を使って参照することができます。以下の例では、`light` ノードの要素は、LIGHT ライブラリの中の `light` 要素を参照しています。

```
<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>
```

URI パスの構文

URI の構文は、インターネット技術タスクフォース (IETF) 文書 RFC3986 「Uniform Resource Identifier (URI): Generic Syntax」で定義されています。この文書は <http://www.ietf.org/rfc/rfc3986.txt> から入手できます。RFC3986 では、URI を、スキーム、オーソリティ、パス、クエリ、フラグメントの 5 つの階層から構成されるものとして定義しています。この構文を BNF で表すと、以下のようになります。

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

この `scheme` と `hierarchy-part` は必須です。ただし、`hierarchy-part` は空のパスでもかまいません。

URI 構文では、パス名の階層 (ディレクトリなど) をスラッシュ (/) で区切り、それ以外のパス中の特殊文字は、スペースを 16 進表現の「%20」に変換するなどのように、エスケープする必要があります。

ネイティブのファイルシステムパスのように、IETF フォーマットに準拠しない絶対パスは、準拠するように修正する必要があります。たとえば、URI 構文定義により、Windows の絶対パス「C:\foo\bar\my file#27.dae」は、現在の基底 URI への相対パス（「C」で始まる）として解釈することができます。さらに、バックスラッシュは、有効なセパレータではなく、他のテキストキャラクタと同じように扱うこともできます。アプリケーションの中には、Windows パスを探して、有効な URI に変換するものもありますが、あらゆるアプリケーションがそうするわけではありません。したがって、常に、有効な URI 構文を使うようにしてください。この例では、「/C:/foo/bar/my%20file%2327.dae」になります。

注：できるだけ、絶対パスよりも、参照元の文書の位置からの相対パスを使うのが、よいエンコード習慣です。

スコープ付き識別子 (SID) 参照方式

スコープ付き識別子 (sid) は、XML Schema 言語の NCName 型 (`xs:NCName`) として定義された `sid_type` 値で、その値は、幅優先走査を使って検索されるので、親要素のスコープ内の、同じ階層レベルの sid の集合の中で、一意でなくてはならないという制約があります。sid は、`<technique>` 要素全体の中では、一意でないことがあります。

SID へのパス含むもしくは参照する属性や要素は、すべて、`sidref_type` 型の値を使います。これは、COLLADA で定義された、`id` 属性および `sid` 属性がインスタンス文書中の要素を特定するための参照方式です。そのような属性や要素には、`target` 属性や `ref` 属性、`<SIDREF>` 要素や `<SIDREF_array>` 要素などの SID パスを含む、もしくは参照している属性や要素が含まれます。

SID 参照構文

スコープ付き識別子参照方式（以前はターゲットアドレッシングと呼ばれていた）の構文は、複数の部分に分かれています。

- 最初の部分は、インスタンス文書中の要素の `id`、またはこれが相対アドレスであるということを示すドット・セグメント (.) です。
- その後には、任意の数のスコープ付き識別子 (SID) が続きます。各識別子の先頭には、パスの区切り文字として、スラッシュ文字定数 (/) が付加されます。もしこの部分が空ならば、スラッシュ文字定数はありません。スコープ付き識別子は、最初の部分によって特定される要素の子要素を指定します。入れ子になった要素の場合、対象となる要素へのパスを指定するために、複数のスコープ付き識別子が使われることもあります。
- 最後の部分はオプションです。この部分は、要素の値を参照するための、C/C++スタイルの構造体メンバ選択構文です。この部分が存在しない場合には、`target` 要素の全メンバの値（たとえば、行列すべての値）が対象となります。この部分が存在する場合、以下の2つのうちのいずれかの形式をとることができます。
 - シンボルアクセスを示すメンバ値 (フィールド) の名前。この記法は、以下の要素から構成されます。
 - メンバ選択アクセスを示すピリオド記号 (.)。
 - メンバ値 (フィールド) の記号名。この章に記載された一般用語集サブセクションには、共通プロファイル中のこのフィールドの値が記載されています。アプリケーション定義の値も可能ですが、その場合、相互運用性は保証されません。
 - 配列アクセスを示すメンバ値 (フィールド) の基数的な位置。配列アクセス構文を利用できるのは、1次元のベクトルと2次元の行列中のフィールドを示すときだけです。この記法は、以下の要素から構成されます。
 - 配列選択アクセスを表す左括弧記号「(」。
 - ゼロ (最初のフィールド) で始まるフィールド番号。
 - 式の終わりを示す右括弧記号「)」。

SID 参照の例

以下は、スコープ付き識別子参照の例です。

```
<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>
```

以下の例では、3つの<channel>要素が、X、Y、Zで表される<translate>要素のメンバ値の各成分を対象にしています。同様に<rotate>要素のANGLEメンバは、それぞれシンボリック構文と、配列構文を使って2回対象にされます。

```
<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>
```

柔軟性と簡潔性を向上させるために、SIDのアドレッシングメカニズムはXML要素をスキップすることができます。id属性とsid属性をすべての間に位置する要素に割り当てる必要はありません。

たとえば、<optics>と<technique>の要素にsid属性を追加せずに、カメラのYを対象とすることが可能です。要素の中には、id属性やsid属性が利用できないものもあります。

また、対象となるテクニクのそれぞれに追加のアニメーションチャンネルを作成することなく、そのカメラの中の複数のテクニクの<yfov>を対象とすることもできます（テクニクとは、「スイッチ」であり、インポート時には、どちらか一方が選ばれ、両方が選ばれるわけではないので、単一の対象に解決されます）。以下に例を示します。

```
<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>
```

それぞれのテクニクでパラメータの名前が異なっていますが、それでも同じsid="YFOV"属性が使用されている点に注意してください。このようにすることも可能なのです。

スキップすることができなければ、要素を対象とすることは不安定なメカニズムになり、長い属性や多くの余分なアニメーション・チャンネルが必要となってしまうでしょう。異なるテクニックの対象パラメータが異なる値を必要とする場合、ディベロッパが個別のアニメーション・チャンネルを使用できることは言うまでもありません。

インスタンス化と外部参照

特定のオブジェクトの実際のデータ表現は、一度しか保存されないこともあります。しかしそのオブジェクトはシーン中に複数回出現することができます。オブジェクトは、表示されるたびに、さまざまな方法で変形される場合があります。シーン中での個々の表示は、オブジェクトのインスタンスと呼ばれる `instance_*` 要素の仲間を利用すると、COLLADA 文書で、オブジェクトデータのインスタンス化や共有を効率的に記述することができます。

各インスタンスは、位置、方向、スケールを決定するために、親のローカル座標系および対応する `<unit>` や `<up_axis>` の設定を継承します。

オブジェクトの各インスタンスは、他のインスタンスとまったく異なっても、あるいは、同じデータを共有していてもかまいません。ユニークなインスタンスはオブジェクトデータの独自のコピーを持ち、独立して操作することが可能です。固有ではない（共有）インスタンスは、データの一部もしくは全部を、そのオブジェクトの他インスタンスと共有します。ある共有されたインスタンスに変更を加えると、そのデータを共有している他のすべてのインスタンスに影響を及ぼすことになります。

この効果を生み出すメカニズムが現在のシーンやリソースにとってローカルな場合には、これはインスタンス化と呼ばれます。現在のシーンやリソースにとって外部のものである場合には、外部参照と呼ばれます。

注：COLLADA では、インスタンス間のデータ共有の方法を規定していません。データ共有ポリシーはランタイムアプリケーションに任されています。

COLLADA には複数の `instance_*` 要素が含まれており、この要素が対応する要素をインスタンス化します。たとえば、`<instance_animation>` は `<animation>` のインスタンスを記述します。インスタンス要素の `url` 属性は、適切な対応する型の要素を指します。

COLLADA コアの場合、インスタンス要素は以下のようになります。

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

COLLADA Physics の場合、インスタンス要素は以下のようになります。

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

COLLADA FX の場合、インスタンス要素は以下のようになります。

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

COLLADA キネマティックスの場合、インスタンス要素は以下のようになります。

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

共通プロファイル

COLLADA スキーマでは、設定プロファイルに準拠する情報表現のためのコンテキストを設定する `<technique>` 要素を定義しています。このプロファイル情報は、現在のところ、COLLADA スキーマの スコープ外にあります。

COLLADA デザインの1つの特徴として、共通プロファイル用のテクニックが存在することがあります。 `<technique_common>` と `<profile_COMMON>` の要素が、このプロファイルを明示的に起動します。 COLLADA コンテンツを解析するツールはすべて、この共通プロファイルを認識できなければなりません。したがって、COLLADA は共通プロファイルの定義を提供しています。

命名規則

COLLADA 共通プロファイルでは、正規の名前を定める規則として次のような命名規則を採用しています。

- パラメータの名前は大文字にします。たとえば、`<param>` 要素の `name` 属性の値は、次のようにすべて大文字になります。

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

- パラメータの型が COLLADA スキーマ、XML スキーマ、C/C++ 言語のいずれかの基本型に対応する場合には、小文字にします。それ以外の型名は、名前を構成する単語単位で頭文字を大文字にします。たとえば、`<param>` 要素の `type` 属性の値は、次のようにこの規則にしたがいます。

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

- 入力とパラメータのセマンティックス名は大文字にします。たとえば、`<input>` と `<newparam>` の要素の `semantic` 属性の値は、すべて大文字になります。

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

共通プロファイル要素

COLLADA 共通プロファイルは、`<technique_common>` または `<profile_COMMON>` 要素で宣言します。

```
<technique_common>
<!-- この範囲は共通プロファイルです -->
</technique_common>
```

<technique_common>要素の範囲外にある要素は、共通プロファイルはもちろん、いかなるプロファイルにも含まれません。たとえば、<polygons>要素の範囲内に存在する<input>要素は共通プロファイルには含まれず、テクニックやプロファイルによって変わりません。

テクニックの例と議論

より一般的には、<technique_common>および<technique>は、一緒に、COLLADA の複数表現や、代替プロファイルによる拡張のための、設計イディオムとして連携して使われます。COLLADA では、1 個の <technique_common>要素とゼロまたは任意個数の<technique>要素を使って、さまざまな要素に対して複数の表現を提供することができます。共通テクニック (technique_common、必須) は、要素の強く型付けされた表現です。

それ以外のテクニックは、代替表現を提供するベンダによって定義されます。各<technique>には、拡張が適用されるプラットフォーム (商品名など) を示す profile 属性があります。

拡張要素の代替表現は、それぞれ、対応するプロファイルの要素を記述する値を含んでいる必要があります。この表現は、プロファイルの間で一貫性を持たせることもできますが、かならずしも必須ではありません。言い換えれば、<technique_common>が環境光を記述している場合に、その特定のプロファイルにおいて、その光の<technique>が、それ以外のエリアライトなどを記述することも可能です。

```
<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>
```

注:

- 消費側のアプリケーションは、かならず<technique_common>を認識する必要があります。このテクニックの情報は、現在のランタイムで他のテクニックを認識できないときに、信頼性の高いフォールバックとして利用するように設計されています。
- エクスポートアプリケーションが<technique>要素を使っている場合には、<technique_common>が含まれている必要があります。
- 特定の場所にあるテクニックは、すべて、同一の概念やオブジェクトやプロセスを表していますが、そこに含まれる情報は、ターゲットアプリケーションによって、まったく異なっていることがあります。
- 消費側のアプリケーションは、複数の<technique>から利用するテクニックを選択することができます。明示的に選択しなかった場合には、<technique_common>がデフォルトになります。

共通用語集

このセクションでは、共通プロファイル中にあるパラメータやセマンティックスの正規名の一覧を記載しておきます。また、target 属性での参照に使われる、メンバ指定シンボル名の一覧もあります。

共通プロファイル中の<param>要素（データフロー）の name 属性と<newparam>要素の semantic 属性の値は、以下の通りです。

name 属性および semantic 属性の値	種類	代表的な コンテキスト	解説	デフォルト値
A	float_type	<material>、 <texture>	アルファカラー成分	なし
ANGLE	float_type	<animation>、 <light>	オイラー角	なし
B	float_type	<material>、 <texture>	青カラー成分	なし
DOUBLE_SIDED	論理値	<material>	レンダリングステート	なし
G	float_type	<material>、 <texture>	緑カラー成分	なし
P	float_type	<geometry>	3 番目のテクスチャ座標	なし
Q	float_type	<geometry>	4 番目のテクスチャ座標	なし
R	float_type	<material>、 <texture>	赤カラー成分	なし
S	float_type	<geometry>	1 番目のテクスチャ座標	なし
T	float_type	<geometry>	2 番目のテクスチャ座標	なし
TIME	float_type	<animation>	時間（秒）	なし
U	float_type	<geometry>	1 番目の汎用パラメータ	なし
V	float_type	<geometry>	2 番目の汎用パラメータ	なし
W	float_type	<animation>、 <controller>、 <geometry>	4 番目のデカルト座標	なし
X	float_type	<animation>、 <controller>、 <geometry>	1 番目のデカルト座標	なし
Y	float_type	<animation>、 <controller>、 <geometry>	2 番目のデカルト座標	なし
Z	float_type	<animation>、 <controller>、 <geometry>	3 番目のデカルト座標	なし

共通プロファイル中の<channel>要素の target 属性メンバの値は、以下の通りです。

target 属性の値	種類	解説
\(' # `)' [\(` # `)']	float_type	行列またはベクトルのフィールド
A	float_type	アルファカラー成分
ANGLE	float_type	オイラー角
B	float_type	青カラー成分
G	float_type	緑カラー成分
P	float_type	3 番目のテクスチャ座標
Q	float_type	4 番目のテクスチャ座標
R	float_type	赤カラー成分
S	float_type	1 番目のテクスチャ座標
T	float_type	2 番目のテクスチャ座標

target 属性の値	種類	解説
TIME	float_type	時間 (秒)
U	float_type	1 番目の汎用パラメータ
V	float_type	2 番目の汎用パラメータ
W	float_type	4 番目のデカルト座標
X	float_type	1 番目のデカルト座標
Y	float_type	2 番目のデカルト座標
Z	float_type	3 番目のデカルト座標

対象ベクトルおよび行列のフィールドについては、左右の括弧を使った配列添字の記法を利用できるということに注意してください。

このページは空白です。

4章 プログラミングガイド

概要

この章では、COLLADA プログラミングについて少し詳しく説明します。

COLLADA のパラメータについて

COLLADA FX 中の<param>要素 (core) や<newparam>要素は、指定されたスコープの中でバインド可能なパラメータを宣言します。バインドが可能であるためには、かならずしもパラメータの型が厳密に一致する必要はありません。ただし、この場合のパラメータの型は、integer から floating-point、float3_type から float4_type、Boolean から integer といった簡単な(アプリケーションにとって意味のある)変換やキャストで変換できるような互換性を持つ必要があります。

COLLADA では、一般的なパラメータ操作のために、次の要素を提供しています。

- <param>(core) : パラメータを定義して、即値として使う際の型および値を設定します。

COLLADA では、FX やキネマティックのパラメータ操作に、以下の基本要素を提供しています。

- <newparam> : パラメータを作成します。
- <setparam> : パラメータの型および値を変更または設定します。
- <param> (reference) : <newparam>によって作成された既存のパラメータを参照します。

FX 中のパラメータの詳細については、7 章 FX 入門を参照してください。

曲線補間

このセクションでは、<geometry>/<spline>および<animation>/<sampler>曲線の実装を曖昧さがないように記述するための情報を提供します。

概要

<geometry>/<spline>と<animation>/<sampler>は、どちらも曲線を定義します。1 番目の方は、表示できる曲線を表します。2 番目の方は、アニメーションの作成に使われる曲線を表します。

COLLADA では、曲線の補間に必要なデータを特定する<input>要素に semantic 属性を定義しています。この属性の値としては、POSITION、INTERPOLATION、LINEAR_STEPS、INPUT、OUTPUT、IN_TANGENT、OUT_TANGENT、CONTINUITY などがあります。さらに、ソース中の<Name_array>は、処理する曲線の種類をアプリケーションが指定することを可能にします。共通プロファイルでは、BEZIER、LINEAR、BSPLINE、および HERMITE という値が定義されています。このセクションでは、これらのセマンティックや曲線名が COLLADA でどのように使用されるかについて説明します。

スプライン曲線 (<geometry>/<spline>)

COLLADA のアニメーションの曲線の仕様 (<animation>/<sampler>) は、三次多項式曲線 (<geometry>/<spline>) の描画プリミティブのデータフロー定義を継承しています。

曲線は複数のセグメントによって定義されます。各セグメントは 2 つの端点によって定義されます。あるセグメントの終了点は、その次のセグメントの開始点でもあります。segment[i]の端点は、POSITION

$[i]$ と $POSITION[i+1]$ で与えられます。したがって、 n 個のセグメントを持つ曲線には、 $n+1$ 個の位置があります。点は、2次元でも3次元でも定義できます。

端点の間の曲線の挙動は、指定された補間法および追加の係数によって決まります。セグメントは、それぞれ、異なる方法で補間することができます。セグメントの補間法は、慣行として最初の点に関連付けられるので、 $segment[i]$ の補間法は $INTERPOLATION[i]$ に保存されます。 n セグメント曲線が開いている場合には、 $INTERPOLATION[n+1]$ は使われませんが、曲線が閉じている（最後のセグメントの終了点が最初のセグメントの開始点に接続されている）場合には、 $INTERPOLATION[n+1]$ がこの追加セグメントの補間法になります。`<spline>`要素の `closed` 属性は、曲線が閉じている (`true`) か、それとも開いている (`false`、デフォルト) かを示しています。

`LINEAR_STEPS` は、必要な曲線の補間精度を示すオプションの `<input>` セマンティックです。一般に、特定セグメント内の複雑な曲線は、線分の再分割を利用した近似によって実現されます。`LINEAR_STEPS` は、その再分割の回数を決めます。

COLLADA におけるスプライン定義の例を、次に示します。

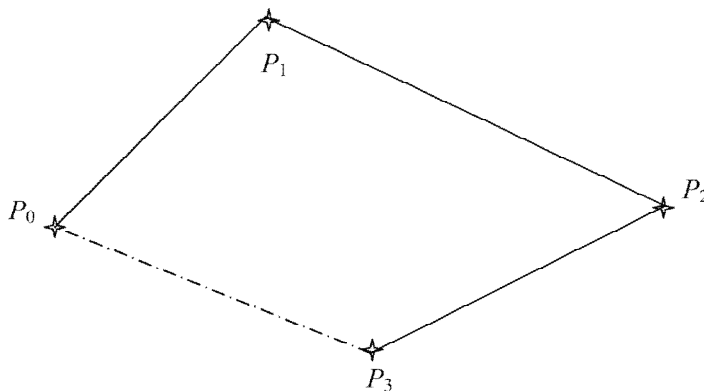
```
<spline closed="true">
  <source id= "positions" >
    <!-- n+1 個の値を持つ --> </source>
  <source id="interpolations" >
    <!-- n+1 個の値を持つ --> </source>
  <source ... >
    <!-- n+1 個の値 --> </source>
  <source ... >
    <!-- n+1 個の値 --> </source>
  <control_vertices>
    <input semantic="POSITION" source = "#positions"/>
    <input semantic="INTERPOLATION" source="#interpolations"/>
    <input ...<!--補間方式によっては追加の入力を記述 -->
```

`CONTINUITY` は、曲線のデザイン時に、接線に対して設定された制約を示すオプションの `<input>` セマンティックです。`CONTINUITY` の有効な値は次のとおりです。

- C_0 : 点に関して連続。曲線は、接続点において同じ点を共有します。
- C_1 : 導関数が連続。曲線は、接続点においてパラメータの微分を共有します。
- G_1 (幾何学的に連続) : C_0 の条件に加えて、さらに、接線が同じ方向を向く必要があります。

1 次スプライン

線形補間のもっとも単純な補間法です。任意のセグメントの曲線が、その開始点と終了点の間の直線になるからです。セグメント内に制御点を追加する必要はまったくありません。次の図は、4 つの位置 (P_0 、 P_1 、 P_2 、 P_3) のそれぞれの組の間を `LINEAR` で補間した、3 セグメントの閉じた `<spline>` を示しています。これは閉じたスプラインなので、最後 (4 番目) のセグメントは P_3 と P_0 の間になります。



1次スプラインの式は、次のようになります。

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

この式の別の表現方法としては、次のように行列形式を用いる方法があります。

$$L(s) = SMC$$

$$S = [s \ 1]$$

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

$$C = [P_0 \ P_1]$$

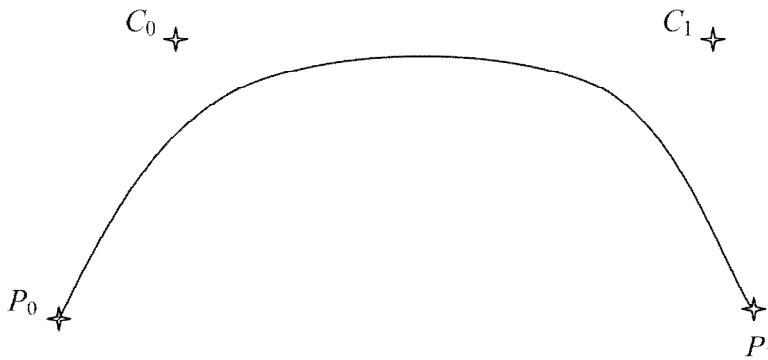
COLLADA では、**LINEAR** segment $[i]$ のジオメトリベクトルは、次のように定義されます。

- P_0 は、**POSITION** $[i]$
- P_1 は、**POSITION** $[i+1]$

ベジェおよびエルミートスプライン

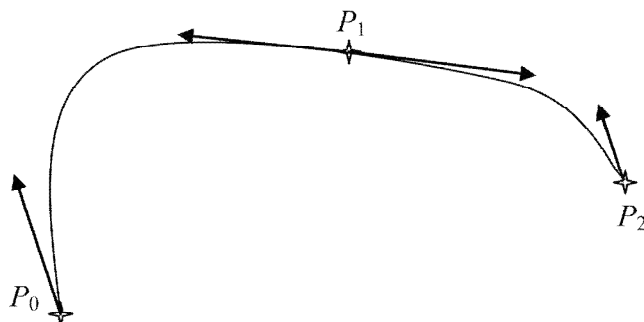
セグメントは、3次ベジェスプラインにすることもできます。この場合、曲線の振る舞いを制御するための点が、さらに2つ必要です。以下の例は、**BEZIER** を使って補間した1つのセグメントを示しています。このセグメントも、開始点と終了点は先ほどと同じ (P_0 、 P_1) ですが、曲線を計算するための追加情報を提供する制御点がさらに2つ (C_0 、 C_1) あります。

注：COLLADA 1.4.1 がサポートするのは3次のベジェ曲線だけなので、どのセグメントにもかならず制御点が2つ存在します。



HERMITE は **BEZIER** と等価ですが、制御点 C_0 、 C_1 の代わりに、接線 T_0 、 T_1 を指定します。

次の図は、3次エルミート補間を用いた2つのセグメントから成る曲線を示したものです。



点 P_1 には、2本の接線が関連付けられています。2番目のセグメントの開始部分を定義する接線は、 P_1 から出てくる (OUT) セグメント用なので、**OUT_TANGENT** と呼ばれます。1番目のセグメントの終了部分を定義する接線は、 P_1 に入っていく (IN) セグメント用なので、**IN_TANGENT** と呼ばれます。

つまり、`IN_TANGENT[1]`は segment[0]の終了接線、`OUT_TANGENT[1]`は segment[1]の開始接線です。
HERMITE と **BEZIER** は同一の多項式補間であり、次の変数の変化は次の式によって表わされます。

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

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

式はパラメトリック式として与えられます。曲線上のあらゆる点は、0 から 1 に変化するパラメータ s を使って計算されます。 s が 0 のときには、この式は P_0 を表します。 s が 1 のときには、この式は P_1 を表します。この範囲の外の値に対しては、式は定義されません。

COLLADA では、`<input>`セマンティックス `IN_TANGENT` および `OUT_TANGENT` は、補間法に応じて、接線もしくは制御点のいずれかを保存するために使われます。

3 次ベジェスプラインの式は、次のようになります。

$$B(s) = P_0(1-s)^3 + 3C_0s(1-s)^2 + 3C_1s^2(1-s) + P_1s^3, s \in [0,1]$$

この式を表現するためによく使われる別の方法としては、次のように行列形式を用いる方法があります。

$$B(s) = SMC$$

$$S = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix}$$

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

$$C = \begin{bmatrix} P_0 & C_0 & C_1 & P_1 \end{bmatrix}$$

COLLADA では、ベジェ segment[i]のジオメトリベクトルは、次のように定義されます。

- P_0 は、`POSITION[i]`
- C_0 は、`OUT_TANGENT[i]`
- C_1 は、`IN_TANGENT[$i+1$]`
- P_1 は、`POSITION[$i+1$]`

以下に、2 つのセグメントを持つ 2 次元 (X、Y) ベジェ曲線の COLLADA サンプルを示します。

```
<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>    <!-- 開いた曲線の
    場合は最後のエントリを無視 -->
    <technique_common>
      <accessor>
        ... <param name="INTERPOLATION" type="Name" ...
      </accessor> </technique_common> </source>
    </technique_common>
  </source>
  <source id="in_tangents" >
    <float_array count="6" ...><!-- 開いた曲線の場合は最初のエントリを無視 -->
```

```

    <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" ...><!-- 開いた曲線の場合は最後のエントリを無視 -->
  </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>

```

3次エルミートスプラインの式は、次のようになります。

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

これは、行列形式では次のようになります。

$$H(s) = SMC$$

$$S = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix}$$

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

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

where :

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

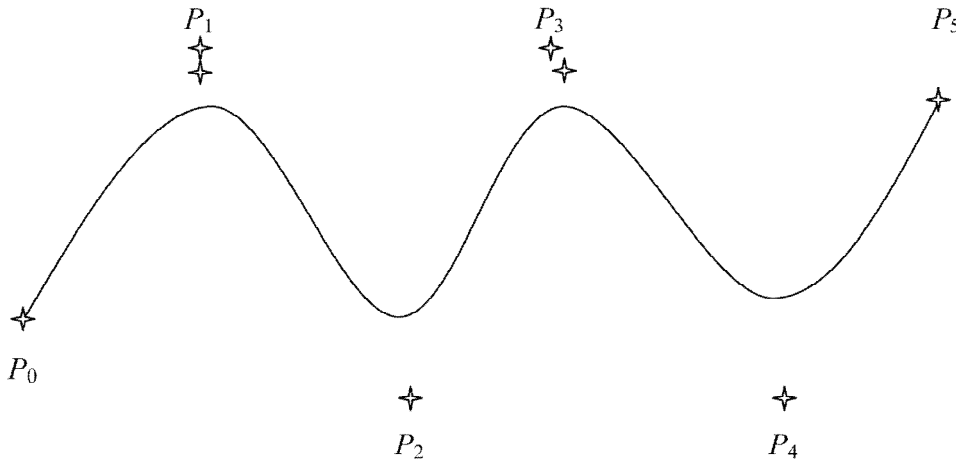
$$T_1 = 3(P_1 - C_1)$$

COLLADA では、**HERMITE** segment $[i]$ のジオメトリベクトルは、次のように定義されます。

- P_0 は、**POSITION** $[i]$
- P_1 は、**POSITION** $[i+1]$
- T_0 は、**OUT_TANGENT** $[i]$
- T_1 は、**IN_TANGENT** $[i+1]$

B スプライン

B スプライン (基底スプライン) は一連の制御点によって定義されます。B スプラインでは次の図のように、曲線が最初と最後の点のみを通ることが保証されます。



COLLADA 1.4.1 では一様な 3 次 B スプライン補間が定義されています。端点 P_1 、 P_2 の間の曲線の挙動は、直前の点 (P_0) および直後の点 (P_2) を使って、下記の式によって与えられます。開いた曲線の場合、 P_0 より前の点は存在しません。したがって、 P_0 について P_1 と点対称の点が使われます。同じ論理が最後の点についても適用されます。 P_4 について P_5 と点対称の点が上の式で使われます。

P_i と P_{i+1} の間のセグメントの B スプラインは、以下のような行列形の式によって記述されます。

$$B_i(s) = SMC$$

$$S = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix}$$

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

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

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

COLLADA で、この B スプラインジオメトリベクトルを定義するために必要なのは、POSITION と INTERPOLATION の 2 つの <input> 要素を使うことです。

- $P_i = \text{POSITION}[i]$
- $\text{INTERPOLATION}[i] = \text{BSPLINE}$.

カーディナルスプライン

カーディナルスプラインとは、その接線が端点とテンションパラメータで定義される 3 次エルミートスプラインのことです。接線は、次のように、セグメントの直前の点と直後の点から計算されます。

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

ただし、 c はテンションパラメータで、接線の長さを調節するための定数です。COLLADA 1.4 では、このパラメータは別個に指定されるのではなく、サンブラへの OUT_TANGENT 入力および IN_TANGENT 入力によって指定される接線に組み込まれます。

カーディナルスプラインは、BSPLINE と同じジオメトリベクトル C を使って、行列形式で表すことができます。

$$D_i(s) = SMC$$

$$S = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix}$$

$$t = (1-c)/2$$

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

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

COLLADA のスキン変形 (スキニング)

スキニングは、<node>要素によって表される変換のセットに対して、頂点を線形に重み付けすることにより、ジオメトリを変形するテクニックです。特定のジオメトリに影響を与えるノードは、通常、「スケルトン」と呼ばれる単一の階層構造にまとめられます。ただし、影響するノードがこの階層構造と無関係な部分に存在することも可能です。そのような階層構造のノードは、スケルトンの「ジョイント」を表します。「ジョイント」と「ボーン」を混同しないようにしてください。「ボーン」は、2つのジョイントを結ぶ架空の線分のことです。

このセクションには、COLLADA 内のスキニングに関する説明や式が記載されています。

概要

スキニング<controller>は、ジオメトリをスケルトンに関連づけます。スケルトンは、静止位置 (バインドポーズ) にあるとみなされます。バインドポーズというのは、スケルトンがジオメトリにバインドされたときの、各ジョイントの世界空間内の位置および方向のことです。この世界空間は、他の世界空間座標系と区別するために、バインドポーズ空間とも呼ばれます。

スキニング<instance_controller>は、スキニング<controller>をインスタンス化して、ランタイムスケルトンに関連づけます。COLLADA では、スキニングをオブジェクト空間で定義するので、<instance_controller>階層中の<node>の位置は、最終的な頂点場所に影響します。オブジェクト空間スキニングは、最大限の柔軟性をもたらします。オブジェクト空間スキニングの出力は、<instance_controller>を含む<node>座標系のオブジェクト空間の頂点です。

オブジェクト空間内で頂点をスキニングする場合、スキニングされた同一のジオメトリを、複数の場所でレンダリングするのが簡単かつ効率的です。このことは、同じポーズの複数のアクターを、異なる場所で同時に表示する際に重要です。このような状態は、大群衆、並行マシン、複数アクターの振付けなどのアニメーションでよく発生します。同じポーズのアクターは、それぞれ、スキニングされた同じ頂点データを共有します。

スキニングの定義

COLLADA 内のスキニングに関する定義

- バインド形状：<skin>要素の source 属性によって参照されるメッシュの頂点。
- バインド形状行列：メッシュがスケルトンにバインドされたときの、バインド形状の変換を表す単一の行列。この行列は、バインド形状をオブジェクト空間からバインド空間に変換します。

- ジョイント：スケルトンのボーンは、ジョイントによって定義されます。各ボーンのベースは、次のジョイントにまで伸びています。バインド空間では、ジョイントはバインドポーズの中にあります。バインドポーズはスケルトンのジョイントがバインド形状にバインドされた時の位置および方向です。<visual_scene>の中では、ジョイントの向きは、アクターのポーズとアニメーションによって定められます。ワールド空間内でのジョイントの位置は、メッシュの位置と一致しないことがあります。一致するかどうかは、メッシュをオブジェクト空間に変換するために使われるルート行列に依存します。
- 重み：特定のジョイントが、頂点の最終的な位置に影響を与える度合い。頂点は、任意の数のジョイントに対して、重みの合計が1に等しくなるように重み付けされます。頂点の変換は、各ジョイントについて個別に行われます。複数の頂点変換結果は、重みによって線形結合され、スキニング後の頂点が生成されます。
- 逆バインドポーズ行列：このジョイントにバインド形状がバインドされた際の、ジョイントのバインド空間変換行列の逆行列。

スキニングの式

バインド形状中の各頂点 v に対するスキニングの計算は、以下の通りです。

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

- n : 頂点 v に影響するジョイントの数
- BSM: バインド形状行列
- $IBMi$: ジョイント i の逆バインドポーズ行列
- JMi : ジョイント i の変換行列
- JW : 頂点 v に対するジョイント i の影響の重み

注: v 、BSM、 $IBMi$ 、 JW は、骨格アニメーションに関する定数です。アプリケーションによっては、あらかじめ BSM に $IBMi$ を掛けたり、 v に BSM を掛けておくことがあります。

式についての注意

ワールド空間スキニングとオブジェクト空間スキニングの主な違いは、 JMi の定義にあります。

- ワールド空間スキニングでは、 JMi は、オブジェクト空間からワールド空間へのジョイントの変換行列です。
- オブジェクト空間スキニングでは、 JMi は、オブジェクト空間から別のオブジェクト空間へのジョイントの変換行列です。最初のオブジェクト空間変換は、バインド形状が最初に定義されたオブジェクト空間におけるジオメトリの変換です。二番目のオブジェクト空間変換は、<instance_controller><skeleton>によって選択された、対象オブジェクト空間への変換です。

この変換を概念的に説明するには、これらの空間の間にある空間を考えて、最終的な行列を構築するのが最も簡単です。1つの方法は、ワールド空間スキニングで見たように、ジオメトリオブジェクト空間からワールド空間に移動してから、スケルトンのワールド空間行列の逆行列を使って、ワールド空間からスケルトンのオブジェクト空間への変換を行うことです。

ここで参照しているスケルトンの行列が、バインド形状行列ではないことに留意することが大切です。この行列は、<instance_controller><skeleton>によって参照されている<visual_scene>の中の<node>であり、同じ値を持たない可能性があります。<instance_controller><skeleton>によって参照される<node>を使うと、シーン中のスケルトンの位置を決め、アニメーションを行う際に、最大限の柔軟性が得られます。この柔軟性により、シーン中のスキンのバインド空間や、スケルトンのオブ

ジェクト空間に対するあらゆる制限が取り除かれます。これは、アニメーションの位置が、常に、選択したルートノードとの相対的位置で決まるためです。

仮に、その代わりにバインド形状行列を使ったとすると、スケルトンの位置やアニメーションは、常に、シーン中のバインド形状行列の位置や方向と相対的に決めなければならなくなるでしょう。複数のキャラクターに対して同時にアニメーションを実行している場合、キャラクター同士が重なり合う可能性が高いので、位置や方向がおかしくなる可能性があります。`<instance_controller><skeleton>`によって参照されるノードの世界空間行列を、スキンのバインド形状行列と同一にして、先に述べた動作に一致させる、もしくは、単位行列と同一にして、ワールド空間スキニングの動作と一致させられることには注意すべきです。これらのオプションを有効にすると、オブジェクト空間スキニングは最も柔軟性のあるモデルになります。

前述の式の結果は、スケルトン相対オブジェクト空間の頂点なので、最終的な頂点を生成するには、オブジェクト空間からワールド空間への変換を掛け算する必要があります。通常、この最後の手順は、頂点シェーダで行われ、使われる行列は、`<instance_controller>`を所有するノード用のワールド空間変換行列になります。

スケルトンとその`<instance_controller>`に対して同時にアニメーションを行う簡単な方法が1つあります。`<skeleton>`のルートの中に`<instance_controller>`を配置すれば、最後の2つの行列は互いに打ち消し合うので、ワールド空間スキニングと同じような解決が得られるというわけです。メッシュは、常にスケルトンに従います。

このページは空白です。

5章 コア要素のリファレンス

概要

このセクションでは、効果（FX）フレームワークおよびフィジックスフレームワーク以外の、COLLADAスキーマの基本機能およびインフラストラクチャを表すコア要素を取り扱います。その中には、ジオメトリ、アニメーション、スキニング、アセット、およびシーンを記述する要素が含まれています。

要素のカテゴリ分類

この章では、要素をアルファベット順に記載します。関連する要素を見つけやすくするため、以下の表では、要素をカテゴリ別に記載しています。

アニメーション

animation	アニメーション情報の宣言をカテゴリ化します。
animation_clip	アニメーションクリップとしてまとめて利用するアニメーション曲線のセクションを定義します。
channel	アニメーションの出力チャンネルを宣言します。
instance_animation	COLLADA アニメーションリソースをインスタンス化します。
library_animation_clips	<animation_clip> 要素を保存するライブラリを提供します。
library_animations	<animation> 要素を保存するライブラリを提供します。
sampler	アニメーション用の補間サンプリング関数を宣言します。

カメラ

camera	シーン階層またはシーングラフへのビューを宣言します。<camera>要素には、カメラの光学的な特性や撮影装置を表す要素が含まれます。
imager	フィルムや CCD といったカメラの画像センサーを表します。
instance_camera	COLLADA カメラリソースをインスタンス化します。
library_cameras	<camera> 要素を保存するライブラリを提供します。
optics	カメラに搭載されている、画像を画像センサーの上に投影する装置を表します。
orthographic	正投影カメラの視野を記述します。
perspective	透視カメラの視野を記述します。

コントローラ

controller	汎用的な制御情報の宣言をカテゴリ化します。
instance_controller	COLLADA コントローラリソースをインスタンス化します。
joints	ジョイントノード（スケルトンノード）と属性データを関連付けます。
library_controllers	<controller> 要素を保存するライブラリを提供します。
morph	静的メッシュの複数セットをブレンドするのに必要なデータを記述します。
skeleton	スキンコントローラが必要なジョイントノードの検索をどこから始めるのかを指定します。
skin	重み付けブレンドを用いたスキニングの記述に必要な頂点とプリミティブの情報

	を含みます。
<code>targets</code>	モーフィングターゲットと、その重み付け、さらに関連したユーザ定義属性を宣言します。
<code>vertex_weights</code>	スキニング処理で利用される関節と重み付けの組み合わせを記述します。

データフロー

<code>accessor</code>	<code><float_array></code> 、 <code><int_array></code> 、 <code><Name_array></code> 、 <code><bool_array></code> 、 <code><IDREF_array></code> のいずれかの配列要素へのアクセスパターンを宣言します。
<code>bool_array</code>	Boolean 値の同種の配列の保存場所を宣言します。
<code>float_array</code>	浮動小数点値の値を含んだ同種の配列の保存場所を宣言します。
<code>IDREF_array</code>	ID 参照値を含んだ同種の配列の保存場所を宣言します。
<code>int_array</code>	整数値を保持する同種の配列を格納します。
<code>Name_array</code>	シンボル名値を保持する同種の配列を格納します。
<code>param(core)</code>	親要素のパラメータに関する情報を宣言します。
<code>SIDREF_array</code>	スコープ付き識別子の参照値を含んだ同種の配列の格納場所を宣言します。
<code>source</code>	<code><source></code> 要素は、その <code><source></code> 要素を参照する <code><input></code> 要素のセマンティックスにしたがって値を提供するデータのリポジトリを宣言します。
<code>input(shared)</code>	データソースの入力セマンティックスを宣言します。
<code>input(unshared)</code>	データソースの入力セマンティックスを宣言します。

拡張性

<code>extra</code>	親要素に関する任意の追加情報を提供します。
<code>technique(core)</code>	コンテンツの一部の処理するために使われる情報を宣言します。各テクニックは、対応するプロファイルに適合させておく必要があります。
<code>technique_common</code>	すべての COLLADA 実装がサポートしなければならない共通プロファイルの特定の要素の情報を指定します。

ジオメトリ

<code>control_vertices</code>	スプラインの制御頂点 (CV) を記述します。
<code>geometry</code>	シーン内のオブジェクトの視覚的形狀や外観を記述したものです。
<code>instance_geometry</code>	COLLADA ジオメトリリソースをインスタンス化します。
<code>library_geometries</code>	<code><geometry></code> 要素を保存するライブラリを提供します。
<code>lines</code>	<code><mesh></code> 要素においてジオメトリプリミティブと頂点属性の結び付きを宣言します。
<code>linestrips</code>	<code><mesh></code> 要素においてジオメトリプリミティブと頂点属性の結び付きを宣言します。
<code>mesh</code>	頂点とプリミティブの情報を使って基本的なジオメトリメッシュを記述します。
<code>polygons</code>	<code><mesh></code> 要素においてジオメトリプリミティブと頂点属性の結び付きを宣言します。
<code>polylist</code>	<code><mesh></code> 要素においてジオメトリプリミティブと頂点属性の結び付きを宣言します。
<code>spline</code>	複数セグメントから構成されるスプラインを記述します。制御点 (CV) とセグメント情報が含まれます。
<code>triangles</code>	頂点属性をバインドし、それらの頂点を個々の三角形に編成するために必要な情報を提供します。
<code>trifans</code>	頂点属性をバインドし、それらの頂点を連結された三角形に編成するために必要な情報を提供します。
<code>tristrips</code>	頂点属性をバインドし、それらの頂点を連結された三角形に編成するために必要な情報を提供します。

<code>vertices</code>	メッシュ頂点の属性や識別情報を宣言します
-----------------------	----------------------

ライティング

<code>ambient(core)</code>	環境光源を記述します。
<code>color</code>	親光源要素のカラーを表します。
<code>directional</code>	並行光源を表します。
<code>instance_light</code>	COLLADA 光源リソースをインスタンス化します。
<code>library_lights</code>	<image>要素を保存するライブラリを提供します。
<code>light</code>	シーンを照らす光源を宣言します。
<code>point</code>	点光源を記述します。
<code>spot</code>	スポットライトソースを表します。

数学

<code>formula</code>	式を定義します。
<code>instance_formula</code>	COLLADA 式リソースをインスタンス化します。
<code>library_formulas</code>	<formula>要素を保存するライブラリを提供します。

メタデータ

<code>asset</code>	その親要素に関連したアセット管理情報を定義します。
<code>COLLADA</code>	COLLADA スキーマに準拠したいくつかのコンテンツを含むドキュメントのルートを宣言します。
<code>contributor</code>	アセット管理用の作成者情報を定義します。
<code>geographic_location</code>	アセット管理のためにアセットの場所を定義します。

パラメータ

<code>newparam</code>	新たに名前付きパラメータオブジェクトを作成して、型と初期値を割り当てます。
<code>param(reference)</code>	定義済みのパラメータを参照します。
<code>setparam</code>	先に定義されたパラメータに新しい値を割り当てます。

シーン

<code>evaluate_scene</code>	<visual_scene>の評価方法を指定する情報を宣言します。
<code>instance_node</code>	COLLADA ノードリソースをインスタンス化します。
<code>instance_visual_scene</code>	COLLADA <code>visual_scene</code> リソースをインスタンス化します。
<code>library_nodes</code>	<node>要素を保存するライブラリを提供します。
<code>library_visual_scenes</code>	<visual_scene>要素を保存するライブラリを提供します。
<code>node</code>	シーン中の各要素の階層構造的な関係を表します。
<code>scene</code>	COLLADA リソースのコンテンツで視覚化可能な情報セット全体を表します。
<code>visual_scene</code>	COLLADA リソースのコンテンツで視覚化可能な情報セット全体を表します。

変換

<code>Lookat</code>	カメラの照準を定めるのに適した位置や方向の変換情報が含まれます。
<code>matrix</code>	座標系中の各点や、座標系そのものに対する数学的な変更を表します。

rotate	オブジェクトを軸のまわりで回転させる際の回転方法を指定します。
scale	オブジェクトのサイズを変更する方法を指定します。
skew	オブジェクトを特定の軸に沿って変形させる方法を指定します。
translate	座標系中の物体を回転させることなく、位置だけを変更します。

accessor

カテゴリ: データフロー

概要

accessor 要素は、配列データソースからの値のストリームを記述します。

コンセプト

`<accessor>`要素は、`<float_array>`、`<int_array>`、`<Name_array>`、`<bool_array>`、`<IDREF_array>`のいずれかの配列要素、もしくは、外部配列ソースに対する、アクセスパターンを宣言します。配列の構成は、`offset` および `stride` 属性を変えることによって、インタリーブ方式にも非インタリーブ方式にもできます。

アクセッサの出力は、その子要素である`<param>`要素によって記述されます。

属性

`<accessor>`要素には、以下の属性があります。

<code>count</code>	<code>uint_type</code>	配列へのアクセス回数。必須。
<code>offset</code>	<code>uint_type</code>	配列から最初に読み出す値のインデックス。デフォルト値は0です。オプション。
<code>source</code>	<code>xs:anyURI</code>	URI 表現を利用してアクセスする配列の場所を表します。必須。この要素は、COLLADA の配列要素を参照することも、インスタンス文書のスコープ外の配列データソースを参照することもできます。ソースは、COLLADA 文書である必要はありません。
<code>stride</code>	<code>uint_type</code>	配列へ毎回アクセスする際に、1つの単位としてみなす値の数。デフォルト値は1で、これは単一の値にアクセスすることを意味します。オプション。

関連要素

`<accessor>`要素は、以下の要素と関連性があります。

親要素	<code>source / technique_common</code>
子要素	下記サブセクションを参照してください。
その他	<code>bool_array</code> , <code>float_array</code> , <code>IDREF_array</code> , <code>int_array</code> , <code>Name_array</code> , <code>mesh</code> , <code>convex_mesh</code> , <code>SIDREF_array</code>

子要素

名前/例	解説	デフォルト値	出現回数
<code><param></code> (データフロー)	<code><param></code> 要素の <code>type</code> 属性は、この要素が <code><accessor></code> 要素の子であるときには、 <code>int</code> 、 <code>float</code> 、 <code>Name</code> 、 <code>bool</code> 、 <code>IDREF</code> 、 <code>SIDREF</code> の配列型に限定されます。主見出し項目を参照してください。	なし	0以上

詳細

`<param>`要素の数と順序は、`<accessor>`要素の出力を定義します。各パラメータは、パラメータと値が指定された順序に応じて値にバインドされます。データの並べ替えは行われません。name 属性のない`<param>`要素は、その値が出力の一部ではないので、要素がバインドされないことを示しています。

stride 属性の値は、`<param>`要素の数以上である必要があります。stride の値で示されているよりも`<param>`要素の数が少ない場合、バインドされていないデータソースの値はスキップされます。

例

以下に`<accessor>`要素の基本的な使い方の例を示します。

```
<source>
  <int_array name="values" count="9">
    1 2 3 4 5 6 7 8 9
  </int_array>
  <technique_common>
    <accessor source="#values" count="9">
      <param name="A" type="int"/>
    </accessor>
  </technique_common>
</source>
```

default stride = 1										
	array offset →	0	1	2	3	4	5	6	7	8
	values →	1	2	3	4	5	6	7	8	9
accessor count	1st pass	A								
	2nd pass		A							
	3rd pass			A						
	etc.									
	9th pass									A

また、以下は3つの対の整数値からなるストリームを表した`<accessor>`要素の例です。2番目の`<param>`要素には name 属性がないため、配列中の2番目の値をすべて読み飛ばしています。

```
<source>
  <int_array name="values" count="9">
    1 0 1 2 0 2 3 0 3
  </int_array>
  <technique_common>
    <accessor source="#values" count="3" stride="3">
      <param name="A" type="int"/>
      <param type="int"/>
      <param name="B" type="int"/>
    </accessor>
  </technique_common>
</source>
```

stride = 3										
	array offset →	0	1	2	3	4	5	6	7	
	values →	1	0	1	2	0	2	3	0	
accessor count	1st pass	A		B						
	2nd pass				A		B			
	3rd pass							A		

さらに以下の例では、stride 属性が3であっても、3番目の値と出力とをバインドする<param>要素がないため、3番目の値を読み飛ばしています。

```
<source>
  <int_array name="values" count="9">
    1 1 0 2 2 0 3 3 0
  </int_array>
  <technique_common>
    <accessor source="#values" count="3" stride="3">
      <param name="B" type="int"/>
      <param name="A" type="int"/>
    </accessor>
  </technique_common>
</source>
```

stride = 3										
	array offset →	0	1	2	3	4	5	6	7	
	values →	1	1	0	2	2	0	3	3	
accessor count	1st pass	B	A							
	2nd pass				B	A				
	3rd pass							B	A	

この例では、stride 属性はやはり3ですが、3番目の値と出力とをバインドする<param>要素がないため、3番目の値は読み飛ばされます。また、この例では、最初の3つの値 (offset=3なので4番目の値から始まる)、および最後の3つの値 (count が4なので、値は12個しか読み込まれない)は無視されます。

```
<source>
  <int_array name="values" count="18">
    1 1 0 2 2 0 3 3 0 4 4 0 5 5 0 6 6 0
  </int_array>
  <technique_common>
    <accessor source="#values" offset="3" count="4" stride="3">
      <param name="A" type="int"/>
      <param name="C" type="int"/>
    </accessor>
  </technique_common>
</source>
```

stride = 3																			
	array offset →	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	values →	1	1	0	2	2	0	3	3	0	4	4	0	5	5	0	6	6	0
accessor count	1st pass				A	C													
	2nd pass							A	C										
	3rd pass										A	C							
	4th pass													A	C				

<input>のセマンティックスは、ソース中の特定のデータ順序 (X、Y、ZやR、G、Bなど)を意味します。<source>の<accessor>中の<param>の実際の名前は、重要ではありません。<param>の名前はいかなるバインディングをも意味しませんが、名前や<param>全体 (リストの最後にある場合)が存在しないことは、データが読み飛ばされることを意味します。

<accessor>からソースを正しく読みとるために、プログラムは、特定のセマンティックが期待するデータを考慮し、それをストライド、オフセット、および名前がヌルでないparamの数と比較して、読み取るべき値の数を判断する必要があります。そして、読み取りの際には、名前のない<param>に対応するデータを読み飛ばす必要があります。

プログラムの中に、頂点マップのような配列があって、その中にジオメトリを書き込もうとしています。

```
struct
{
    float x_pos, y_pos, z_pos, x_norm, y_norm, z_norm, tex1_U,
    tex1_V, tex2_U tex2_V;
} my_array[1000];
```

このようなソースが与えられたとします。

```
<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>
```

その入力は以下の通りだとします。

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

`my_array` へのデータ読み込みをシーケンシャルに行う場合、アクセッサのストライドは3で、`<param>` のすべてに名前があるので、`<source>`は三次元の位置を含むと仮定され、`my_array` への読み込みは以下のように行われます。

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							

アクセッサを以下のように変更します。

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

2番目の`<param>`には名前がないので、読み飛ばされます。名前のある`<param>`は2つだけなので、`<source>`は二次元の位置を含むと仮定され、読み込みは以下のように行われます。

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								

ここで、頂点配列全体を単一の浮動小数点配列にパックしたい場合。

```
<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 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>

```


各<accessor>中の<param>の数を基にすれば、読み取ったのは三次元の位置、三次元の法線、および二次元のテクスチャ座標であると推定できるはずです。

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

また、<accessor>の offset 属性を使うと、データの前にあるフィールドを読み飛ばせることに注意してください。たとえば、前述の例の、source id=texture1 のソースの中にある<accessor>は、以下のように書いても同じように動作します。

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

ambient

(core)

カテゴリ: ライティング

概要

環境光源を記述します。

注:<ambient>には2種類あります。8章「FXリファレンス」の「[fx_common_color_or_texture_type](#)」も参照してください。

コンセプト

<ambient> 要素は、環境光源を表すために必要なパラメータを宣言します。環境光源は、位置や方向に関係なく、すべてに対してに照明が当たる光のことです。

属性

<ambient>要素には属性はありません。

関連要素

<ambient>要素は、以下の要素と関連性があります。

親要素	light / technique_common
子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<color>	光源のカラーを表す3つの浮動小数点数を含みます。主見出し項目を参照してください。	なし	1

詳細

例

以下は、`<ambient>`要素の例です。

```

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

```

animation

カテゴリ: **アニメーション**

概要

アニメーション情報の宣言をカテゴリ化します。

コンセプト

アニメーションの階層構造には、アニメーションのキーフレームデータとサンプリング関数を記述する要素が含まれます。これらの要素は、同時に実行すべきアニメーションがグループ化されるように並べられています。

アニメーションは、時間の経過に伴うオブジェクトまたは値の変化を記述します。通常、アニメーションは、動いている錯覚を与えるために利用されます。一般的なアニメーションの技法としては、キーフレームアニメーションがあります。

キーフレームはデータを2次元(2D)サンプリングしたものです。最初の次元は入力と呼ばれ、通常は時間が使用されますが、他の任意の値でもかまいません。2番目の次元は出力と呼ばれ、アニメートされている最中の値を表します。キーフレームと補間アルゴリズムを組み合わせることで、キーフレーム間の時間に対する中間の値が計算され、キーフレーム間の区間全体の出力値のセットが生成されます。キーフレームのセットとその間の補間は、「アニメーション曲線」もしくは「関数曲線」と呼ばれる2次元の関数を定義します。この曲線が、`<animation>`要素によって表現されます。

アニメーション曲線の補間については、4章「プログラミングガイド」の「曲線補間」を参照してください。

属性

`<animation>`要素には、以下の属性があります。

id	xs:ID	<code><animation></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<animation>`要素は、以下の要素と関連性があります。

親要素	<code>library_animations</code> , <code>animation</code>
-----	--

子要素	下記サブセクションを参照してください。
その他	instance_animation

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>	主見出し項目を参照してください。	なし	0 または 1
<animation>	関連するアニメーションの階層構造を構築することができます。主見出し項目を参照してください。	なし	0 以上 (「詳細」を参照)
<source> (core)	主見出し項目を参照してください。	なし	0 以上 (「詳細」を参照)
<sampler>	アニメーションの補間サンプリング関数を記述します。主見出し項目を参照してください。	なし	0 以上 (「詳細」を参照)
<channel>	アニメーションの出力チャンネルを記述します。主見出し項目を参照してください。	なし	0 以上 (「詳細」を参照)
<extra>	主見出し項目を参照してください。	なし	0 以上

詳細

[<animation>](#)要素には、アニメーションツリーを形成するためのアニメーションデータを表す要素が含まれます。データの実際の型および複雑さは、子要素によって詳細に表されます。

子要素は以下のルールに従います。

- [<animation>](#)要素は、次のいずれかを 1 つ以上含んでいなければなりません。
 - [<animation>](#)
 - [<sampler>](#)および[<channel>](#)
- [<sampler>](#)と[<channel>](#)は、常に一緒に使用されなければなりません。

[<animation_clip>](#)要素から参照されていない[<animation>](#)は、再生時にシーンに適用することができます。それ以外の再生の詳細は、[<animation_clip>](#)を参照してください。

アニメーションターゲットの解決に関する情報は、[<animation_clip>](#)の「詳細」を参照してください。

例

以下は、指定可能な属性を持った空の[<animation>](#)要素の例です。

```
<library_animations>
  <animation name="walk" id="Walk123">
    <!-- 適切な id、source、target 値を使う -->
    <source id="..." />
    <source id="..." />
    <sampler> ... </sampler>
    <channel source="..." target="..." />
  </animation>
</library_animations>
```

以下は、「ジャンプ」するアニメーションを定義した簡単なアニメーションツリーの例です。

```
<library_animations>
  <animation name="jump" id="jump">
    <animation id="skeleton_root_translate">
      <source/><source/><sampler/><channel/>
    </animation>
  </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>

```

以下は、一部のアニメーションを未定義のままにした、より複雑なアニメーションツリーの例です。

```

<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>

```

animation_clip

カテゴリ: アニメーション

概要

アニメーションクリップとしてまとめて利用する、アニメーション曲線や式のセットのセクションを定義します。

コンセプト

アニメーションクリップは、アニメーション曲線や式（もしくはその両方）の単一のセットを、複数の部分に切り分ける目的で利用します。たとえば、歩いていたキャラクターが、途中から走り出すようなアニメーションがあったとします。この歩いているアニメーションと走っているアニメーションは、2つの異なるクリップに切り分けることができます。また、アニメーションクリップは、同じシーン中にいる別々のキャラクターのアニメーションや式を切り分けたり、同じキャラクターの別々の部分（上半身と下半身など）を切り分けるのにも利用できます。

現時点では、アニメーションクリップを COLLADA ドキュメント中でインスタンス化することはできません。エンジンや他のツールで利用するためのものです。

属性

`<animation_clip>`要素には、以下の属性があります。

id	xs:ID	<code><animation></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
start	xs:double	クリップの開始時間を表します(単位は秒)。この時間は、キーフレームデータで利用されているものと同じで、どのキーフレームセットをクリップに含めるのかを判断する目的で利用されます。クリップがいつ再生されるのかを開始時間で指定するわけではありません。指定した時間が、参照されているアニメーションの2つのキーフレーム間に位置する場合には、補間された値が利用されます。デフォルト値は0.0です。オプション。
end	xs:double	クリップの終了時間を表します(単位は秒)。開始時間と同じように利用されます。終了時間を指定しなかった場合、最も長いアニメーションの終了時間が値として利用されます。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<animation_clip>`要素は、以下の要素と関連性があります。

親要素	<code>library_animation_clips</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0または1
<code><instance_animation></code>	主見出し項目を参照してください。	なし	1以上
<code><instance_formula></code>	主見出し項目を参照してください。	なし	1以上
<code><extra></code>	主見出し項目を参照してください。	なし	0以上

詳細

アニメーション、ターゲット、およびシーン

複数の`<animation_clip>`は、同じターゲットをもつ`<animation>`や`<formula>`を参照することができます。また、`<animation_clip>`には参照されていないが、再生時に適用・使用される`<animation>`や`<formula>`も、同じターゲットを持つことが可能です。

`<animation_clip>`の中で参照されかつ使用されている`<animation>`や`<formula>`は、再生時にシーンに適用しないようにしてください。(再生のために)シーンに適用するのは、参照されていない`<animation>`や`<formula>`だけにしてください。

注:プラグイン作成者は、`<animation_clip>`を完全にはサポートしていなくても、この方針に従う必要があります。たとえば、DCC ツールは、`<library_animations>`および`<library_animation_clips>`のコンテンツを、バンクやパレットに格納することができます。参照されていない`<animation>`や`<formula>`は、すべて、アプリケーションランタイムによって処理されます。ロード・再生されるのはこのコンテンツです。

例

以下は、指定可能な属性を持った2つの<animation_clip>要素の例です。

```
<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>
```

asset

カテゴリ: **メタデータ**

概要

その親要素に関連したアセット管理情報を定義します。

コンセプト

コンピュータは、膨大な量の情報を保存しています。アセットは、独立した集合に組織され、まとめて管理される情報セットです。さまざまな属性を用いてアセットを表すことで、この情報を、ソフトウェアツールおよび人間の両方によって管理および理解できるようにします。アセット情報は階層的であることが多く、その場合大きなアセットの各パーツは、それぞれ別のアセットとして管理される、より小さい部分に区分けされます。

属性

<asset>要素には属性はありません。

関連要素

<asset>要素は、以下の要素と関連性があります。

親要素	コア要素: animation , animation_clip , camera , COLLADA , controller , evaluate_scene , extra , geometry , light , node , source , visual_scene FX要素: material , image , effect , profile_* (「プロファイル」を参照), technique (FX)(profile_CG , profile_COMMON , profile_GLES 内) フィジックス要素: force_field , physics_material , physics_scene , physics_model キネマティクス要素: kinematics_scene 全セクション: library_*
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><contributor></code>	その親要素の貢献者に関連したデータを提供します。主見出し項目を参照してください。	なし	0 以上
<code><coverage></code> <code><geographic_location>...</code> <code></geographic_location></code> <code></coverage></code>	物理空間中のビジュアルシーンの場所に関する情報を提供します。この要素には属性はありませんが、子要素として 0 個または 1 個の <code><geographic_location></code> を含むことができます。詳しくは、 <code><geographic_location></code> の主見出し項目を参照してください。	なし	0 または 1
<code><created></code>	親要素が作成された日付と時刻が含まれます。この日時は、XML スキーマの <code>xs:dateTime</code> プリミティブ型として ISO 8601 形式で表現されます。この要素には属性はありません。	なし	1
<code><keywords></code>	親要素の検索条件として使われる単語のリストが含まれます。この要素には属性はありません。	なし	0 または 1
<code><modified></code>	親要素が最後に変更された日付と時刻が含まれます。この日時は、XML スキーマの <code>xs:dateTime</code> プリミティブ型として ISO 8601 形式で表現されます。この要素には属性はありません。	なし	1
<code><revision></code>	親要素のリビジョン情報が含まれます。この要素には属性はありません。	なし	0 または 1
<code><subject></code>	親要素の主題に関する説明が含まれます。この要素には属性はありません。	なし	0 または 1
<code><title></code>	親要素のタイトル情報が含まれます。この要素には属性はありません。	なし	0 または 1
<code><unit></code> <code> meter=...</code> <code> name=...</code> <code></unit></code>	COLLADA 要素やオブジェクト用の距離の単位を定義します。この距離の単位は、よりローカルな <code><asset></code> / <code><unit></code> によってオーバーライドされない限り、 <code><asset></code> の親要素のスコープ内にあるあらゆる空間的尺度に適用されます。この単位の値は自己記述的であり、実世界の測定値と整合性がとれている必要はありません。そのオプションの属性は、 <code>name</code> : 距離単位の名前 (<code>xs:NMTOKEN</code>)。例: 「meter」、「centimeter」、「inches」、「parsec」。これは、実際の測定単位名でも、仮想の単位名でもかまいません。 <code>meter</code> : 1 単位の距離が実際に何メートルになるかを浮動小数点数で表したもの。たとえば、 <code>name</code> が「meter」なら 1.0、 <code>name</code> が「kilometer」なら 1000、 <code>name</code> が「foot」なら 0.3048 になります。詳しくは、6 章「フィジックスリファレンス」の「物理的な単位について」を参照してください。 <code>name</code> : meter	<code>name</code> : <code>meter</code> <code>meter</code> : 1.0	0 または 1

名前/例	解説	デフォルト値	出現回数
<code><up_axis></code>	ジオメトリデータの座標系に関する情報が含まれます。座標はすべて右手系です。有効な値として <code>X_UP</code> 、 <code>Y_UP</code> 、 <code>Z_UP</code> のいずれかが指定できます。この要素は、それぞれの軸を上、どの軸を右、どの軸を内側とみなすのかを指定します。詳しくは、「詳細」を参照してください。この要素には属性はありません。	<code>Y_UP</code>	0 または 1
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

階層的な`<asset>`要素の場合は、親アセットも子アセットも共に、`<unit>`など、同じメタデータに対して値を供給しますが、子アセットの値は子要素のスコープ内では親の値に優先します。これは再帰的に適用されます。

上軸の値

`<up_axis>`要素の値には以下の意味があります。

値	右軸	上軸	内側軸
X-UP	負の y	正の x	正の z
Y_UP	正の x	正の y	正の z
Z_UP	正の x	正の z	負の y

例

以下は、親の`<COLLADA>`要素を表した`<asset>`要素の例です。つまり、ドキュメント全体を表しています。

```
<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>
```

bool_array

カテゴリ: データフロー

概要

Boolean 値の同種の配列の保存場所を宣言します。

コンセプト

`<bool_array>`要素は、COLLADA スキーマで汎用的に利用されるデータ値を保存します。配列そのものは強く型付けされますが、セマンティックスを伴いません。単純に、XML での Boolean 値の並びを記述します。

属性

`<bool_array>`要素には、以下の属性があります。

count	uint_type	配列中の値の数。必須。
id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<bool_array>`要素は、以下の要素と関連性があります。

親要素	source (core)
子要素	なし
その他	accessor

詳細

`<bool_array>`要素には、XML Boolean 値のリストが含まれます。これらの値は、`<source>`要素のデータのレポジトリとして利用されます。

例

以下は、4つの Boolean 値の並びを表した`<bool_array>`要素の例です。

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

camera

カテゴリ: カメラ

概要

ビジュアルシーン階層またはシーングラフへのビューを宣言します。`<camera>`要素には、カメラの光学的な特性や撮影装置を表す要素が含まれます。

コンセプト

カメラは、ビジュアルシーンを見ているビューアの視点を表します。カメラは、シーンの視覚的なイメージを捕らえる装置です。シーン中のカメラには位置と方向があります。これが、カメラの光学系装置やレンズから見たビューポイントとなります。

カメラの光学系は、入射光をまとめて画像を作ります。画像の焦点は、カメラの撮影装置 (フィルム) の面に合わせられます。結果として生じる画像を、撮影装置が記録します。

属性

`<camera>`要素には、以下の属性があります。

id	xs:ID	<code><camera></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

<camera>要素は、以下の要素と関連性があります。

親要素	library_cameras
子要素	下記サブセクションを参照してください。
その他	instance_camera

子要素

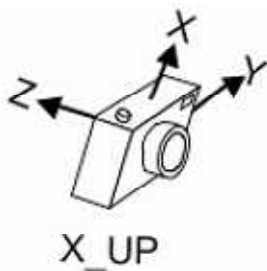
子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<asset>	カメラのビューのための、軸方向と測定単位を定義します。また、この要素の作成に関する情報も含まれています。主見出し項目を参照してください。	なし	0または1
<optics>	標準パラメータを利用して視野角 (FOV) と視錐台を記述します。主見出し項目を参照してください。	なし	1
<imager>	フィルムや CCD といったカメラの画像センサーを表します。主見出し項目を参照してください。	なし	0または1
<extra>	主見出し項目を参照してください。	なし	0以上

詳細

単純なカメラの場合、汎用的なテクニックでは、<optics>要素だけが含まれていれば問題ありません。

カメラは、ローカルな X 軸の正方向が右に、レンズがローカルな Z 軸の負方向に、カメラの上方向がローカルな Y 軸の正方向を向くように定義されます (<Lookat>要素を参照)。この方向は、<asset>要素の<up_axis>値に影響を受けます。



例

以下は、視野角 45 度でのシーンの透視図を記述した<camera>要素の例です。

```
<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>
```

channel

カテゴリ: アニメーション

概要

アニメーションの出力チャンネルを宣言します。

コンセプト

時間の経過に伴うアニメーションのサンプリング値の変化は、チャンネルに出力されてゆきます。チャンネルは、アニメーションエンジンから受け取ったこれらの値を保存する場所を表します。チャンネルの対象は、アニメーションされた値を受け取るデータ構造となります。

属性

<channel>要素には、以下の属性があります。

source	urifragment_type	URL 表現を使って表したサンプリングの場所。必須。
target	sidref_type	サンプリングの出力にバインドされた要素の SID の参照。このテキスト文字列は、3 章「スキーマのコンセプト」の「アドレス構文」のセクションに記述されている単純な形式にしたがったパス名です。必須。

関連要素

<channel>要素は、以下の要素と関連性があります。

親要素	animation
子要素	なし
その他	なし

詳細

この要素には、データは含まれていません。

例

以下は、「Box」という id を持つ要素の値の変化を対象とした<channel>要素の例です。

```
<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>
```

COLLADA

カテゴリ: **メタデータ**

概要

COLLADA スキーマに準拠したいくつかのコンテンツを含むドキュメントのルート宣言します。

コンセプト

COLLADA スキーマは XML ベースなので、整形形式の XML 文書として認識されるために、文書ルート要素または文書実体をちょうど 1 つだけ持つ必要があります。COLLADA 要素は、そのルート要素として動作します。

属性

<COLLADA>要素には、以下の属性があります。

version	列挙	インスタンス文書が準拠している COLLADA スキーマのリビジョンです。有効な値は、1.5.0 だけです。必須。
xmlns	xs:anyURI	インスタンス文書のスキーマを特定するために、XML Schema の名前空間属性を要素に適用します。
base	xs:anyURI	XML Base 仕様は HTML BASE の場合と同様、XML ドキュメントのパーツのベース URI を定義するファシリティを記述します。1 つの属性 <code>xml:base</code> を定義し、相対的な URI 参照を処理する際にその属性を使用するための手順を詳細に記述しています。詳しくは、 http://www.w3.org/XML/1998/namespace を参照してください。

関連要素

<COLLADA>要素は、以下の要素と関連性があります。

親要素	親要素はありません
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<asset>	主見出し項目を参照してください。	なし	1
<i>library_element</i>	以下のような任意のライブラリ要素を、好きなように組合せて好きな数だけ配置することができます。 <library_animation_clips> <library_animations> <library_articulated_systems>(キネマティックス) <library_cameras> <library_controllers> <library_effects>(FX) <library_force_fields>(フィジックス) <library_formulas>	なし	0 以上

名前/例	解説	デフォルト値	出現回数
	<library_geometries> <library_images> (FX) <library_joints> (キネマティックス) <library_kinematics_models> (キネマティックス) <library_kinematics_scenes> (キネマティックス) <library_lights> <library_materials> (FX) <library_nodes> <library_physics_materials> (フィジックス) <library_physics_models> (フィジックス) <library_physics_scenes> (フィジックス) <library_visual_scenes> 主見出し項目を参照してください。		
<scene>	主見出し項目を参照してください。	なし	0 または 1
<extra>	主見出し項目を参照してください。	なし	0 以上

詳細

<COLLADA>要素は、COLLADA インスタンス文書中の文書実体（ルート要素）です。

例

以下は、スキーマのバージョンが「1.5.0」の空の COLLADA インスタンス文書の例です。

```
<?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>
```

color

カテゴリ: ライティング

概要

親光源要素のカラーを表します。

コンセプト

<light>のコンテキストでは、<color>要素にはその親光源要素の RGB カラーを表す 3 つの浮動小数点値が含まれます。

<profile_COMMON>のコンテキストでは、親要素の RGBA カラーを表す 4 つの浮動小数点値が含まれます。

属性

<color>要素には、以下の属性があります。

sid	sid_type	オプション。親要素が <profile_COMMON> の場合のみ。この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
-----	----------	---

関連要素

<color>要素は、以下の要素と関連性があります。

親要素	In <light>: ambient(core)、directional、point、spot In <profile_COMMON>: fx_common_color_or_texture_type 型の要素(ambient、emission、diffuse、reflective、specular、transparent)
子要素	なし
その他	なし

詳細

例

contributor

カテゴリ: **メタデータ**

概要

アセット管理用の作成者情報を定義します。

コンセプト

現在の制作ラインでは、特にアートチームのサイズが着実に大きくなるのに伴って、1つのアセットに複数の人間、複数のツールが関わるが増えています。この要素の情報は、アセット管理システムにとって重要になることがあります。そのコンテンツのフォーマットはアプリケーション定義です。

属性

<contributor>要素には属性はありません。

関連要素

<contributor>要素は、以下の要素と関連性があります。

親要素	asset
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><author></code>	作成者名の文字列が含まれます。この要素には属性はありません。	なし	0 または 1
<code><author_email></code>	RFC 2822 セクション 3.4 に準拠する作成者の完全な電子メールアドレスの文字列が含まれます。この要素には属性はありません。	なし	0 または 1
<code><author_website></code>	この協力者のウェブサイトの URL を表す <code>xs:anyURI</code> が含まれます。この要素には属性はありません。	なし	0 または 1
<code><authoring_tool></code>	オーサリングツールの名前を表す文字列が含まれます。この要素には属性はありません。	なし	0 または 1
<code><comments></code>	この協力者からのコメントを表す文字列が含まれます。この要素には属性はありません。	なし	0 または 1
<code><copyright></code>	著作権情報を表す文字列が含まれます。この要素には属性はありません。	なし	0 または 1
<code><source_data></code>	このアセットで使われるソースデータへの URI 参照 (<code>xs:anyURI</code>) が含まれます。この要素には属性はありません。	なし	0 または 1

詳細

例

以下は、アセットのための `<contributor>` 要素の例です。

```
<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>
```

controller

カテゴリ: コントローラ

概要

動的コンテンツ用の汎用コントロールの情報を定義します。

コンセプト

コントローラとは、別のオブジェクトの操作を制御管理するためのデバイスや仕組みです。`<controller>` 要素は、アクティブ、動的コンテンツを記述するための一般的な汎用メカニズムです。この要素には、

データの操作を記述する要素が含まれます。データの実際の型および複雑さは、子要素によって詳細に表されます。

COLLADA には、ビジュアルシーン中のアクティブなメッシュジオメトリ用として、頂点スキニングおよびメッシュモーフィングの 2 種類のコントローラが記述されています。けれども、コントローラ概念はジオメトリや可視化に限定されているわけではなく、COLLADA 仕様の将来バージョンでは、アニメーションブレンド、物理的シミュレーション、ダイナミクス、ユーザとの対話処理などを記述する他の種類のコントローラが導入される可能性があります。

属性

<controller>要素には、以下の属性があります。

id	xs:ID	<controller> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

<controller>要素は、以下の要素と関連性があります。

親要素	library_controllers
子要素	下記サブセクションを参照してください。
その他	instance_controller

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>	主見出し項目を参照してください。	なし	0 または 1
<i>control_element</i>	コントロールデータを含む要素。つぎのいずれかである必要があります。 <skin> <morph> 主見出し項目を参照してください。	なし	1
<extra>	主見出し項目を参照してください。	なし	0 以上

詳細

<controller>要素を、<node>要素の範囲内でインスタンス化して利用する方法は、<geometry>要素と同様です。

<morph>要素は、メッシュを変形してブレンドするメッシュモーフィングコントローラに情報を提供します。

<skin>要素は、重みつきインフルエンスに基づいて頂点を変換し、滑らかに変化するメッシュを生成する、頂点スキニングコントローラに情報を提供します。

複数のコントローラ間のインタラクション

コントローラは、複数個を同時に適用することができます。そのために、コントローラは別のコントローラをソースとして使うことができます。コントローラを適用する際に、現在適用されているコントローラのソースが別のコントローラである場合、まず、その別のコントローラ（ソース）を先に適用する必要があります。つまり、コントローラの実行は、コントローラ以外のオブジェクトをソースとして持つ

コントローラ（通常はジオメトリ）から始まって、そのコントローラの実行パイプラインはそこから継続されます。

例

以下は、指定可能な属性を持った空の<controller>要素の例です。

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

以下は、まず<morph>コントローラが適用され、その後で<skin>が適用される典型的な例です。

```
<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>
```

control_vertices

カテゴリ: ジオメトリ

概要

スプラインの制御頂点 (CV) を記述します。

コンセプト

制御頂点には、制御頂点、およびそれに関するセグメントの両方に関する情報が格納されます。セグメントデータは、指定された制御頂点から始まるスプラインセグメントに適用されます。

各制御頂点は、位置を持つ必要があります。制御頂点では、その頂点から始まるセグメントで使用する補間法を指定することを強くお勧めします。指定がない場合には、線形補間であると仮定されます。ベジェおよびエルミート補間では、各制御頂点に対して入出力接線を指定する必要があります。

また、各制御頂点は、カスタムデータソースを使って、任意の量のユーザ特定情報を持つことができます。カスタムデータソースは、各制御頂点が1つの値を持てるだけのデータを提供する必要があります。

属性

<control_vertices>要素には属性はありません。

関連要素

<control_vertices>要素は、以下の要素と関連性があります。

親要素	spline , nurbs , nurbs_surface
子要素	下記サブセクションを参照してください。
その他	source

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (unshared)	<code><input></code> (unshared)要素の中には、 <code>semantic</code> 属性が <code>POSITION</code> である要素が、最低でも1つは存在する必要があります。主見出し項目を参照してください。	なし	1以上
<code><extra></code>	主見出し項目を参照してください。	なし	0以上

詳細

COLLADA の`<control_vertices>`では、多項式補間の種類として、`LINEAR`、`BEZIER`、`CARDINAL`、`HERMITE`、`BSPLINE` を認識します。これらの記号名は、`<source>`要素の中の`<Name_array>`で使われます。この値は、`semantic` 属性の値が `POSITION` である`<input>`要素によって、制御頂点に渡されます。

詳しくは、4章「プログラミングガイド」の「曲線補間」を参照してください。

共通プロファイルでは、`<control_vertices>`用の`<input>`のセマンティックスとして、以下の値を定義しています。

<code><input></code> の <code>semantic</code> 値	種類	解説	デフォルト値
<code>POSITION</code>	任意の多次元浮動小数点数	制御頂点の位置。	なし
<code>INTERPOLATION</code>	<code>xs:Name</code>	この制御頂点から始まるセグメントを表す多項式補間の種類。共通プロファイルで定義されている種類は、 <code>LINEAR</code> 、 <code>BEZIER</code> 、 <code>HERMITE</code> 、 <code>CARDINAL</code> 、 <code>BSPLINE</code> です。	<code>LINEAR</code>
<code>IN_TANGENT</code>	任意の多次元浮動小数点数	制御頂点 (<code>BEZIER</code> 、 <code>HERMITE</code>) の前のセグメントの形状を制御する接線。このソースの値の次元数は、 <code>POSITION</code> のソースの次元数に一致する必要があります。	なし
<code>OUT_TANGENT</code>	任意の多次元浮動小数点数	制御頂点 (<code>BEZIER</code> 、 <code>HERMITE</code>) の後のセグメントの形状を制御する接線。このソースの値の次元数は、 <code>POSITION</code> のソースの次元数に一致する必要があります。	なし
<code>CONTINUITY</code>	<code>xs:Name</code>	(オプション) 制御点における連続性の制約を定義します。共通プロファイルで定義された種類としては、 <code>C0</code> 、 <code>C1</code> 、 <code>G1</code> があります。	なし
<code>LINEAR_STEPS</code>	<code>int_type</code>	(オプション) この制御頂点の後に続くスプラインセグメントで使われる区分的線形近似ステップの数。	なし

詳細

- 混合補間スプラインにおいて、補間の種類が `BEZIER` もしくは `HERMITE` であるセグメントが最低でも1つある場合、各制御頂点について、`IN_TANGENT` 値と `OUT_TANGENT` 値を1つずつ指定する必要があります。
- 子要素のデータ型は、すべて同一である必要があります。たとえば、補間の種類が `BEZIER` や `HERMITE` の場合、
- 有効： `POSITION`、`IN_TANGENT`、`OUT_TANGENT` のすべてを `float2_type` として定義。
- 無効： `POSITION` が `float2_type` で、`IN_TANGENT` や `OUT_TANGENT` が `float3_type`。

- 子要素の間にも制約があります。たとえば、POSITION 子要素の数は、INTERPOLATION 要素の数に等しい必要があります。詳細は<spline>を参照してください。

例

<spline>を参照してください。

directional

カテゴリ: ライティング

概要

並行光源を表します。

コンセプト

<directional>要素は、並行光源を表すために必要なパラメータを宣言します。並行光源とは、位置に関係なく、同じ方向からすべてを照らす光源のことです。

ローカル座標での光源のデフォルトの方向ベクトルは[0,0,-1]で、負のZ軸方向に向いています。実際の放射方向は、光源がインスタンス化されるノードの座標変換として定義されます。

属性

<directional>要素には属性はありません。

関連要素

<directional>要素は、以下の要素と関連性があります。

親要素	light / technique_common
子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<color>	光源のカラーを表す3つの浮動小数点数を含みます。主見出し項目を参照してください。	なし	1

詳細

例

以下は、<directional>要素の例です。

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

evaluate_scene

カテゴリ: シーン

概要

`<visual_scene>`の評価方法を指定する情報を宣言します。

コンセプト

属性

`<evaluate_scene>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。
sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、アドレス構文 3 章の「スキーマのコンセプト」を参照してください。
enable	xs:boolean	評価が有効かどうか。評価を無効にすると、デバッグの際に便利ことがあります。デフォルト値は true です。オプション。

関連要素

`<evaluate_scene>`要素は、以下の要素と関連性があります。

親要素	<code>visual_scene</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	アセット管理情報用。主見出し項目を参照してください。	なし	0 または 1
<code><render></code>	シーンをレンダリングするためのエフェクトパスを記述します。FX の主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

例

extra

カテゴリ: **拡張性**

概要

親要素に関する任意の追加情報を提供します。

コンセプト

スキーマに拡張性を持たせるには、ユーザが任意の情報を指定できる方法が必要となります。このような追加情報によって、アプリケーションが使用する実データやセマンティックス（メタ）データを表現することが可能です。

COLLADA では、このような追加情報を、任意の XML 要素やデータを含んだテクニック（technique）として表現します。

属性

<extra>要素には、以下の属性があります。

id	xs:ID	<extra> 要素のユニークな識別子を含んだテキスト文字列。この値は、そのインスタンス文書中で一意である必要があります。オプション。
name	xs:token	この要素の文字列の名。オプション。
type	xs:NMTOKEN	特定の <extra> 要素が表す情報の種類に関するヒント。このテキスト文字列は、アプリケーションの理解可能な形式でなければなりません。オプション。

関連要素

<extra>要素は、以下の要素と関連性があります。

親要素	<p>コア要素: <code>animation</code>, <code>animation_clip</code>, <code>camera</code>, <code>COLLADA</code>, <code>controller</code>, <code>control_vertices</code>, <code>evaluate_scene</code>, <code>geometry</code>, <code>imager</code>, <code>joints</code>, <code>light</code>, <code>lines</code>, <code>linestrips</code>, <code>mesh</code>, <code>morph</code>, <code>node</code>, <code>optics</code>, <code>polygons</code>, <code>polylist</code>, <code>scene</code>, <code>skin</code>, <code>spline</code>, <code>targets</code>, <code>triangles</code>, <code>trifans</code>, <code>tristrips</code>, <code>vertex_weights</code>, <code>vertices</code>, <code>visual_scene</code></p> <p>フィジックス要素: <code>Attachment</code>, <code>box</code>, <code>capsule</code>, <code>convex_mesh</code>, <code>cylinder</code>, <code>force_field</code>, <code>physics_material</code>, <code>physics_model</code>, <code>physics_scene</code>, <code>plane</code>, <code>ref_attachment</code>, <code>rigid_body</code>, <code>rigid_constraint</code>, <code>shape</code>, <code>sphere</code></p> <p>FX 要素: <code>bind_material</code>, <code>effect</code>, <code>image</code>, <code>material</code>, <code>pass</code>, <code>profile_BRIDGE</code>, <code>profile_CG</code>, <code>profile_COMMON</code>, <code>profile_GLES</code>, <code>profile_GLES2</code>, <code>profile_GLSL</code>, <code>render</code>, <code>sampler_state</code>, <code>sampler_*</code> (「テクスチャリング」を参照), <code>shader</code>, <code>technique</code> (FX) (<code>profile_*</code>内), <code>texture</code>, <code>texture_pipeline</code></p> <p>B-Rep 要素: <code>brep</code>, <code>circle</code>, <code>cone</code>, <code>curves</code>, <code>cylinder</code> (B-Rep), <code>edges</code>, <code>ellipse</code>, <code>faces</code>, <code>hyperbola</code>, <code>line</code>, <code>nurbs</code>, <code>nurbs_surface</code>, <code>parabola</code>, <code>pcurves</code>, <code>shells</code>, <code>solids</code>, <code>surfaces</code>, <code>surface_curves</code>, <code>swept_surface</code>, <code>torus</code>, <code>wires</code></p> <p>キネマティックス要素: <code>articulated_system</code>, <code>joint</code>, <code>kinematics</code>, <code>kinematics_model</code>, <code>kinematics_scene</code>, <code>motion</code></p> <p>全セクション: <code>instance_*</code>, <code>library_*</code></p>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code><technique></code> (core)	主見出し項目を参照してください。	なし	1 以上

詳細

例

以下は、構造化された追加コンテンツと構造化されていないものの両方を含む`<extra>`要素の例です。

```
<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>
```

以下の例は、`<extra>`と`<technique>`を連携させる方法を示しています。

```
<light>
  <!-- アプリケーションは、以下の 3 テクニックの 1 つを選択します -->
  <technique_common> ... </technique_common>
  <technique profile="ProductA"> ... </technique>
  <technique profile="ProductB"> ... </technique>
  <!-- アプリケーションは、以下の 2 つの 0 個以上の extras、さらに各 extras ごとに -->
  <!-- 1 個の technique を選択します -->
  <extra type="basic">
    <technique profile="ProductA"> ... </technique>
    <technique profile="ProductB"> ... </technique>
  </extra>
  <extra type="bonus">
    <technique profile="ProductB"> ... </technique>
  </extra>
```

2 つの選択の例:

- `technique_common`、extra 「basic」 (製品 B)、extra 「bonus」 (製品 B)
- `technique` (製品 B)、extra 「basic」 (製品 B)、extra 「bonus」 (製品 B)

float_array

カテゴリ: データフロー

概要

浮動小数点値の値を含んだ同種の配列の保存場所を宣言します。

コンセプト

`<float_array>`要素は、COLLADAスキーマで汎用的に利用されるデータ値を格納します。配列そのものは強く型付けされますが、セマンティックスを伴いません。単純に、浮動小数点値の並びを記述します。

属性

`<float_array>`要素には、以下の属性があります。

count	uint_type	配列中の値の数。必須。
id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。
digits	xs:unsignedByte	配列中に含めることが可能な浮動小数点値の有効桁数(10進数)。最小値は1、最大値は17です。デフォルト値は6です。オプション。
magnitude	xs:short	配列中に含めることが可能な浮動小数点値の最大指数。最大値は308、最小値は-324です。デフォルト値は38です。オプション。

関連要素

`<float_array>`要素は、以下の要素と関連性があります。

親要素	source(core)
子要素	なし
その他	accessor

詳細

`<float_array>`要素には、浮動小数点値の値のリストが含まれます。これらの値は、`<source>`要素のデータのレポジトリとして利用されます。

例

以下は、9つの浮動小数点値の並びを表した`<float_array>`要素の例です。

```
<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>
```

formula

カテゴリ: 数学

概要

式を定義します。

コンセプト

式を記述するにはさまざまな方法があります。COLLADA は、一般的なテクニックとして MathML を使います。

属性

<formula>要素には、以下の属性があります。

id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。
sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この文字列は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、アドレス構文 3 章の「スキーマのコンセプト」を参照してください。

関連要素

<formula>要素は、以下の要素と関連性があります。

親要素	library_formulas , animation_clip , kinematics_model/technique_common , kinematics/axis_info
子要素	下記サブセクションを参照してください。
その他	instance_formula , library_formulas

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<newparam>	主見出し項目を参照してください。	なし	0 以上
<target>	式の結果変数を指定する common_float_or_param_type (11 章 型を参照) が含まれます。通常はパラメータです。	なし	1
<technique_common>	すべての COLLADA 実装がサポートしなければならない共通プロファイルの式を指定します。詳しくは、下の子要素詳細のサブセクション、「共通プロファイル」の使い方情報のセクション、および主見出し項目を参照してください。	なし	1
<technique>	各<technique>>は、<technique>の profile 属性として指定され特定のプロファイルのための式を指定します 主見出し項目を参照してください。	なし	0 以上

<formula> / <technique_common>の子要素

MathML は、主に、プレゼンテーションとコンテンツの 2 つの部分に分かれています。COLLADA でサポートしているのは、コンテンツの部分だけです。該当するスキーマが利用できる場合には、<technique_common>子要素には、任意の有効な MathML XML を含むことができます。

詳細

COLLADA Mathematics は、COLLADA の中で関数を定義する機能を提供します。COLLADA Mathematics は MathML に基づいて実装されており、表現とコンテンツの定義を提供します。

MathML が選ばれた理由は、次の通りです。

- MathML が、標準 (W3C) として認められていること。
- MathML では、スキーマが有効であること。

COLLADA の中で MathML を利用するには、MathML スキーマにアクセス可能である必要があります。COLLADA の共通プロファイルでは、MathML のバージョン 2.0 を採用しています。

例

以下は、<formula>要素の例です。

```
<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>
```

geographic_location

カテゴリ: メタデータ

概要

`<asset>`要素の親要素が存在する地理的位置の情報を定義します。

コンセプト

属性

`<geographic_location>`要素には属性はありません。

関連要素

`<geographic_location>`要素は、以下の要素と関連性があります。

親要素	asset/coverage
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><longitude></code>	WGS 84 世界測地系の定義に準拠した、アセットの経度を指定する浮動小数点数が含まれます。有効な値の範囲は-180.0 ~ 180.0 です。	なし	1
<code><latitude></code>	WGS 84 世界測地系の定義に準拠した、アセットの緯度を指定する浮動小数点数が含まれます。有効な値の範囲は-90.0 ~ 90.0 です。	なし	1
<code><altitude mode=" "></code>	アセットの高度を浮動小数点数で指定します。高度は、WGS 84 の高度計算ではなく、Keyhole マークアップ言語 (KML) の規格に従います。したがって、高度は地形もしくは海水面からの相対値になります。必須属性は、以下の通りです。 <ul style="list-style-type: none"> <code>mode</code>: 高度値 (メートル) が、海水面からの距離か、該当する緯度/経度点の地形の高度からの距離かどちらであるかを示します。有効な値は、以下の通りです。 <code>absolute</code> <code>relativeToGround</code> こちらがデフォルトです。 	なし	1

詳細

例

以下は、ビジュアルシーンの地理的位置を指定するアセットを含むビジュアルシーンの例です。

```
<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> <!-- オプション。それ以外の場合には親アセットから継承 -->
    </asset>
    <node id="Light" name="Light">
      <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>
```

geometry

カテゴリ: ジオメトリ

概要

シーン内のオブジェクトの視覚的形狀や外観を記述したものです。

コンセプト

`<geometry>`要素は、ジオメトリ情報の宣言をカテゴリ化します。ジオメトリ(幾何学)とは、点、線、角度、面、立体の大きさや特性、相互関係についての研究を行う数学の一分野です。`<geometry>`要素には、メッシュ、凸メッシュ、スプラインの宣言が含まれます。

ジオメトリは、さまざまな形式で記述されます。基本的に、コンピュータグラフィックスのハードウェアは、カラーや法線などのさまざまな属性を持った頂点の位置情報を受け付けるよう標準化が進んでいます。ジオメトリの記述は、この頂点データを直接的で効率的に表現するための方法です。以下に、一般的なジオメトリ形式の一部を挙げます。

- B スプライン
- ベジエ曲線
- メッシュ
- NURBS
- パッチ

もちろん、これですべてというわけではありません。現時点では、COLLADA はポリゴンメッシュとスプラインのみをサポートしています。

属性

`<geometry>` 要素には、以下の属性があります。

id	xs:ID	<code><geometry></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。

関連要素

`<geometry>` 要素は、以下の要素と関連性があります。

親要素	<code>library_geometries</code>
子要素	下記サブセクションを参照してください。
その他	<code>instance_geometry</code>

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code>geometric_element</code>	ジオメトリデータを記述する要素。かならず以下のいずれかである必要があります。 <code><convex_mesh></code> (6章 フィジックスリファレンスを参照) <code><mesh></code> <code><spline></code> <code><brep></code> (9章 B-Rep リファレンスを参照) 主見出し項目を参照してください。	なし	1
<code><extra></code>	<code><geometry></code> 要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0 以上

詳細

`<geometry>` 要素は、ジオメトリデータを表す要素を含みます。データの実際の型および複雑さは、子要素によって詳細に表されます。

例

以下は、指定可能な属性を持った空の`<geometry>`要素の例です。

```
<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>
```

IDREF_array

カテゴリ: データフロー

概要

ID 参照値を含んだ同種の配列の保存場所を宣言します。

コンセプト

<IDREF_array>要素は、インスタンス文書中の ID を参照する文字列値を保存します。

属性

<IDREF_array>要素には、以下の属性があります。

count	uint_type	配列中の値の数。必須。
id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

<IDREF_array>要素は、以下の要素と関連性があります。

親要素	source(core)
子要素	なし
その他	accessor

詳細

<IDREF_array>要素には、XML の IDREF の値のリストが含まれます。これらの値は、<source>要素のデータのレポジトリとして利用されます。

例

以下は、ドキュメント中の<node>要素を参照する<IDREF_array>要素の例です。

```
<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>
```

imager

カテゴリ: カメラ

概要

フィルムや CCD といったカメラの画像センサーを表します。

コンセプト

カメラが画像を（通常は平面の）センサーに投影する光学系です。

<imager>要素は、このセンサーによる、光のカラーと強度を数値化する方法を定義します。

現実世界では、光の輝度が極めて広いダイナミックレンジを持つことがあります。たとえば、野外のシーンでは、太陽の明るさは、木陰よりも桁違いに明るくなります。光子に含まれる波長の組み合わせも無数にあります。

これに対し、表示装置では、非常に限定されたダイナミックレンジを利用します。通常は、可視範囲内の3つの波長、つまり、赤 (Red)、緑 (Green)、青 (Blue) の3原色しか考慮しません。これらは、一般に3つの8ビット値として表されます。

したがって、画像センサーは、以下の2つの処理を行うようになっています。

- スペクトルサンプリング
- ダイナミックレンジのリマップ

この2つの処理の組み合わせはトーンマッピングと呼ばれ、画像合成（レンダリング）の最後のステップとして実行されます。

レイトレーサなどの高品質のレンダラは、スペクトルの強度を内部的に浮動小数点値として表し、実際のピクセルのカラーを `float3_type`（マルチスペクトルレンダラの場合は浮動小数点数の配列）として格納します。その後、トーンマッピングを実行して、グラフィックスハードウェアやモニタで表示可能な24ビットのRGB画像を作成します。

多くのレンダラでは、ハイダイナミックレンジ（HDR）のオリジナル画像を保存して、後で「再露光」することもできます。

属性

<imager>要素には属性はありません。

関連要素

<imager>要素は、以下の要素と関連性があります。

親要素	<code>camera</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><technique></code> (core)	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	<code><imager></code> 要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0 以上

詳細

`<imager>`要素は、オプションです。この要素は、共通プロファイルでは省略されます（この要素には `<technique_common>`がない）。省略時の解釈は、以下の通りです。

- 輝度の線形マッピング
- 0~1の範囲にクランプ（各成分 8 ビットのフレームバッファでは、0..255 にマッピングされる）
- R、G、B のスペクトルサンプリング

マルチスペクトル・レンダラは、少なくともスペクトルサンプリングを定義するために、`<imager>`要素を指定する必要があります。

例

以下は、CCD センサーを持つ現実的なカメラを記述した `<camera>`要素の例です。

```
<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>
      <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>
```

input

(shared)

カテゴリ: データフロー

概要

データソースの入力セマンティックスを宣言して、そのソースにコンシューマを接続します。

注:<input>には2種類あります。詳しくは、「<input>(unshared)」を参照してください。

コンセプト

<input>要素は、入力からコンシューマが必要とするデータソースへの接続を宣言します。データソースは、データを文書の中で再利用するための、意味情報のない生データのコンテナです。このデータを利用するには、コンシューマは、必要な意味情報を持つデータソースへの接続を宣言します。

<source>要素と<input>要素は、COLLADA データフローモデルの一部です。また、このモデルは、ストリーム処理、パイプ、プロデューサー・コンシューマなどとも呼ばれます。入力接続は、<source>からシンク (<polylist>のような<input>の親であるデータフローコンシューマ) へのデータフローパスです。

COLLADA では、入力はすべて、インデックス値によって操作されます。コンシューマは、入力にインデックス値を指定することによって、入力をサンプリングします。同じインデックス値を共有できる複数の入力を持つコンシューマもあります。同じ offset 属性値を持つ入力は、同じインデックス値によってコンシューマから操作されます。これは、コンシューマが格納する必要があるインデックスの総数を減らす最適化です。これらの入力は、このセクションでは共有入力として記述されていますが、非共有入力の場合でも同じように動作します。

属性

<input>要素には、以下の属性があります。

offset	uint_type	親要素の<p>もしくは<v>サブ要素によって定義される、インデックスリストへのオフセット。2つの <input> 要素で同じオフセットが使用されている場合は、同じインデックスの示す値が両方の要素で使用されます。これは、インデックスリストの簡単な圧縮形式で、入力が使われる順序も定義しています。必須。
semantic	xs:NMTOKEN	入力の結び付きの意味をユーザ定義するものです。必須。COLLADA スキーマに列挙されている <input>共通の semantic 属性値の一覧については、「詳細」を参照してください。
source	urifragment_type	データソースの場所。必須。
set	uint_type	単一のセットとしてグループ化する入力。これは、複数の入力で同じセマンティックスを使用する場合に役立ちます。オプション。

関連要素

<input>要素は、以下の要素と関連性があります。

親要素	コア要素: lines , linestrips , polygons , polylist , triangles , trifans , tristrips , vertex_weights B-Rep 要素: edges , faces , pcurves , shells , solids , wires
-----	--

子要素	なし
その他	<code>p</code> (<code>lines</code> 、 <code>linestrips</code> 、 <code>polygons</code> 、 <code>polylist</code> 、 <code>triangles</code> 、 <code>trifans</code> 、 <code>tristrips</code> 内) <code>v</code> (<code>vertex_weights</code> 内)

詳細

各入力接続は、`offset` 属性によって、親要素のスコープ内で一意に特定されます。

共通プロファイル中の<input>要素の `semantic` 属性値は、以下の通りです。

semantic 属性の値	解説
<code>BINORMAL</code>	ジオメトリの従法線ベクトル
<code>COLOR</code>	カラー座標ベクトル。カラー入力は RGB (<code>float3_type</code>) です。
<code>CONTINUITY</code>	制御頂点 (CV) における連続性の制約です。詳しくは、4章「プログラミングガイド」の「曲線補間」を参照してください。
<code>IMAGE</code>	ラスタまたは MIP レベルの入力。
<code>INPUT</code>	サンブラ入力。詳しくは、4章「プログラミングガイド」の「曲線補間」を参照してください。
<code>IN_TANGENT</code>	先行する制御点の接線ベクトル。詳しくは、4章「プログラミングガイド」の「曲線補間」を参照してください。
<code>INTERPOLATION</code>	サンブラの補間種類。詳しくは、4章「プログラミングガイド」の「曲線補間」を参照してください。
<code>INV_BIND_MATRIX</code>	ローカル座標からワールド座標への変換行列の逆行列。
<code>JOINT</code>	スキンのインフルエンス識別子
<code>LINEAR_STEPS</code>	この制御点の次のスプラインセグメントで利用される区分線形近似のステップ数。詳しくは、4章「プログラミングガイド」の「曲線補間」を参照してください。
<code>MORPH_TARGET</code>	メッシュ・モーフィング用のモーフィングターゲット
<code>MORPH_WEIGHT</code>	メッシュ・モーフィングの加重値
<code>NORMAL</code>	法線ベクトル
<code>OUTPUT</code>	サンブラ出力。詳しくは、4章「プログラミングガイド」の「曲線補間」を参照してください。
<code>OUT_TANGENT</code>	後続の制御点の接線ベクトル。詳しくは、4章「プログラミングガイド」の「曲線補間」を参照してください。
<code>POSITION</code>	ジオメトリの座標ベクトル。詳しくは、4章「プログラミングガイド」の「曲線補間」を参照してください。
<code>TANGENT</code>	ジオメトリの接線ベクトル
<code>TEXBINORMAL</code>	テクスチャの従法線ベクトル
<code>TEXCOORD</code>	テクスチャ座標ベクトル
<code>TEXTANGENT</code>	テクスチャ接線ベクトル
<code>UV</code>	一般的なパラメータベクトル
<code>VERTEX</code>	メッシュ頂点
<code>WEIGHT</code>	スキンのインフルエンスの重みの値

例

以下は、1つの<polygons>要素に対する6つの<input>要素の例です。これらの<input>要素は、頂点位置、法線、テクスチャ座標とテクスチャ空間接線のセット2組を表しています。`offset` 属性は、入力がソースデータをサンプリングするために使う、<p>要素からのインデックスを示しています。複数の<input>要素が同じオフセット値を持っている場合、それらの要素は、<p>要素の中の同じインデッ

クスを共有することを意味しています。これは、文書中の領域を節約するための、簡単なインデックス圧縮形式です。

set 属性は、情報の同じ論理セットに属する<input>要素の、論理的な構成を示しています。この例では、TEXCOORD と TEXTANGENT の対が2セットあります。

```
<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>
```

input

(unshared)

カテゴリ: データフロー

概要

データソースの入力セマンティックスを宣言して、そのソースにコンシューマを接続します。

注:<input>には2種類あります。詳しくは、「<input>(shared)」を参照してください。

コンセプト

<input>要素は、コンシューマが必要とする入力の結び付きを宣言します。データソースは、データを文書の中で再利用するための、意味情報のない生データのコンテナです。このデータを利用する際には、コンシューマは、必要な意味情報を持つデータソースへの接続を宣言します。

<source>要素と<input>要素は、COLLADA データフローモデルの一部です。また、このモデルは、ストリーム処理、パイプ、プロデューサー・コンシューマなどとも呼ばれます。入力接続は、<source>からシンク (<vertices>のような<input>の親であるデータフローコンシューマ) へのデータフローパスです。

COLLADA では、入力はすべて、インデックス値によって操作されます。コンシューマは、入力にインデックス値を指定することによって、入力をサンプリングします。一意のインデックス値によって操作される簡単な入力を持つコンシューマもあります。これらの入力は、このセクションでは非共有入力として記述されていますが、共有入力の場合でも同じように動作します。

属性

`<input>`要素には、以下の属性があります。

semantic	xs:NMTOKEN	入力の結び付きの意味をユーザ定義するものです。必須。3章「スキーマのコンセプト」の「共通用語集」の中の、共通 <code><input></code> semantic 属性値のリストを参照してください。COLLADA スキーマに列挙されている <code><input></code> 共通の semantic 属性値の一覧については、「 <code><input>(shared)</code> 」を参照してください。
source	urifragment_type	データソースの場所。必須。

関連要素

`<input>`要素は、以下の要素と関連性があります。

親要素	<code>joints, sampler, targets, vertices, control_vertices</code>
子要素	なし
その他	なし

詳細

それぞれの入力は、親要素の範囲内で `offset` 属性によってユニークに識別されます。

`<sampler>` アニメーション曲線において役立つ semantic 値の説明は、4章「プログラミングガイド」の「曲線補間」を参照してください。

例

以下は、`<sampler><input>`の例です。

```

<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>

```

instance_animation

カテゴリ: アニメーション

概要

COLLADA アニメーションリソースをインスタンス化します。

コンセプト

`<instance_animation>`要素は、`<animation>`要素によって記述されたオブジェクトをインスタンス化します。`<instance_animation>`要素の中では、複数の`<animation_clip>`要素がグループ化されることにより、アニメーションクリップを構成します。

COLLADA のインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

`<instance_animation>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	インスタンス化する <code><animation></code> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。

関連要素

`<instance_animation>`要素は、以下の要素と関連性があります。

親要素	<code>animation_clip</code>
子要素	下記サブセクションを参照してください。
その他	<code>animation</code>

子要素

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	<code><animation></code> 要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、「anim」という ID で識別されるローカル定義の<animation>要素を参照する<instance_animation>要素の例です。このインスタンスは、オリジナルのオブジェクトの位置から少し平行移動されたものとなります。

```
<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>
```

instance_camera

カテゴリ: カメラ

概要

COLLADA カメラリソースをインスタンス化します。

コンセプト

<instance_camera>要素は、<camera>要素によって記述されるオブジェクトをインスタンス化して、ビジュアルシーンの中でアクティブにします。カメラオブジェクトは、親<node>のローカル座標系の中でインスタンス化されて、この座標系によって位置、方向、スケールが決まります。

COLLADA のインスタンス要素の詳細は、3 章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

<instance_camera>要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3 章「スキーマのコンセプト」の「アドレス構文」を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	インスタンス化する <camera> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。

関連要素

<instance_camera>要素は、以下の要素と関連性があります。

親要素	node
子要素	下記サブセクションを参照してください。
その他	camera

子要素

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	<code><instance_camera></code> 要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、「cam」という ID で識別されるローカル定義の `<camera>` 要素を参照する `<instance_camera>` 要素の例です。このインスタンスは、オリジナルのオブジェクトの位置から少し平行移動されたものとなります。

```

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

```

instance_controller

カテゴリ: コントローラ

概要

COLLADA コントローラリソースをインスタンス化します。

コンセプト

`<instance_controller>` 要素は、`<controller>` 要素によって記述されたオブジェクトをインスタンス化します。コントローラオブジェクトは、親 `<node>` のローカル座標系の中でインスタンス化されて、この座標系によって位置、方向、スケールが決まります。コントローラは、スキニングアニメーションやモーフィングアニメーションに基づいて、メッシュを変形することができます。

COLLADA のインスタンス要素の詳細は、3 章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

`<instance_controller>` 要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、アドレス構文 3 章の「スキーマのコンセプト」を参照してください。
name	xs:token	この要素の文字列の名。オプション。

url	xs:anyURI	<p>インスタンス化する <code><controller></code> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。</p> <p>ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。</p> <p>外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。</p>
-----	-----------	---

関連要素

`<instance_controller>`要素は、以下の要素と関連性があります。

親要素	node
子要素	下記サブセクションを参照してください。
その他	controller

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><skeleton></code>	必要な関節ノードの検索をスキンコントローラが開始すべき場所、を示します。この要素は、モーフィングコントローラに対しては意味を持ちません。主見出し項目を参照してください。	なし	0 以上
<code><bind_material></code>	FX の主見出し項目を参照してください。	なし	0 または 1
<code><extra></code>	<code><instance_controller></code> 要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、「skin」という ID で識別されるローカル定義の`<controller>`要素を参照する`<instance_controller>`要素の例です。このインスタンスは、オリジナルのオブジェクトの位置から少し平行移動されたものとなります。

```

<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>

```

また以下は、2つの`<instance_controller>`要素で、skin という ID で識別される同じくローカルに定義された`<controller>`要素を参照する例です。この2つの skin インスタンスは、`<skeleton>`要素を使ってスケルトンの別々のインスタンスにバインドされます。

```

<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>

```


instance_formula

カテゴリ: 数学

概要

COLLADA 式リソースをインスタンス化します。

コンセプト

COLLADA のインスタンス要素の詳細は、3 章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

<instance_formula>要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3 章「スキーマのコンセプト」の「アドレス構文」を参照してください。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。
url	xs:anyURI	インスタンス化すべきオブジェクトの場所の URL。必須。#文字で始まる、相対 URI の断片的な識別子を利用してローカルなインスタンスへの参照を表します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<instance_formula>要素は、以下の要素と関連性があります。

親要素	animation_clip , kinematics/axis_info , kinematics_model/technique_common
子要素	下記サブセクションを参照してください。
その他	formula , library_formulas

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<setparam>	インスタンス化された式の、ソース(引数)とデスティネーション(結果)を指定します。主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<instance_formula>`要素の例です。

```
<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>
```

instance_geometry

カテゴリ: ジオメトリ

概要

COLLADA ジオメトリリソースをインスタンス化します。

コンセプト

`<instance_geometry>`要素は、`<geometry>`要素によって記述されたオブジェクトをインスタンス化します。ジオメトリオブジェクトは、親`<node>`もしくは`<shape>`のローカル座標系の中でインスタンス化されるので、この座標系によって位置、方向、スケールが決まります。COLLADA では、凸メッシュ、メッシュ、スプラインプリミティブをサポートしています。

COLLADA のインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

`<instance_geometry>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	インスタンス化する <code><geometry></code> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。

関連要素

`<instance_geometry>`要素は、以下の要素と関連性があります。

親要素	<code>node</code> , <code>shape</code>
子要素	下記サブセクションを参照してください。

その他	geometry
-----	--------------------------

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><bind_material></code>	マテリアルシンボルをマテリアルインスタンスにバインドします。これにより、1つのジオメトリを、各回異なる外観を伴い、シーンに複数回インスタンス化することができます。主見出し項目を参照してください。	なし	0または1
<code><extra></code>	<code><instance_geometry></code> 要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0以上

詳細

例

以下は、「cube」という ID で識別されるローカル定義の[geometry](#)要素を参照する `<instance_geometry>`要素の例です。このインスタンスは、オリジナルのオブジェクトの位置から少し平行移動されたものとなります。

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

instance_light

カテゴリ: **ライティング**

概要

COLLADA 光源リソースをインスタンス化します。

コンセプト

`<instance_light>`要素は、`<light>`要素によって記述されるオブジェクトをインスタンス化して、ビジュアルシーンの中でアクティブにします。ディレクショナルライト、ポイントライト、スポットライトオブジェクトは、親`<node>`のローカル座標系の中でインスタンス化されて、この座標系によって位置、方向、スケールが決まります。例外は、環境光です。環境光は、全方向に等しく放射されるので、このような空間的変換には影響を受けません。

COLLADA のインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

`<instance_light>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	インスタンス化する <code><light></code> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。

関連要素

`<instance_light>`要素は、以下の要素と関連性があります。

親要素	<code>node</code>
子要素	下記サブセクションを参照してください。
その他	<code>light</code>

子要素

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	<code><instance_light></code> 要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0 以上

詳細

COLLADA では、インスタンス同士でデータを共有する際のポリシーを規定していません。データ共有ポリシーはランタイムアプリケーションに任されています。

例

以下は、「light」という ID で識別されるローカル定義の`<light>`要素を参照する`<instance_light>`要素の例です。このインスタンスは、オリジナルのオブジェクトの位置から少し平行移動されたものとなります。

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

instance_node

カテゴリ: シーン

概要

COLLADA ノードリソースをインスタンス化します。

コンセプト

`<instance_node>`は、`<node>`要素によって記述されたオブジェクトのインスタンスを作成します。`<node>`要素の各インスタンスは、オブジェクトをシーン中に配置するために定義された、固有のローカル座標系を持つ、`<node>`階層構造の要素を参照しています。

COLLADA のインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

`<instance_node>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、アドレス構文 3章の「スキーマのコンセプト」を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	インスタンス化する <code><node></code> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。
proxy	xs:anyURI	オプション。この属性の仕組みや使い方は、アプリケーション定義です。たとえば、バウンディングボックスや詳細度に利用することができます。詳しくは、「詳細」を参照してください。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。

関連要素

`<instance_node>`要素は、以下の要素と関連性があります。

親要素	<code>node</code>
子要素	下記サブセクションを参照してください。
その他	<code>node</code>

子要素

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	<code><instance_node></code> 要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0 以上

詳細

アプリケーションは、url 属性の URL と、proxy 属性の URL のどちらを解決するかを選択することができます。どちらも、<node>要素に解決されます。この属性の仕組みや使い方はアプリケーション定義なので、アプリケーションが追跡するパスを決定する方法に関する情報は、<instance_node>の<extra>要素に格納する必要があります。

以下に例を示します。

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

以下の例では、アプリケーションは、新しい文書をロードして、解析して、複雑なビルド (complex_building) を管理する代わりに、簡単な box ノード階層 (単純な<instance_geometry>など) を利用することができます。この複雑なビルド自体も、url と proxy をもつ<instance_node>を含んでいて、データを階層的に管理することができます。

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

以下のサンプルには、同一の COLLADA 文書に定義された<node>および<instance_node>、およびそれを参照する url および proxy があります。この構造は、使用するノードや詳細度 (LOD) の基本的な構造について、アプリケーションに複数の選択肢を提供します。

```
<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>
```

例

以下は、「myNode」という ID で識別されるローカル定義の<node>要素を参照する<instance_node>要素の例です。このインスタンスは、オリジナルのオブジェクトの位置から少し平行移動されたものとなります。

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

instance_visual_scene

カテゴリ: シーン

概要

COLLADA `visual_scene` リソースをインスタンス化します。

コンセプト

`<instance_visual_scene>`は、シーンの視覚的な面をインスタンス化します。`<scene>`要素に含むことのできる`<instance_visual_scene>`要素は、最高でも1つだけです。この制約により、文書、最上位のシーン、およびその視覚的な記述の間の関係は1対1に限定されます。この制約は、アプリケーションやツール、特にシーンを1つだけしかサポートしないものに対して、ロードするメインのシーンの目安を提供します。

COLLADA のインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

`<instance_visual_scene>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、アドレス構文 3章の「スキーマのコンセプト」を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	インスタンス化する <code><visual_scene></code> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。

関連要素

`<instance_visual_scene>`要素は、以下の要素と関連性があります。

親要素	<code>scene</code>
子要素	下記サブセクションを参照してください。
その他	<code>visual_scene</code>

子要素

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	<code><instance_visual_scene></code> 要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、「vis_scene」という ID で識別されるローカル定義の<visual_scene>要素を参照する<instance_visual_scene>要素の例です。

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

int_array

カテゴリ: データフロー

概要

整数値を保持する同種の配列を保存します。

コンセプト

<int_array>要素は、COLLADA スキーマで汎用的に利用されるデータ値を保存します。配列そのものは強く型付けされますが、セマンティックスを伴いません。単純に、整数値の並びを記述します。

属性

<int_array>要素には、以下の属性があります。

count	uint_type	配列中の値の数。必須。
id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値は、そのインスタンス文書中で一意である必要があります。オプション。
name	xs:token	この要素の文字列の名。オプション。
minInclusive	xs:integer	配列に保持可能な最も小さい整数値。デフォルトは -2147483648。オプション。
maxInclusive	xs:integer	配列に保持可能な最も大きい整数値。デフォルト値は 2147483647 です。オプション。

関連要素

<int_array>要素は、以下の要素と関連性があります。

親要素	source(core)
子要素	なし
その他	accessor

詳細

<int_array>要素には、整数値のリストが含まれます。これらの値は、<source>要素へのデータのレポジトリとして利用されます。

例

以下は、5 つの整数値の並びを記述する `<int_array>` 要素の例です。

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

joints

カテゴリ: コントローラ

概要

ジョイントノードと属性データとの関連性を宣言します。

コンセプト

ジョイントノード (スケルトンノード) と属性データを関連付けます。COLLADA では、スケルトン中の各ジョイントの逆バインド行列 (インフルエンス) で指定します。

属性

`<joints>` 要素には属性はありません。

関連要素

`<joints>` 要素は、以下の要素と関連性があります。

親要素	<code>skin</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (unshared)	<code><input></code> 要素の中には、semantic が JOINT であるものが、最低でも 1 つ必要です。 semantic が JOINT である input が参照する <code><source></code> は、ジョイントノードを特定する SID を含む <code><Name_array></code> を含んでいる必要があります。スキンコントローラを複数回インスタンス化して、各インスタンスを別々にアニメーションできるようにするために、IDREF の代わりに SID が使われます。主見出し項目を参照してください。詳しくは、3 章「スキーマのコンセプト」の「アドレス構文」を参照してください。	なし	2 以上
<code><extra></code>	<code><joints></code> 要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、ジョイントとそれぞれのバインド位置を関連付ける<joints>要素の例です。

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

library_animation_clips

カテゴリ: アニメーション

概要

<animation_clip>要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

<library_animation_clips>要素には、以下の属性があります。

id	xs:ID	<library_animation_clips> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

<library_animation_clips>要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<asset>	主見出し項目を参照してください。	なし	0 以上
<animation_clip>	主見出し項目を参照してください。	なし	1 以上
<extra>	<library_animation_clips>要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_animation_clips>`要素の例です。

```
<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>
```

library_animations

カテゴリ: アニメーション

概要

`<animation>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_animations>`要素には、以下の属性があります。

id	xs:ID	<code><library_animations></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_animations>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0または1
<code><animation></code>	主見出し項目を参照してください。	なし	1以上

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	<code><library_animations></code> 要素に関する任意の追加情報を提供します。主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_animations>`要素の例です。

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

library_cameras

カテゴリ: カメラ

概要

`<camera>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_cameras>`要素には、以下の属性があります。

id	xs:ID	<code><library_cameras></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_cameras>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code><camera></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_cameras>`要素の例です。

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

library_controllers

カテゴリ: コントローラ

概要

`<controller>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_controllers>`要素には、以下の属性があります。

id	xs:ID	<code><library_controllers></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_controllers>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code><controller></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_controllers>`要素の例です。

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

library_formulas

カテゴリ: 数学

概要

`<formula>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_formulas>`要素には、以下の属性があります。

id	xs:ID	<code><library_formulas></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。

関連要素

`<library_formulas>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	instance_formula

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code><formula></code>	その引数と結果パラメータで式を指定します。主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_formulas>`要素の例です。

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

library_geometries

カテゴリ: ジオメトリ

概要

`<geometry>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_geometries>`要素には、以下の属性があります。

id	xs:ID	<code><library_geometries></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_geometries>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code><geometry></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_geometries>`要素の例です。

```
<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>
```

library_lights

カテゴリ: ライティング

概要

`<light>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_lights>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_lights>`要素は、以下の要素と関連性があります。

親要素	COLLADA
-----	-------------------------

子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code><light></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_lights>`要素の例です。

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

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

library_nodes

カテゴリ: シーン

概要

`<node>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_nodes>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_nodes>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code><node></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_nodes>`要素の例です。

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

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

library_visual_scenes

カテゴリ: シーン

概要

`<visual_scene>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_visual_scenes>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
----	-------	---

name	xs:token	この要素の文字列の名。オプション。
------	----------	-------------------

関連要素

`<library_visual_scenes>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code><visual_scene></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_visual_scenes>`要素の例です。

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

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

light

カテゴリ: **ライティング**

概要

シーンを照らす光源を宣言します。

コンセプト

光源は、視覚的なシーンを照らす照明の源を表します。光源は、シーン内に配置することも、無限遠に設定することもできます。光源にはさまざまな属性があり、さまざまなパターンや周波数で光を放射します。COLLADA は次のようなことをサポートします。

- 環境光源は、あらゆる方向から光を放射します。環境光源の輝度は減衰しません。
- 点光源は、空間中の既知の位置から全方向に光を放射します。点光源の輝度は、光源までの距離が大きくなるのに伴って減衰します。

- 平行光源は、空間中の無限遠にある既知の方向から特定の方向に光を放射します。平行光源の輝度は減衰しません。
- スポット光源は、空間の既知の位置から特定の方向に光を放射します。スポットライトからの光は、円錐形に放射されます。放射角が光源の方向から広がるにつれて、光の輝度は減衰します。また、スポット光源の輝度は、光源からの距離が増加するにつれて減衰します。

属性

`<light>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<light>`要素は、以下の要素と関連性があります。

親要素	<code>library_lights</code>
子要素	下記サブセクションを参照してください。
その他	<code>instance_light</code> 、 <code>ambient</code> (core)、 <code>directional</code> 、 <code>point</code> 、 <code>spot</code>

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code> <code><technique_common></code>	主見出し項目を参照してください。 すべての COLLADA 実装がサポートしなければならない共通プロファイルの光源情報を、指定します。使い方の情報については「共通プロファイル」のセクションを参照してください。 <code><ambient></code> (core)、 <code><directional></code> 、 <code><point></code> 、または <code><spot></code> の要素が1つだけ含まれていなければなりません。主見出し項目を参照してください。	なし なし	0 または 1 1
<code><technique></code> (core)	各 <code><technique></code> は、 <code><technique></code> の <code>profile</code> 属性として指定された特定のプロファイルのための光源情報を指定します。主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

例

以下は、シーン中に平行光源をインスタンス化して、回転させることで夕暮れの描写を表した、`<light>`要素を含んだ`<library_lights>`要素の例です。

```
<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>

```

lines

カテゴリ: ジオメトリ

概要

メッシュが頂点属性をバインドして頂点を個々のラインに編成するために必要な情報を提供します。

コンセプト

`<lines>`要素は、`<mesh>`要素のジオメトリ・プリミティブと頂点属性の結びつきを宣言します。

頂点配列の情報は`<mesh>`要素の属性配列として別に提供され、`<lines>`要素でインデックス付けされます。

メッシュによって記述される各直線には、頂点が2つずつあります。最初の直線は、1番目と2番目の頂点から形づくられます。二番目の直線は、3番目と4番目頂点から形づくられます。

属性

`<lines>`要素には、以下の属性があります。

name	xs:token	この要素の文字列の名。オプション。
count	uint_type	線のプリミティブの数。必須。
material	xs:NCName	マテリアルのシンボルを宣言します。このシンボルはインスタンス化の際にマテリアルにバインドされます。詳しくは、 <code><instance_geometry></code> や <code><bind_material></code> を参照。オプション。指定しなかった場合、ライティングやシェーディングの結果はアプリケーションに依存します。

関連要素

`<lines>`要素は、以下の要素と関連性があります。

親要素	<code>mesh</code> , <code>convex_mesh</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (shared)	<code>input</code> が少なくとも1つ存在する場合には、そのうちのいずれかの <code>input</code> で <code>semantic="VERTEX"</code> を指定する必要があります。主見出し項目を参照してください。	なし	0 以上

名前/例	解説	デフォルト値	出現回数
<code><p></code>	任意の数の直線の頂点属性を記述するインデックスが含まれています。 <code><p></code> (「primitive」) 要素中のインデックスは、順序に応じて異なる入力を参照します。 <code><p></code> 要素の最初の値は offset 属性の値として 0 が指定された入力によって参照されます。2 番目の値は offset 属性として 1 が指定された入力によって参照されます。線の頂点は、その頂点に対する入力である値によって構成されます。すべての入力が参照された後は、次の値はオフセット 0 の値として入力によって再度参照され、新しい頂点を開始することになります。	なし	0 または 1
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

例

以下は、`<lines>` 要素の例です。3 つの `<input>` 要素が、2 つの独立した直線にまとめられています。最後の 2 つの input は同じオフセットを使っています。

```
<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>
```

linestrips

カテゴリ: ジオメトリ

概要

頂点属性をバインドし、それらの頂点を連結されたラインストリップに編成するために必要な情報を提供します。

コンセプト

`<linestrips>` 要素は、`<mesh>` 要素のジオメトリ・プリミティブと頂点属性の結びつきを宣言します。

頂点配列の情報は `<mesh>` 要素の属性配列として別に提供され、`<linestrips>` 要素でインデックス付けされます。

メッシュで記述された各ラインストリップは、任意の数の頂点を持ちます。ラインストリップ内の各線分は、現在の頂点、およびそれに先行する頂点から形づくられます。

属性

`<linestrips>`要素には、以下の属性があります。

name	xs:token	この要素の文字列の名。オプション。
count	uint_type	ラインストリップのプリミティブの数。必須。
material	xs:NCName	マテリアルのシンボルを宣言します。このシンボルはインスタンス化の際にマテリアルにバインドされます。詳しくは、 <code><instance_geometry></code> や <code><bind_material></code> を参照。オプション。指定しなかった場合、ライティングやシェーディングの結果はアプリケーションに依存します。

関連要素

`<linestrips>`要素は、以下の要素と関連性があります。

親要素	<code>mesh</code> , <code>convex_mesh</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (shared)	<code>input</code> が少なくとも1つ存在する場合には、そのうちのいずれかの <code>input</code> で <code>semantic="VERTEX"</code> を指定する必要があります。主見出し項目を参照してください。	なし	0 以上
<code><p></code>	接続された任意の数の線分の頂点属性を記述するインデックスが含まれています。 <code><p></code> (「primitive」) 要素中のインデックスは、順序によって異なる入力参照します。 <code><p></code> 要素の最初の値は <code>offset</code> 属性の値として0が指定された入力によって参照され、2番目の値は <code>offset</code> 属性として1が指定された入力によって参照されます。線の頂点は、その頂点に対する入力である値によって構成されます。すべての入力を使用した後、次の値はオフセット0の値として入力によって再度参照され、新しい頂点を開始することになります。	なし	0 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

`<linestrips>`要素には、`<p>`要素の並びが含まれます。

例

以下は、3つの頂点属性を持ち、すべての入力で同じオフセットを利用している2つの線分を表した `<linestrips>`要素の例です。

```
<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>

```

Lookat

カテゴリ: 変換

概要

カメラの照準を定めるのに適した位置や方向の変換情報が含まれます。

コンセプト

<Lookat>要素には、`float3x3_type` が含まれており、以下の3つの数学的なベクトルを記述します。

1. オブジェクトの位置
2. 注視点の位置
3. 「上」の方向

シーン中のカメラやオブジェクトの位置・方向を行列で指定すると、煩雑になりがちです。lookat 変換を利用すれば、視点や注視点や方向を直感的に指定できます。

属性

<Lookat>要素には、以下の属性があります。

sid	sid_type	
		この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。

関連要素

<Lookat>要素は、以下の要素と関連性があります。

親要素	<code>node</code>
子要素	なし
その他	なし

詳細

<Lookat>要素には、9個の浮動小数点値で構成されるリストが含まれます。この値は、OpenGL®ユーティリティ(GLU)実装と同様、以下の3つのベクトルを構成しています。

1. 視点を指定する P_x 、 P_y 、 P_z 。
2. 注視点を指定する I_x 、 I_y 、 I_z 。
3. 上方向を指定する UP_x 、 UP_y 、 UP_z 。

OpenGL と等価のビューイング行列を計算する際、視点は原点、注視点は z 軸のマイナス方向に割り当てられます。また、上方向の軸は、視平面 y 軸のプラス方向に割り当てられます。

このベクトルの値は、ローカルな、オブジェクト座標で指定されます。

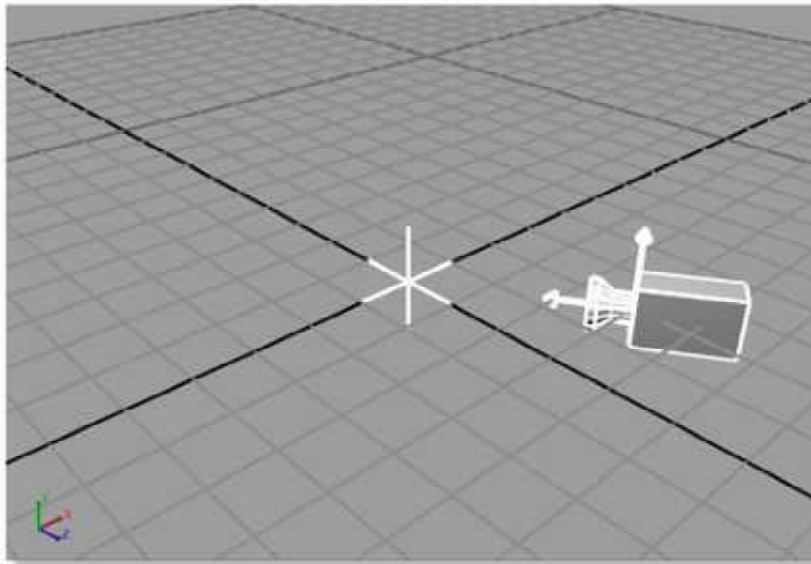
transformation 要素の適用方法の詳細については、<node>を参照してください。

例

以下は、[10,20,30]にローカル座標の原点をおき、y 軸方向を上とする<Lookat>要素の例です。

```
<node id="Camera">
  <lookat>
    2.0  0.0  3.0  <!-- 視点 (X,Y,Z)      -->
    0.0  0.0  0.0  <!-- 注視点 (X,Y,Z)   -->
    0.0  1.0  0.0  <!-- 上方向ベクトルの位置 (X,Y,Z) -->
  </lookat>
  <instance_camera url="#camera1"/>
  ...
</node>
```

図 5-1: <Lookat>要素 : 3D のクロスヘアで注視点の位置を表しています。



matrix

カテゴリ: 変換

概要

座標系中の各点や、座標系そのものに対する数学的な変更を表します。

コンセプト

<matrix>要素には、float4x4_type が含まれています。これは、浮動小数点値の 4x4 行列です。

コンピュータグラフィックスでは、データの変換に線形代数を利用します。3次元座標系は、一般的に 4x4 の行列式で表されます。これらの行列式を、シーングラフを介して階層構造化することによって、基準座標系のつながりを表すことができます。

COLLADA での行列は、数学的には「列行列」です。これらの行列は、人間が読みやすいように行優先で記述します。See the example.

属性

`<matrix>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。
-----	----------	---

関連要素

`<matrix>`要素は、以下の要素と関連性があります。

親要素	<code>node</code>
子要素	なし
その他	なし

詳細

`<matrix>`要素には、16個の浮動小数点値のリストが含まれています。この値は、行列合成に適した4×4の列順序行列に編成されます。

transformation 要素の適用方法の詳細については、`<node>`を参照してください。

例

以下は、x軸方向に2単位、y軸方向に3単位、z軸方向に4単位の平行移動を行う変換行列を表した`<matrix>`要素の例です。

```

<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>

```

mesh

カテゴリ: ジオメトリ

概要

頂点とプリミティブの情報を使って基本的なジオメトリメッシュを記述します。

コンセプト

メッシュは、主に頂点とプリミティブの情報で構成される、幾何学的形状を記述するための一般的な形式です。

頂点情報とは、メッシュ面上の特定の点に結び付けられた属性セットです。それぞれの頂点には、以下のような属性データが含まれます。

- 頂点の位置
- 頂点のカラー
- 頂点の法線
- 頂点のテクスチャ座標

またメッシュには、メッシュの幾何学的な形状を作成するために頂点をどのように組織化するのかという情報も含まれます。メッシュの頂点は、ジオメトリプリミティブ (polygons、triangles、lines など) の集合として組織化されます。

属性

<mesh>要素には属性はありません。

関連要素

<mesh>要素は、以下の要素と関連性があります。

親要素	geometry
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<source>	メッシュの頂点データの集まりを提供します。主見出し項目を参照してください。	なし	1 以上
<vertices>	メッシュの頂点属性を記述して、トポロジ的な位置を確立します。主見出し項目を参照してください。	なし	1
<i>primitive_elements</i>	入力からの値を頂点属性データにアセンブルするジオメトリプリミティブ。以下を任意の順序で任意に組合せることができます。		
<lines>	直線プリミティブを含みます。主見出し項目を参照してください。	なし	0 以上
<linestrips>	ラインストリッププリミティブを含みます。主見出し項目を参照してください。	なし	0 以上
<polygons>	穴を持つことができるポリゴンプリミティブを含みます。主見出し項目を参照してください。	なし	0 以上
<polylist>	穴を持ってないポリゴンプリミティブを含みます。主見出し項目を参照してください。	なし	0 以上
<triangles>	三角形プリミティブを含みます。主見出し項目を参照してください。	なし	0 以上
<trifans >	三角形ファンプリミティブを含みます。主見出し項目を参照してください。	なし	0 以上
<tristrips>	三角形ストリッププリミティブを含みます。主見出し項目を参照してください。	なし	0 以上
<extra>	主見出し項目を参照してください。	なし	0 以上

詳細

頂点データによって構成されるジオメトリプリミティブを記述する<mesh>要素は、<lines>、<linestrips>、<polygons>、<polylist>、<triangles>、<trifans>、そして<tristrips> というようなプリミティブ要素を 0 個以上持つことができます。

<mesh>要素中の<vertices>要素は、メッシュの頂点を記述するために利用されます。ポリゴンや三角形などは、位置を直接記述するのではなく、メッシュの頂点を示すインデックスを指定します。メッシュの頂点には、semantic 属性の値が POSITION である<input>要素(unshared)が最低でも 1 つは含まれていなければなりません。

テクスチャ座標に対しては、COLLADA の右手座標系が適用されます。したがって、[0,0]の ST 座標は、テクスチャ画像がプロフェッショナル用途の2次元テクスチャ・ビューア/エディタにロードされた際に、テクスチャ画像の左下のテクセルにマッピングされます。

例

以下は、指定可能な属性を持った空の<mesh>要素の例です。

```
<mesh>
  <source id="box-Pos"/>
  <vertices id="box-Vtx">
    <input semantic="POSITION" source="#box-Pos">
  </vertices>
</mesh>
```

異なる set 属性を持ちつつ、インデックスデータを共有したい、つまり、インデックスデータを最適化したい場合には、<input>要素を<vertices>要素から根源の要素に移動して、VERTEX という semantic を持つ input の offset 属性値を再利用することができます。

```
<vertices>
  <input semantic="POSITION"/>
  <input semantic="TEXCOORD"/>
  <input semantic="NORMAL"/>
</vertices>
<polygons>
  <input semantic="VERTEX" offset="0"/>
  ...
```

以下を利用します。

```
<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"/>
  ...
```

morph

カテゴリ: コントローラ

概要

静的メッシュの複数セットをブレンドするのに必要なデータを記述します。

コンセプト

モーフは、ベースメッシュを任意の数の「モーフターゲット」メッシュとブレンドして、最終的なメッシュを作成するために使われます。作成したメッシュは、スキニング操作の入力として使ったり、そのままレンダリングしたりすることができます。DCC ツールの中には、モーフをデフォーマと呼ぶものもあります。モーフの一般的な使い方の1つに、キャラクタに表情を適用することがあります。モーフが操作するのは、ベースメッシュおよびターゲットメッシュの中の<vertices>要素によって参照されるソースだけです。それ以外の情報 (<polylist>その他のプリミティブタグ) は、常にベースメッシュから取得されます。ターゲットメッシュには、<mesh>要素において許されるものなら何でも含むことができますが、モーフで使われるのは、<vertices>要素のコンテンツだけです。

<morph> には、ベースメッシュとターゲットメッシュのブレンド方法を記述するは重みが含まれています。シングルターゲットの場合、重み0のモーフは、ベースメッシュと一致する頂点を出力します。この重みが1に近づくにつれて、出力頂点は、ベースメッシュの値から、モーフターゲット中の対応する頂点の値に、徐々に変化していきます (たとえば、ベースの頂点5は、ターゲットの頂点5に近づいていきます)。重みが1に達すると、モーフの出力頂点は、ターゲットの頂点に一致するようになります。

ベースメッシュおよびターゲットメッシュの<vertices>要素は、すべて、同じセマンティックスをもつ<input>要素を、同じ数、同じ順序で含んでいる必要があります。また、モーフメッシュの中に存在する頂点は、すべて、同じ数である必要があります。よい結果を得るには、ベースメッシュの頂点とターゲットメッシュの頂点が、一対一に対応している必要があります。たとえば、ベースメッシュとモーフターゲットの双方がキャラクタの顔である場合、ベースメッシュの頂点5とモーフターゲットの頂点5は、どちらも顔の上のだいたい同じ場所 (目尻など) を表している必要があります。

<vertices>要素に含まれているものは、すべて、モーフによってブレンドされます。<vertices>要素の中に、セマンティックスが POSITION、NORMAL、TEXCOORD その他の数値である<input>があれば、これらはすべてモーフィングされます。

<morph>の中の<targets>の要素は、モーフターゲットの<mesh>要素リストを含む1つの<source>要素、および重みの<float_array>を含むもう1つの<source>を指しています。

<morph>要素のmethod属性は、これらのメッシュを組み合わせるのに使われる式を指定します。モーフターゲットを組み合わせる方法としては、さまざまな方法を利用できます。一般的なものは、以下の2つの方法です。

- **NORMALIZED**

$$0 \quad (\text{Target1, Target2, ...}) * (w1, w2, ...) = \\ (1-w1-w2-...) * \text{BaseMesh} + w1 * \text{Target1} + w2 * \text{Target2} + \dots$$

- **RELATIVE**

$$0 \quad (\text{Target1, Target2, ...}) + (w1, w2, ...) = \text{BaseMesh} + w1 * \text{Target1} + \\ w2 * \text{Target2} + \dots$$

属性

`<morph>`要素には、以下の属性があります。

source	xs:anyURI	ベースメッシュを記述する <code><geometry></code> を参照します。必須。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
method	列挙	どのブレンド方法を利用するのかを指定します。指定可能な値は、NORMALIZEDと RELATIVE です。デフォルト値は NORMALIZED です。オプション。

関連要素

`<morph>`要素は、以下の要素と関連性があります。

親要素	controller
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><source></code>	モーフ重み、およびモーフターゲットのデータ。主見出し項目を参照してください。	なし	2 以上
<code><targets></code>	ブレンドする入力メッシュ(モーフターゲット)。このメッシュの中には、最低でも、セマンティックが MORPH_WEIGHT の子 <code><input></code> 要素が 1 つ、およびセマンティックが MORPH_TARGET の子 <code><input></code> 要素が 1 つ必要です。主見出し項目を参照してください。	なし	1
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

注釈付サンプルについては、http://collada.org/mediawiki/index.php/Skin_and_morph を参照してください。

例

以下は、空の `<morph>` 要素の例です。

```
<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>
```

Name_array

カテゴリ: データフロー

概要

シンボル名値を保持する同種の配列を格納します。

コンセプト

`<Name_array>`要素には、COLLADA スキーマで汎用的に利用される名前値が、データとして格納されます。この配列は、強く型付けされていますが、セマンティックスはありません。ただ、XML の名前値の並びを格納しているだけです。

属性

`<Name_array>`要素には、以下の属性があります。

count	uint_type	配列中の値の数。必須。
id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<Name_array>`要素は、以下の要素と関連性があります。

親要素	<code>source</code> (core)
子要素	なし。
その他	<code>accessor</code>

詳細

`<Name_array>`要素には、XML の名前値(`xs:Name`)のリストが含まれます。これらの値は、`<source>`要素へのデータのレポジトリとして利用されます。アプリケーションは、アプリケーション定義の任意の名前値を指定することができます。

たとえば、`<Name_array>`を曲線補間の記述のソースとして使えば、処理する曲線の種類をアプリケーションが指定することができます。共通プロファイルでは、`BEZIER`、`LINEAR`、`BSPLINE`、および`HERMITE`という値が定義されています。

例

以下は、4 つの名前値の並びを表した`<Name_array>`要素の例です。

```
<Name_array id="names" name="myNames" count="4">
  Node1 Node2 Joint3 WristJoint
</Name_array>
```

以下は、サンブラに補間の種類を渡す例です。

```
<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>

```

newparam

カテゴリ: パラメータ

プロファイル: External、Effect、CG、COMMON、GLES、GLES2、GLSL

概要

新たに名前付きパラメータオブジェクトを作成して、型と初期値を割り当てます。

コンセプト

パラメータは、コンパイラや実行時の関数から利用できる、型付きのデータオブジェクトです。

FXの場合、パラメータは、FX ランタイムの中で作成されますが、宣言時に追加の属性を割り当てることもできます。

キネマティックスの場合、パラメータは、インスタンス化されたキネマティックスオブジェクトの特定の属性へのアクセスを提供します。

属性

<newparam>要素には、以下の属性があります。

sid	sid_type	説明
		パラメータの識別子（つまり変数名）。必須。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。

関連要素

<newparam>要素は、以下の要素と関連性があります。

親要素	FX 要素: <code>effect</code> , <code>profile_CG</code> , <code>profile_COMMON</code> , <code>profile_GLSL</code> , <code>profile_GLES</code> , <code>profile_GLES2</code> キネマティックス要素: <code>instance_kinematics_model</code> , <code>instance_articulated_system</code> , <code>instance_kinematics_scene</code> , <code>axis_info</code> , <code>effector_info</code> , <code>kinematics_model/technique_common</code> コア要素: <code>formula</code>
子要素 その他	下のサブセクションを参照 <code>param</code> (reference) <code>setparam</code>

FX 内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<annotate>	主見出し項目を参照（COMMON 内では無効）	なし	0 以上

名前/例	解説	デフォルト値	出現回数
<code><semantic></code>	主見出し項目を参照	なし	0 または 1
<code><modifier></code>	主見出し項目を参照 (COMMON 内では無効)	なし	0 または 1
<code>parameter_type _element</code>	<p>パラメータの型。11章「型」の「パラメータの型要素」に記載された、該当するグループの要素のいずれかである必要があります。</p> <p>CG:<code>cg_param_group</code></p> <p>GLSL:<code>gls1_value_group</code></p> <p><code><effect></code>中: <code>fx_newparam_group</code></p> <p>GLES: <code>gles_param_group</code></p> <p>GLES2:<code>gles2_value_group</code></p> <p>COMMON スコープ内では、以下のいずれかである必要があります。</p> <p><code><float></code></p> <p><code><float2></code></p> <p><code><float3></code></p> <p><code><float4></code></p> <p><code><sampler2D></code></p>	なし	1

キネマティクスおよび`<formula>`/`<newparam>`の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code>parameter_type</code>	<p>パラメータの型。以下の要素のいずれかをちょうど1個含む必要があります。</p> <p><code><float></code></p> <p><code><int></code></p> <p><code><bool></code></p> <p><code><SIDREF></code></p>	なし	1

例

以下は、FX の場合の例です。

```
<newparam sid="diffuseColor">
  <annotate name="UIWidget"><string>none</string></annotate>
  <semantic>DIFFUSE</semantic>
  <modifier>EXTERN</modifier>
  <float3> 0.30 0.56 0.12 </float3>
</newparam>
```

以下は、キネマティクスの場合の例です。

```
<instance_kinematics_model url="#KINEMATICS_MODEL_ARM" sid="model">
  <newparam sid="kinematics.model">
    <SIDREF>model</SIDREF>
  </newparam>
</instance_kinematics_model>
```

node

カテゴリ: シーン

概要

シーン中の興味のある点を宣言します。

コンセプト

`<node>`要素は、シーン中の興味のある点を宣言することによって、シーン中の要素の階層関係を表します。ノードは、シーン・グラフの枝上にある、特定の点を表しています。実際には、`<node>`要素はシーングラフ全体の部分グラフのルートとなります。

シーングラフの概念には、アーク（弧）とノード（節）があります。ノードは、グラフ内の情報を表す点です。弧は、ノードを他のノードにつなげている部分です。さらに、ノードは内部ノード（枝）と外部ノード（葉）に分かれます。COLLADA では、内部ノードを表す場合に「ノード」という用語を利用しています。また、弧は路とも呼ばれます。

属性

`<node>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。
sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。
type	列挙	<code><node></code> 要素の型。有効な値は、JOINT または NODE です。デフォルトは、NODE です。オプション。
layer	list_of_names_type	そのノードが属しているレイヤの名前。たとえば、「foreground glowing」という値は、このノードが、foreground という名前のレイヤと、glowing という名前のレイヤの両方に属していることを示します。デフォルト値は空で、ノードがどのレイヤにも属していないことを意味します。オプション。

関連要素

`<node>`要素は、以下の要素と関連性があります。

親要素	library_nodes , node , visual_scene
子要素	下記サブセクションを参照してください。
その他	instance_node

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	アセット管理情報を表現する。主見出し項目を参照してください。	なし	0 または 1

名前/例	解説	デフォルト値	出現回数
<i>transformation_elements</i>	以下の transformation 要素の任意の組合せ。 <Lookat> <matrix> <rotate> <scale> <skew> <translate> 主見出し項目を参照してください。	なし	0 以上
<instance_camera>	カメラオブジェクトをインスタンス化する。主見出し項目を参照してください。	なし	0 以上
<instance_controller>	コントローラオブジェクトをインスタンス化する。主見出し項目を参照してください。	なし	0 以上
<instance_geometry>	ジオメトリオブジェクトをインスタンス化する。主見出し項目を参照してください。	なし	0 以上
<instance_light>	光源オブジェクトをインスタンス化する。主見出し項目を参照してください。	なし	0 以上
<instance_node>	他のノードの階層構造をインスタンス化する。主見出し項目を参照してください。	なし	0 以上
<node>	階層構造を再帰的に定義する。主見出し項目を参照してください。	なし	0 以上
<extra>	階層構造を再帰的に定義する。主見出し項目を参照してください。	なし	0 以上

詳細

<node>要素は、シーングラフ・トポロジーの基礎となります。そのため、<node>要素には、<node>要素自身を含め、さまざまな子要素が記述できます。

<node>要素は、各子変換要素が出現順で合成されるコンテキストを表します。他の子要素には、<node>要素の範囲内で累積された変換が等しく適用されます。

変換要素は<node>要素の座標系を変換することになります。これは、数学的には、変換要素が行列に変換されてから、<node>の中で指定された順序で掛け算され、座標系を構成するということを意味します。

例

以下の例は、<node>要素を 2 つもつ、<visual_scene>要素の概略を示しています。2 つのノードの名前は、それぞれ「earth」と「sky」です。

```
<visual_scene>
  <node name="earth">
</node>
  <node name="sky">
</node>
</visual_scene>
```

optics

カテゴリ: カメラ

概要

カメラに搭載されている、画像を画像センサーの上に投影する装置を表します。

コンセプト

光学系 (optics) は、1 つ以上の光学系要素で構成されます。光学系要素は、通常、光の経路をどのように変更するのかによって分類されます。

- 反射要素: 鏡など (ニュートン式望遠鏡の凹面主鏡やクロム・ボールなどで、環境マップを取り込むために利用されます)
- 屈折要素: レンズやプリズムなど

特定のカメラ光学系では、これらを複雑に組合せたものが内蔵されることがあります。たとえば、シュミット式望遠鏡には凹面レンズと凹面主鏡の両方と、さらに接眼部のレンズも含まれています。

現実の可変焦点式の「ズームレンズ」には、実際には 10 枚を超えるレンズと可変絞り (アイリス絞り) が含まれることがあります。

コンピュータグラフィックスで一般に利用されている「透視投影」カメラモデルは、「ズームレンズ」の単純な近似で、無限に小さな絞り (レンズ関連の値である焦点距離の代わりに) 直接指定された視野を持ちます。

属性

<optics>要素には属性はありません。

関連要素

<optics>要素は、以下の要素と関連性があります。

親要素	camera
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<technique_common>	すべての COLLADA 実装がサポートしなければならない共通プロファイルの光学の (optics) 情報を、指定します。詳しくは、「共通プロファイル」の使い方情報のセクション、および以下の子要素詳細についてのサブセクションを参照してください。主見出し項目も参照してください。	なし	1
<technique> (core)	各<technique>は、<technique>の profile 属性によって指定され、特定のプロファイルのための光学情報を指定します。主見出し項目を参照してください。	なし	0 以上
<extra>	主見出し項目を参照してください。	なし	0 以上

<optics> / <technique_common>の子要素

名前/例	解説	デフォルト値	出現回数
<orthographic> または、 <perspective>	投影の種類。主見出し項目を参照してください。	なし	1

詳細

共通プロファイルでは<perspective>と<orthographic>の光学タイプが定義されています。それ以外の <optics>要素のタイプはプロファイル固有の <technique>で指定されていなければなりません。

例

以下は、視野角 45 度での scene の透視図を記述する <camera>要素の例です。

```
<camera name="eyepoint">
  <optics>
    <technique_common>
      <perspective>
        <yfov>45.0</yfov>
        <aspect_ratio>1.33333</aspect_ratio>
      <znear>0.1</znear>
      <zfar>32767</zfar>
    </perspective>
  </technique_common>
</optics>
</camera>
```

orthographic

カテゴリ: カメラ

概要

正投影カメラの視野を記述します。

コンセプト

正投影とは、2次元の面上に3次元シーンを描画するための方法です。正投影では、オブジェクトの外観上のサイズはカメラからの距離に依存しません。

<perspective>と比較してみてください。

属性

<orthographic>要素には属性はありません。

関連要素

<orthographic>要素は、以下の要素と関連性があります。

親要素	optics / technique_common
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

注:<orthographic>要素は、以下のいずれかを含む必要があります。

- 単一の<xmag>要素
- 単一の<yimag>要素
- <xmag>要素と<yimag>要素の両方
- <aspect_ratio>要素、および<xmag>か<yimag>のいずれか。

これらの要素は、カメラの視野を記述するためのものです。<aspect_ratio>要素が存在しない場合、アスペクト比は、<xmag>または<yimag>の要素と、現在のビューポートから計算されます。

名前/例	解説	デフォルト値	出現回数
<xmag sid="...">	ビューの水平(X)拡大率を表す浮動小数点数が含まれます。sid属性はオプションです。	なし	「注」を参照してください。
<yimag sid="...">	ビューの垂直(Y)拡大率を表す浮動小数点数が含まれます。sid属性はオプションです。	なし	「注」を参照してください。
<aspect_ratio sid="...">	視野のアスペクト比を表す浮動小数点値が含まれます。sid属性はオプションです。	なし	「注」を参照してください。
<znear sid="...">	前方クリップ面への距離を表す浮動小数点値が含まれます。sid属性はオプションです。	なし	1
<zfar sid="...">	後方クリップ面への距離を表す浮動小数点値が含まれます。sid属性はオプションです。	なし	1

詳細

X倍率およびY倍率は単純なスケール係数であり、正投影ビューポートのXコンポーネントおよびYコンポーネントに適用されます。これより、デフォルトの正投影ビューポートが、OpenGLやDirectXなどの場合と同様に[[**-1,1**],[**-1,1**]]であるとした場合、COLLADAの正投影ビューポートは[[**-xmag,xmag**],[**-ymag,ymag**]]になります。このことにより、**xmag/2**の正投影幅および**ymag/2**の正投影高さが与えられます。

中央のスクリーンピクセルは、スクリーン座標で(0,0)であると仮定されます。

例

以下は、標準的なビュー（拡大縮小なし、標準のアスペクト比）を指定した<orthographic>要素の例です。

```
<orthographic>
  <xmag sid="animated_zoom">1.0</xmag>
  <aspect_ratio>0.1</aspect_ratio>
  <znear>0.1</znear>
  <zfar>1000.0</zfar>
</orthographic>
```

param

(データフロー)

カテゴリ: データフロー

概要

親要素のパラメータに関する情報を宣言します。

注:他の要素の<param>については、「<param> (参照)」を参照してください。

コンセプト

関数形式やプログラム形式では、ユーザが何らかの方法でパラメータ情報を指定できなければなりません。この情報は、関数のパラメータ(引数)データを表します。

マテリアルシェーダプログラムには、頂点プログラムやピクセルプログラムを表すコードが含まれる場合があります。これらのプログラムでは、状態情報の一部としてパラメータを必要とします。

基本的なパラメータ宣言では、パラメータの名前、データ型、値データを記述します。パラメータ名は、パラメータを関数やプログラムに対して識別させることを可能にします。パラメータの型は、値の符号化方法を示しています。<param>要素には、パラメータの実際の値を表す `xs:string` 型の情報が含まれます。

属性

<param>要素には、以下の属性があります。

<code>name</code>	<code>xs:token</code>	この要素の文字列の名。オプション。
<code>sid</code>	<code>sid_type</code>	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。
<code>type</code>	<code>xs:NMTOKEN</code>	データ値の型。このテキスト文字列は、アプリケーションの理解可能な形式でなければなりません。必須。
<code>semantic</code>	<code>xs:NMTOKEN</code>	パラメータの意味をユーザ定義するためのものです。オプション。

関連要素

<param>要素は以下の要素と関連があります。

親要素	<code>accessor, bind_material</code>
子要素	なし
その他	なし

詳細

<param>要素は、汎用的なデータ制御のためのパラメータを記述します。

例

以下は、<accessor>の出力を表す2つの<param>要素の例です。

```
<accessor source="#values" count="3" stride="3">
```

```
<param name="A" type="int"/>
<param name="B" type="int"/>
</accessor>
```

param

(リファレンス)

カテゴリ: パラメータ

プロファイル: External、COMMON、CG、GLES2、GLSL

概要

定義済みのパラメータを参照します。

注:<accessor>や<bind_material>の中のこの要素については、「<param>(データフロー)」を参照してください。

コンセプト

パラメータは、ランタイム中に作成された型付けされたデータオブジェクトで、ランタイム時にコンパイラや関数で利用することができます。

FXの場合、これはシェーダバインディング宣言において事前に定義されたパラメータを指します。

キネマティックスの場合、これはバインディング宣言において事前に定義されたパラメータを指します。このパラメータは、キネマティックスオブジェクトの特定の属性へのアクセスを提供します。

属性

詳しくは、「詳細」サブセクションを参照してください。

関連要素

<param>要素は、以下の要素と関連性があります。

親要素	詳しくは、「詳細」サブセクションを参照してください。
子要素	なし
その他	modifier , newparam , setparam , usertype

詳細

<param>は、<newparam>を使って作成された既存のパラメータのSIDを参照します。SIDを参照する方法は、<param>の親要素によって異なります。

SIDの詳細は、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。

fx_common_color_or_texture_type、common_float_or_param_type、<bind_uniform>型の要素の中

シェーダ属性要素 (<ambient>、<diffuse>など) および<bind_uniform>の中。

<param>要素には、以下の属性があります。

ref	sidref_type	必須。既存のパラメータのSIDへのパス。
-----	-------------	----------------------

`<param>`要素は、以下の要素と関連性があります。

親要素	<code>ambient (FX)</code> 、 <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>
-----	---

`<param>`要素には、いかなる情報も含まれません。言い換えれば、`<param></param>`は常に空です。

レンダーターゲットおよびキネマティックスの場合

`<param>`要素には、以下の属性があります。

<code>ref</code>	<code>xs:token</code>	必須。既存のパラメータの ID を参照します。
------------------	-----------------------	-------------------------

`<param>`要素は、以下の要素と関連性があります。

親要素	<code>color_target</code> 、 <code>depth_target</code> 、 <code>stencil_target</code> 、 <code>bind (キネマティックス)</code> 、 <code>bind_kinematics_model</code>
-----	---

`<param>`要素には、いかなる情報も含まれません。言い換えれば、`<param></param>`は常に空です。

`<texture*>`内

`<param>`要素には属性はありません。

`<param>`要素は、以下の要素と関連性があります。

親要素	<code>texture1D</code> 、 <code>texture2D</code> 、 <code>texture3D</code> 、 <code>textureCUBE</code> 、 <code>textureRECT</code> 、 <code>textureDEPTH</code>
-----	--

`<param>`要素には、既存のパラメータの SID を表わす、`sidref_type` 型の情報が含まれます。

`<bind_material>`および`<accessor>`内

`<param>`要素には、以下の属性があります。

<code>name</code>	<code>xs:token</code>	オプション。この要素の文字列の名。
<code>sid</code>	<code>sid_type</code>	オプション。
<code>semantic</code>	<code>xs:NMTOKEN</code>	オプション。ユーザ定義のパラメータの意味。
<code>type</code>	<code>xs:NMTOKEN</code>	必須。データ値の型。このテキスト文字列は、アプリケーションの理解可能な形式でなければなりません。 <code><param></code> が <code><accessor></code> 要素の子であるときには、この属性は、 <code>int</code> 、 <code>float</code> 、 <code>Name</code> 、 <code>bool</code> 、 <code>IDREF</code> 、 <code>SIDREF</code> の配列型に限定されます。

`<param>`要素は、以下の要素と関連性があります。

親要素	<code>bind_material</code> 、 <code>accessor</code>
-----	--

`<param>`要素には、既存のパラメータの SID を表わす、`xs:string` 型の情報が含まれます。

例

以下は、シェーダの例です。

```

<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>

```

以下は、キネマティックスの場合の例です。

```
<instance_articulated_system sid="system" url="#MOTION">
  <bind symbol="motion.kinematics.model">
    <param ref="kinematics.model"/>
  </bind>
</instance_articulated_system>
```

perspective

カテゴリ: カメラ

概要

透視カメラの視野を記述します。

コンセプト

透視図とは、視点からの距離によって物体の大きさが相対的に変化する関係を表します。コンピュータグラフィックスでは、3次元物体を2次元の平面上にレンダリングして表示モニタ上に適切な比率の像を生成するために、透視投影の技法を使います。

[<orthographic>](#)と比べてみてください。

属性

[<perspective>](#)要素には属性はありません。

関連要素

[<perspective>](#)要素は、以下の要素と関連性があります。

親要素	optics/technique_common
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

注:[<perspective>](#)要素は、以下のいずれかを含む必要があります。

- 単一の[<xfov>](#)要素
- 単一の[<yfov>](#)要素
- [<xfov>](#)要素と[<yfov>](#)要素の両方
- [<aspect_ratio>](#)要素、および[<xfov>](#)か[<yfov>](#)のいずれか。

これらの要素は、カメラの視野を記述するためのものです。最初の2つの指定では、アプリケーションは、カメラのアスペクト比を、ビューポートのアスペクト比に基いて計算することができます。

名前/例	解説	デフォルト値	出現回数
<xfov sid="...">	水平視野を度数で表す浮動小数点数が含まれています。 sid 属性はオプションです。	なし	「注」を参照してください。

名前/例	解説	デフォルト値	出現回数
<code><yfov sid="..."></code>	垂直視野を度数で表す浮動小数点値が含まれています。 sid 属性はオプションです。	なし	「注」を参照してください。
<code><aspect_ratio sid="..."></code>	視野のアスペクト比を表す浮動小数点値が含まれます。 sid 属性はオプションです。	なし	「注」を参照してください。
<code><znear sid="..."></code>	前方クリップ面への距離を表す浮動小数点値が含まれます。 sid 属性はオプションです。	なし	1
<code><zfar sid="..."></code>	後方クリップ面への距離を表す浮動小数点値が含まれます。 sid 属性はオプションです。	なし	1

詳細

`<aspect_ratio>`要素が指定されていない場合、アスペクト比は、`<xfov>`または`<yfov>`の要素と、現在のビューポートから計算されます。アスペクト比は、視野の幅と高さの比として定義されます。したがって、アスペクト比は、 $\text{aspect_ratio} = \text{xfov} / \text{yfov}$ という式にしたがって、視野パラメータから導出したり、視野パラメータの導出に利用したりすることができます。

中央のスクリーンピクセルは、スクリーン座標で (0,0) であると仮定されます。

クリップ面までの距離は、この要素のスコープ中の`<asset/>``<unit>`によって定義される、現在の単位によって指定されます。

例

以下は、水平視野角 90 度を指定し、アニメーションのターゲットにもなる`<perspective>`要素の例です。

```
<perspective>
<xfov sid="animated_zoom">90.0</xfov>
<aspect_ratio>1.333</aspect_ratio>
<znear>0.1</znear>
  <zfar>1000.0</zfar>
</perspective>
```

point

カテゴリ: ライティング

概要

点光源を記述します。

コンセプト

`<point>`要素は、点光源を記述するために必要なパラメータを宣言します。点光源は、空間中の既知の位置から全方向に光を放射します。点光源の輝度は、光源までの距離が大きくなるのに伴って減衰します。

光源の位置は、インスタンス化されるノードの座標変換で定義されます。

属性

`<point>`要素には属性はありません。

関連要素

`<point>`要素は、以下の要素と関連性があります。

親要素	<code>light / technique_common</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><color></code>	光源のカラーを表す3つの浮動小数点数を含みます。主見出し項目を参照してください。	なし	1
<code><constant_attenuation sid="..."></code>	詳しくは、「詳細」を参照してください。sid属性はオプションです。	1.0	0または1
<code><linear_attenuation sid="..."></code>	詳しくは、「詳細」を参照してください。sid属性はオプションです。	0.0	0または1
<code><quadratic_attenuation sid="..."></code>	前方クリップ面への距離を表す浮動小数点値が含まれます。sid属性はオプションです。	なし	1

詳細

特定の距離における総合的な減衰は、`<constant_attenuation>`、`<linear_attenuation>`、`<quadratic_attenuation>`を利用して計算されます。使われる式は、次の通りです。

$$A = \text{constant_attenuation} + (\text{Dist} * \text{linear_attenuation}) + ((\text{Dist}^2) * \text{quadratic_attenuation})$$

例

以下は、`<point>`要素の例です。

```
<light id="blue">
  <technique_common>
    <point>
      <color>0.1 0.1 0.5</color>
      <linear_attenuation>0.3</linear_attenuation>
    </point>
  </technique_common>
</light>
```

polygons

カテゴリ: ジオメトリ

概要

メッシュが頂点属性をバインドして頂点を個々のポリゴンに編成するために必要な情報を提供します。

コンセプト

`<polygons>`要素は、`<mesh>`要素のジオメトリ・プリミティブと頂点属性の結びつきを宣言します。

注: ポリゴンは、データを格納する方法としては推奨される方法ではありません。ジオメトリを最も効率的に表現するには、`<triangles>`や `<polylist>`を利用してください。`<polygons>` を使うのは、穴が必要な場合だけ、その際も、穴のある部分だけで使うようにしてください。

頂点配列の情報は`<mesh>`要素の属性配列として別に提供され、`<polygons>`要素でインデックス付けされます。

記述されたポリゴンは、任意の数の頂点を持ちます。ポリゴンは凸図形であることが理想ですが、凹図形であってもかまいません。ポリゴンには穴を含めることも可能です。4つ以上の頂点を含んでいるポリゴンプリミティブは、非平面である場合もあります。

さまざまな操作で、サーフェスのポイントの正確な方向が必要となります。この方向を法線ベクトルで部分的に定義できますが、法線自体の周りの「回転」については不明瞭のままです。この曖昧な余分の回転をなくす1つの方法は、同じポイントでのサーフェスの接線を指定することです。

座標系の種類がすでにわかっていると仮定すると(たとえば、右手系)、この方法でサーフェスの方向を完全に指定できます。つまり、オブジェクト空間とサーフェス空間との間の変換を行う 3x3 行列が定義できます。

接線と法線で、サーフェス座標系の2つの座標軸(行列の2つの列)を指定します。従法線と呼ばれる3番目の座標軸は、接線と法線の外積として計算できます。

2つの異なる種類の接線は、以下のように文書中で異なる目的と論理的な配置を持つので、COLLADAでは、それら2つの接線をサポートしています。

- テクスチャ空間の接線: semantic 属性として `TEXTANGENT` と `TEXBINORMAL` を記述し、set 属性を記述した`<input>`(shared)要素で指定します。
- 標準の(幾何学的)接線: semantic 属性として `TANGENT` と `BINORMAL` を記述した`<input>`(shared)要素で指定します。

属性

`<polygons>`要素には、以下の属性があります。

<code>count</code>	<code>uint_type</code>	ポリゴンプリミティブの数。必須。
<code>material</code>	<code>xs:NCName</code>	マテリアルのシンボルを宣言します。このシンボルはインスタンス化の際にマテリアルにバインドされます。詳しくは、 <code><instance_geometry></code> や <code><bind_material></code> を参照。オプション。指定しなかった場合、ライティングやシェーディングの結果はアプリケーションに依存します。
<code>name</code>	<code>xs:token</code>	オプション。

関連要素

`<polygons>`要素は、以下の要素と関連性があります。

親要素	<code>mesh</code> , <code>convex_mesh</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (shared)	input が少なくとも1つ存在する場合には、そのうちのいずれかの input で <code>semantic="VERTEX"</code> を指定する必要	なし	0 以上

名前/例	解説	デフォルト値	出現回数
	があります。主見出し項目を参照してください。		
<code><p></code>	特定のポリゴンの頂点属性（インデックス）を指定する <code>uint_type</code> 値のリストが含まれます。詳しくは、「詳細」を参照してください。	なし	0 以上
<code><ph></code>	1 つ以上の穴を含むポリゴンを記述します。下記サブセクションを参照してください。	0	0 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

polygons / ph 子要素

`<ph>`要素に属性はありません。

`<ph>`に子要素が存在する場合、その子要素は以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><p></code>	特定のポリゴンの頂点属性（インデックス）を指定する <code>uint_type</code> 値のリストが含まれます。詳しくは、「詳細」を参照してください。	なし	1
<code><h></code>	<code><p></code> によって指定されたポリゴンの、穴のインデックスを指定する <code>uint_type</code> 値のリストを含みます。詳しくは、「詳細」を参照してください。	なし	1 以上

詳細

`<p>`（「primitive」）（もしくは`<h>`）要素中の各インデックスは入力に参照されますが、参照時には各値の記述された順番が使用されます。`<p>`要素の最初のインデックスは `offset` 属性の値として 0 が指定された入力によって参照され、2 番目のインデックスは `offset` 属性として 1 が指定された入力によって参照されます。ポリゴンの頂点は、その頂点に対する入力である値によって構成されます。すべての入力が参照された後は、次の値はオフセット 0 の値として入力によって再度参照され、新しい頂点を開始することになります。

生成される頂点の順番は反時計回りで、それぞれのポリゴンのサーフェスを記述します。

プリミティブが頂点法線を持たずに構成される場合には、ライティングを可能にするために、プリミティブごとの法線をアプリケーションが生成する場合もあります。

例

以下は、単一の正方形を記述する`<polygons>`要素の例です。`<polygons>`要素には2つの`<source>`要素が含まれており、それぞれには`<input>`(shared)要素のセマンティクスに応じた位置と法線データが含まれています。`<p>`要素の値の記述された順序によって、入力値の利用される順序が決まります。

```

<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>

```

以下は、ジオメトリの接線を指定する方法の簡単な例です（法線と接線の入力は、どちらも `offset` が 1 であるため、`<p>` 要素中の同じエントリが参照される点に注意してください）。

```
<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>
```

以下は、テクスチャ空間の接線を指定する方法を示した例です（テクスチャ空間の接線は、入力への `offset` や順番ではなく、`set` 属性によって、特定のテクスチャ座標セットと関連付けられている点に注意してください）。

```
<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>
```

polylist

カテゴリ: ジオメトリ

概要

メッシュが頂点属性をバインドして頂点を個々のポリゴンに編成するために必要な情報を提供します。

コンセプト

`<polylist>` 要素は、`<mesh>` 要素のジオメトリ・プリミティブと頂点属性の結びつきを宣言します。

頂点配列の情報は `<mesh>` 要素の属性配列として別に提供され、`<polylist>` 要素でインデックス付けされます。

`<polylist>` に記述されているポリゴンには、任意の数の頂点を含めることができます。4 つ以上の頂点を含んでいる `polylist` プリミティブは、非平面である可能性もあります。

さまざまな操作で、サーフェスのポイントの正確な方向が必要となります。この方向を法線ベクトルで部分的に定義できますが、法線自体の周りの「回転」については不明瞭のままです。この曖昧な余分の回転をなくす1つの方法は、同じポイントでのサーフェスの接線を指定することです。

座標系の種類がすでにわかっていると仮定すると（たとえば、右手系）、この方法でサーフェスの方向を完全に指定できます。つまり、オブジェクト空間とサーフェス空間との間の変換を行う 3x3 行列が定義できます。

接線と法線で、サーフェス座標系の2つの座標軸（行列の2つの列）を指定します。従法線と呼ばれる3番目の座標軸は、接線と法線の外積として計算できます。

2つの異なる種類の接線は、以下のように文書中で異なる目的と論理的な配置を持つので、COLLADA では、それら2つの接線をサポートしています。

- テクスタ空間の接線：semantic 属性として `TEXTANGENT` と `TEXBINORMAL` を記述し、set 属性を記述した `<input>` (shared) 要素で指定します。
- 標準の（幾何学的）接線：semantic 属性として `TANGENT` と `BINORMAL` を記述した `<input>` (shared) 要素で指定します。

属性

`<polylist>` 要素には、以下の属性があります。

<code>name</code>	<code>xs:token</code>	この要素の文字列の名。オプション。
<code>count</code>	<code>uint_type</code>	ポリゴンプリミティブの数。必須。
<code>material</code>	<code>xs:NCName</code>	マテリアルのシンボルを宣言します。このシンボルはインスタンス化の際にマテリアルにバインドされます。詳しくは、 <code><instance_geometry></code> や <code><bind_material></code> を参照。オプション。指定しなかった場合、ライティングやシェーディングの結果はアプリケーションに依存します。

関連要素

`<polylist>` 要素は、以下の要素と関連性があります。

親要素	<code>mesh</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (shared)	<code>input</code> が少なくとも1つ存在する場合には、そのうちのいずれかの <code>input</code> で <code>semantic="VERTEX"</code> を指定する必要があります。主見出し項目を参照してください。	なし	0 以上
<code><vcount></code>	<code><polylist></code> 要素で記述されているポリゴンの頂点の数を表す整数値のリストが含まれます。	なし	0 または 1
<code><p></code>	特定の <code>polylist</code> の頂点属性（インデックス）を指定する整数値のリストが含まれます（「p」は「primitive」の頭文字です）	なし	0 または 1
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

生成される頂点の順番は反時計回りで、それぞれのポリゴンのサーフェスを記述します。

プリミティブが頂点法線を持たずに構成される場合には、ライティングを可能にするために、プリミティブごとの法線をアプリケーションが生成する場合があります。

例

以下は、2つの四角形と1つの三角形を表した<polylist>要素の例です。<polylist>要素には2つの<source>要素が含まれており、それぞれには<input>(shared)要素のセマンティックスに応じた位置と法線データが含まれています。<p>要素の値の記述された順序によって、入力値の利用される順序が決まります。

```
<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>
```

rotate

カテゴリ: 変換

概要

オブジェクトを軸のまわりで回転させる際の回転方法を指定します。

コンセプト

回転は、平行移動を行わずに、座標系の中の物体の向きだけを変更します。コンピュータグラフィックスでは、向きを変える際に、座標系を基準として値を変更せずに済むように回転移動を利用します。言い換えると、<rotate>要素は、ローカル座標の原点を中心に座標軸を変換することを意味します。

この要素には、角度と回転軸を表す数学的なベクトルが含まれます。

属性

<rotate>要素には、以下の属性があります。

sid	sid_type	
		この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。

関連要素

`<rotate>`要素は、以下の要素と関連性があります。

親要素	コア要素: <code>node</code> フィジックス要素: <code>rigid_body</code> 内の <code>technique_common/mass_frame</code> 、 <code>instance_rigid_body</code> 、 <code>shape</code> 、 <code>ref_attachment</code> 、 <code>Attachment</code> キネマティクス要素: <code>frame_object</code> 、 <code>frame_origin</code> 、 <code>frame_tcp</code> 、 <code>frame_tip</code> 、 <code>link</code>
子要素	なし
その他	なし

詳細

`<rotate>`要素には、OpenGL[®]や RenderMan[®]の回転の仕様と同様に、4つの浮動小数点値のリストが含まれています。これらの値は、回転軸を指定する列ベクトル[X, Y, Z]と、度数で表された角度を表します。

transformation 要素の適用方法の詳細については、`<node>`を参照してください。

例

以下は、Y軸を基軸とした90度の回転を表す`<rotate>`要素の例です。

```
<rotate>
  0.0 1.0 0.0 90.0
</rotate>
```

sampler

カテゴリ: **アニメーション**

概要

アニメーション用の補間サンプリング関数を宣言します。

コンセプト

COLLADA では、アニメーションの関数曲線を1次元の `<sampler>`要素で表します。`<sampler>`要素では、サンプリングポイントと、それらの補間方法を定義します。アニメーション・チャンネルの値の計算に利用する際には、アニメーションのキーフレームがサンプリングポイントとなります。

サンプリングポイント(キーフレーム)は、補間の種類のシンボル名ですが、サンブラへの入力データソースです。アニメーション・チャンネルは、サンブラの出力データ値をターゲットに出力します。

アニメーション曲線(`<animation>`/`<sampler>`)

アニメーションでは、対象となるパラメータが時間の経過に従ってどのように変化するかを定義するために曲線が使用されます。曲線の定義は、`<geometry>`/`<spline>`の定義に似ていますが、異なるのは、アニメーションキーを含む特殊な一次元の軸が存在することです。このキーによって、ある1つまたは一連のパラメータが、アニメーション中に時間の経過に従ってどのように変化するかが決まります。

キーは、通常はTIME 値ですが、ほかの任意の変数にすることもできます。たとえば、列車の車輪の回転をその列車のレール上の位置に関連付けてもかまいません。そうすれば、列車を前後に動かすことで車輪やその他のメカニズムを自動的に動かすことができます。

キー軸内のアニメーションは単調曲線に制限されます。つまり、アニメーションキーは、入力の昇順にソートされている必要があり、重複があってはなりません。これは、アニメーション曲線が閉じてはいけないことを意味します。

キーは<source>配列に格納され、これが、<geometry>/<spline>の全 POSITION 入力 の 1 番目の軸の代わりになります。同一キー値を持つ複数の異なった曲線を使用すれば、複数のパラメータをアニメートできます。それらのパラメータは OUTPUT 配列によって与えられます。

要約すると、次のようになります。

$$\text{POSITION}[i].X = \text{INPUT}[i]$$

$$\text{POSITION}[i].Y = \text{OUTPUT}[i]$$

n 個の曲線がある場合、曲線 j の点 i は次のように表されます。

$$\text{POSITION}[j][i] = \text{INPUT}[j][i]$$

$$\text{POSITION}[j][i+1] = \text{OUTPUT}[j][i]$$

属性

<sampler>要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
pre_behavior	列挙	<p>オプション。最初のキー以前のサンプル値のあるべき挙動を示します。有効な値は次のとおりです。</p> <p>UNDEFINED: デフォルト値。前後の挙動は定義されません。</p> <p>CONSTANT: 最初の(behavior_before)または最後の(behavior_after)専用の値が返されます。</p> <p>GRADIENT: 値は、サンプル中の最後の 2 つのキーによって決まる直線に従います (Maya®の LINEAR と同じです)</p> <p>CYCLE: アニメーションが循環するように、[first_key , last_key] の区間にキーがマッピングされます。</p> <p>OSCILLATE: アニメーションが往復するように、[first_key , last_key] の区間にキーがマッピングされます。</p> <p>CYCLE_RELATIVE: アニメーションが無限に繰り返されます。</p> <p>詳しい情報については「詳細」を参照してください。</p>
post_behavior	列挙	<p>オプション。最後のキー以後のサンプル値がいくらであるべきかを示します。有効な値は、pre_behavior と同じです。</p>

関連要素

<sampler>要素は、以下の要素と関連性があります。

親要素	animation
子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<code><input></code> (unshared)	<code><input></code> (unshared)要素の中には、semantic 属性が INTERPOLATION である要素が、最低でも 1 つは存在する必要があります。主見出し項目を参照してください。	なし	1 以上

詳細

サンプリングポイントは、`<source>`要素を参照する`<input>`要素で記述します。`<input>`要素の semantic 属性には、INPUT、INTERPOLATION、IN_TANGENT、OUT_TANGENT、OUTPUT いずれかを指定するか、もしくは他のものを指定する場合があります。

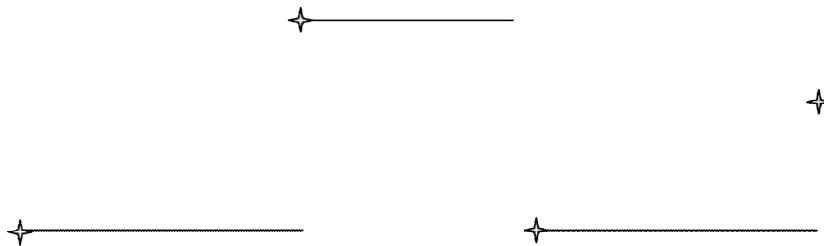
COLLADA では、次の種類の補間法を認識します。LINEAR、BEZIER、CARDINAL、HERMITE、BSPLINE、および STEP。これらのシンボル名は、シンボル名が格納されている`<Name_array>`を含む`<source>`要素に保持されます。これらの値は、INTERPOLATION `<input>`要素によってサンブラに供給されます。

完全な`<sampler>`要素には、semantic 属性が INTERPOLATION の `<input>`要素を 1 つ含まれていなければなりません。COLLADA では、デフォルトの補間方式が指定されていません。つまり、補間の種類を指定しなかった場合、`<sampler>`の動作はアプリケーションの定義したものとなります。

詳しくは、4 章「プログラミングガイド」の「曲線補間」を参照してください。

STEP 補間

アニメーション曲線では、「STEP」という別の補間タイプも使えます。この場合、次のセグメントが始まるまで値がセグメントの最初の点と同じ値に維持されます。具体的には、次の曲線ようになります。



これに対する COLLADA コードは、次のとおりです。

```

<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"/>

```

線形アニメーション曲線

LINEAR 補間は STEP に似ていますが、この補間では、各キー値間でパラメータの値が線形補間されません。

ベジェ / エルミートアニメーション曲線

ベジェ補間とエルミート補間は、<input>の semantic が POSITION ではなく INPUT と OUTPUT であることを除けば、<spline>の説明と同じです。INPUT および OUTPUT セマンティックは常に 1-D パラメータです。前述したとおり、OUTPUT が 2 次元以上の場合には、複数のパラメータが同じキー値に基づいて、それぞれ独立して補間されます。IN_TANGENT セマンティックと OUT_TANGENT セマンティックはそれぞれ、1 つのキー値を持ち、その後パラメータごとに値を 1 つずつ持ちます。

3 次ベジェ・エルミート補間の式としては、<spline>用に定義されたものと同じ数式が使われます。パラメータ j 、segment $[i]$ に対するジオメトリベクトルは、以下のようになります。

ベジェの場合：

- P_0 は、(INPUT $[i]$, OUTPUT $[j][i]$)
- C_0 (もしくは T_0) は、(OUT_TANGENT $[0][i]$, OUT_TANGENT $[j][i]$)
- C_1 (もしくは T_1) は、(IN_TANGENT $[0][i+1]$, IN_TANGENT $[j][i+1]$)
- P_1 は、(INPUT $[i+1]$, OUTPUT $[j][i+1]$)

特殊な場合:接線値が 1 次元

エクスポートによっては、接線の退化した曲線をエクスポートすることがあります。これは COLLADA 仕様ではサポートされていません。このような退化は、関連するエクスポートの更新に伴ってなくなっていくはずですが、以下の情報は、あくまでも参考のために提供するにすぎません。

1 次元の接線データというこの特殊なケースでは、OUT_TANGENT と IN_TANGENT はキー値を含まず、したがって OUTPUT 配列と同じ次元を持ちます。

不足しているキー値は、INPUT セグメントが提供するキーの線形補間として提供されます。ジオメトリベクトル値は次のように、通常アニメーション曲線の場合と同様に提供されます。

- P_0 は、(INPUT $[i]$, OUTPUT $[j][i]$)
- C_0 は、(INPUT $[i]/3 + INPUT[i+1] * 2/3$, OUT_TANGENT $[j][i]$)
- C_1 は、(INPUT $[i]*2/3 + INPUT[i+1]/3$, IN_TANGENT $[j][i+1]$)
- C_1 は、(INPUT $[i+1]$, OUTPUT $[j][i+1]$)

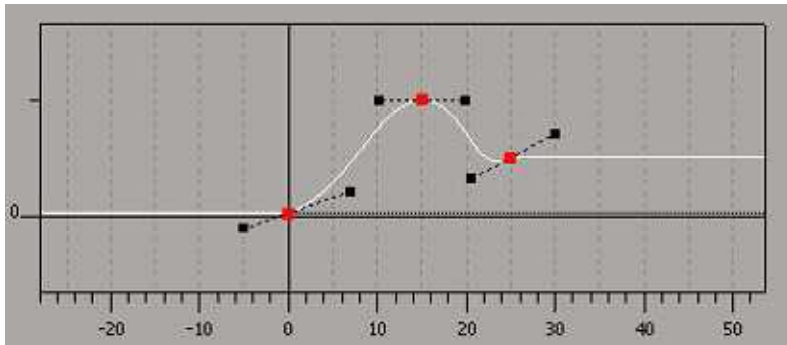
B スプライン / カーディナルアニメーション曲線

BSPLINE および CARDINAL 曲線については、先に説明したのと同じ原則が当てはまります。POSITION は、INPUT と OUTPUT を結合することで得られます。これらのアニメーション曲線には、先に定義したのと同じ式が適用されます。

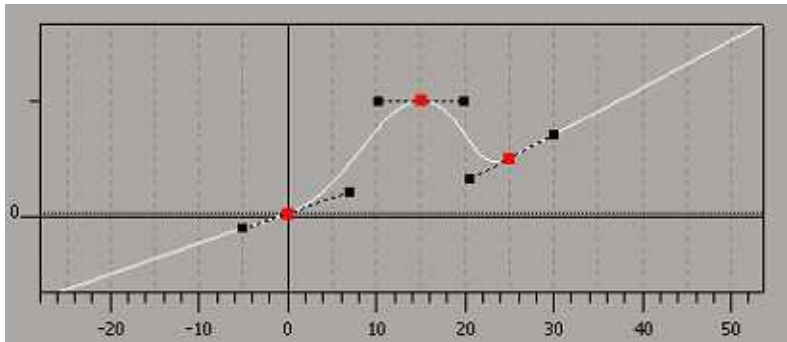
前後の挙動

2 つのオプション属性 pre_behavior と post_behavior は、最初のキーの前、および最後のキーの後のサンプル値の、あるべき値を表します。以下の図および擬似コードは、さまざまな挙動オプションの例を示しています。

CONSTANT の挙動は、次の通りです。



GRADIENT の挙動は、以下の通りです。



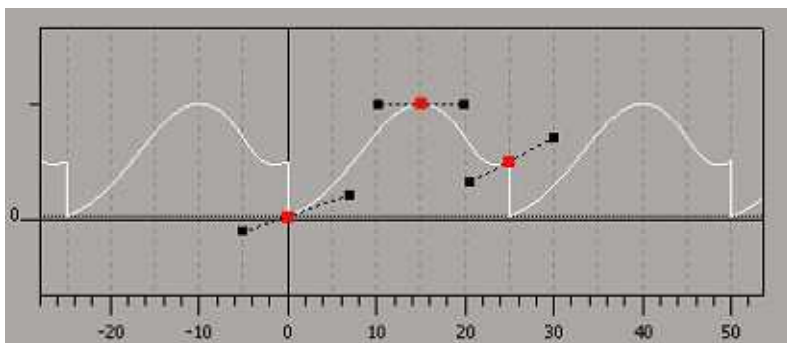
```

++ post_behavior 用の擬似コード: key > last_key > first_key
if 補間によって接線値が得られる場合:
    tangent = tangent[last_key]
else
    tangent = value[last_key]-value[penultimate_key] / (last_key -
penultimate_key)
return value[last_key] + tangent * (key-last_key)

++ pre_behavior 用の擬似コード: key < first_key < last_key
if 補間によって接線値が得られる場合:
    tangent = tangent[first_key]
else
    tangent = value[second_key]-value[first_key] / (second_key- first_key)
return value[fist_key] + tangent * (key-fist_key)

```

CYCLE の挙動は、以下の通りです。



```

++ post_behavior 用の擬似コード: 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]

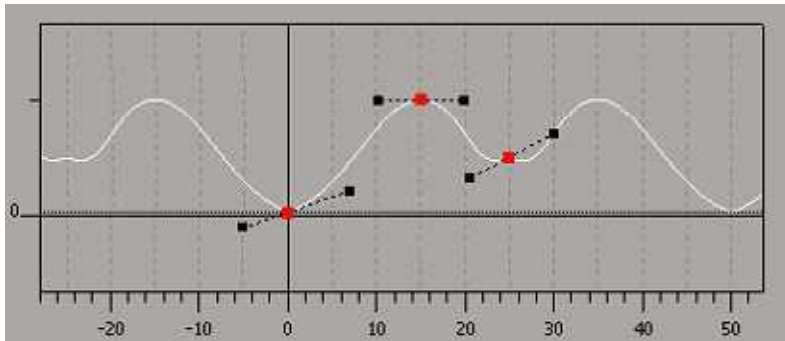
```

```

++ pre_behavior 用の擬似コード: 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]

```

OSCILLATE の挙動は、以下の通りです。



```

++ post_behavior 用の擬似コード: 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 // アニメーションを逆方向に再生
    new_key = first_key + last_key - (key - (last_key - first_key) * repeat);
return value[new_key]

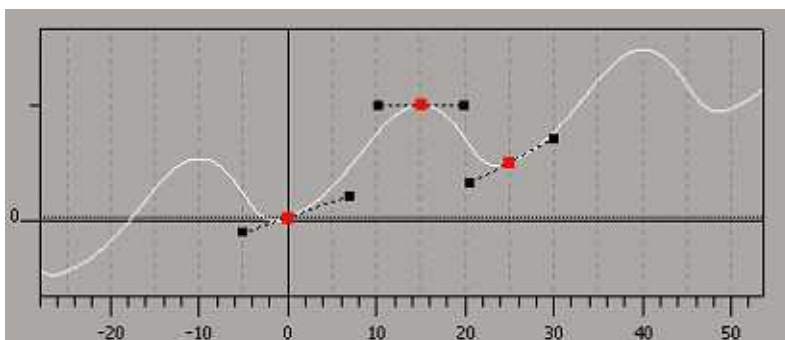
```

```

++ pre_behavior 用の擬似コード: key < first_key < last_key
repeat = (int) (first_key - key) / (last_key - first_key)
if (repeat is odd) // CYCLE と同じ
    new_key = key + (last_key - first_key) * (repeat + 1);
else // アニメーションを逆方向に再生
    new_key = first_key + last_key - (key + (last_key - first_key) * (repeat + 1));
return value[new_key]

```

CYCLE_RELATIVE の挙動は、以下の通りです。



```

++ post_behavior 用の擬似コード: 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

```

```

++ pre_behavior 用の擬似コード: 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)

```

例

以下は、「group1_translate-anim-outputY」という id を持つキーフレームの source 要素の y 軸方向の値を評価する `<sampler>` 要素の例です。INTERPOLATION 入力は、`<source>` 要素に、より明確にするため以下のように示されます。

```
<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>
```

scale

カテゴリ: 変換

概要

オブジェクトのサイズを変更する方法を指定します。

コンセプト

拡大縮小は、回転や平行移動を伴わずに、座標系内の物体の大きさを変更します。コンピュータグラフィックスでは、座標系軸に対する値の大きさや比率を変更するために拡大縮小変換を利用します。

この要素には、座標系の x 軸、y 軸、z 軸の相対的な比率を表す数学的なベクトルが含まれます。

属性

`<scale>` 要素には、以下の属性があります。

sid	sid_type	説明
		この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。

関連要素

`<scale>`要素は、以下の要素と関連性があります。

親要素	node
子要素	なし
その他	なし

詳細

`<scale>`要素には、3つの浮動小数点値のリストが含まれています。これらの値は、行列合成に適した列ベクトルに編成されます。

スケール値がゼロの場合、結果はその軸に垂直な平面上への投影になります。たとえば、 $z=0$ の場合、あらゆる点が (x,y) 平面の上に投影されます。スケール値が負の場合、結果は、該当する軸上での鏡映を伴います。

`transformation`要素の適用方法の詳細については、[<node>](#)を参照してください。

例

以下は、物体（座標系）の大きさを一律に2の倍数で増やす変換を記述した`<scale>`要素の例です。

```
<scale>
  2.0 2.0 2.0
</scale>
```

scene

カテゴリ: シーン

概要

COLLADA リソースのコンテンツで視覚化可能な情報セット全体を表します。

コンセプト

各 COLLADA 文書は、最高1個の`<scene>`要素を含むことができます。

`<scene>`要素は、シーン階層（シーングラフ）のベースを宣言します。シーンには、オーサリングツールで作成された視覚的情報や変換情報といったコンテンツの大部分を提供する要素が含まれます。

シーンの階層構造は、シーングラフで構成されます。シーングラフは、視覚情報および関連データをノードとして含む有向非巡回グラフ（DAG）、つまりツリー構造のデータです。シーングラフ構造は、データ処理の最適化やレンダリングに役立つため、コンピュータグラフィックスでは幅広く利用されています。

属性

`<scene>`要素には属性はありません。

関連要素

`<scene>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。

その他	なし
-----	----

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><instance_physics_scene></code>	フィジックスの主見出し項目を参照してください。	なし	0 以上
<code><instance_visual_scene></code>	主見出し項目を参照してください。	なし	0 または 1
<code><instance_kinematics_scene></code>	Kinematics の主見出し項目を参照してください。	なし	0 または 1
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

`<scene>` 文書要素の中には、`<COLLADA>` 要素 (ルート要素) が最大 1 つ宣言されます。シーングラフは、`<scene>` 下でインスタンス化された `<visual_scene>` 要素から構成されます。インスタンス化された `<physics_scene>` 要素は、シーンに適用される任意のフィジックス (物理) を表します。

例

以下は、id が「world」の視覚的シーンをインスタンス化する単純な `<scene>` 要素の例です。

```
<COLLADA>
<scene>
  <instance_visual_scene url="#world"/>
</scene>
</COLLADA>
```

setparam

カテゴリ: パラメータ

FX プロファイル: External、Effect、CG、GLES2

概要

先に定義されたパラメータに新しい値を割り当てます。

コンセプト

パラメータはランタイム時に `<newparam>` として定義することができ、もしくはソースコードや事前にコンパイルされたバイナリ中のグローバルパラメータとしてコンパイル/リンク時に発見することができます。それぞれの `<setparam>` は推測的な呼び出しで、以下の効果を狙っています。

- 「X」と呼ばれるシンボルを検索して、現在の範囲内に見つかった場合には、このデータ型の値を割り当てます。シンボルが見つからなかった場合や値を割り当てることができなかった場合には、無視してロードを続けます。

FX の高度な言語プロファイルでは、`<setparam>` を使って、`<array length="N"/>` 要素を利用したサイズ指定のない配列に具体的な配列サイズを割り当てたり、`<usertype>` のパラメータのインスタンスを抽象インタフェース型のパラメータと結び付けたりすることができます。

FXの外では、`<setparam>`は、COLLADA 共通のデータ型のプールからのみ、値を割り当てることができます。

属性

`<setparam>`要素には、以下の属性があります。

ref	xs:token	値セットを持つ事前に定義されたパラメータの ID を参照します。必須。
-----	----------	-------------------------------------

関連要素

`<setparam>`要素は、以下の要素と関連性があります。

親要素	FX 要素: <code>instance_effect</code> 、 <code>usertype</code> キネマティクス要素: <code>instance_articulated_system</code> 、 <code>instance_kinematics_scene</code> 、 <code>axis_info</code> 、 <code>effector_info</code> 、 <code>instance_kinematics_model</code> 、 コア要素: <code>instance_formula</code>
子要素	下記サブセクションを参照してください。
その他	<code>newparam</code> 、 <code>param</code> (reference)

子要素

名前/例	解説	デフォルト値	出現回数
<code>parameter_type_element</code>	対応するスコープ内で有効なパラメータ型の要素については、章末の「パラメータの型要素」を参照してください。 CG: <code>cg_param_group</code> GLS2: <code>gles2_value_group</code> <code><instance_effect></code> : <code>fx_setparam_group</code> キネマティクス、および <code><instance_formula></code> は、以下のいずれかである必要があります。 <code><float></code> <code><int></code> <code><bool></code> <code><SIDREF></code> <code><connect_param></code> : Kinematics の主見出し項目を参照してください。	なし	1

詳細

FX ランタイムローダでは、失敗した`<setparam>`をレポートするかどうか規定されていませんが、失敗した際に効果のロードを中断すべきではありません。

例

以下は、FX の場合の例です。

```
<setparam ref="light_Direction">
  <float3> 0.0 1.0 0.0 </float3>
</setparam>
```

以下は、キネマティクスの場合の例です。

```
<instance_articulated_system url="#MOTION_SYSTEM" sid="model">
```

```

<setparam ref="motion.model.elbow.x.locked">
  <bool>true</bool>
</setparam>

</instance_articulated_system>

```

SIDREF_array

カテゴリ: データフロー

概要

スコープ付き識別子の参照値を含んだ同種の配列の格納場所を宣言します。

コンセプト

`<SIDREF_array>`要素は、インスタンス文書中のスコープ付き識別子 (SID) を参照する値を格納します。

属性

`<SIDREF_array>`要素には、以下の属性があります。

id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値は、そのインスタンス文書中で一意である必要があります。オプション。
名前	xs:token	この要素の文字列の名。オプション。
count	uint_type	配列中の値の数。必須。

関連要素

`<SIDREF_array>`要素は、以下の要素と関連性があります。

親要素	<code>source</code> (core)
子要素	なし
その他	<code>accessor</code>

詳細

`<SIDREF_array>`要素には、COLLADA のスコープ付き識別子アドレス値 (`sidref_type`) のリストが含まれます。これらの値は、`<source>`要素へのデータのレポジトリとして利用されます。

スコープ付き識別子の詳細は、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。

例

以下は、`<SIDREF_array>`要素の例です。

```

<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>

```

skeleton

カテゴリ: コントローラ

概要

スキンコントローラが必要なジョイントノードの検索をどこから始めるのかを指定します。

コンセプト

シーングラフが複雑になるにともなって、シーン中に同じオブジェクトを何度も表示しなければならない場合があります。そのため、スペースを節約するために、オブジェクトの実際のデータ表現を一度保存しておき、複数の場所で参照することが可能です。しかしながら、シーンに毎回表示する際に、さまざまにオブジェクトを変形したい状況も発生します。スキンコントローラの場合、オブジェクトの変形は外部のノードセットから派生されます。

また、同じスキンコントローラの複数のインスタンスに、ノードセットの別々のインスタンスを参照させたい場合も発生するでしょう。その場合、個々のコントローラを個別にアニメーション化する必要があります。スキンコントローラをアニメーション化するためには、それに影響を及ぼすノードをアニメーション化しなければならないからです。

さらに、別々のスキンコントローラのインスタンスに、同じノードセットを参照させる必要が出てくる場合もあります。たとえば、キャラクターに衣装や防護具をアタッチする場合です。そうした場合には、単一セットのノードの操作によって両方のコントローラの変形が行えます。

属性

<skeleton>要素には属性はありません。

関連要素

<skeleton>要素は、以下の要素と関連性があります。

親要素	instance_controller
子要素	なし
その他	なし

詳細

この要素には、`xs:anyURI` 型の URI が含まれます。

例

以下は、<skeleton>要素がどのようにして、「skin」という名前で特定されるローカル定義された同一の<controller>要素を参照する2つのコントローラインスタンスを、特定のスケルトンの異なるインスタンスにバインドするのかを示した例です。

```
<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"/>
  </controller>
</library_controllers>
```

```

        <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>

```

skew

カテゴリ: 変換

概要

オブジェクトを特定の軸に沿って変形させる方法を指定します。

コンセプト

スキュー処理（剪断変形）とは、特定の座標軸に沿って物体を変形させることです。変形の影響を受ける値は、その値の使用する座標軸に対して平行移動されます。コンピュータグラフィックスでは、物体を変形させたり、画像の歪みを補正したりするために、この剪断変形変換を利用します。

この要素には、角度と、回転軸と平行移動軸を表す 2 つ数学的なベクトルが含まれています。

属性

<skew>要素には、以下の属性があります。

sid	sid_type	
		この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。

関連要素

`<skew>`要素は、以下の要素と関連性があります。

親要素	<code>node</code>
子要素	なし
その他	なし

詳細

`<skew>`要素には、RenderMan[®]の仕様と同様に、7つの浮動小数点値のリストが含まれています。これらの値は、度数で表された角度と、それに続く回転軸と平行移動軸を指定する2つの列ベクトルを表します。

`transformation`要素の適用方法の詳細については、`<node>`を参照してください。

例

以下は、Y軸に対して45度回転させ、X軸に沿ったポイントの変位を表す`<skew>`要素の例です。

```
<skew>
  45.0 0.0 1.0 0.0 1.0 0.0 0.0
</skew>
```

skin

カテゴリ: コントローラ

概要

重み付けブレンドを用いたスキニングの記述に必要な頂点とプリミティブの情報を含みます。

コンセプト

キャラクターのスキニングを行う場合には、アニメーションエンジンによって、スキニングされたキャラクターの関節（スケルトン）が動かされます。関節と、スキンのトポロジーを構成するメッシュの頂点との関連付けは、スキンメッシュによって記述します。制御アルゴリズムによってスキンメッシュの頂点を変換するうえで、関節が変換に影響を与えます。

一般的なスキニングアルゴリズムでは、隣接する関節の影響を重みの値に応じてブレンドします。

古典的なスキニングアルゴリズムでは、ジオメトリの点（メッシュの頂点など）を、ノード（関節とも呼ばれます）の行列で変換し、スカラー加重を利用して、その結果の加重平均を計算します。スキニングの影響を受けるジオメトリはスキンと呼ばれ、変換（ノード）およびそれに対応する重みの組合せは、インフルエンスと呼ばれ、影響を与えるノードの集合（一般に階層構造を持つ）は骨格と呼ばれます。

スキニング処理には、以下の2つのステップが必要となります。

- 「骨格とスキンのバインド」と呼ばれる前処理
- 骨格のポーズの変化に応じてスキン形状を変更するための、スキニング・アルゴリズムの実行。前処理によって作成される「スキニング情報」は、以下の情報から構成されます。
- バインド形状：「デフォルトの形状」とも呼ばれます。これは、スキンをスケルトンにバインドした時点でのスキンの形状です。バインド形状には、`<mesh>`の各頂点に対応する位置が必須として含まれるほか、オプションとして頂点の属性を含めることができます。
- インフルエンス：`<mesh>`の各頂点に対応する、ノードと重みの対の可変長リストです。

- **バインドポーズ**：バインド時点での各インフルエンスの座標変換を表します。これは、ノードごとの情報として保存される「バインド行列」で表現されます。バインド行列とは、バインド時にノードをローカル座標からワールド座標へ変換するための行列です。

スキニングアルゴリズムにおいては、変換はすべてバインドポーズに対して相対的に行われます。この相対変換は、通常、スケルトンの各ノードについて事前に計算され、スキニング行列として保存されません。

頂点の新しい（スキニングされた）位置を導き出すために、それぞれ影響を受けるノードのスキニング行列は、頂点のバインドシェイプ位置を変換して、その結果をブレンド加重値で平均化します。

スキニング行列を導き出す最も簡単な方法は、ノードの現在のローカル座標からワールドへの変換行列に、ノードのバインド行列の逆行列を掛け合わせることです。この操作により、各ノードによるバインド・ポーズ変換は事実上打ち消され、スキン共通のオブジェクト空間で作業することが可能になります。

バインド処理は、通常、以下の処理を必要とします。

- スキンの現在の形状をバインド形状として格納すること
- バインド行列を計算して保存すること
- フォールオフ機能をいくつか備えたデフォルトのブレンド加重値の生成。特定の頂点からより遠くのジョイントが得られ、影響力がさらに少なくなります。また、重みが0の場合、インフルエンスは無視してかまいません。

このような処理によって、アーティストが重みを変更できるようになります。通常、重みの編集は、メッシュ上への「ペイント」によって行います。

属性

<skin>要素には、以下の属性があります。

source	xs:anyURI	ベースメッシュ（静的メッシュまたはモーフィングメッシュ）を参照する URI。スキンメッシュのバインド形状も表します。必須。
--------	-----------	---

関連要素

<skin>要素は、以下の要素と関連性があります。

親要素	controller
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<bind_shape_matrix>	バインド前のベースメッシュの位置と方向に関して追加情報を指示します。4x4 の行列を表す 16 個浮動小数点数が列優先順序で含まれています。COLLADA 文書では、人間にとって読みやすいように、行優先順序で書かれます。<bind_shape_matrix>要素が指定されていない場合、<bind_shape_matrix>として識別情報行列が利用されます。この要素には属性はありません。	なし	0 または 1
<source>	特定のベースメッシュのスキニングに必要なほとんどのデータを指定します。主見出し項目を参照してください。	なし	3 以上

名前/例	解説	デフォルト値	出現回数
<code><joints></code>	このスキンに必要なジョイントごとの情報を集めたものです。主見出し項目を参照してください。	なし	1
<code><vertex_weights></code>	スキンで利用される関節と重み付けの頂点ごとの組み合わせを表します。関節の配列への-1というインデックスは、バインド形状を参照します。重み付けは、利用する前に正規化するべきです。主見出し項目を参照してください。	なし	1
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

バインド形状中の各頂点 v に対するスキニングの計算は、以下の通りです。

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

- n : 頂点 v に影響するジョイントの数
- BSM: バインド形状行列
- IBM_i : ジョイント i の逆バインド行列
- JMi : ジョイント i のジョイント行列
- JW: 頂点 v に対するジョイント i のジョイント影響・重み

通常の最適化では

- $(v * BSM)$ は、ロード時に計算され、格納されます。

COLLADA 内のスキニングに関する定義

- バインド形状 (またはベースメッシュ) : `<skin>` 要素の `source` 属性によって参照されるメッシュの頂点。
- ジョイント: `semantic="JOINT"` をもつ `<input>` 要素 (unshared) によって参照された `<source>` の中で、SID によって指定されたノード。SID は、通常、`<Name_array>` の中に格納されます。この中では、名前と SID (ノード) が 1 対 1 に対応しています。`<skeleton>` 要素は、スキンコントローラがインスタンス化を行う際に、SID ルックアップを開始する位置を定義します。関節行列は、ランタイム時にこれらのノードから取得できます。
- 重み: `semantic="WEIGHT"` をもつ `<input>` 要素 (unshared) によって参照された `<source>` の中の値。通常は、`<float_array>` に格納され、1 度に 1 個の浮動小数点数が取り出されます。`<vertex_weights>` 要素は、スキンにより使用されるジョイント、重みの組合せを記述します。
- 逆バインド行列: `semantic="INV_BIND_MATRIX"` をもつ `<input>` 要素 (unshared) によって参照された `<source>` の中の値。通常は、`<float_array>` に格納され、1 度に 16 個の浮動小数点数が取り出されます。`<joints>` 要素は、ジョイントを逆バインドポーズ行列に関連付けます。
- バインド形状行列: バインド形状のスキニング前の変換を表す単一の行列。

例

以下は、指定可能な属性を持った `<skin>` 要素の例です。

```
<controller id="skin">
  <skin source="#base_mesh">
    <source id="Joints">
      <Name_array count="4"> Root Spinel Spine2 Head </Name_array>
      ...
    </source>
  </skin>
</controller>
```

```

<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>

```

source

(core)

カテゴリ: データフロー

概要

<source>要素は、その<source>要素を参照する<input>要素のセマンティックスにしたがって値を提供するデータのレジストリを宣言します。

注:<ampler*>要素の中の<source>については、該当する要素を参照してください。

コンセプト

データソースとは、確立された通信チャンネルを介してアクセス可能な既知の情報源のことです。

データソースは、情報へのアクセス方式を提供します。このアクセス方式としては、情報の表現に応じて各種のものが実装されています。情報は、データの配列としてローカルに保存することも、データを生成するプログラムとして使用することもできます。

属性

<source>要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。必須。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<source>`要素は、以下の要素と関連性があります。

親要素	コア要素: animation , mesh , morph , skin , spline フィジックス要素: convex_mesh B-Rep 要素: brep , nurbs , nurbs_surface
子要素	下記サブセクションを参照してください。
その他	accessor

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code>array_element</code>	データ配列要素。以下のいずれかになります。 <code><bool_array></code> <code><float_array></code> <code><IDREF_array></code> <code><int_array></code> <code><Name_array></code> <code><SIDREF_array></code> <code><token_array></code> 主見出し項目を参照してください。	なし	0 または 1
<code><technique_common></code>	すべての COLLADA 実装がサポートしなければならない共通プロファイルのソース情報を、指定します。詳しくは、「共通プロファイル」の使い方情報のセクション、およびそれに続く子要素詳細についてのサブセクションを参照してください。	なし	0 または 1
<code><technique></code> (core)	各 <code><technique></code> は、 <code><technique></code> の <code>profile</code> 属性として指定された特定のプロファイルのためのソース情報を指定します。主見出し項目を参照してください。	なし	0 以上

`<source>` / `<technique_common>`の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	1

詳細

例

以下は、1つのRGBカラーを構成する浮動小数点値の配列を含んだ`<source>`要素の例です。

```
<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>

```

spline

カテゴリ: ジオメトリ

概要

複数セグメントから構成されるスプラインを記述します。制御点(CV)とセグメント情報が含まれます。

コンセプト

<spline>の構成は<mesh>と非常によく似ています。<spline>には、属性を提供する <source> 要素と属性ストリームを組み立てる <control_vertices> 要素が含まれています。各セグメントに関する情報は、その直前の制御頂点に関する情報と一緒に格納されます。

属性

<spline>要素には、以下の属性があります。

closed	xs:boolean	最初と最後の制御点を結び付けているセグメントがあるかどうかを表します。デフォルト値は「false」で、開いたスプラインであることを意味します。オプション。
--------	------------	---

関連要素

<spline>要素は、以下の要素と関連性があります。

親要素	geometry
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<source>	スプラインの制御点とセグメントの値を提供します。主見出し項目を参照してください。	なし	1 以上
<control_vertices>	スプラインの制御点を表します。主見出し項目を参照してください。	なし	1
<extra>	主見出し項目を参照してください。	なし	0 以上

詳細

詳しくは、以下を参照してください。

- <control_vertices>
- 4章「プログラミングガイド」の「曲線補間」

例

以下は、指定可能な属性を持った空の<spline>要素の例です。

```
<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>
```

spot

カテゴリ: ライティング

概要

スポットライトソースを表します。

コンセプト

スポット光源は、空間中の既知の位置から、特定の方向に向かって円錐形に光を放射します。放射角が光源の方向から広がるにつれて、光の輝度は減衰します。また、スポット光源の輝度は、光源からの距離が増加するにつれて減衰します。

ローカル座標での光源のデフォルトの方向ベクトルは[0,0,-1]で、負のZ軸方向に向いています。実際の放射方向は、光源がインスタンス化されるノードの座標変換によって定義されます。

属性

<spot>要素には属性はありません。

関連要素

<spot>要素は、以下の要素と関連性があります。

親要素	light / technique_common
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<color>	光源のカラーを表す 3つの浮動小数点数が含まれます。主見出し項目を参照してください。	なし	1
<constant_attenuation sid="...">	sid 属性はオプションです。	1.0	0または1
<linear_attenuation sid="...">	sid 属性はオプションです。	0.0	0または1

名前/例	解説	デフォルト値	出現回数
<code><quadratic_attenuation sid="..."></code>	sid 属性はオプションです。	0.0	0 または 1
<code><falloff_angle sid="..."></code>	sid 属性はオプションです。	180.0	0 または 1
<code><falloff_exponent sid="..."></code>	sid 属性はオプションです。	0.0	0 または 1

詳細

特定の距離における総合的な減衰は、`<constant_attenuation>`、`<linear_attenuation>`、`<quadratic_attenuation>`を利用して計算されます。使われる式は、次の通りです。

$$A = \text{constant_attenuation} + (\text{Dist} * \text{linear_attenuation}) + ((\text{Dist}^2) * \text{quadratic_attenuation})$$

光の方向を基準として減衰量を指定するには、`<falloff_angle>`と`<falloff_exponent>`を利用します。

例

以下は、`<spot>`要素の例です。

```
<light id="blue">
  <technique_common>
    <spot>
      <color>0.1 0.1 0.5</color>
      <linear_attenuation>0.3</linear_attenuation>
    </spot>
  </technique_common>
</light>
```

targets

カテゴリ: コントローラ

概要

モーフィングターゲットと、その重み付け、さらに関連したユーザ定義属性を宣言します。

コンセプト

`<targets>`要素では、モーフィングターゲットとモーフィングの重み付けを宣言します。`<input>`要素で、ブレンドするメッシュセットを定義し、さらにブレンド時に利用される重み付けの配列も定義します。モーフィングターゲットに関連付ける詳細な情報を指定するのにも利用できます。

属性

`<targets>`要素には属性はありません。

関連要素

`<targets>`要素は、以下の要素と関連性があります。

親要素	<code>morph</code>
子要素	下記サブセクションを参照してください。

その他	なし
-----	----

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (unshared)	1つは <code>semantic="MORPH_TARGET"</code> とともに、もう1つは <code>semantic="MORPH_WEIGHT"</code> とともに記述しなければなりません。主見出し項目を参照してください。	なし	2 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<targets>`要素の完全な例です。

```
<targets>
  <input source="#morph-targets" semantic="MORPH_TARGET">
  <input source="#morph-weights" semantic="MORPH_WEIGHT">
</targets>
```

technique

(core)

カテゴリ: **拡張性**

概要

コンテンツの一部の処理するために使われる情報を宣言します。

`<profile_*>`要素中の`<technique>`については、「`<technique>` (FX)」を参照してください。

コンセプト

テクニックとは、特定のプラットフォームやプログラムで必要となる情報を記述するためのものです。プラットフォームやプログラムは、`profile` 属性を使って指定されます。各テクニックは、対応するプロファイルに適合させておく必要があります。テクニックのコンテキストを定めるのは、そのインスタンス文書のプロファイル、および親要素の 2 つです。

通常、テクニックは「スイッチ」として機能します。コンテンツ中の特定の部分に対して1つ以上のテクニックが指定されている場合、インポート時に、普通は両方ではなく、どちらか一方が選ばれます。使用されるプロファイルの選択は、インポートを行うアプリケーションのサポートしているプロファイルを基準にして選ばれます。

テクニックには、アプリケーション・データとプログラムが含まれており、1つの単位として管理できるアセットとなっています。

属性

`<technique>`要素には、以下の属性があります。

<code>profile</code>	<code>xs:NMTOKEN</code>	プロファイルの種類。ベンダ定義された文字列で、プラットフォームまたは <code>technique</code> の機能ターゲットを表します。必須。
<code>xmlns</code>	<code>xs:anyURI</code>	この XML Schema 名前空間の属性は、このインスタンス文書のコンテンツを検証するために使う、追加のスキーマを特定します。オプション。

関連要素

`<technique>`要素は、以下の要素と関連性があります。

親要素	<code>extra</code> 、 <code>source(core)</code> 、 <code>light</code> 、 <code>optics</code> 、 <code>imager</code> 、 <code>force_field</code> 、 <code>physics_material</code> 、 <code>physics_scene</code> 、 <code>rigid_body</code> 、 <code>rigid_constraint</code> 、 <code>instance_rigid_body</code> 、 <code>bind_material</code> 、 <code>motion</code> 、 <code>kinematics</code> 、 <code>kinematics_model</code>
子要素	「詳細」を参照してください。
その他	なし

詳細

`<technique>` 要素には、任意の整形式の XML データを含めることができます。整形式のデータは、すべて COLLADA スキーマに対して検証されます。データ検証に別のスキーマを利用するように指定することも可能です。それ以外も正しいものとみなされますが、実際に検証することはできません。

例

以下は、1つの`<technique>`で行える別々の処理を示した例です。

```
<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>
```

以下のサンプルは、「OTHER」という名前のプラットフォーム（プロファイル）用、およびそれ以外の全プラットフォーム（`<technique_common>`の情報）用の、ほとんど同一の操作を示しています。

```
<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>
```


technique_common

カテゴリ: 拡張性

概要

あらゆる COLLADA 実装がサポートしなければならない共通プロファイルの特定の要素の情報を指定します。

コンセプト

そのアプリケーション固有のテクニックが COLLADA 文書の中に存在しない場合に、消費側のアプリケーションが利用できるテクニック情報を指定します。

逆に言えば、要素の中に特定のプロファイル用の<technique>子要素が存在する場合には、COLLADA 文書を読み込むアプリケーションは、そのアプリケーションに最も適したテクニックを使うべきです。指定された<technique>の中に該当するものが存在せず、なおかつ、<technique_common>要素が指定されている場合には、アプリケーションはその<technique_common>要素を代わりに利用する必要があります。

<technique_common> 要素の属性や子要素は、親要素によって異なります。詳しくは、各親要素を参照してください。

属性

詳しくは、親要素の主見出し項目を参照してください。

関連要素

<technique_common>要素は、以下の要素と関連性があります。

親要素	bind_material 、 instance_rigid_body 、 light 、 optics 、 physics_material 、 physics_scene 、 rigid_body 、 rigid_constraint 、 source (core)、 motion 、 kinematics 、 kinematics_model
子要素	詳しくは、親要素の主見出し項目を参照してください。
その他	technique

備考

共通プロファイルおよびカスタマイズされたプロファイルについての詳細は、「共通プロファイル」セクションを参照してください。

例

親要素を参照してください。

translate

カテゴリ: 変換

概要

物体のローカル座標系における位置を変更します。

コンセプト

この要素には、x 軸、y 軸、z 軸の各方向への距離を表す数学的なベクトルが含まれます。

コンピュータグラフィックスでは、座標系を基準として位置を決めたり、値を移動したりするために、平行移動の座標変換を利用します。

属性

<translate>要素には、以下の属性があります。

sid	sid_type	
		この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。

関連要素

<translate>要素は、以下の要素と関連性があります。

親要素	コア要素: node , フィジックス要素: shape 、 technique_common / mass_frame (rigid_body 内) 、 instance_rigid_body 、 ref_attachment 、 Attachment キネマティックス要素: frame_object 、 frame_origin 、 frame_tcp 、 frame_tip 、 link
子要素	なし
その他	なし

詳細

<translate>要素には、3つの浮動小数点値のリストが含まれています。これらの値は、行列合成に適した列ベクトルの形式に構成されます。

transformation 要素の適用方法の詳細については、<node>を参照してください。

例

以下は、X 軸方向に 10 単位の移動を表す<translate>要素の例です。

```
<translate>
  10.0 0.0 0.0
</translate>
```

triangles

カテゴリ: ジオメトリ

概要

メッシュが頂点属性をバインドして頂点を個々の三角形に編成するために必要な情報を提供します。

コンセプト

`<triangles>`要素は、`<mesh>`要素のジオメトリ・プリミティブと頂点属性の結びつきを宣言します。

頂点配列の情報は、属性配列として別個に供給され、その後で`<triangles>`要素によってインデックス参照されます。

メッシュによって記述される各三角形には、頂点が3つずつあります。1番目の三角形は、1番目、2番目、3番目の頂点から形成されます。2番目の三角形は4番目、5番目、6番目の頂点から形成され、以下同様に続きます。

属性

`<triangles>`要素には、以下の属性があります。

name	xs:token	この要素の文字列の名。オプション。
count	uint_type	三角形プリミティブの数。必須。
material	xs:NCName	マテリアルのシンボルを宣言します。このシンボルはインスタンス化の際にマテリアルにバインドされます。詳しくは、 <code><instance_geometry></code> や <code><bind_material></code> を参照。オプション。指定しなかった場合、ライティングやシェーディングの結果はアプリケーションに依存します。

関連要素

`<triangles>`要素は、以下の要素と関連性があります。

親要素	mesh , convex_mesh
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (shared)	input が少なくとも1つ存在する場合には、そのうちのいずれかのinputで <code>semantic="VERTEX"</code> を指定する必要があります。主見出し項目を参照してください。	なし	0 以上
<code><p></code>	(「p」は「primitive」の頭文字です。) 任意の数の三角形の頂点属性を記述するインデックスが含まれています。このインデックスは、 <code><input></code> 要素によって参照されている親 <code><source></code> 要素を参照しています。この要素には属性はありません。詳しくは、「詳細」を参照してください。	なし	0または1
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

`<p>`要素中のインデックスは異なる力に参照されますが、参照時には各値の記述された順番が使用されます。`<p>`要素の最初のインデックスは、`offset` 属性の値として 0 が指定された入力によって参照されます。2 番目の値は `offset` 属性として 1 が指定された入力によって参照されます。三角形のそれぞれの頂点は、その頂点に対するそれぞれの入力へのインデックスによって構成されます。すべての入力が参照された後は、次の値は `offset0` の値として入力によって再度参照され、新しい頂点を開始することになります。

生成される頂点の順番は反時計回りで、これによってそれぞれの三角形のサーフェスを記述します。

プリミティブが頂点法線を持たずに構成される場合には、ライティングを可能にするために、プリミティブごとの法線をアプリケーションが生成する場合があります。

例

以下は、2 つの三角形を記述する `<triangles>` 要素の例です。この中には、位置および法線のデータを含む 2 つの `<source>` 要素があり、そのセマンティックスは `<input>` 要素によって決定されます。`<p>` 要素の値の記述された順序によって、入力値の利用される順序が決まります。

```
<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>
```

trifans

カテゴリ: ジオメトリ

概要

メッシュが頂点属性をバインドして頂点を個々の連結した三角形に編成するために必要な情報を提供します。

コンセプト

`<trifans >`要素は、`<mesh>` 要素のジオメトリ・プリミティブと頂点属性の結びつきを宣言します。

頂点配列の情報は `<mesh>` 要素の属性配列として別に提供され、`<trifans >`要素でインデックス付けされます。

メッシュによって記述される各三角形には、頂点が 3 つずつあります。1 番目の三角形は、1 番目、2 番目、3 番目の頂点から形成されます。それ以降の三角形は、現在の頂点に加えて、1 番目の頂点と、現在の頂点の直前の頂点の再利用によって構成されます。

属性

`<trifans >`要素には、以下の属性があります。

name	xs:token	この要素の文字列の名。オプション。
count	uint_type	三角形ファンプリミティブの数。必須。
material	xs:NCName	マテリアルのシンボルを宣言します。このシンボルはインスタンス化の際にマテリアルにバインドされます。詳しくは、 <code><instance_geometry></code> や <code><bind_material></code> を参照。オプション。指定しなかった場合、ライティングやシェーディングの結果はアプリケーションに依存します。

関連要素

`<trifans >`要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<code>mesh</code> , <code>convex_mesh</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (shared)	input が少なくとも1つ存在する場合には、そのうちのいずれかの input で <code>semantic="VERTEX"</code> を指定する必要があります。主見出し項目を参照してください。	なし	0 以上
<code><p></code>	(「p」は「primitive」の頭文字です。) 連結された任意の数の三角形の頂点属性を記述するインデックスが含まれています。このインデックスは、 <code><input></code> 要素によって参照されている親 <code><source></code> 要素を参照しています。この要素には属性はありません。詳しくは、「詳細」を参照してください。	なし	0 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

`<trifans >`要素には、`<p>`要素の並びが含まれます。

`<p>`要素中の各値は入力に参照されますが、参照時には各値の記述された順番が使用されます。`<p>`要素の最初の値は `offset` 属性の値として0が指定された入力によって参照されます。2番目の値は `offset` 属性として1が指定された入力によって参照されます。三角形の頂点は、その頂点に対する入力の値によって構成されます。すべての入力が参照された後は、次の値は `offset0` の値として入力によって再度参照され、新しい頂点を開始することになります。

生成される頂点の順番は反時計回りで、これによってそれぞれの三角形のサーフェスを記述します。

プリミティブが頂点法線を持たずに構成される場合には、ライティングを可能にするために、プリミティブごとの法線をアプリケーションが生成する場合もあります。

例

以下は、2つの三角形を記述する<trifans>要素の例です。この中には、位置および法線のデータを含む2つの<source>要素があり、そのセマンティクスは<input>要素によって決定されます。<p>要素の値の記述された順序によって、入力値の利用される順序が決まります。

```
<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>
```

tristrips

カテゴリ: ジオメトリ

概要

メッシュが頂点属性をバインドして頂点を個々の連結した三角形に編成するために必要な情報を提供します。

コンセプト

<tristrips>要素は、<mesh>要素のジオメトリ・プリミティブと頂点属性の結びつきを宣言します。

頂点配列の情報は<mesh>要素の属性配列として別に提供され、<tristrips>要素でインデックス付けされます。

メッシュによって記述される各三角形には、頂点が3つずつあります。1番目の三角形は、1番目、2番目、3番目の頂点から形成されます。以降の三角形は、現在の頂点、および直前の2つの頂点を再利用することにより形成されます。

属性

<tristrips>要素には、以下の属性があります。

name	xs:token	この要素の文字列の名。オプション。
count	uint_type	三角形ストリッププリミティブの数。必須。
material	xs:NCName	マテリアルのシンボルを宣言します。このシンボルはインスタンス化の際にマテリアルにバインドされます。詳しくは、<instance_geometry>や<bind_material>を参照。オプション。指定しなかった場合、ライティングやシェーディングの結果はアプリケーションに依存します。

関連要素

<tristrips>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	mesh, convex_mesh

子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (shared)	input が少なくとも 1 つ存在する場合には、そのうちのいずれかの input で <code>semantic="VERTEX"</code> を指定する必要があります。主見出し項目を参照してください。	なし	0 以上
<code><p></code>	(「p」は「primitive」の頭文字です。) 連結された任意の数の三角形の頂点属性を記述するインデックスが含まれています。このインデックスは、 <code><input></code> 要素によって参照されている親 <code><source></code> 要素を参照しています。この要素には属性はありません。詳しくは、「詳細」を参照してください。	なし	0 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

`<tristrips>` 要素には、`<p>` 要素の並びが含まれます。

`<p>` 要素中の各値は入力に参照されますが、参照時には各値の記述された順番が使用されます。`<p>` 要素の最初の値は `offset` 属性の値として 0 が指定された入力によって参照されます。2 番目の値は `offset` 属性として 1 が指定された入力によって参照されます。三角形の頂点は、その頂点に対する入力の値によって構成されます。すべての入力が参照された後は、次の値は `offset0` の値として入力によって再度参照され、新しい頂点を開始することになります。

生成される頂点の順番は、最初 (および 3 番目、5 番目...) の三角形では反時計回りで、2 番目 (および 4 番目、6 番目...) の三角形では時計回りとなり、これによってそれぞれの三角形のサーフェスを記述します。

プリミティブが頂点法線を持たずに構成される場合には、ライティングを可能にするために、プリミティブごとの法線をアプリケーションが生成する場合があります。

例

以下は、2 つの三角形を記述する `<tristrips>` 要素の例です。この中には、位置および法線のデータを含む 2 つの `<source>` 要素があり、そのセマンティックスは `<input>` 要素によって決定されます。`<p>` 要素の値の記述された順序によって、入力値の利用される順序が決まります。

```
<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>
```

vertex_weights

カテゴリ: コントローラ

概要

スキニング処理で利用される関節と重み付けの組み合わせを記述します。

コンセプト

`<vertex_weights>`要素は、ベースメッシュ中の各頂点に対して、それぞれの関節と重み付けを関連付けます。

属性

`<vertex_weights>`要素には、以下の属性があります。

count	uint_type	ベースメッシュ中の頂点の数。必須。
-------	-----------	-------------------

関連要素

`<vertex_weights>`要素は、以下の要素と関連性があります。

親要素	skin
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (shared)	<code><input></code> 要素の1つは、 <code><vertex_weights></code> の子として、 <code>semantic="JOINT"</code> を指定する必要があります。 <code><input></code> 要素は、関連付ける関節と属性を表します。主見出し項目を参照してください。	なし	2 以上
<code><vcount></code>	<code><vertex_weights></code> によって定義された特定のインフルエンسに関連付けられたボーンの数指定する、整数のリストが含まれます。この要素には属性はありません。	なし	0 または 1
<code><v></code>	各頂点とボーンや属性との関連付けを記述するインデックスのリストが含まれます。関節の配列への-1というインデックスは、バインド形状を参照します。重み付けは、利用する前に正規化するべきです。この要素には属性はありません。	なし	0 または 1
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

例

以下は、空の`<vertex_weights>`要素の例です。

```
<skin>
  <vertex_weights count="">
    <input semantic="JOINT"/>
  </vertex_weights>
```



```

    </v/>
  </extra/>
</vertex_weights>
</skin>

```

以下は、より具体的な<vertex_weights>要素の例です。<vcount>要素で、最初の頂点にはボーンが3つあり、2番目の頂点には2つあるといったことを指定している点に注意してください。また、<v>要素では、最初の頂点は、バインド形状に対してweights[0]で重み付けを行い、同様にweights[1]はボーン0に対して、weights[2]はボーン1に対して重み付けを行うことも指定しています。

```

<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>

```

vertices

カテゴリ: ジオメトリ

概要

メッシュ頂点の属性や識別情報を宣言します

コンセプト

<vertices>要素は、メッシュ中のメッシュ頂点を記述します。メッシュ頂点は、メッシュを構成している頂点の識別情報、およびテッセレーションに対して不変な他の頂点属性を表します。

属性

<vertices>要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。必須。
name	xs:token	この要素の文字列の名。オプション。

関連要素

<vertices>要素は、以下の要素と関連性があります。

親要素	mesh, convex_mesh, brep
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><input></code> (unshared)	1つの <code><input></code> 要素は、メッシュ中の各頂点のトポロジ的な位置を確立するために、 <code>semantic="POSITION"</code> を指定する必要があります。主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

`<brep>`要素の中の`<vertices>`要素は、辺、もしくは、辺のない単一の点の境界を指定します。B-repの場合、色、テクスチャ、その他の追加データは評価されません。

例

以下は、メッシュ頂点を表す`<vertices>`要素の例です。

```
<mesh>
  <source id="position"/>
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
</mesh>
```

visual_scene

カテゴリ: シーン

概要

COLLADA リソースのコンテンツで視覚化可能な情報セット全体を表します。

コンセプト

`visual_scene` の階層構造は、シーングラフにまとめられます。シーングラフは、視覚情報および関連データをノードとして含む有向非巡回グラフ (DAG)、つまりツリー構造のデータです。シーングラフ構造は、データ処理の最適化やレンダリングに役立つため、コンピュータグラフィックスでは幅広く利用されています。

属性

`<visual_scene>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<visual_scene>`要素は、以下の要素と関連性があります。

親要素	<code>library_visual_scenes</code>
-----	------------------------------------

子要素	下記サブセクションを参照してください。
その他	instance_visual_scene

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code><node></code>	主見出し項目を参照してください。	なし	1 以上
<code><evaluate_scene></code>	<code><evaluate_scene></code> 要素は、この <code>visual_scene</code> を評価する方法を指定する情報を宣言します。主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

詳細

`<visual_scene>`要素は、シーングラフのトポロジーのルートを構成することになります。

1つの`<library_visual_scenes>`要素の中に`<visual_scene>`要素を複数記述してもかまいません。`<scene>`要素の中の`<instance_visual_scene>`要素—`<COLLADA>`ルート要素の下に宣言されているものは、どの`<visual_scene>`要素をドキュメントで利用するのかを宣言します。

例

以下は、子要素を持たない`<visual_scene>`要素を含んだ COLLADA リソースの簡単な例です。シーンの名前は「world」です。

```
<?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>
  <scene>
    <instance_visual_scene url="#world"/>
  </scene>
</COLLADA>
```

COLLADA では、レイヤや可視性をサポートしています。各ノードには、`xs:NCName` のリストである `layer` 属性があります。ノードは、この属性のリストに含まれる各レイヤに属しています。そして、ビジュアルシーンの中には、シーンのレンダリング方法を記述する`<evaluate_scene>`が存在します。ここは、フルスクリーンのエフェクトが使われる場所でもあります。

以下の文書の断片は、この仕組みを示しています。このソリューションではレイヤを利用しています。これは、「可視性」のソリューションとしては理想ではないかもしれませんが、同じ結果を実現することができます。

```
<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>
```

6章 フィジックスリファレンス

概要

このセクションでは、COLLADA フィジックスを構成する要素を扱います。

要素のカテゴリ分類

この章では、要素をアルファベット順に記載します。関連する要素を見つけやすくするため、以下の表では、要素をカテゴリ別に記載しています。

解析形状

box	ローカル座標の原点を中心とし、軸方向を向いたボックスを宣言します。
capsule	ローカル座標の原点を中心として y 軸方向を向いたカプセルを宣言します。
convex_mesh	基本的なジオメトリメッシュを記述する情報を含む、もしくは、参照しています。
cylinder	ローカル座標の原点を中心として y 軸方向を向いた円柱形のカプセルを宣言します。
plane	無限平面のプリミティブを宣言します。
shape	< rigid_body >のコンポーネントを記述します。
sphere	ローカル座標の原点を中心とする球プリミティブを宣言します。

物理マテリアル

instance_physics_material	定義済みの< physics_material >要素を使って、形状の表面プロパティを指定します。
library_physics_materials	< physics_material >要素を保存するライブラリを提供します。
physics_material	パラメータ付きのテクニク/プロファイルを使って、オブジェクトの物理的なプロパティを定義します。

物理モデル

Attachment	rigid_body または node へのアタッチメントを定義します。
ref_attachment	剛体やノードに、基準座標系として使うアタッチメントを定義します。
instance_physics_model	物理モデルを別の物理モデルの中に埋め込んだり、物理モデルを物理シーンの中でインスタンス化したりします。
instance_rigid_body	< rigid_body >< instance_physics_model >を使ってインスタンス化された< physics_model >の特定の< rigid_body >に対して、インタフェースを提供します。
instance_rigid_constraint	束縛のある物理モデルをインスタンス化することによって生成された束縛に対して、インタフェースを提供します。
library_physics_models	< physics_model >要素を保存するライブラリを提供します。
physics_model	何度もインスタンス化される剛体と制約の複雑な組み合わせを構築することができます。
rigid_body	形状を崩さないシミュレーションした物体を記述します。
rigid_constraint	< rigid_body >のようなコンポーネントを、可動パーツを持った複雑なフィジックスモデルに結び付けます。

物理シーン

<code>force_field</code>	force-fields に使用する汎用的なコンテナを提供します。
<code>instance_force_field</code>	<force_field>のインスタンス化を宣言します。
<code>instance_physics_scene</code>	<physics_scene>のインスタンス化を宣言します。
<code>library_force_fields</code>	<force_field>要素を保存するライブラリを提供します。
<code>library_physics_scenes</code>	<physics_scene>要素を保存するライブラリを提供します。
<code>physics_scene</code>	フィジックスオブジェクトがインスタンス化/シミュレーション化される環境を指定するためのものです。

概要

3D アプリケーションでは、仮想世界の物理シミュレーションに使われるデータやプロセスが、レンダリングに使われるものとは異なっていることが珍しくありません。COLLADA フィジックスを使うと、コンテンツ製作者は、このような物理シーンを記述することができます。

COLLADA フィジックスは、現在のところ、基本的な剛体力学をサポートしています。剛体は、ニュートンの物理学の基本法則にしたがって他の剛体と相互作用する、形状（ジオメトリ）と質量特性をもつ変形不可能な物体です。物理ベースのシステムには、通常、関節式アニメーションモデルで使われるような階層構造や親子関係の概念はありません。その代わりに、剛体は、束縛によって、他の剛体やワールドと連動させることができます。

束縛は、特定の物体が動く際の、他の物体との相対関係を指定することができます。たとえば、車輪をシャーシ（筐体）に束縛することができます。そうすると、車輪はx軸にのみ沿って回転し、他の軸に沿って移動することも回転することもありません。一般に、剛体の束縛には、さまざまな角度自由度や線形自由度を制限できる、さまざまなパラメータがあります。関連する剛体と束縛は、さらに、物理モデルとして論理的にグループ化されて、車やキャラクタの持つラグドールのような複雑な物理システムを定義します。最後に、物理モデルのライブラリの中で定義された物理モデルは、ビジュアルジオメトリがビジュアルシーンの中でインスタンス化されるのと同じように、物理シーンの中でインスタンス化されます。

物理シーン中でシミュレートされたモデルは、インスタンス化されたモデル中の剛体が、ビジュアルシーン中のノードの配置を直接制御することによって、視覚化されます。このようなノードは、物理シーンで使用されているものと異なるジオメトリを表示することもできるし、このノードを、スキニングされたメッシュをレンダリングするために使われるボーンにすることもさえ可能です。剛体は動力的である、つまり、その挙動を完全に物理シミュレーションによって決定することができます。また、剛体は運動力学的である、つまり、その位置や方向をアニメーションによって制御することもできます。その場合でも、シミュレーション中の他の動力的物体には影響を与えることができます。また、束縛された2つの物体の間の望ましい位置や方向をアニメーションデータの対象にすることにより、アニメーションから物理シミュレーションに間接的な影響を与えることもできます。

物理的な単位について

COLLADA では、特定の長さの目盛を強制することはありません。データは、インチ、フィート、メートル、マイルなどの単位で格納することができます。このことは、フィジックスでも同じように当てはまります。従って、速度、力、質量特性などの尺度は、そのファイルで指定された単位によって決まります。たとえば、距離と長さをメートル、質量をキログラム、時間を秒単位で指定した場合、力の単位はニュートンになります。距離がインチの場合、速度はインチ毎秒です。密度は、1立方単位当たりの質量の単位として指定されます。たとえば、ポンドとフィートを使った場合、密度は、物質の立方フィート当たりのポンド数になるということです。さまざまな尺度が利用できることの落とし穴の1つは、距離の単位がメートルで質量の単位がキログラムのメートル法を使うと、密度の単位が、メートル法標準の

密度定義であるキログラム毎立方デシメートル(リットル)ではなく、キログラム毎立方メートルになってしまいます。

必要であれば、COLLADA ドキュメントの「ベース」から単位を取り出すことができます。COLLADA では、以下の測定単位を使用します。

測定	単位
Time	秒 (標準単位)
角度	束縛/ジョイント制限に使われる度数 (標準単位)
質量	キログラム (標準単位)
距離	メートル (デフォルトの単位)。<asset>要素は<unit>要素を含んでおり、この要素によって、距離の大きさを対応するアセットに対して再定義することができます。

ジオメトリの種類

物理シミュレーションでは、剛体のジオメトリを記述するために、通常、ある種のメッシュ表現を使います。COLLADA には、視覚的なデータのジオメトリを記述する方法がすでに存在しているので、物理表現でも同じシステムを使います。実際、剛体に使われるメッシュは、ビジュアルノードからも参照できます。物理シミュレーションに必要なのは、基本的な頂点の位置と三角形の情報だけなので、COLLADA のメッシュ用のスキーマは、物理的な用途からは複雑に見えるかもしれません。

物理エンジンが衝突を正しく処理するためには、メッシュが凸である必要があります。メッシュが凸である、あるいは、指定されたメッシュに対して凸包を生成する必要があることを示すために、COLLADA では、明示的な要素 (<convex_mesh>) を提供しています。

物理エンジンでは、コリジョンボリュームとして、一般的なメッシュや凸包に加えて、解析的な形状(ボックス、球、カプセル)をよく使います。解析的な形状の利用は、物理シミュレーションにおいて、特定の曲面表現を改善したり、パフォーマンスを向上したり、メモリー使用量を削減したりするのに役立ちます。したがって、COLLADA では、物理シミュレーションでの利用を想定して、<box>、<sphere>、<capsule>、<cylinder>などのプリミティブを少しだけ追加しています。

これらのプリミティブは、ジオメトリのサイズを指定するために、半径、高さ、範囲などの子要素を持っています。これらのプリミティブは、どれも原点を中心とし、軸方向を向いています。ジオメトリ要素は、その要素の剛体の中での位置や方向を指定する形状要素の子要素です。また、形状要素も、表面特性を指定するための子要素を持っています。剛体は、以下の形状を1つ以上含んでいます。

- <box>
- <capsule>
- <convex_mesh>
- <cylinder>
- <plane>
- <sphere>

Attachment

カテゴリ: 物理モデル

概要

剛体束縛の中の剛体やノードに、アタッチされた座標系を定義します。

コンセプト

`<rigid_constraint>`は、2つの剛体を互いに連結(して両者の間の運動を制限)します。`<Attachment>`は、2番目の剛体、`<ref_attachment>`は1番目の剛体を参照します。たとえば、ドアと壁の間の蝶番束縛の場合、一方が参照アタッチメント(この場合には壁)、もう一方がアタッチメント(ドア)です。

また、`<Attachment>`は、その連結先のためのローカルな座標系を、`<translate>`および`<rotate>`要素を利用して、剛体(ノード)との相対位置として定義します。たとえば、蝶番(剛体制約)は、ドア(剛体)のローカル座標の原点からの相対位置として、ドアの縁の中央に連結されます。

属性

`<Attachment>`要素には、以下の属性があります。

<code>rigid_body</code>	<code>xs:anyURI</code>	<code><rigid_body></code> や <code><node></code> を参照するスコープ付き識別子。この属性は、 <code><ref_attachment></code> もしくは <code><Attachment></code> のどちらかの <code><rigid_body></code> のSIDを参照する必要があります。両方が <code><node></code> であってははいけません。必須。
-------------------------	------------------------	--

関連要素

`<Attachment>`要素は、以下の要素と関連性があります。

親要素	<code>rigid_constraint</code>
子要素	下記サブセクションを参照してください。
その他	<code>ref_attachment</code>

子要素

子要素が存在する場合には、任意の順で記述できます。

名前/例	解説	デフォルト値	出現回数
<code><translate></code>	接続点の位置を変更します。コアの主見出し項目を参照してください。	なし	0 以上
<code><rotate></code>	接続点の位置を変更します。コアの主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

```
<attachment rigid_body="./SomeRigidBody">
  <translate/>
  <rotate/>
  <extra/>
</attachment>
```

より完全な例は、`<rigid_constraint>`を参照してください。

box

カテゴリ: 解析形状

概要

ローカル座標の原点を中心とし、軸方向を向いたボックスを宣言します。

コンセプト

ボックスは、COLLADA フィジックスのジオメトリプリミティブの1つで、同等の8つの頂点を持つメッシュを使うよりも、効率的な衝突検出を可能にします。詳しくは、この章の冒頭の「ジオメトリの種類」のセクションを参照してください。

属性

<box>要素には属性はありません。

関連要素

<box>要素は、以下の要素と関連性があります。

親要素	shape
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<half_extents>	ボックスの範囲を表す3つの浮動小数点値が含まれます。ボックスの寸法は、この値（範囲の半分）の2倍になります。この要素には属性はありません。	なし	1
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

例

5x2x2 のボックスは、次のように表されます。

```

<shape>
  <box>
    <half_extents> 2.5 1.0 1.0 </half_extents>
  </box>
</shape>

```

capsule

カテゴリ: 解析形状

概要

ローカル座標の原点を中心として y 軸方向を向いたカプセルプリミティブを宣言します。

コンセプト

カプセルは、フィジックス用に特別に追加されたジオメトリプリミティブで、一般に衝突検出のために使われます。詳しくは、この章の冒頭の「ジオメトリの種類」のセクションを参照してください。

カプセルは丸い蓋のついた円柱です。厳密には、2つの球によって生成される凸包、もしくは球と線分のミンコフスキー和（直線上をスイープした球）として記述することができます。カプセルは球形のものが最も一般的ですが、COLLADA では、両端が楕円体のカプセルも可能なように一般化されています。

属性

`<capsule>`要素には属性はありません。

関連要素

`<capsule>`要素は、以下の要素と関連性があります。

親要素	<code>shape</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><height></code>	ふたになっている半球（楕円体）の中心同士をつなぐ線分の長さを表す浮動小数点値が含まれます。この要素には属性はありません。	なし	1
<code><radius></code>	カプセル（楕円体のこともある）の x 、 y 、 z 半径を表す3つの浮動小数点値が含まれます。この要素には属性はありません。	なし	1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

```
<capsule>
  <height> 4.0 </height>
  <radius> 1.0 2.0 3.0 </radius>
</capsule>
```

球形のカプセルは、3つの半径をすべて等しく設定することによって作成できます。

```
<capsule>
  <height> 2.0 </height>
  <radius> 1.0 1.0 1.0 </radius>
</capsule>
```

convex_mesh

カテゴリ: 解析形状

概要

基本的なジオメトリックメッシュを記述するのに十分な情報を含むか、または参照します。

コンセプト

<convex_mesh>の定義は<mesh>と同じですが、<source>、<vertices>、<polygons>などのような完全な記述ではなく、別の<geometry>要素を単に指し示すことで形状を構成している点で異なります。別の<geometry>要素を使用する場合には、それらの凸包を計算しなければならず、このような凸包はオプションの convex_hull_of 属性で示されます。

<convex_mesh>を使用すると、レンダリングに利用される<mesh>をフィジックス用途で再利用できるため、ドキュメントのサイズを最小限に抑えたり、オリジナルの<mesh>へのリンクを維持できる、といった利点があります。

<convex_mesh>要素の記述を最小限に抑える方法は、頂点を (<vertices>要素と対応するソースを用いて) 指定し、その頂点群 (ポイントクラウド) の凸包をインポータに計算させることです。

属性

<convex_mesh>要素には、以下の属性があります。

convex_hull_of	xs:anyURI	<geometry>の URI 文字列。指定された場合、指定されたメッシュの凸包を計算します。この場合、アプリケーションは、<source>、<vertices> が指定されたとしても、無視する必要があります。オプション。
----------------	-----------	---

関連要素

<convex_mesh>要素は、以下の要素と関連性があります。

親要素	geometry
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素は不要です。ただし、子要素が含まれる場合には、以下の順序と指定された回数で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<source>	メッシュの頂点データの集まりを提供します。convex_hull_of が指定されていない場合に必要です。コアの主見出し項目を参照してください。	なし	1 以上
<vertices>	メッシュの頂点属性を記述して、トポロジ的な位置を確立します。convex_hull_of が指定されていない場合に必要です。コアの主見出し項目を参照してください。	なし	1
primitive_elements	プリミティブ要素としては、以下を任意に組み合わせることができます。		
	<lines>	直線プリミティブを含みます。コアの主見出し項目を参照してください。	0 以上

名前/例	解説	デフォルト値	出現回数
	<code><linestrips></code> ラインストリッププリミティブを含みます。コアの主見出し項目を参照してください。	なし	0 以上
	<code><polygons></code> 穴を持つことができるポリゴンプリミティブを含みます。コアの主見出し項目を参照してください。	なし	0 以上
	<code><polylist></code> 穴を持ってないポリゴンプリミティブを含みます。コアの主見出し項目を参照してください。	なし	0 以上
	<code><triangles></code> 三角形プリミティブを含みます。コアの主見出し項目を参照してください。	なし	0 以上
	<code><trifans ></code> 三角形ファンプリミティブを含みます。コアの主見出し項目を参照してください。	なし	0 以上
	<code><tristrips></code> 三角形ストリッププリミティブを含みます。コアの主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

`convex_hull_of` 属性を使わない場合には、有効な`<convex_mesh>`を定義するために、子要素`<source>`および`<vertices>`を指定します。

例

```
<geometry id="myConvexMesh">
  <convex_mesh>
    <source>...</source>
    <vertices>...</vertices>
    <polygons>...</polygons>
  </convex_mesh>
</geometry>
```

または、

```
<geometry id="myArbitraryMesh">
  <mesh>
    ...
  </mesh>
</geometry>
<geometry id="myConvexMesh">
  <convex_mesh convex_hull_of="#myArbitraryMesh"/>
</geometry>
```

cylinder

カテゴリ: 解析形状

概要

ローカル座標の原点を中心としてローカルな y 軸方向を向いた円柱形のプリミティブを宣言します。

注: この要素が `<surface>` の中で使われる場合については、9 章「<B-Rep リファレンス> (B-rep)」の `cylinder` を参照してください。

コンセプト

フィジックスのコリジョン形状としては、ジオメトリプリミティブ (解析的な形状とも呼ばれる) が最も便利です。詳しくは、この章の冒頭の「ジオメトリの種類」のセクションを参照してください。

属性

`<cylinder>` 要素には属性はありません。

関連要素

`<cylinder>` 要素は、以下の要素と関連性があります。

親要素	<code>shape</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><height></code>	円柱の y 軸方向の長さを表す浮動小数点が含まれます。この要素には属性はありません。	なし	1
<code><radius></code>	円柱 (楕円形も可能) の半径を表す 2 つの浮動小数点値が含まれます。この要素には属性はありません。	なし	1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

```
<cylinder>
  <height> 2.0 </height>
  <radius> 1.0 1.0 </radius>
</cylinder>
```

force_field

カテゴリ: 物理シーン

概要

force-fields に使用する汎用的なコンテナを提供します。

コンセプト

force_field は剛体などの物理オブジェクトに対して影響を及ぼします。physics_scene もしくは physics_model のインスタンスの下でインスタンス化されます。

属性

<force_field>要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	オプション。

関連要素

<force_field>要素は、以下の要素と関連性があります。

親要素	library_force_fields
子要素	下記サブセクションを参照してください。
その他	instance_force_field

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<asset>	コアの主見出し項目を参照してください。	なし	0 または 1
<technique> (core)	コアの主見出し項目を参照してください。	なし	1 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

現時点では、<force_field>用の共通プロファイル/テクニックはありません。<technique>要素には、整形形式の任意の XML データを含めることができます。

例

```
<library_force_fields>
  <force_field>
    <technique profile="SomePhysicsProfile">
      <program url="#SomeWayToDescribeAForceField">
        <param> ... </param>
      </program>
    </technique>
  </force_field>
</library_force_fields>
```

instance_force_field

カテゴリ: 物理シーン

概要

`<force_field>`要素によって記述されるオブジェクトをインスタンス化します。

コンセプト

COLLADA のインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

`<instance_force_field>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、アドレス構文 3章「スキーマのコンセプト」の構文を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	インスタンス化する <code><force_field></code> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。

関連要素

`<instance_force_field>`要素は、以下の要素と関連性があります。

親要素	<code>instance_physics_model</code> , <code>physics_scene</code>
子要素	下記サブセクションを参照してください。
その他	<code>force_field</code>

子要素

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

この要素には標準仕様はありません。使い方は、アプリケーションに依存します。

例

```
<instance_force_field url="#my_force_field">
</instance_force_field>
```

instance_physics_material

カテゴリ: 物理マテリアル

概要

定義済みの<physics_material>要素を使って、形状の表面プロパティを指定します。

コンセプト

効率をよくするため、物理エンジンの中には、このようなプロパティを形状ごとに格納する代わりに、物理マテリアルのパレットを参照するものもあります。

COLLADA のインスタンス要素一般についての情報は、3 章「スキーマのコンセプト」のインスタンス化と外部参照を参照してください。

属性

<instance_physics_material>要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、アドレス構文 3 章の「スキーマのコンセプト」を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	インスタンス化する <physics_material> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。

関連要素

<instance_physics_material>要素は、以下の要素と関連性があります。

親要素	rigid_body / technique_common, instance_rigid_body / technique_common, shape
子要素	下記サブセクションを参照してください。
その他	physics_material

子要素

名前/例	解説	デフォルト値	出現回数
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

例

```
<shape>
  <sphere> <radius> 1 </radius> </sphere>
  <instance_physics_material url="#my_force_field" />
</shape>
```


instance_physics_model

カテゴリ: 物理モデル

概要

物理モデルを別の物理モデルの中に埋め込んだり、物理モデルを物理シーンの中でインスタンス化したりします。

コンセプト

この要素の目的は、あるフィジックスモデルの定義中に別のフィジックスモデルを階層的に埋め込むこと、およびフィジックスモデルをフィジックスシーン中でインスタンス化することです。どちらの場合も、含まれている剛体のパラメータと制約をオーバーライドできます。

物理モデルを物理シーンの中でインスタンス化するには、その物理モデルに含まれる剛体を、ビジュアルシーンの変換ノードとリンクすることにより、フィジックスに表示されるメッシュのアニメーションを行わせることができます。同じように、`rigid_body` が動力学的でなく運動力学的である場合、アプリケーションは、`rigid_body` を物理環境の中で運動させるために、アニメーションによって影響を受けるターゲットノードから変換情報を取得することができます。

さらに、インスタンス化されたフィジックスモデルの親属性を指定することもできます。この親は、フィジックスモデルの最初の位置と方向を決めることとなります（その剛体に関しても同様です）。親（または祖父など）は一部のアニメーションコントローラのターゲットとなることも可能で、非力学剛体のキーフレーム運動を物理シミュレーションと組み合わせることができます。

COLLADA のインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

<instance_physics_model>要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。この属性を利用することで、<animation>に <instance_physics_model>インスタンスの要素を指定できます。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	どの <physics_model> をインスタンス化するかを表します。必須。ローカルなインスタンスや外部参照を参照することができます。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント 識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。
parent	xs:anyURI	ビジュアルシーン内のノードの id を指し示します。この属性を利用することで、フィジックスモデルを特定の変換ノードの下にインスタンス化することができます。この変換ノードは、フィジックスモデルの最初の位置と方向を指定することになり、アニメートして剛体の運動に影響を与えるようにすることが可能です。オプション。 デフォルトで、フィジックスモデルは、特定の変換ノードではなく、ワールドの下にインスタンス化されます。このパラメータは、現在の<physics_model>の親要素が<physics_scene>の場合にだけ意味を持ちます。

関連要素

`<instance_physics_model>`要素は、以下の要素と関連性があります。

親要素	<code>physics_scene</code> , <code>physics_model</code>
子要素	下記サブセクションを参照してください。
その他	<code>physics_model</code>

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><instance_force_field></code>	<code><force_field></code> 要素をインスタンス化して、このフィジックスモデルに影響を及ぼします。主見出し項目を参照してください。	なし	0 以上
<code><instance_rigid_body target="#SomeNode"></code>	<code><rigid_body></code> 要素をインスタンス化して、そのプロパティの一部またはすべてのオーバーライドを可能にします。 <code>target</code> 属性は、この剛体インスタンスでオーバーライドされた変換を持つ <code><node></code> 要素を定義します。主見出し項目を参照してください。	なし	0 以上
<code><instance_rigid_constraint></code>	<code><rigid_constraint></code> 要素をインスタンス化して、プロパティの一部をオーバーライドします。 <code><rigid_constraint></code> 子要素が、対象としている <code><node></code> 要素を定義するので、この要素には <code>target</code> 属性がありません。主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

instance_physics_scene

カテゴリ: 物理シーン

概要

`<physics_scene>`要素によって記述されるオブジェクトをインスタンス化します。

コンセプト

COLLADA のインスタンス要素の詳細は、3 章 スキーマのコンセプトの「インスタンス化と外部参照」を参照してください。

属性

`<instance_physics_scene>`要素には、以下の属性があります。

<code>sid</code>	<code>sid_type</code>	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3 章「スキーマのコンセプト」のアドレス構文を参照してください。
<code>name</code>	<code>xs:token</code>	この要素の文字列の名。オプション。

url	xs:anyURI	<p>インスタンス化する <code><physics_scene></code> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。</p> <p>ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPather の短縮ポインタです。</p> <p>外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。</p>
-----	-----------	---

関連要素

`<instance_physics_scene>`要素は、以下の要素と関連性があります。

親要素	<code>scene</code>
子要素	下記サブセクションを参照してください。
その他	<code>physics_scene</code>

子要素

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

インスタンス化に関しては、`<scene>`が COLLADA ファイルの出発点です。COLLADA フィジックスシーンは、`<scene>`要素を利用して、`<instance_physics_scene>`要素の中にインスタンス化されます。

例

```

<scene>
  <instance_physics_scene url="my_physics_scene" />
</scene>

```

instance_rigid_body

カテゴリ: 物理モデル

概要

`<instance_physics_model>`を使ってインスタンス化された`<physics_model>`の特定の`<rigid_body>`に対して、インタフェースを提供します。

コンセプト

フィジックスと豊富なグラフィックスを同時に使うアプリケーションでは、剛体は、`<node>`の変換を最終的に`<scene>`の中で設定します。インスタンス化された物理モデルのスケルトンがまだ存在しない場合、`<instance_rigid_body>`要素は、特定の剛体インスタンスをビジュアルシーン中の特定のノードに接続するために便利です。

この`<instance_rigid_body>`要素は、以下の3つの目的に利用します。

- `<node>`要素へのリンクを指定するため
- 特定のインスタンス内の`<rigid_body>`のパラメータをオーバーライドするため (オプション)
- `<rigid_body>`インスタンスの初期状態 (線速度と角速度) を指定するため

COLLADA のインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

`<instance_rigid_body>`要素には、以下の属性があります。

<code>sid</code>	<code>sid_type</code>	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。この属性を利用することで、 <code>rigid_body</code> に <code><animation></code> インスタンスの要素を指定できます。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
<code>name</code>	<code>xs:token</code>	この要素の文字列の名。オプション。
<code>body</code>	<code>sidref_type</code>	インスタンス化する <code><rigid_body></code> の SID への参照。必須。
<code>target</code>	<code>xs:anyURI</code>	どの <code><node></code> がこの <code><rigid_body></code> インスタンスの影響を受けるのかを表します。必須。 ローカルなインスタンスや外部参照を参照することができます。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。

関連要素

`<instance_rigid_body>`要素は、以下の要素と関連性があります。

親要素	<code>instance_physics_model</code>
子要素	下記サブセクションを参照してください。
その他	<code>rigid_body</code>

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><technique_common></code>	すべての COLLADA 実装がサポートしなければならない共通プロファイルの剛体情報を、指定します。詳しくは、「共通プロファイル」の使い方情報のセクション、およびそれに続く子要素詳細についてのサブセクションを参照してください。	なし	1
<code><technique></code> (core)	各 <code><technique></code> は、 <code><technique></code> の <code>profile</code> 属性として指定された特定のプロファイルのための剛体情報を指定します。コアの主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	<code><instance_rigid_body></code> に情報を追加する複数表現可能なユーザ定義データ(<code><technique></code> 要素のように、ベースデータを切り替えるのとは違う操作です)。コアの主見出し項目を参照してください。	なし	0 以上

<instance_rigid_body< / <technique_common>の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<angular_velocity>	rigid_body インスタンスの初期スピンや角速度を指定する 3 次元ベクトル(3 つの浮動小数点値)が含まれます。また、このベクトルは、回転率(ラジアン毎秒)と同じ大きさを持つ回転軸としても知られます。回転の方向は、座標系の掌性(右手系か左手系か)に従います。たとえば、右手系の場合、見ている人の方を向いた回転ベクトルは、その人から見て反時計回りに回転する物体に対応しています。この要素には属性はありません。	0 0 0	0 または 1
<velocity>	rigid_body インスタンスの初期線速度を指定する 3 次元ベクトル(3 つの浮動小数点値)が含まれます。この要素には属性はありません。	0 0 0	0 または 1
<dynamic sid="..."> false</dynamic>	rigid_body が移動可能かどうかを指定する論理値が含まれます。sid 属性はオプションです。	true	0 または 1
<mass sid="..."> 0.5</mass>	rigid_body の総質量を指定する浮動小数点値が含まれます。sid 属性はオプションです。	密度 × 形状全体の体積から導き出されます。	0 または 1
<mass_frame> <translate> ... </translate> <rotate> ... </rotate> </mass_frame>	「ルート」図形のローカルの原点に対して相対的な剛体の重心と方向を定義します。これは、慣性テンソルの非対角成分(慣性の積)をすべて 0 にし、対角成分(慣性モーメント)だけを保存することができます。 <translate>子要素と<rotate>子要素の出現回数は、任意の回数が可能です。最低でもどちらか一方は存在する必要があります。コアの主見出し項目を参照してください。	identity(重心はローカルの原点に位置し、主軸はローカル軸となります)	0 または 1
<inertia sid="..."> 1 1 1 </inertia>	慣性テンソル(慣性モーメント)の対角成分を重心のローカル座標系で表した、3 つの浮動小数点数が含まれています。先の説明を参照してください。sid 属性はオプションです。	質量、形状ボリューム、重心から派生	0 または 1
<physics_material> または、 <instance_physics_material>	rigid_body のための physics_material を定義または参照します。主見出し項目を参照してください。	なし	0 または 1
<shape>	主見出し項目を参照してください。	なし	0 以上

例

```
<physics_scene id="ColladaPhysicsScene">
  <instance_physics_model sid="firstCatapultAndRockInstance"
    url="#catapultAndRockModel" parent="#catapult1">
<!--この physics_model の中で rigid_body の属性をオーバーライドする -->
```

```

<!--そして、rigid_body の初速度を指定する -->
  <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>
<!--このインスタンスは、rigid_body をそのノードに割り当てるだけである。オーバーライドはしない -->
  <instance_rigid_body body="./catapult/base" target="#baseNode">
    <technique_common/>
  </instance_rigid_body>
</instance_physics_model>
</physics_scene>

```

instance_rigid_constraint

カテゴリ: 物理モデル

概要

束縛のある物理モデルをインスタンス化することによって生成された束縛に対して、インタフェースを提供します。

コンセプト

2つの剛体間の剛体束縛には、さまざまなプロパティがあり、これを実行時に調節することにより、シミュレーションにさらなる多様性を加えることができます。たとえば、関節モデルからのアニメーションデータは、対応するジョイントに対するアタッチされたフレームのアラインメントを更新するのに利用できます。このジョイントに最大トルクがある場合、これは基本の物理ベースの基本的なキャラクターモーションシステムを提供します。

COLLADA のインスタンス要素一般についての情報は、3章「スキーマのコンセプト」のインスタンス化と外部参照を参照してください。

属性

`<instance_rigid_constraint>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりませんこの属性を利用することで、アニメーションターゲットに <code><rigid_constraint></code> インスタンスの要素を指定できます。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
name	xs:token	この要素の文字列の名。オプション。
constraint	sidref_type	インスタンス化する <code><rigid_constraint></code> の SID への参照。必須。

関連要素

`<instance_rigid_constraint>`要素は、以下の要素と関連性があります。

親要素	<code>instance_physics_model</code>
子要素	下記サブセクションを参照してください。
その他	<code>rigid_constraint</code>

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	<code><instance_rigid_constraint></code> に情報を追加する、ユーザ定義の複数表現が可能なデータ。コアの主見出し項目を参照してください。	なし	0 以上

詳細

要素が明示的に含まれている場合には、`<instance_physics_model>`は対応する`<physics_model>`を反映しています。`<instance_rigid_constraint>`によって連結された2つの剛体のインスタンスは、対応する`<rigid_constraint>`によって参照されている剛体に対応しています。

例

```
<instance_physics_model>
  <instance_rigid_constraint sid="my_joint">
    <extra> <maximum_torque> 100 </maximum_torque> </extra>
  </instance_rigid_constraint>
</instance_physics_model>
```

library_force_fields

カテゴリ: 物理シーン

概要

`<force_field>`要素を保存するライブラリを提供します。

コンセプト

アプリケーションの中には、剛体の運動に作用を与えるために力の場を利用するものもあります。

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存可能です。

属性

`<library_force_fields>`要素には、以下の属性があります。

id	xs:ID	<code><library_force_fields></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_force_fields>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><force_field></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_force_fields>`要素の例です。

```
<library_force_fields>
  <force_field>
    <technique profile="AGEIA"/>
  </force_field>
</library_force_fields>
```

library_physics_materials

カテゴリ: 物理マテリアル

概要

`<physics_material>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_physics_materials>`要素には、以下の属性があります。

id	xs:ID	<code><library_physics_materials></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_physics_materials>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><physics_material></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_physics_materials>`要素の例です。

```
<library_physics_materials>
  <physics_material id="phymat1">
    ...
  </physics_material>

  <physics_material id="phymat2">
    ...
  </physics_material>
</library_physics_materials>
```

library_physics_models

カテゴリ: 物理モデル

概要

`<physics_model>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_physics_models>`要素には、以下の属性があります。

<code>id</code>	<code>xs:ID</code>	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
<code>name</code>	<code>xs:token</code>	この要素の文字列の名。オプション。

関連要素

`<library_physics_models>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><physics_model></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_physics_models>`要素の例です。

```

<library_physics_models>
  <physics_model id="phymod1">
    ...
  </physics_model>

  <physics_model id="phymod2">
    ...
  </physics_model>
</library_physics_models>

```

library_physics_scenes

カテゴリ: 物理シーン

概要

`<physics_scene>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存可能です。

属性

`<library_physics_scenes>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_physics_scenes>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><physics_scene></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、`<library_physics_scenes>`要素の例です。

```
<library_physics_scenes>
  <physics_scene id="physce1">
    ...
  </physics_scene>

  <physics_scene id="physce2">
    ...
  </physics_scene>
</library_physics_scenes>
```

physics_material

カテゴリ: 物理マテリアル

概要

オブジェクトの物理特性を定義します。

コンセプト

この要素では、オブジェクトの物理的な性質を定義するために、パラメータ付きのテクニック/プロファイルを利用します。共通プロファイルでは、[static_friction](#) といった組み込み名を定義しています。

物理マテリアルは、形状の中にインラインで使うこともできるし、[<library_physics_materials>](#) 要素の下に格納しておいて、[<instance_physics_material>](#) を使って形状からインスタンス化することもできます。

属性

[<physics_material>](#) 要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

[<physics_material>](#) 要素は、以下の要素と関連性があります。

親要素	library_physics_materials , shape , instance_rigid_body / technique_common , rigid_body / technique_common
子要素	下記サブセクションを参照してください。
その他	instance_physics_material

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<asset>	コアの主見出し項目を参照してください。	なし	0 または 1
<technique_common>	すべての COLLADA 実装がサポートしなければならない共通プロファイルの物理マテリアル情報を指定します。下記サブセクションを参照してください。	なし	1
<technique> (core)	各 <technique> は、 <technique> の <code>profile</code> 属性として指定された特定のプロファイルのための物理マテリアル情報を指定します。コアの主見出し項目を参照してください。	なし	0 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

<physics_material> / <technique_common>の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><dynamic_friction sid="..."> 0.23 </dynamic_friction></code>	動摩擦係数を指定する浮動小数点数が含まれます。sid 属性はオプションです。	0	0 または 1
<code><restitution sid="..."> 0.2 </restitution></code>	衝突時に保存される運動エネルギーの比率を表す浮動小数点数(通常、0.0~1.0の範囲)が含まれます。「反発」とか「弾性」とも呼ばれます。sid 属性はオプションです。	0	0 または 1
<code><static_friction sid="..."> 0.23 </static_friction></code>	静摩擦係数を指定する浮動小数点数が含まれます。sid 属性はオプションです。	0	0 または 1

例

```
<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>
```

physics_model

カテゴリ: 物理モデル

概要

何度もインスタンス化される剛体と制約の複雑な組み合わせを構築することができます。

コンセプト

視覚的なシーングラフノードの階層構造には、ルートノードに基づく自然なグループ分けが存在しています。剛体には、関節による階層構造はありません。その代わりに、剛体はシミュレーションによってすべて同じレベルにあると見なされます。<physics_model>要素は、剛体と束縛の集合に対し、論理的なグループ化のメカニズムを提供します。物理モデルは、単一の剛体のような単純なものもあれば、ジョイント(束縛)によって連結されたボーン(剛体)をもつキャラクタのように複雑なものもあります。<node>は、<physics_model>とは違って、位置や方向を指定するための変換子要素がありません。

フィジックスモデルの中で定義されている各子要素は、id 属性の代わりに、sid 属性を持ちます。この sid 属性は、インスタンス化時にフィジックスモデルのコンポーネントにアクセスしてオーバーライドするのに利用されます。

`<physics_model>`をシミュレーションの中で利用するには、`<instance_physics_model>`要素を使って`<physics_scene>`の中にインスタンス化する必要があります。

物理モデルには、ノードがビジュアルシーン中の他のノードを参照したり再利用するための仕組みと同じような、他の定義済みの物理モデルを含むための仕組みがあります。たとえば、家の物理モデルは、レンガ造りの壁のように、インスタンス化された複数の物理モデル(たとえば)を含むことができます。この要素はそのようなモデルの構造を定義します。つまり`<instance_physics_model>`要素は、`<physics_model>`をインスタンス化し、`<physics_model>`のパラメータの多くをオーバーライドすることができます。`<instance_physics_model>`要素には、`<physics_model>`親要素の中における位置や方向を示す子要素があります。

属性

`<physics_model>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<physics_model>`要素は、以下の要素と関連性があります。

親要素	<code>library_physics_models</code>
子要素	下記サブセクションを参照してください。
その他	<code>instance_physics_model</code>

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><rigid_body></code>	<code><rigid_body></code> 要素を定義して、デフォルト以外のプロパティを設定します。主見出し項目を参照してください。	なし	0 以上
<code><rigid_constraint></code>	<code><rigid_constraint></code> 要素を定義して、一部またはすべてのプロパティをオーバーライドします。主見出し項目を参照してください。	なし	0 以上
<code><instance_physics_model></code>	指定された url からフィジックスモデルをインスタンス化して、他の子要素と区別するために、sid を割り当てます。主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

```
<library_physics_models>
  <!-- 他の Physics_model または physics_scene で再利用したり修正できるように、 -->
  <!-- カタパルトの physics_model を定義する。 -->
  <physics_model id="catapultModel">
    <!-- インラインに定義されるカタパルトのベース。 -->
    <rigid_body sid="base">
      <technique_common>
```

```

        <dynamic>false</dynamic>
        <instance_physics_material url="#catapultBasePhysicsMaterial"/>
    </shape>
        <instance_geometry url="#catapultBaseConvexMesh"/>
        <!-- カタパルトのモデルへの相対的なベースのローカル位置-->
        <translate> 0 -1 0 </translate>
    </shape>
    </technique_common>
</rigid_body>

    <!-- 同じように、カタパルトの上部（または腕）を定義する。 -->
    <rigid_body sid="top">
        <technique_common>
            <dynamic>true</dynamic>
        </technique_common>
    </rigid_body>
        <instance_geometry url="#catapultTopConvexMesh"/>
        <translate> 0 3 0 </translate>
    </shape>
    </technique_common>
</rigid_body>

    <!-- カタパルトの動きを制御する角スプリングを定義する。
         オプションで、剛体の制約を他の physics_model からコピーするために
         URL を指定する。 -->
    <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>

    <!-- この physics_model は、事前に定義した 2 つのモデルを組み合わせる。 -->
    <physics_model id="catapultAndRockModel">
        <!-- この石は、事前に定義されている physics_model のライブラリから取得する。 -->
        <instance_physics_model sid="rock"
            url="http://feelingssoftware.com/models/rocks.dae#rockModels/bigRock">
            <!-- catapultAndRockModel 空間のカタパルト上に石を配置する。 -->
            <translate> 0 4 0 </translate>
        </instance_physics_model>
    </physics_model>

```

```

</instance_physics_model>
  <instance_physics_model sid="catapult" url="#catapultModel" />
</physics_model>
</library_physics_models>

```

physics_scene

カテゴリ: 物理シーン

概要

フィジックスオブジェクトがインスタンス化/シミュレーション化される環境を指定するためのものです。

コンセプト

COLLADA では、主に以下のような理由で、複数のシミュレーションを同時に実行できるようになっています。

- 複数のシミュレーションで別々のグローバルな設定が必要な場合があり、他のフィジックスエンジンや別のハードウェア上で実行される場合さえあります。
- 高レベルなグループ化機能があれば、パフォーマンスを最適化するために相互作用を最小限に抑えることができます。たとえば、あるフィジックスシーンでの剛体は、他のフィジックスシーンの剛体とコリジョンしないことがわかっているため、それらの間でコリジョンテストを行う必要はありません。
- 複数の詳細レベル (LOD) をサポートできます。

文字列テクスチャ `<physics_scene>` 要素には、`technique` と `extra` の要素と、`<instance_physics_model>` 要素のリストを含めることができます。

アクティブな `<physics_scene>` は (シミュレートされているものは)、メインの `<scene>` の基でインスタンス化することで示されます。

属性

`<physics_scene>` 要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<physics_scene>` 要素は、以下の要素と関連性があります。

親要素	<code>library_physics_scenes</code>
子要素	下記サブセクションを参照してください。
その他	<code>instance_physics_scene</code>

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><instance_force_field></code>	<code><force_field></code> 要素をインスタンス化してフィジックスシーンに影響を与えます。主見出し項目を参照してください。	なし	0 以上
<code><instance_physics_model></code>	<code><physics_model></code> 要素をインスタンス化して、その子の一部または全部をオーバーライドすることができます。主見出し項目を参照してください。	なし	0 以上
<code><technique_common></code>	すべての COLLADA 実装がサポートしなければならない共通プロファイルの物理シーン情報を指定します。詳しくは、「共通プロファイル」の使い方情報のセクション、および続く子要素詳細についてのサブセクションを参照してください。	なし	1
<code><technique></code> (core)	各 <code><technique></code> は、 <code><technique></code> の <code>profile</code> 属性として指定された特定のプロファイルのための物理シーン情報を指定します。コアの主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

`<physics_scene>` / `<technique_common>`の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><gravity sid=" " ></code>	シーンの重力場のベクトル表現。この重力場によりもたらされた加速度の大きさと方向を表す3つの浮動小数点値の非正規化方向ベクトルとして、与えられます。 <code>sid</code> 属性はオプションです。	なし	0 または 1
<code><time_step sid=" " ></code>	フィジックスシーンの、秒単位で測定された積分時間ステップ。この値はエンジン固有のもので、省かれている場合は、フィジックスエンジンのデフォルト値が使用されます。浮動小数点値が含まれます。 <code>sid</code> 属性はオプションです。	なし	0 または 1

例

```

<library_physics_scenes>
<!--通常の physics scene -->
<physics_scene id="ColladaPhysicsScene">
  <instance_physics_model sid="firstCatapultAndRockInstance">
    url="#catapultAndRockModel" parent"#catapult1">
<!-- オーバーライドした physics_model のインスタンス
現在の変換行列で、ワールド空間中の physics_model の
初期位置と方向を決める。 -->
  <instance_rigid_body body="./rock/rock" target="#rockNode">
    <technique_common>
      <velocity>0 -1 0</velocity> <!-- オプションのオーバーライド -->
      <mass>10</mass>

```

```

<!-- 重くする -->
    </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>
<!-- 2つの物理的にシミュレーションされたカタパルトの「アーミー」がインスタンス化されるシーン -->
<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>
  </visual_scene>
</library_visual_scenes>
<scene>
  <!-- 親ノードの1つをインスタンス化して physics_model を複製できる。 -->
    <node id="catapult2">
      <translate/>
      <!-- 2番目のカタパルトの位置を別の場所に位置させる -->
      <rotate/>
      <instance_node url="#catapultNode1"/> <!-- physics_model と表示を複製 -->
    </node>
  </visual_scene>
</library_visual_scenes>
<scene>
  <!-- visual_scene に physics_scene が適用可能であることを示す。 -->
  <instance_physics_scene url="#ColladaPhysicsScene"/>
  <instance_visual_scene url="#battlefield"/>
</scene>

```

plane

カテゴリ: 解析形状 (フィジックス)
 曲面 (B-Rep)

概要

無限平面を定義します。

コンセプト

フィジックスの中の平面は、動力的な剛体が境界を貫通したり永遠に落下したりすることを防ぐための、静的コリジョンオブジェクトとしてよく使われます。詳しくは、この章の冒頭の「ジオメトリの種類」のセクションを参照してください。

B-Rep では、平面は曲面の一種です。B-Rep における平面座標系の説明は、9 章 B-Rep リファレンスの [surface](#) を参照してください。

属性

`<plane>` 要素には属性はありません。

関連要素

`<plane>` 要素は、以下の要素と関連性があります。

親要素	shape 、 surface (B-Rep)
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><equation></code>	平面方程式の係数を表す 4 つの浮動小数点値が含まれます。 $Ax + By + Cz + D = 0$ この要素には属性はありません。	なし	1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

```

<rigid_body>
    <dynamic>false</dynamic>
    <shape>
        <plane>
            <!-- 平面の方程式: Ax + By + Cz + D = 0 -->
            <!-- A, B, C, D 係数 (normal & D) -->
            <equation> 0.0 1.0 0.0 0.0 </equation> <!-- x-z 平面 (グラウンド) -->
        </plane>
    </shape>
</rigid_body>
    
```

ref_attachment

カテゴリ: 物理モデル

概要

剛体束縛の中の剛体やノードに、アタッチされた基準系を定義します。

コンセプト

`<rigid_constraint>`は、2つの剛体を互いに連結（して両者の間の運動を制限）します。
`<ref_attachment>`は1番目の剛体、`<Attachment>`は2番目の剛体を参照します。たとえば、ドアと壁の間の蝶番束縛の場合、一方が参照アタッチメント（この場合には壁）、もう一方がアタッチメント（ドア）です。

また、`<ref_attachment>`は、その連結先のためのローカルな座標系を、`<translate>`および`<rotate>`要素を利用して、剛体（ノード）との相対位置として定義します。たとえば、蝶番（剛体束縛）は、壁（剛体）のローカル座標の原点からの相対位置として、壁の縁の中央に連結されます。

属性

`<ref_attachment>`要素には、以下の属性があります。

<code>rigid_body</code>	<code>xs:anyURI</code>	<code><rigid_body></code> や <code><node></code> を参照するスコープ付き識別子。この属性は、 <code><rigid_body></code> もしくは <code><Attachment></code> のどちらかの中の <code><ref_attachment></code> のSIDを参照する必要があります。両方が <code><node></code> であってはなりません。必須。
-------------------------	------------------------	---

関連要素

`<ref_attachment>`要素は、以下の要素と関連性があります。

親要素	<code>rigid_constraint</code>
子要素	下記サブセクションを参照してください。
その他	<code>Attachment</code>

子要素

子要素が存在する場合には、任意の順で記述できます。

名前/例	解説	デフォルト値	出現回数
<code><translate></code>	この変換の位置は、対応する <code><rigid_body></code> の接続点を示しています。コアの主見出し項目を参照してください。	なし	0 以上
<code><rotate></code>	この変換の方向は、その <code><rigid_body></code> のジョイントフレームの方向を示しています。コアの主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

```
<ref_attachment rigid_body="./SomeRigidBody">
  <translate/>
  <rotate/>
  <extra/>
</ref_attachment>
```

より完全な例は、`<rigid_constraint>`を参照してください。

rigid_body

カテゴリ: 物理モデル

概要

形状を崩さないシミュレーションした物体を記述します。

コンセプト

リジッドボディは、各種の束縛（蝶番やボールジョイントなど）で結び付けられている場合も、また結び付けられていない場合もあります。複雑なモデルをインスタンス化できるように、リジッドボディや制約などは<physics_model>要素にカプセル化されます。

リジッドボディは、コリジョン検出のためのパラメータと形状のコレクションで構成されています。それぞれの形状は、複雑なコリジョン形状（バウンディング形状）を構築できるように、回転、変形可能となっています。これらの形状は1つ以上の<shape>要素で記述されています。

属性

<rigid_body>要素には、以下の属性があります。

sid	sid_type	<rigid_body>要素のスコープ付き識別子を含むテキスト文字列。この値は兄弟関係にあるどの要素に対してもユニークでなければなりません。<node> がインスタンス化される際に、それぞれのリジッドボディを視覚的な<physics_model>に関連付けるのに利用されます。必須。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。
name	xs:token	この要素の文字列の名。オプション。
id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。

関連要素

<rigid_body>要素は、以下の要素と関連性があります。

親要素	physics_model
子要素	下記サブセクションを参照してください。
その他	instance_rigid_body

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<technique_common>	すべてのCOLLADA実装がサポートしなければならない共通プロファイルのリジッドボディ情報を、指定します。詳しくは、「共通プロファイル」の使い方情報のセクション、およびそれに続く子要素詳細についてのサブセクションを参照してください。	なし	1
<technique> (core)	各<technique>は、<technique>のprofile属性として指定された特定のプロファイルのためのリジッドボディ情報を指定します。コアの主見出し項目を参照してください。	なし	0 以上

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	<code><rigid_body></code> に情報を追加する複数表現可能なユーザ定義データ <code><technique></code> 要素のように、ベースデータを切り替えるのとは違う操作です。コアの主見出し項目を参照してください。	なし	0 以上

`<rigid_body>` / `<technique_common>`の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><dynamic sid="...">>false</dynamic></code>	<code>rigid_body</code> が移動可能かどうかを指定する論理値が含まれます。 <code>sid</code> 属性はオプションです。	なし	0 または 1
<code><mass sid="...">0.5</mass></code>	<code>rigid_body</code> の総質量を指定する浮動小数点値が含まれます。 <code>sid</code> 属性はオプションです。	密度 × 形状全体の体積から導き出されず。	0 または 1
<code><mass_frame> <translate> ... </translate> <rotate> ... </rotate> </mass_frame></code>	COLLADA で変換を表すためのメカニズムを利用して、 <code>rigid_body</code> の重心および主軸の方向を指定します。 <code><translate></code> および <code><rotate></code> は複数の要素が任意の順序で存在できるので、その結果である変換のアフィン部分は抽出する必要があります。この最終的な平行移動と回転は、それぞれ、重心と主軸に対応しています。 <code><translate></code> 子要素と <code><rotate></code> 子要素の出現回数は、0 回または任意の回数が可能です。最低でもどちらか一方は存在する必要があります。コアの主見出し項目を参照してください。	identity (重心はローカルの原点に位置し、主軸はローカル軸となります)	0 または 1
<code><inertia sid="..."> 1 1 1</inertia></code>	主軸となる座標系で計算された、リジッドボディの慣性テンソル行列の対角成分である、3つの浮動小数点数が含まれます。このような座標系では、テンソル行列の非対角要素はゼロになります。 <code>sid</code> 属性はオプションです。	float3: 慣性テンソル (慣性モーメント) の対角成分で、重心のローカルフレームで表現されます (上記を参照)。	0 または 1
<code><physics_material></code> または <code><instance_physics_material></code>	<code>rigid_body</code> のための <code>physics_material</code> を定義または参照します。主見出し項目を参照してください。	なし	0 または 1
<code><shape></code>	主見出し項目を参照してください。	なし	1 以上

密度、質量、慣性の指定規則

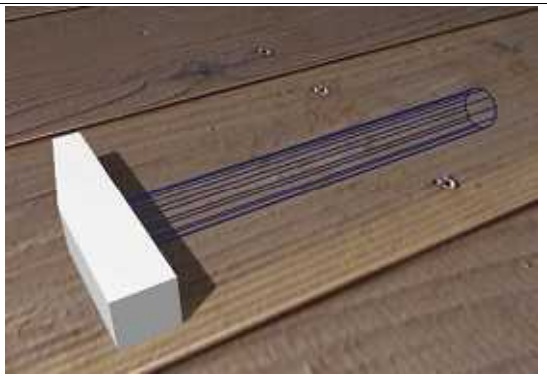
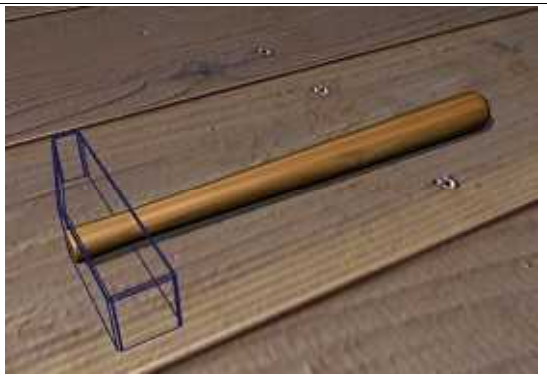
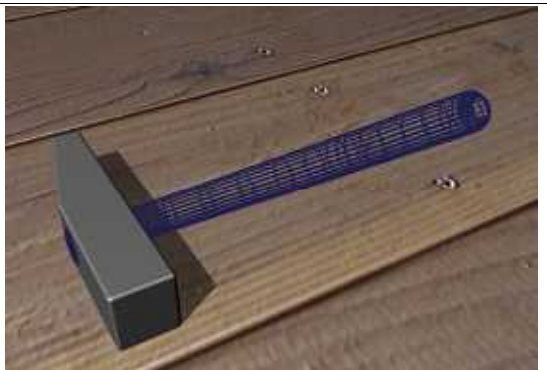
さまざまな質量特性の指定方法としては、以下の3つの方法が可能です。

- リジッドボディの質量を指定しない。リジッドの質量、慣性、質量系は、形状の体積および質量を使って計算します。指定された慣性や座標系の変換は、すべて、この計算中に上書きされます。このレベルにおける質量や密度の指定に関する詳細は、`<shape>`を参照してください。

- リジッドボディの質量は指定するが、慣性を指定しない。質量を指定しない場合と同じように、慣性や質量系は、形状体積および質量を使って計算します。また、導出された慣性テンソルは、指定された質量と体積積分によって推定された質量の比によって、スケーリングされます。
- リジッドボディの質量と慣性の両方を指定する。この場合、質量系も同時に指定されることが想定されますが、指定されなかった場合には、恒等変換が使われます。これは、物体の重心がローカル座標の原点であり、質量系が主軸と同じ方向を向いていることを意味します。
- リジッドボディとその形状のどちらも、質量または密度を指定してもかまいません。どちらも定義しなかった場合、密度のデフォルトは1.0となり、質量は形状の全体の体積を利用して計算されます。

例

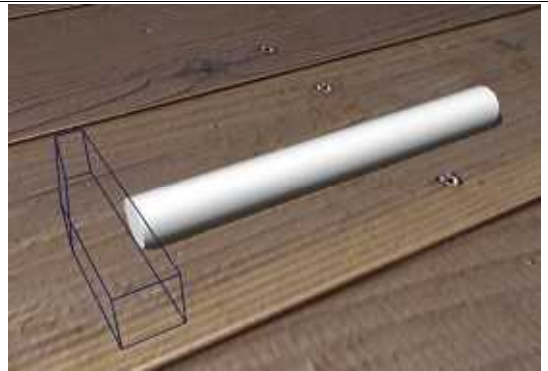
以下は、複合化したリジッドボディの例です。形状の違いがフィジックス（シリンダプリミティブと単純な凸包）を意味し、レンダリングの対象（テクスチャ化され、すばまった柄の部分と、傾斜された頭の部分）となる点に注意してください。

<pre><library_geometries> <geometry id="hammerHeadForPhysics"> <mesh> ... </mesh> </geometry></pre>	
<pre><geometry id="hammerHandleToRender"> <mesh> ... </mesh> </geometry></pre>	
<pre><geometry id="hammerHeadToRender"> <mesh> ... </mesh> </geometry> </library_geometries></pre>	

```

<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"/>
<!-- このジオメトリは小さくて、他の場所では使われてい
ないで、インライン化されている -->
          <cylinder>
            <height> 8.0 </height>
            <radius> 0.5 0.5 </radius>
          </cylinder>
        </shape>
        <shape>
          <mass> 1.0 </mass>
          <!-- このジオメトリは、インラインではなく
参照される -->
          <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>

```



rigid_constraint

カテゴリ: 物理モデル

概要

リジッドボディを、他のリジッドボディもしくはワールドに対して束縛します。

コンセプト

コンストレイントは、相対運動の線形自由度や角度自由度に対する制限として設定できます。リジッドボディとコンストレイントの集合は、可動部分をもつ複雑な物理モデルを構成することができます。

役に立つ物理モデルを作成するということは、通常、バネやボールジョイントなど、さまざまな種類の束縛を利用して複数のリジッドボディを結び付けることを意味します。

COLLADA では、2つのリジッドボディをリンク、またはリジッドボディとシーン階層中(例: ワールド空間)の座標フレームをリンクするための制約をサポートしています。COLLADA では、コンストレイントプリミティブ要素のさまざまな組み合わせを定義する代わりに、非常に柔軟な要素、つまり、自由度(DOF) 6を持った汎用的な制約を1つ提供しています。この汎用的な制約を利用して、もっと簡単な制約(たとえば、線形スプリングや角スプリング、ボールジョイント、ヒンジなど)を表現することができます。

制約は、以下のように指定します。

- リジッドボディのローカル空間、もしくはシーン階層中の座標フレームへの相対的な平行移動と方向を利用して定義された2つのフレームをアタッチします。COLLADAの他の部分と一貫性を取るために、これは標準の<translate>要素と<rotate>要素を利用して表現します。
- 自由度 (DOF) で指定します。DOF では、並進の特定の軸または回転の軸に沿って発生する可変性を、アタッチされたフレームの空間中で表現して指定します。たとえば、ドアのヒンジ部分は、通常、特定の回転軸に沿って自由度が1です。それとは反対に、スライダのジョイント部分は、単一の並進軸に沿って自由度が1となります。
自由度と制限は、非常に柔軟な<limits>要素で指定します。

属性

<rigid_constraint>要素には、以下の属性があります。

sid	sid_type	<rigid_constraint>要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。必須。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。
name	xs:token	この要素の文字列の名。オプション。

関連要素

<rigid_constraint>要素は、以下の要素と関連性があります。

親要素	physics_model
子要素	下記サブセクションを参照してください。
その他	instance_rigid_constraint


子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<ref_attachment>	リジッドボディコンストレイントの中の (rigid_body やノードに) アタッチされた基準系を定義します。主見出し項目を参照してください。	なし	1
<Attachment>	リジッドボディコンストレイントの中の (リジッドボディやノードに) アタッチされた座標系を定義します。主見出し項目を参照してください。	なし	1
<technique_common>	すべての COLLADA 実装がサポートしなければならない共通プロファイルのリジッドボディコンストレイント情報を、指定します。詳しくは、「共通プロファイル」の使い方情報のセクション、およびそれに続く子要素詳細についてのサブセクションを参照してください。	なし	1
<technique> (core)	各<technique>は、<technique>の profile 属性として指定された特定のプロファイルのためのリジッドボディコンストレイント情報を指定します。コアの主見出し項目を参照してください。	なし	0 以上
<extra>	ユーザ定義された複数の表現が可能。コアの主見出し項目を参照してください。	なし	0 以上

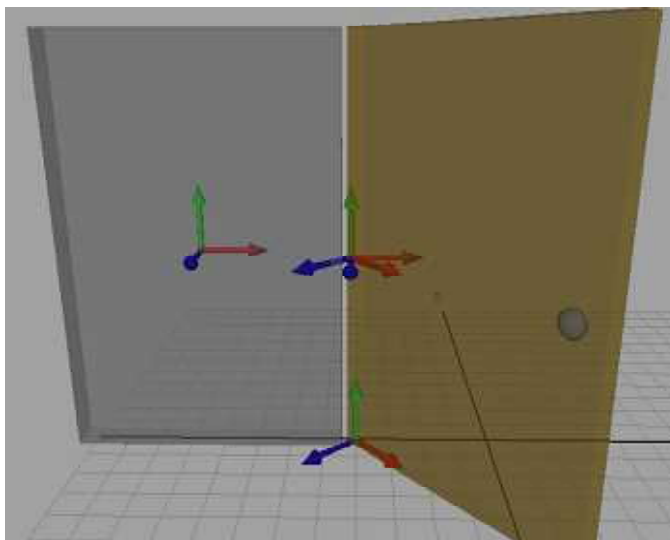
<rigid_constraint> / <technique_common>の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<pre><enabled sid="..."> true</enabled></pre>	論理値が含まれます。false の場合、<constraint> はリジッドボディに対していかなる力や作用も及ぼしません。sid 属性はオプションです。	True	0 または 1
<pre><interpenetrate sid="...">true </interpenetrate></pre>	論理値が含まれます。true の場合、アタッチされたリジッドボディが互いに突き抜ける場合があることを表します。sid 属性はオプションです。	False	0 または 1
<p>「スイングする円錐とねじれ」の角度制限を持つ 2 つの束縛。</p> <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> 	<p><limits>要素は、コンストレイント制限（自由度と範囲）を指定するための柔軟性のある方法を提供します。この要素には属性はありません。この要素は、オプションとして子要素 <swing_cone_and_twist> または <linear>、もしくはこの両方を含むことができます。両方を使う場合には、例の順序で出現する必要があります。このような制限の記述では不十分な場合には、カスタム<technique>を利用してください。</p> <p><linear>要素は、それぞれの軸に沿った線形の（平行移動的な）上限を表します。</p> <p><swing_cone_and_twist>要素は、それぞれの回転軸に沿った角度を度数で表します。</p> <p><min>、<max>要素はオプションですが、使用する場合には、例の順序で出現する必要があります。これらの要素の sid 属性はオプションです。これらの要素には、それぞれ、x、y、z の制限を表す 3 つの浮動小数点値が含まれます。また、+/-無限大に対応する値である INF と -INF は、その軸に沿った制限がないことを示すために利用できます。</p> <p>それぞれの上限は、ref_attachment の空間で表現します。</p> <p><swing_code_and_twist>の中では、x と y の上限は「スイングする円錐」を表し、z の上限は「ねじれの角度」の範囲を表します（左側の図を参照）。</p>	<p>linear:</p> <p>最小 0.0 0.0 0.0</p> <p>最大 0.0 0.0 0.0</p> <p>swing_cone_and_ねじれ:</p> <p>最小 0.0 0.0 0.0</p> <p>最大 0.0 0.0 0.0</p> <p>これは、完全に固定されたリジッドボディ制約に対応します。つまり、2 つのリジッドボディは互いに相対的に動くことはありません（回転も変形も許されていません）。</p>	0 または 1

名前/例	解説	デフォルト値	出現回数
<p>例 1:</p> <pre><spring> <linear> <stiffness sid="...">5.4544 </stiffness> <damping sid="...">0.4132 </damping> <target_value sid="...">3 </target_value> </linear> </spring></pre> <p>例 2:</p> <pre><spring> <angular> <stiffness>5.4544 </stiffness> <damping>0.4132 </damping> <target_value>90 </target_value> </angular> </spring></pre>	<p>スプリングは、距離 (<linear>) と角度 (<angular>) のいずれか、もしくは、その両方に依存します。両方を指定する場合には、<angular> が先に出現する必要があります。これらの要素は、オプションで 3 つの子要素を持つことができます。この子要素を利用する場合には、例の順序で出現する必要があります。これらの要素には、それぞれ、単一の浮動小数点値が含まれます。これらの要素の sid 属性はオプションです。</p> <p><stiffness> (スプリング係数とも呼ばれます) の単位は、<linear> の場合には力/距離、<angular> の場合には力/角度数になります。</p> <p>スプリングは、ref_attachment 空間で表されます。</p>	<p>stiffness: 1.0 damping: 0.0 target_value: 0.0</p> <p>これは、「無限リジッドボディ」コンストレイント(つまりスプリングではない)に対応していません。</p>	<p>0 または 1</p>

例



上記は、ヒンジ部分を持ったドアの例です。壁のリジッドボディ(グレー部分、右手)では、中央にローカル空間のフレームがあります。ドアはフロア上のローカル空間があり、Y 軸に対して 45 度回転されています。ヒンジ部分の制約は、Y 軸に対して +/- 90 度の回転に制限されています。アタッチされた個々のフレームは、リジッドボディのローカル空間で定義された変形と回転を持ちます。

```

<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>
<!-- sid 属性を追加することで、アニメーションから制限をターゲットにすることが可能となる-->
      <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>
    </rigid_constraint>
  </physics_model>
</library_physics_models>

```

shape

カテゴリ: 物理モデル

概要

<rigid_body>のコンポーネントを記述します。

コンセプト

リジッドボディには、コリジョン検出用に1つの形状または形状のコレクションを含めることができます。各形状は、複雑なコリジョン形状（バウンディング形状）を構築できるように、回転や平行移動が可能となっています。

これらの形状は<shape>要素で記述し、それぞれに対して以下の情報を含めることができます。

- 表面の跳ね返りや摩擦特性を記述する<physics_material>定義もしくはインスタンス
- 物理的性質（質量や密度）
- 変換（<rotate>、<translate>）
- <geometry>のインスタンスまたはインライン化された定義

形状レベルの質量特性は、物理シミュレーションでは関係ありません。関係するのは、リジッドボディレベルの質量特性だけです。必要があれば、親リジッドが子形状を積分することによって物体の質量特性を導出できるように、shape 要素には質量や密度を指定することができます。COLLADA は、用交換用のフォーマットを用途として想定しています。コンテンツクリエイターにとっては、通常、ジオメトリのグループに対して質量、重心、慣性モーメントを正しく割り当てるよりも、グループ中の各物体に重量を

割り当てる方が簡単です。質量特性の計算は、通常、コンテンツエクスポートパイプラインの中の、アート作成の後、かつ、物理エンジンの中でインスタンス化する前に、行われます。

形状は中空、つまり、質量が体積全体に分布しているのではなくて、表面だけに存在していることもあります。この場合の密度（指定された場合）は、単位正方形当たりの質量を示します。中空か中空でないかが影響するのは、質量、慣性、重心の計算だけです。

属性

`<shape>`要素には属性はありません。

関連要素

`<shape>`要素は、以下の要素と関連性があります。

親要素	rigid_body/technique_common , instance_rigid_body/technique_common
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><hollow sid="..."> true</hollow></code>	論理値が含まれます true の場合、質量は形状のサーフェスに沿って分布します。 sid 属性はオプションです。	なし	0 または 1
<code><mass sid="..."> 0.5 </mass></code>	形状の質量を指定する浮動小数点数が含まれます。 sid 属性はオプションです。	密度 x 形状の体積から導き出されます	0 または 1
<code><density sid="..."> 0.5 </density></code>	形状の密度を指定する浮動小数点数が含まれます。sid 属性はオプションです。	質量/形状の体積から導き出されます	0 または 1
インライン定義またはインスタンス: <code><physics_material></code> または <code><instance_physics_material></code>	この形状に使用される <code><physics_material></code> 。	<code><shape></code> でインスタンス化または定義されたジオメトリから導き出されます	0 または 1
形状のジオメトリ。	これは、以下のいずれかです。 <ul style="list-style-type: none"> 以下のいずれかの要素を使ったインライン定義。 <code><plane></code>、 <code><box></code>、 <code><sphere></code>、 <code><cylinder></code>、 <code><capsule></code>。 <code><instance_geometry></code>要素を使って、他の種類のジオメトリ (<code><mesh></code>、 <code><convex_mesh></code>、 <code><spline></code>など) を参照するジオメトリインスタンス。 	なし	1
<code><rotate></code> , <code><translate></code>	形状の変換。これらの要素の任意の組合せ。さらなる情報は、コアの主見出し項目、および <code><node></code> を参照してください。	変換は行われません。	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

詳しくは、`<rigid_body>`要素を参照してください。

例

```
<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>
```

sphere

カテゴリ: **解析形状 (フィジックス)**
曲面 (B-Rep)

概要

ローカル座標の原点を中心とする球を記述します。

コンセプト

フィジックスでは、球のようなジオメトリプリミティブや解析的な形状は、通常コリジョン形状として役立ちます。詳しくは、この章の冒頭の「ジオメトリの種類」のセクションを参照してください。

B-rep では、球は半径によって定義され、球の中心を原点とする座標系によって空間に配置される曲面の一種です。

属性

`<sphere>`要素には属性はありません。

関連要素

`<sphere>`要素は、以下の要素と関連性があります。

親要素	<code>shape</code> 、 <code>surface</code> (B-Rep)
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><radius></code>	球の半径を指定する浮動小数点値が含まれます。この要素には属性はありません。	なし	1

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

```
<shape>
  <sphere>
    <radius> 1.0 </radius>
  </sphere>
</shape>
```

このページは空白です。

7章 FX 入門

概要

COLLADA FX は、カラーをビジュアルシーンに適用する方法を作成者が記述することを可能にします。これは、多くのプラットフォームとアプリケーションプログラミングインタフェース(API)すべてにわたって、マテリアルプロパティを記述するための柔軟な概念です。

COLLADA スキーマの FX 要素で記述できるものには、以下のものがあります。

- 単一パスおよび複数パスのエフェクト。抽象マテリアルの定義です (例: プラスチック)
- エフェクトパラメタリゼーション (`<newparam>` を使用)
- エフェクトメタデータ
- シーングラフとのバインディング
- 複数のテクニック
- インラインおよび外部のソースコードまたはバイナリ

`<profile_*>` 要素により、複数のアプリケーションプログラミングインタフェース (API) やシェーダ言語がサポートされているので、各エフェクトを複数のプラットフォーム用に記述することができます。各プラットフォームを、プラットフォーム固有のデータ型、レンダリングステート、および機能を用いて完全に記述できます。

各プラットフォームでは、各エフェクトを多くのテクニックを用いて記述できます。テクニックとは、レンダリングのスタイル (例: `daytime`、`nighttime`、`magic`、`superhero_mode`)、異なる詳細レベル、または計算方法 (例: 「`approximate`」、 「`accurate`」、 「`high_LOD`」、 「`low_LOD`」) のユーザがラベル付けした (例で示したような) 表記のことです。

より高いレベルで、マテリアルシステムは、エフェクト定義のデフォルト値以外の値をパラメータに提供することで、あらかじめ定義されたエフェクトを特定のインスタンスに特化できるようにします。このことにより、単一の効果を多くのさまざまなマテリアルの基礎として使用することが可能になります。

最後に、マテリアルは、シーングラフの1つ以上のポイントにバインドされるとき、使用するために挿入されます。シーングラフのジオメトリ内の `<bind_material>` 要素は、1つ以上のマテリアルをインスタンス化し、それらをジオメトリのセグメントに接続することができます。マテリアルインスタンス内で、シーングラフのリソース (光源やカメラなど) とバインドしたり、テクスチャ座標をペアにしたりすることで、エフェクトをさらに特化させることができます。

プラットフォーム固有のエフェクトにプロファイルを利用する

`<profile_*>` 要素を使うと、各エフェクトを複数のプラットフォーム用に記述することができます。

プロファイルについて

`<profile_*>` 要素は、特定のプロファイル中のエフェクトのための、プラットフォーム固有の値と宣言をすべてカプセル化します。この要素は、プラットフォーム固有の具体的なデータ型と、文書中のそれ以外の場所で利用されている COLLADA の抽象データ型との間の、明確なインタフェースを定義します。

COLLADA FX では、以下のプロファイルをサポートしています。

- **<profile_COMMON>**: プラットフォームに依存しない固定機能シェーダのためのあらゆる値および宣言をカプセル化します。**<profile_COMMON>**は、どのプラットフォームでもサポートする必要があります。このプロファイル中のエフェクトは、現在のエフェクトランタイムが他のプロファイルを認識できないときに、信頼性の高いフォールバックとして使うことを想定して設計されています。
- **<profile_CG>**: Cg 高水準言語で使うためのエフェクトをカプセル化します。
- **<profile_GLES>**: OpenGL ES で使うためのエフェクトをカプセル化します。
- **<profile_GLES2>**: OpenGL ES 2.0 で使うためのエフェクトをカプセル化します。
- **<profile_GLSL>**: OpenGL シェーディング言語で使うためのエフェクトをカプセル化します。
- **<profile_BRIDGE>**: COLLADA FX エフェクトと、NVIDIA® Cg FX や Microsoft FX のような外部エフェクトファイル形式との間の橋渡しをする機能を提供します。

注:

- テクニックとは異なり、他のプロファイルを指定した場合には、**<profile_COMMON>**を指定する必要はありません。
- **<profile_COMMON>**は、あらゆる環境で望ましいとは限らない共通のエフェクト定義ですが、**<technique_common>**は異なる拡張性メカニズムの一部です。

プロファイル中の FX 要素の属性および構造

各 FX 要素の特定の属性および子要素は、使用されるプロファイルスコープによって異なることがあります。プロファイルスコープは、以下のようにグループ化できます。

- エフェクトの外部（エフェクトをデータに関連付けたり、利用シナリオに適用したりするために必要な要素）
- エフェクト（要素は、特定のプロファイルの外のエフェクトスコープ内で有効）
- 共通プロファイル
- Cg プロファイル
- OpenGL ES (GLES) プロファイル
- GLSL プロファイル
- OpenGL ES 2.0 (GLES2) プロファイル

以下の表は、どの要素がどのスコープの中で有効かを示しています。

要素	外部	エフェクト	共通	Cg	GLES	GLSL	GLES2
<alpha>	–	–	–	–	有効	–	–
<ambient> (FX)	–	–	有効	–	–	–	–
<annotate>	有効	有効	有効	有効	有効	有効	有効
<argument>	–	–	–	–	有効	–	–
<array>	–	–	–	有効	–	有効	有効
<binary>	–	–	–	–	–	–	有効
<bind> (FX)	有効	–	–	–	–	–	–
<bind_attribute>	–	–	–	–	–	有効	有効
<bind_material>	有効	–	–	–	–	–	–
<bind_uniform>	–	–	–	有効	–	有効	有効
<bind_vertex_input>	有効	–	–	–	–	–	–
<blinn>	–	–	有効	–	–	–	–

要素	外部	エフェクト	共通	Cg	GLES	GLSL	GLS2
<code>	-	-	-	有効	-	有効	有効
<color_clear>	-	-	-	有効	有効	有効	有効
<color_target>	-	-	-	有効	有効	有効	有効
<compiler>	-	-	-	有効	-	-	有効
<constant> (FX)	-	-	有効	-	-	-	-
<constant> (コンパイナ)	-	-	-	-	有効	-	-
<create_2d>	有効	-	-	-	-	-	-
<create_3d>	有効	-	-	-	-	-	-
<create_cube>	有効	-	-	-	-	-	-
<depth_clear>	-	-	-	有効	有効	有効	有効
<depth_target>	-	-	-	有効	有効	有効	有効
<diffuse>	-	-	有効	-	-	-	-
<draw>	-	-	-	有効	有効	有効	有効
<effect>	-	有効	-	-	-	-	-
<emission>	-	-	有効	-	-	-	-
<evaluate>	-	-	-	有効	有効	有効	有効
<format>	有効	-	-	-	-	-	-
<image>	有効	-	-	-	-	-	-
<include>	-	-	-	有効	-	有効	有効
<index_of_refraction>	-	-	有効	-	-	-	-
<init_from>	有効	-	-	-	-	-	-
<instance_effect>	有効	-	-	-	-	-	-
<instance_image>	有効	有効	有効	有効	有効	有効	有効
<instance_material> (ジオメトリ)	有効	-	-	-	-	-	-
<instance_material> (レンダリング)	有効	-	-	-	-	-	-
<lambert>	-	-	有効	-	-	-	-
<library_effects>	有効	-	-	-	-	-	-
<library_images>	有効	-	-	-	-	-	-
<library_materials>	有効	-	-	-	-	-	-
<linker>	-	-	-	-	-	-	有効
<material>	有効	-	-	-	-	-	-
<modifier>	有効	有効	有効	有効	有効	有効	有効
<newparam>	-	有効	有効	有効	有効	有効	有効
<param>(reference)	-	-	有効	有効	-	有効	有効
<pass>	-	-	-	有効	有効	有効	有効
<phong>	-	-	有効	-	-	-	-
<profile_BRIDGE>	有効	-	-	-	-	-	-
<profile_CG>	-	-	-	有効	-	-	-
<profile_COMMON>	-	-	有効	-	-	-	-
<profile_GLES>	-	-	-	-	有効	-	-
<profile_GLES2>	-	-	-	-	-	-	有効

要素	外部	エフェクト	共通	Cg	GL ES	GLSL	GL ES2
<profile_GLSL>	-	-	-	-	-	有効	-
<program>	-	-	-	有効	-	有効	有効
<reflective>	-	-	有効	-	-	-	-
<reflectivity>	-	-	有効	-	-	-	-
<render>	有効	-	-	-	-	-	-
<RGB>	-	-	-	-	有効	-	-
<sampler1D>	-	-	有効	有効	-	有効	-
<sampler2D>	有効	有効	有効	有効	有効	有効	有効
<sampler3D>	有効	有効	有効	有効	-	有効	有効
<samplerCUBE>	有効	有効	有効	有効	-	有効	有効
<samplerDEPTH>	有効	有効	有効	有効	-	有効	-
<samplerRECT>	有効	有効	有効	有効	-	有効	-
<sampler_image>	有効	-	-	-	-	-	-
<semantic>	有効	有効	有効	有効	有効	有効	有効
<setparam>	有効	-	-	有効	-	-	有効
<shader>	-	-	-	有効	-	有効	有効
<shininess>	-	-	有効	-	-	-	-
<specular>	-	-	有効	-	-	-	-
<states>	-	-	-	有効	有効	有効	有効
<stencil_clear>	-	-	-	有効	有効	有効	有効
<stencil_target>	-	-	-	有効	有効	有効	有効
<technique> (FX)	-	-	有効	有効	有効	有効	有効
<technique_hint>	有効	-	-	-	-	-	-
<texcombiner>	-	-	-	-	有効	-	-
<texenv>	-	-	-	-	有効	-	-
<texture_pipeline>	-	-	-	-	有効	-	-
<transparency>	-	-	有効	-	-	-	-
<transparent>	-	-	有効	-	-	-	-
<usertype >	-	-	-	有効	-	-	有効

FXのパラメータについて

COLLADA パラメータ全般の情報は、4章 プログラミングガイドの「COLLADAのパラメータについて」を参照してください。

COLLADA FXにおいて、<newparam>要素は、指定されたスコープの中でバインド可能なパラメータを宣言します。バインドを成功させるためには、かならずしもパラメータの型同士が厳密に一致する必要はありません。ただし、この場合のパラメータの型は、integer から `float_type`、`float3_type` から `float4_type`、Boolean から `int_type` といった簡単（かつアプリケーションにとって意味のある）な変換やキャストで変換できるような互換性を持つ必要があります。COLLADA FXでは、型キャストについてアプリケーションが何をサポートすべきで何をすべきでないかについて、具体的なルールを定めていません。したがって、できるだけ多くのアプリケーションで安全に利用できるファイルを作成するには、キャストに期待するよりも、型を正しく一致させた方がよいでしょう。

`<effect>`のスコープ内のパラメータは、どのプラットフォームでも利用できますが、特定の`<profile_*>`（「プロファイル」参照）ブロック内で宣言されたパラメータを利用できるのは、そのプロファイル中のシェーダだけです。この範囲の外側で宣言されたパラメータを`<profile_*>`（「プロファイル」参照）ブロックの中で利用するには、キャストが必要になることがあります。

COLLADA では、エフェクト内のパラメータ操作用に、以下の要素を提供しています。

- `<newparam>`: パラメータを作成します。
- `<setparam>`: パラメータの型および値を変更または設定します。
- `<modifier>`: パラメータの揮発性やリンク（`UNIFORM` や `SHARED` など）を指定します。
- `<array>`: `<newparam>`や`<setparam>`の中で、パラメータを配列として定義します。
- `<usertype>`: `<newparam>`や `<setparam>`の中で、パラメータを構造体として定義します。
- `<annotate>`: パラメータや FX 中の他の場所で使う、`symbol=value` という形式のオブジェクトを表します。
- `<param>`(reference): `<newparam>`によって作成された既存のパラメータを参照します。

`<bind>`や`<bind_vertex_input>`の中のパラメータを特定する

`<bind>`要素や`<bind_vertex_input>`要素は、ターゲットを`<effect>`の中のパラメータにバインドします。`<effect>`の中のパラメータを特定する検索文字列は、`semantic` 属性によって指定されます。`<effect>`の中のパラメータを特定する際には、以下の順序で検索します。

- COLLADA FX パラメータをセマンティックによって探します。
- プロファイルの中にシェーディング言語コードが含まれていて、その言語がセマンティックスをサポートしている場合には、シェーダの中からセマンティックによってパラメータを探します。
- COLLADA FX パラメータを `sid` によって探します。
- プロファイルの中にシェーディング言語コードが含まれている場合には、シェーダの中から名前によってパラメータを探します。

シェーダ

COLLADA は、シェーダを記述する要素を複数提供しています。

- `<blinn>`
- `<constant>`
- `<lambert>`
- `<phong>`

さらに、シェーダの性質を記述する要素もいくつか提供しています。

- `<bind_uniform>`
- `<code>`
- `<compiler>`
- `<include>`
- `<shader>`

レンダリング

透明度（不透明度）を判定する

<transparent>もしくは<transparency>が存在する場合、透明レンダリングが有効になるので、レンダラをアルファブレンディングモードに切り替える必要があります。2つの値を組み合わせる方法は、以下の式で定義されます。<transparent>の不透明度設定に基づいて正しい結果を取得するには、下記の数式を使います。ただし fb はフレームバッファ（つまり、レンダリング対象の背景にある画像）、mat は透明度を計算する前のマテリアルの色です。

- 不透明モード A_ONE の場合:

```
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)
```

- 不透明モード RGB_ZERO の場合:

```
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)
```

- 不透明モード A_ZERO の場合:

```
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)
```

- 不透明モード RGB_ONE の場合:

```
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)
```

ただし、輝度 (luminance) は、ISO/CIE 色標準 (ITU-R 勧告 BT.709-4 を参照) に基づいた関数で、以下のように色チャンネルを平均した値です。

```
luminance = (color.r * 0.212671) +
(color.g * 0.715160) +
(color.b * 0.072169)
```

<transparent>と<transparency>の関係は以下のようになります。

- `<transparent>`が存在しない場合には、`<transparent>`は式の結果に影響を与えません。不透明モードは、デフォルトの不透明モードのままです。つまり、次の式と同じことになります。

```
transparent = <color> 1.0 1.0 1.0 1.0 </color>
```

- `<transparency>`が存在しない場合、`<transparency>`は式の結果に影響を与えません。つまり、係数が 1.0 であるのと同じことになります。

```
transparency = <float> 1.0 <float>
```

- `<transparent>`と`<transparency>`の両方が存在する場合には、どちらも尊重されます。

A_ONE は、透明度情報を RGB チャンネルではなくアルファチャンネルから取得することを指定するので、以下の例では、色としては指定どおりの色が使われますが、透明度の計算では RGB 値は無視されます。

```
<transparent opaque=A_ONE><color>1 0 0.5 0</color></transparent>
```

テクスチャリング

<profile_COMMON>のテクスチャマッピング

このセクションでは、サンプラとイメージについて概説します。

イメージをテクスチャとして使用するには、次のような要素関係を使用します。

```
texture->sampler->image
```

以下では、最小の要素から順に説明します。

- `<image>`要素は、画像データを格納するために設計された、単一の密接構造を構成しています。現在の画像フォーマットの多くは、MIP マッピング、キューブマップ、ボリウムスライスなどの追加情報を格納することができるので、`<image>`要素はこのような 3D ハードウェアに関する概念も含むことができます。`<image>`には、ファイルデータが埋め込まれているか、もしくは、参照されています。これは、従来の 2D 平面の形式 (BMP など) でもかまいませんし、複数のイメージ平面から成る複雑な 3D 形式 (DDS や OpenEXR など) でもかまいません。
- `<sampler*>`には、`<image>`の中の特定の 1D、2D、3D 座標からデータを読み込む方法に関する指示が含まれます。この要素は、`<image>`を参照して、指定された座標からデータをサンプリングするために、どのような操作を行うべきかを指定します。サンプラの指示には、座標からイメージへのマッピング方法 (ラップやミラーなど) に関する情報が含まれます。また、この指示には、最終的に出力する色を生成するために、近傍座標の複数のテクセルをどう合成すればよいかを指定するフィルタリングモードも含まれます。
- `profile_COMMON` の`<texture>`の役割は、ジオメトリのテクスチャ座標セット (配列) を `<sampler*>`にバインドすることにより、サンプラが正しい色を取得できるようにすることです。`<texture>`の `texcoord` 属性は、実際にはセマンティックの名前です。`<instance_geometry>` の`<instance_material><bind_vertex_input>`は、`<texture>`の `texcoord` 属性と、メッシュのテクスチャ座標配列との間の関連を設定するものと想定されています。

また、DCC アプリケーションの中には、`TEXCOORD` がサンプラにプラグインされる前に、これを変更する `offsetU`、`offsetV`、`rotateUV`、`noise` などの`<extra>`情報を指定するものもあります。

以下は、`<sampler2D>`パラメータによって提供された`<image>`をもつ材料をインスタンス化するための`<instance_material>`および関連する要素を利用したテクスチャリングの例です。

```
...
<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>
      ...
    </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>

```


8章 FX リファレンス

概要

このセクションでは、COLLADA FX を構成する要素を扱います。

要素のカテゴリ分類

この章では、要素をアルファベット順に記載します。関連する要素を見つけやすくするため、以下の表では、要素をカテゴリ別に記載しています。

効果

<code>annotate</code>	強く型付けされた注釈を親オブジェクトに追加します。
<code>bind_vertex_input</code>	インスタンス化の際に、ジオメトリ頂点入力をエフェクト頂点入力にバインドします。
<code>effect</code>	COLLADA エフェクトの記述を表します。
<code>instance_effect</code>	COLLADA エフェクトをインスタンス化します。
<code>library_effects</code>	<code><effect></code> アセットを保存するライブラリを提供します。
<code>technique</code> (FX)	ある方法を利用してエフェクトをレンダリングするために必要なテクスチャやサンブラ、シェーダ、パラメータ、パスに関する記述を保持します。
<code>technique_hint</code>	エフェクトで利用するテクニックのプラットフォームのヒントを追加します。

マテリアル

<code>bind</code> (FX)	値をシェーダへのユニフォーム入力にバインドします。もしくは、インスタンス化時にエフェクトパラメータに値をバインドします。
<code>bind_material</code>	特定のマテリアルをジオメトリの一部にバインドし、同時に可変パラメータとユニフォームパラメータをバインドします。
<code>instance_material</code> (ジオメトリ)	COLLADA マテリアルリソースをインスタンス化します。
<code>library_materials</code>	<code><material></code> アセットを保存するライブラリを提供します。
<code>material</code>	ジオメトリの外観、および画面領域の画像処理に必要な数式を定義します

パラメータ

<code>array</code>	1次元配列型のパラメータを作成します。
<code>modifier</code>	<code><newparam></code> 宣言の可変性またはリンクに関する追加情報を指定します。
<code>newparam</code>	新たに名前付きパラメータオブジェクトを作成して、型と初期値を割り当てます。詳しくは、5章 コア要素のリファレンスを参照してください。
<code>param</code> (reference)	定義済みのパラメータを参照します。詳しくは、5章 コア要素のリファレンスを参照してください。
<code>sampler_image</code>	サンブラの対象となる画像をインスタンス化します。
<code>sampler_states</code>	マテリアルからエフェクトのサンブラ状態を変更することを可能にします。
<code>semantic</code>	パラメータ宣言の目的を記述するメタデータを指定します。
<code>setparam</code>	先に定義されたパラメータに新しい値を割り当てます。5章 コア要素のリファレンスの主見出し項目を参照してください。

<code>usertype</code>	パラメータ用の構造化されたクラスのインスタンスを作成します。
-----------------------	--------------------------------

プロファイル

<code>profile_BRIDGE</code>	外部規格で書かれたエフェクトプロファイルの参照に対するサポートを提供します。
<code>profile_CG</code>	NVIDIA® Cg 言語で書かれたエフェクトのプラットフォーム固有の表現を宣言します。
<code>profile_COMMON</code>	プラットフォームに関係のない共通の固定機能シェーダの宣言ブロックをオープンします。
<code>profile_GLES</code>	OpenGL ES のためのプラットフォーム固有のデータ型および<technique>を宣言します。
<code>profile_GLES2</code>	OpenGL ES 2.0 のためのプラットフォーム固有のデータ型および<technique>を宣言します。
<code>profile_GLSL</code>	OpenGL シェーディング言語のためのプラットフォーム固有のデータ型および<technique>を宣言します。

レンダリング

<code>blinn</code>	Blinn BRDF 近似を使って陰影のある面を生成します。
<code>color_clear</code>	レンダリングターゲット画像をクリアするかどうか、クリアする場合にはどんな値を利用するのかを指定します。
<code>color_target</code>	特定のパスで、カラーの情報を出力から受け取る<image>を指定します。
<code>fx_common_color_or_texture_type</code> 以下の要素が含まれます。 <code>ambient (FX)</code> <code>diffuse</code> <code>emission</code> <code>reflective</code> <code>specular</code> <code>transparent</code>	<profile_COMMON>効果内の固定機能シェーダ要素のカラー属性を記述するタイプ。
<code>fx_common_float_or_param_type</code> 以下の要素が含まれます。 <code>index_of_refraction</code> <code>reflectivity</code> <code>shininess</code> <code>transparency</code>	<profile_COMMON>エフェクト内の固定機能シェーダ要素のスカラ属性を記述するタイプ。主見出し項目を参照してください。
<code>constant</code>	ライティングに影響されない、固定的な陰影のある面を生成します。
<code>depth_clear</code>	レンダリングターゲット画像をクリアするかどうか、クリアする場合にはどんな値を利用するのかを指定します。
<code>depth_target</code>	深度情報をパス出力から受け取る<image>を指定します。
<code>draw</code>	どんな種類のジオメトリを提出するのかを FX ランタイムに対して指示します。
<code>evaluate</code>	レンダリングパス用の評価要素が含まれます。
<code>instance_material (レンダリング)</code>	画面エフェクト用の COLLADA マテリアルリソースをインスタンス化します。
<code>lambert</code>	ライティングに影響されない拡散シェーディングのある面を生成します。
<code>pass</code>	1つのレンダリングパイプラインのすべてのレンダリングステートとシェーダと設定を静的に宣言します。
<code>phong</code>	Phong BRDF 近似に従って鏡面反射をシェーディングした、陰影のある面を生成します。
<code>render</code>	シーンを評価するための単一のエフェクトパスを記述します。
<code>states</code>	親パスのために設定されたあらゆるレンダリング状態が含まれます。

<code>stencil_clear</code>	レンダリングターゲット画像をクリアするかどうか、クリアする場合にはどんな値を利用するのかを指定します。
<code>stencil_target</code>	このパスの出力からステンシル情報を受け取る<image>を指定します。

シェーダ

<code>binary</code>	シェーダをバイナリ形式で特定もしくは提供します。
<code>bind_attribute</code>	シェーダの頂点属性入力に、セマンティックスをバインドします。
<code>bind_uniform</code>	値をシェーダへのユニフォーム入力にバインドします。もしくは、インスタンス化時に効果パラメータに値をバインドします。
<code>code</code>	ソースコードのインラインブロック
<code>compiler</code>	シェーダコンパイラ用のコマンドラインもしくはランタイム呼出しオプションが含まれます。
<code>include</code>	外部リソースを参照して、ソースコードまたは事前にコンパイルされたバイナリシェーダをFXランタイムにインポートします。
<code>linker</code>	シェーダをプログラムに組み込むためのシェーダリンク用の、コマンドラインもしくはランタイム呼出しオプションが含まれます。
<code>program</code>	複数のシェーダをリンクして、ジオメトリ処理用のパイプラインを生成します。
<code>shader</code>	<pass>のレンダリングパイプラインで実行するシェーダを宣言して準備します。
<code>sources</code>	1つまたは複数のソースからのシェーダ用ソースコードを連結します。

テクスチャリング

<code>alpha</code>	合成モードテクスチャリング用の<texture_pipeline>コマンドのアルファ部分を定義します。
<code>argument</code>	テクスチャユニットの合成スタイルのテクスチャコマンドが持つRGBまたはアルファコンポーネントの引数を定義する。
<code>create_2d</code>	2次元<image>アセットの手作業での作成を支援します。
<code>create_3d</code>	3次元<image>アセットの手作業での作成を支援します。
<code>create_cube</code>	立方体<image>アセットを初期化します。
<code>format</code>	<image>アセットに期待されるフォーマットやメモリレイアウトを記述します。
<code>image</code>	オブジェクトのグラフィカルな表現の保存場所を宣言します。
<code>init_from</code>	画像全体もしくは画像の一部を、参照されたもしくは埋め込まれたデータで初期化します。
<code>instance_image</code>	シェーダの中で使う画像をインスタンス化します。
<code>library_images</code>	<image>アセットを保存するライブラリを提供します。
<code>RGB</code>	合成モードテクスチャリング用の<texture_pipeline>コマンドのRGB部分を定義します。
<code>fx_sampler_common</code>	<sampler*>要素のサンプリング状態を記述する型。
<code>sampler1D</code>	1次元のテクスチャサンプラを宣言します。
<code>sampler2D</code>	2次元のテクスチャサンプラを宣言します。
<code>sampler3D</code>	3次元のテクスチャサンプラを宣言します。
<code>samplerCUBE</code>	キューブマップ用のテクスチャサンプラを宣言します。
<code>samplerDEPTH</code>	奥行きマップのテクスチャサンプラを宣言します。
<code>samplerRECT</code>	RECT テクスチャサンプラを宣言します。
<code>texcombiner</code>	合成モードテクスチャリング用の<texture_pipeline>コマンドを定義します。

<code>texenv</code>	単純な非合成モードテクスチャリング用の< <code>texture_pipeline</code> >コマンドを定義します。
<code>texture_pipeline</code>	通常モードと合成モードで <code>glTexEnv</code> を利用してマルチテクスチャ操作に変換されるテクスチャコマンドセットを定義します。

COLLADA FX について

詳しくは、7章 FX 入門を参照してください。

alpha

カテゴリ: テクスチャリング

プロファイル: GLES

概要

合成モードテクスチャリング用の<`texture_pipeline`>コマンドのアルファ部分を定義します。

コンセプト

割り当てと全体的なコンセプトの詳細に関しては<`texcombiner`>の解説を参照してください。

属性

<`alpha`>要素には、以下の属性があります。

<code>operator</code>	列挙	<code>glTexEnv(TEXTURE_ENV, COMBINE_ALPHA, operator)</code> での利用を想定。オプション。詳細は < <code>texcombiner</code> >を参照してください。有効な値は、以下の通りです。REPLACE MODULATE ADD ADD_SIGNED INTERPOLATE SUBTRACT
<code>scale</code>	<code>float_type</code>	<code>glTexEnv(TEXTURE_ENV, ALPHA_SCALE, scale)</code> での利用を想定。オプション。詳細は < <code>texcombiner</code> >を参照してください。

関連要素

<`alpha`>要素は、以下の要素と関連性があります。

親要素	<code>texcombiner</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
< <code>argument</code> >	実行する特定の操作に必要な引数を設定します。主見出し項目を参照してください。	なし	1 から 3

詳細

詳細は<`texcombiner`>を参照してください。

例

`<texture_pipeline>`を参照してください。

annotate

カテゴリ: 効果

プロファイル: すべて、および外部

概要

強く型付けされた注釈を親オブジェクトに追加します。

コンセプト

注釈は、`symbol = value` という形式のオブジェクトを表わします。ただし、`symbol` は、`name` 属性で指定されるユーザ定義の識別子、`value` は、子要素として指定される強く型付けされた値です。注釈は、エフェクトランタイムからアプリケーションに対してメタデータを知らせるためだけに利用され、COLLADA ドキュメントでは解釈されません。

属性

`<annotate>`要素には、以下の属性があります。

name	xs:token	<code>symbol = value</code> 形式のオブジェクト内の <code>symbol</code> を表す要素のテキスト文字列名。必須。

関連要素

`<annotate>`要素は、以下の要素と関連性があります。

親要素	<code>effect</code> 、 <code>technique</code> (FX)、 <code>pass</code> 、 <code>newparam</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<code>value_element</code>	<code>symbol = value</code> 形式のオブジェクト内の <code>value</code> を表す、強く型付けされた値。その型の値を含む COLLADA 型要素から構成されています。有効な型要素は、以下の通りです。 <code>bool</code> 、 <code>bool2</code> 、 <code>bool3</code> 、 <code>bool4</code> 、 <code>int</code> 、 <code>int2</code> 、 <code>int3</code> 、 <code>int4</code> 、 <code>float</code> 、 <code>float2</code> 、 <code>float3</code> 、 <code>float4</code> 、 <code>float2x2</code> 、 <code>float3x3</code> 、 <code>float4x4</code> 、 <code>string</code> 詳しくは、11 章 型を参照してください。	なし	1

詳細

現時点では、注釈の標準セットはありません。

例

```
<annotate name="UIWidget"> <string> slider </string> </annotate>
```

```
<annotate name="UIMinValue"> <float> 0.0 </float> </annotate>
<annotate name="UIMaxValue"> <float> 255.0 </float> </annotate>
```

argument

カテゴリ: テクスチャリング

プロファイル: GLES

概要

テクスチャユニットの合成スタイルのテクスチャコマンドが持つ RGB またはアルファコンポーネントの引数を定義する。

コンセプト

割り当てと全体のコンセプトの詳細に関しては、<texture_pipeline>の解説を参照してください。

この要素は、親要素を基にしたコンテキストに依存します。

属性

<argument>要素には、以下の属性があります。

注:以下の表で、##は結合を意味し、*idx* は、引数が親コマンド (<texenv>または<texcombiner>) 内に記述されたインデックスを表します。sourcevalue は、値を指定する場所を意味します。

source	列挙	オプション。引数のソースデータがどこから来るのかを表します。 親が <RGB> の場合、glTexEnv(TEXTURE_ENV, SRC##idx##_RGB, sourcevalue) の呼び出しを意味します。 親が <alpha> の場合、glTexEnv(TEXTURE_ENV, SRC##idx##_ALPHA, sourcevalue) の呼び出しを意味します。 TEXTURE CONSTANT PRIMARY PREVIOUS を指定できます。デフォルト値はありません。
operand	列挙	オプション。ソースから値を読み込む方法の詳細を提供します。 親が <RGB> の場合、glTexEnv(TEXTURE_ENV, OPERAND##idx##_RGB, sourcevalue) の呼び出しを意味し、指定できる値は SRC_COLOR ONE_MINUS_SRC_COLOR SRC_ALPHA ONE_MINUS_SRC_ALPHA。デフォルト値は、SRC_COLOR です。 親が <alpha> の場合、glTexEnv(TEXTURE_ENV, OPERAND##idx##_ALPHA, sourcevalue) の呼び出しを意味し、指定できる値は SRC_ALPHA ONE_MINUS_SRC_ALPHA。デフォルト値は、SRC_ALPHA です。
sampler	xs:NCName	オプション。ソースを読み込むサンプラの <newparam> の名前。 source="TEXTURE" の場合にだけ利用できます。指定可能な値は、シェーダの設計対象となる OpenGL ES のバージョンに依存します。 GL ES 1.0 の場合、<texenv> 要素中のすべての引数は、同じ <newparam> を参照しなければなりません。合成クロスバーがないからです。 GL ES 1.1 の場合、テクスチャの合成クロスバーが利用可能であるため、この属性は任意の <newparam> を参照することができます。

関連要素

`<argument>`要素は、以下の要素と関連性があります。

親要素	RGB, alpha
子要素	なし
その他	なし

詳細

`<argument>`は、特定の操作を実行する際に必要となる引数を設定します。

例

`<texture_pipeline>`を参照してください。

array

カテゴリ: パラメータ

プロファイル: CG、GLES2、GLSL

概要

1次元配列型のパラメータを作成します。

コンセプト

配列型のパラメータは、要素の並びをシェーダに渡すために利用されます。配列型は、単一のデータ型を並べたものです。多次元配列を作成するには、配列型の配列として宣言します。

配列の宣言は、サイズ指定をしてもしなくてもかまいません。サイズ指定のない配列は、シェーダのパラメータとして利用する前に、`<setparam>`を利用して具体的なサイズ（とデータ）を設定する必要があります。

属性

`<array>`要素には、以下の属性があります。

<code>length</code>	<code>xs:positiveInteger</code>	必須。配列中の要素の数。
<code>resizable</code>	<code>xs:boolean</code>	オプション。Cg スコープ内でのみ有効です。true の場合、この配列は Cg の可変長配列と連動しているので、変更時にサイズを変更することができます。デフォルトは、false です。

関連要素

`<array>`要素は、以下の要素と関連性があります。

親要素	コア要素: <code>newparam</code> , <code>setparam</code> FX 要素: <code>create_2d</code> , <code>create_3d</code> , <code>create_cube</code>
子要素	<code>create2d</code> , <code>create3d</code> , <code>createcube</code> の場合はなし。それ以外は、以下のサブセクションを参照してください。
その他	なし

CG スコープ内の子要素。

名前/例	解説	デフォルト値	出現回数
<i>parameter_element</i>	<p>対応するスコープ内で有効なパラメータ型の要素については、章末の「パラメータの型要素」を参照してください。</p> <p>CG: cg_param_group</p> <p>GLSL: glsl_value_group</p> <p>GLSL2: gles2_value_group</p> <p>また、各パラメータの型グループには、子要素として<array>要素が含まれています。配列の中で追加の次元を宣言するには、追加の子配列要素を使います。</p> <p>パラメータの型の組合せの制限については、「詳細」の中の「注意」を参照してください。</p>	なし	0 以上

詳細

子要素の記法に関する注意：以下の記法は、スキーマによって強制することはできませんが、配列を有効なものにするには、これらに従う必要があります。

- パラメータの型子要素の値の型は、すべて統一されている必要があります。
- パラメータの型子要素が1つだけ存在するか、もしくは、パラメータの型要素の数が、length 属性に一致している必要があります。前者の場合、そのパラメータの型によって配列の型が設定され、その値を使って配列全体が初期化されます。

作成後の配列要素は、<setparam>宣言の中で、C/C++の通常の配列インデックス指定の構文を使って、直接参照することができます。たとえば、以下の例において、配列の4番目の要素にアクセスするには、「numbers[3]」が使われます。

例

```
<newparam sid="numbers">
  <array length="4">
    <float>1</float>
    <float>2</float>
    <float>3</float>
    <float>4.0</float>
  </array>
</newparam>
<setparam ref="numbers[2]">
  <float>2.5</float>
</setparam>
```

長方形の配列 (2x3) の例。

```
<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>
```

長さが不揃いな配列の例。

```
<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>
```


binary

カテゴリ: シェーダ

プロファイル: CG、GLES2

概要

シェーダをバイナリ形式で特定もしくは提供します。

コンセプト

GLES2 プロファイルは、ソースコードが存在していて、バイナリを再生成できるように設計されていますが、この要素は、プリコンパイルされたシェーダが必要、もしくは、それによって利益を受けるようなプラットフォームのためにあります。

バイナリは、通常、GLES2 の `glShaderBinary` という API 関数やその他のバイナリ拡張と連携して作用する、オフラインコンパイラによって生成されます。`<binary>` 要素は、`<program>` 要素中の 1 つ以上のレベルに出現します。`<program>` 要素の中では、GLES2 用のプロファイルによってシェーダ情報、コンパイラ設定、リンカ設定が記述されているので、バイナリを再生成することができます。

属性

`<binary>` 要素には属性はありません。

関連要素

`<binary>` 要素は、以下の要素と関連性があります。

親要素	<code>compiler, linker</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

以下の要素のいずれかがちょうど 1 個存在する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><ref></code>	バイナリを含むファイルの URI (<code>xs:anyURI</code>) が含まれます。 この要素には属性はありません。	なし	1
<code><hex format=""></code>	バイナリコードが、16 進でエンコードされたバイナリオクテットの並びとして含まれます。 オプションの <code>format</code> 属性は、バイナリのフォーマットを記述する <code>xs:token</code> を指定します。この属性の値は、通常、該当するファイル拡張子です。	なし	1

詳細

バイナリは、`<ref>` 経由で外部ファイルから取得することも、`<hex>` を使ってインスタンス文書に埋め込むこともできます。`<ref>` の場合、ファイル拡張子や埋め込まれた情報を使って、データのフォーマットを示す必要があります。逆に `<hex>` の場合、ファイル拡張子を利用できないので、`format` 属性を使って追加のフォーマット情報を渡します。

例

```
<binary><ref>file://c:/test/vertexShader.bin</ref></binary>

<binary><hex format="COMPANY_PLATFORM">0123456789ABCDEF</hex></binary>
```

bind

(FX)

カテゴリ: **マテリアル**

プロファイル: **外部**

概要

値をシェーダへのユニフォーム入力にバインドします。もしくは、インスタンス化時に効果パラメータに値をバインドします。

コンセプト

ユニフォームパラメータを持ったシェーダは、コンパイル時に各入力に値をバインドすることができ、また実行時には、このユニフォームパラメータに値を割り当てる必要があります。これらの値は、リテラル値、定数パラメータ、ユニフォームパラメータのいずれでもかまいません。定数値の場合、コンパイラは、このようなシェーダパラメータの宣言を利用して、特定の宣言用に最適化されたシェーダを生成します。

また<bind>要素は、事前に定義されたパラメータを実行時に均一の入力にマッピングするのにも利用され、事前に定義されたパラメータのプールから FX ランタイムで自動的に値をシェーダに割り当てることが可能です。

属性

<bind>要素には、以下の属性があります。

semantic	xs:NCName	どの効果パラメータをバインドするかを指定します。必須。
target	sidref_type	指定したセマンティックスにバインドする値の SID の参照です。このテキスト文字列は、「アドレス構文」のセクションに解説されている構文にしたがったパス名です。必須。

関連要素

<bind>要素は、以下の要素と関連性があります。

親要素	instance_material (ジオメトリ)、 instance_material (レンダリング)
子要素	なし
その他	なし

詳細

一部の FX ランタイムコンパイラでは、コンパイルの前にすべてのユニフォーム入力をバインドしておく必要があります。それ以外の FX ランタイムでは、バインドされていない入力をチェックできるように、シェーダを非実行可能形式のオブジェクトコードに部分コンパイルすることができます。

`<bind>`要素や`<bind_vertex_input>`要素は、ターゲットを`<effect>`の中のパラメータにバインドします。`<effect>`の中のパラメータを特定する検索文字列は、`semantic` 属性によって指定されます。`<effect>`の中のパラメータを特定する際には、以下の順序で検索します。

- COLLADA FX パラメータをセマンティックによって探します。
- プロファイルの中にシェーディング言語コードが含まれている場合には、シェーダの中からセマンティックによってパラメータを探します。
- COLLADA FX パラメータを SID によって探します。
- プロファイルの中にシェーディング言語コードが含まれている場合には、シェーダの中から名前によってパラメータを探します。

例

```
<instance_material symbol="RedMat" target="#RedCGEffect">
  <bind semantic="LIGHTPOS0" target="LightNode/translate"/>
</instance_material>
```

bind_attribute

カテゴリ: シェーダ

プロファイル: GLES2、GLSL

概要

シェーダの頂点属性入力に、セマンティックスをバインドします。

コンセプト

頂点属性変数を持つシェーダが使う変数名は、ジオメトリ頂点入力のセマンティック名と一致しない可能性があります。この要素は、シェーダの頂点属性に別のセマンティック名を追加することを可能にし、実行時システムが、頂点属性をより簡単に特定してジオメトリに関連付けられるようにします。

属性

`<bind_attribute>`要素には、以下の属性があります。

symbol	xs:token	シェーダ内の頂点属性変数の識別子(正式な関数パラメータもしくはスコープ内グローバル変数)。必須。
--------	----------	--

関連要素

`<bind_attribute>`要素は、以下の要素と関連性があります。

親要素	program
子要素	semantic
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<code><semantic></code>	ジオメトリ頂点入力へのセマンティックバインディングのための属性変数の別名を含む <code>xs:token</code> 。主見出し項目を参照してください。	なし	1

詳細

特定の変数用の `<bind_attribute>` が存在しない場合、FX ランタイムは、属性名に直接バインドすることを試みる必要があります。

例

```

    <program>
  <shader stage="VERTEX" />
  ...
  <bind_attribute symbol="pos">
    <semantic> POSITION </semantic>
  </bind_attribute>
  <bind_attribute symbol="diffusecol">
    <semantic> COLOR </semantic>
  </bind_attribute>
  </program>

```

bind_material

カテゴリ: **マテリアル**

プロファイル: **外部**

概要

特定のマテリアルをジオメトリの一部にバインドし、同時に可変パラメータとユニフォームパラメータをバインドします。

コンセプト

ジオメトリの一部を宣言する際に、以下のように特定のマテリアルを持つことを要求することができます。

```
<polygons name="leftarm" count="2445" material="bluePaint">
```

この抽象シンボルは、特定のマテリアルインスタンスにバインドされる必要があります。アプリケーションは、`<bind_material>` 要素の中の `<instance_geometry>` 要素を処理する際に、インスタンス化を行います。アプリケーションは、ジオメトリから `material` 属性をスキャンして、`<instance_material>` (ジオメトリ) の `symbol` 属性によって指定される、実際のマテリアルオブジェクトをバインドします。詳細は、下の「例」を参照してください。

マテリアルをバインドしたら、シェーダパラメータも解決する必要があります。たとえば、入力として 2 つの光源の位置が必要な効果で、シーンに 8 つのユニークな光源が含まれている場合、どの 2 つの光源がマテリアルに利用されるのでしょうか？ また、エフェクトがオブジェクトごとに必要とするテクスチャ座標のセットは 1 つなのに、ジオメトリが定義するテクスチャ座標のセットは 2 つある場合、そのエフェクトにはどのセットが利用されるのでしょうか。そういったシーングラフ中の入力の曖昧さを解消するためのメカニズムが `<bind_material>` です。

パラメータにアタッチされたセマンティックスを指定し、また COLLADA の URL シンタックスでシーングラフ中のノードの個々の要素、最後にはベクトルの個々の要素に結び付けることで、それぞれの入力はシーングラフにバインドされます。

属性

`<bind_material>` 要素には属性はありません。

関連要素

`<bind_material>` 要素は、以下の要素と関連性があります。

親要素	<code>instance_geometry</code> , <code>instance_controller</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><param></code> (core)	<code><bind_material></code> で、これらは追加されてアニメーションのターゲットになります。このようなオブジェクトについて、アニメーションターゲットシステムは <code><effect></code> の内部レイアウトを解析する必要はなく、通常の方法で入力パラメータにバインドすることができます。コアの主見出し項目を参照してください。	なし	0 以上
<code><technique_common></code>	すべての COLLADA 実装がサポートしなければならない共通プロファイルのマテリアルのバインド情報を指定します。詳しくは、「共通プロファイル」の使い方情報のセクション、およびそれに続く子要素詳細についてのサブセクションを参照してください。	なし	1
<code><technique></code> (core)	各 <code><technique></code> は、 <code><technique></code> の <code>profile</code> 属性として指定された特定のプロファイルのためのマテリアルバインド情報を指定します。コアの主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

`<bind_material>` / `<technique_common>` の子要素

名前/例	解説	デフォルト値	出現回数
<code><instance_material></code> (ジオメトリ)	主見出し項目を参照してください。	なし	1 以上

詳細

例

```
<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>

```

以下の例は、マテリアルをジオメトリにバインドする<bind_material>を示しています。<material>要素の id 属性と<polygons>要素の material 属性との間の関連付けは、<instance_material> (マテリアル) 要素によって確立されます。

```

...
<material id="MyMaterial"> ... </material>
...
<geometry>
    ...
    <polygons name="leftarm" count="2445" material="bluePaint">
    ...
</geometry>
...
<scene>
    ...
    <instance_geometry ...>
        <bind_material>
            <technique_common>
                <instance_material symbol="bluePaint" target="MyMaterial">
                    ...
                </instance_material>
            </technique_common>
        </bind_material>
    </instance_geometry>
    ...
</scene>

```

bind_uniform

カテゴリ: シェーダ

プロファイル: CG、GLES2、GLSL

概要

値をシェーダへのユニフォーム入力にバインドします。もしくは、インスタンス化時に効果パラメータに値をバインドします。

コンセプト

ユニフォームパラメータを持ったシェーダは、コンパイル時に各入力に値をバインドすることができ、また実行時には、このユニフォームパラメータに値を割り当てる必要があります。これらの値は、リテラル値、定数パラメータ、ユニフォームパラメータのいずれでもかまいません。定数値の場合、コンパイラは、このようなシェーダパラメータの宣言を利用して、特定の宣言用に最適化されたシェーダを生成します。

また<bind_uniform>要素は、事前に定義されたパラメータを実行時に均一の入力にマッピングするのもにも利用され、事前に定義されたパラメータのプールから FX ランタイムで自動的に値をシェーダに割り当てるのが可能です。

属性

<bind_uniform>要素には、以下の属性があります。

symbol	xs:NCName	シェーダへのユニフォーム入力パラメータのための識別子(正式な関数パラメータまたは特定の範囲でのグローバルな識別子)で、外部リソースにバインドされません。必須。
--------	-----------	---

関連要素

<bind_uniform>要素は、以下の要素と関連性があります。

親要素	shader (CG)、program (GLES2 および GLSL)
子要素	下記サブセクションを参照してください。
その他	なし

子要素

注:子要素<param>または値タイプのいずれか1つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<param>(reference)	主見出し項目を参照してください。	なし	「注」を参照してください。

名前/例	解説	デフォルト値	出現回数
<i>parameter_type</i> <i>_element</i>	CG、GLES2、GLSL スコープ内で有効なパラメータ型の要素については、章末の「パラメータの型要素」を参照してください。 CG: <code>cg_param_group</code> GLSL: <code>glsl_value_group</code> GLES2: <code>gles2_value_group</code>	なし	「注」を参照してください。

詳細

一部の FX ランタイムコンパイラでは、コンパイルの前にすべてのユニフォーム入力をバインドしておく必要があります。それ以外の FX ランタイムでは、バインドされていない入力をチェックできるように、シェーダを非実行可能形式のオブジェクトコードに部分コンパイルすることができます。

例

```

        <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>
    </param>
  </bind_uniform>
  </shader>

```

bind_vertex_input

カテゴリ: **マテリアル**

プロファイル: **外部**

概要

インスタンス化の際に、ジオメトリ頂点入力をエフェクト頂点入力にバインドします。

コンセプト

この要素は、たとえば頂点プログラムパラメータを `<source>` にバインドする際に役に立ちます。頂点プログラムは、既にソースから収集されたデータを必要とします。このデータは、`<polygons>` や `<triangles>` などの照合要素の下の `<input>` 要素からもたらされます。入力は、`<source>` 中のデータにアクセスし、そのデータがポリゴン頂点"fetch"に対応していることを保証します。バインドするために `<input>` を参照するには、`<bind_vertex_input>` を使用してください。

属性

`<bind_vertex_input>` 要素には、以下の属性があります。

<code>semantic</code>	<code>xs:NCName</code>	どの効果パラメータをバインドするかを指定します。必須。
<code>input_semantic</code>	<code>xs:NCName</code>	どの入力セマンティックをバインドするかを指定します。必須。
<code>input_set</code>	<code>uint_type</code>	どの入力セットをバインドするかを指定します。オプション。

関連要素

`<bind_vertex_input>`要素は、以下の要素と関連性があります。

親要素	<code>instance_material</code> (ジオメトリ)
子要素	なし
その他	<code>input</code>

詳細

`<bind_vertex_input>`要素は、(`<geometry>`要素内の `<input>`要素として識別された) ジオメトリ頂点ストリームをマテリアルエフェクト頂点ストリームセマンティックスにバインドします。アプリケーションは通常、同一のセマンティック識別子を用いて頂点ストリームの自動バインディングを実行しますが、セマンティック識別子の意味において食い違いが頻繁に起こります。`<bind_vertex_input>`を使用して、通常以下のようなことによってもたらされるこれらの曖昧さを取り除きます。

- 一般化。たとえば、`TEXCOORD0` 対 `DIFFUSE-TEXCOORD`。
- スペリングの違い。たとえば、`COLOR` 対 `COLOUR`
- 省略形
- 冗長性
- 同義語

`<bind>`要素や`<bind_vertex_input>`要素は、ターゲットを`<effect>`の中のパラメータにバインドします。`<effect>`の中のパラメータを特定する検索文字列は、`semantic` 属性によって指定されます。`<effect>`の中のパラメータを特定する際には、以下の順序で検索します。

- COLLADA FX パラメータをセマンティックによって探します。
- プロファイルの中にシェーディング言語コードが含まれている場合には、シェーダの中からセマンティックによってパラメータを探します。
- COLLADA FX パラメータを `sid` によって探します。
- プロファイルの中にシェーディング言語コードが含まれている場合には、シェーダの中から名前によってパラメータを探します。

例

下の例では、濡れた羽毛マテリアルをカモモデルに適用します。カモモデルは、通常のマップテクスチャ座標およびベースカラーテクスチャ座標を持つことができます。通常のマップテクスチャ座標は `TEXCOORD0` (`semantic=TEXCOORD` および `set=0`) と呼ばれ、ベースカラーテクスチャ座標は `TEXCOORD1` と呼ばれます。

テクスチャ座標 (または他のジオメトリストリーム) のセマンティック名がマッチしない環境があります。たとえば、濡れた羽毛マテリアルは、`TEXCOORD1` と呼ばれる通常のマップテクスチャ座標および `TEXCOORD0` と呼ばれるベースカラーテクスチャ座標を持つかもしれませんが。この場合、これら同一名称の意味がそのようにスワップされているので、これらの一致しないオブジェクトをバインドするには、`<bind_vertex_input>` を用います。

`semantic` 属性はマテリアルエフェクト内のセマンティックを参照するのに対し、接頭辞「`input_`」を持つ属性はジオメトリ頂点`<bind_vertex_input>`ストリームを参照し、そのストリームはセマンティック名とセット番号の組み合わせによって識別されることに注意してください。

```
<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>

```

blinn

カテゴリ: レンダリング

プロファイル: 共通

概要

Blinn BRDF 近似を使って陰影のある面を生成します。

コンセプト

<blinn>は、**<profile_COMMON>**エフェクトの中に使われ、Blinn-Torrance-Sparrow 光源モデルや類似の近似にしたがって陰影のついた曲面を生成する、固定機能パイプラインを宣言します。

この式は複雑であり、詳しい説明は ACM のサイトにあるので、ここでは詳しい説明は省きます。詳しくは、「Models of Light Reflection for Computer Synthesized Pictures」、SIGGRAPH 77, pp 192-198 (<http://portal.acm.org/citation.cfm?id=563893>) を参照してください。

最大限の互換性を確保するには:

アプリケーションの互換性を最大限にするために、**<shininess>**が 0~1 の範囲にある場合には、Blinn-Torrance-Sparrow を利用することをお勧めします。COLLADA のディベロッパは、**<shininess>**が 1.0 を超える場合には、Blinn-Phong 光源モデルに従ったアプリケーションを利用していたはずなので、shininess の範囲に応じて両方の Blinn 等式をサポートすることをお勧めします。

Blinn-Phong の等式

Blinn-Phong の等式は、次の通りです。

$$color = \langle emission \rangle + \langle ambient \rangle * al + \langle diffuse \rangle * \max(N \cdot L, 0) + \langle specular \rangle * \max(H \cdot N, 0)^{\langle shininess \rangle}$$

- *al* – シーンからの環境光の寄与分を表す定数。共通プロファイルでは、**<visual_scene>**の中の**<light><technique_common><ambient>**の値の合計になります。
- *N* – 法線ベクトル (正規化済み)
- *L* – 光ベクトル (正規化済み)
- *I* – 視線ベクトル (正規化済み)
- *H* – 半角ベクトル。単位視線ベクトルと単位光ベクトルの中間点として、 $H = \text{normalize}(I+L)$ という式を使って計算される。

属性

<blinn>要素には属性はありません。

関連要素

<blinn>要素は、以下の要素と関連性があります。

親要素	technique (FX) (profile_COMMON 内)
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<emission>	このオブジェクトのサーフェスから放射される光源の量を宣言します。See fx_common_color_or_texture_type .	なし	0 または 1
<ambient> (FX)	このオブジェクトのサーフェスから放射される環境光の量を宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<diffuse>	このオブジェクトのサーフェスから反射される拡散光の量を宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<specular>	このオブジェクトのサーフェスから反射される鏡面反射光の量を宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<shininess>	このオブジェクトのサーフェスから鏡面反射突起部分の鏡面度または荒さを宣言します。 fx_common_float_or_param_type 型。主見出し項目を参照してください。	なし	0 または 1
<reflective>	完全鏡面反射のカラーを宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<reflectivity>	反射光源に追加する完全鏡面反射の量 (0.0~1.0) を宣言します。 fx_common_float_or_param_type 型。主見出し項目を参照してください。	なし	0 または 1
<transparent>	完全鏡面反射光源のカラーを宣言します。 詳しくは、「 fx_common_color_or_texture_type 」および7章「FX 入門」の「透明度 (不透明度) を判定する」を参照してください。	なし	0 または 1
<transparency>	反射カラーに追加する完全鏡面反射光源の量 (0.0~1.0) をスカラー値として宣言します。 fx_common_float_or_param_type 型。主見出し項目、および7章「FX 入門」の「透明度 (不透明度) を判定する」を参照してください。	なし	0 または 1
<index_of_refraction>	単一スカラーインデックスとして完全鏡面反射光源の反射インデックスを宣言します。 fx_common_float_or_param_type 型。主見出し項目を参照してください。	なし	0 または 1

詳細

例

これは、光沢のあるスペキュラハイライトが一点にある暗赤色のエフェクトの例です。

```
<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>
```

code

カテゴリ: シェーダ

プロファイル: CG、GLES2、GLSL

概要

ソースコードのインラインブロック

コンセプト

ソースコードは<effect>宣言の中にインライン化することができ、シェーダをコンパイルするために利用できます。

属性

<code>要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。他の要素からローカルにブロックを参照できるようにするためのソースコードの識別子です。オプション。詳しくは、アドレス構文 3章の「スキーマのコンセプト」を参照してください。
-----	----------	--

関連要素

<code>要素は、以下の要素と関連性があります。

親要素	profile_CG , profile_GLSL , profile_GLES2
子要素	なし
その他	なし

詳細

`xs:string` として取り込まれたインラインソースコードの中では、たとえば、「<」を「<」に変換するなどして、XML 識別子文字をすべてエスケープする必要があります。

例

```
<code sid="lighting_code">
matrix4x4 mat : MODELVIEWMATRIX;
float4 lighting_fn( varying float3 pos : POSITION,
    ...
</code>
```

color_clear

カテゴリ: レンダリング

プロファイル: CG、GLES、GLES2、GLSL

概要

レンダリングターゲット画像をクリアするかどうか、クリアする場合にはどんな値を利用するのかを指定します。

コンセプト

描画を行う前に、レンダリングターゲットの画像を、空のキャンバス、つまりデフォルト値にリセットしなければならない場合もあります。一連の<color_clear>宣言では、どの値を利用するのか指定します。クリアする指示が含まれていない場合には、ターゲットの画像はレンダリングが始まって変更されません。

属性

GLES スコープ内にある <color_clear> 要素には、属性はありません。

この要素が Cg、GLES2、GLSL スコープ内にある場合には、以下のような属性があります。

の	xs:nonNegativeInteger	マルチレンダリングターゲットのどれを設定するのか指定します。デフォルト値は 0 です。オプション。
---	-----------------------	---

関連要素

<color_clear>要素は、以下の要素と関連性があります。

親要素	evaluate
子要素	なし
その他	なし

詳細

この要素には、赤、緑、青、アルファチャンネルを表す 4 つの浮動小数点値が含まれます。

この要素がパスの中に存在するということは、特定のバックバッファやレンダリングターゲットリソースを消去する必要があることをランタイムに知らせる合図になっています。つまり、リソース中の既存の画像データのすべてを、指定された色で書きかえる必要があるということです。この要素は、リソースを何も描かれていない既知の状態にして、以後このリソースを使った操作が予測どおりに行われるようにします。

index 属性は、消去したいリソースの API のレンダリングターゲットリソースインデックスを特定します。インデックス 0 は、プライマリリソースを表します。プライマリリソースは、バックバッファ、もしくは、適当な<*_target>要素 (<color_target>、<depth_target>、<stencil_target>など) によって提供されるオーバーライドです。

現時点のプラットフォームでは、MRT (マルチレンダリングターゲット) を設定するルールがかなり制限されています。たとえば、Direct3D[®] 9 クラスプラットフォームのほとんどにおける MRT は、同じサイズとピクセルフォーマットのカラーバッファを 4 つ、全カラーバッファに対してアクティブなデプスバッファ 1 つとステンシルバッファ 1 つしか持つことができません。COLLADA FX の宣言は、この制限を緩め

るように設計されており、FX ランタイムは、実際に適用する前に<pass>の中に指定されている特定の MRT 宣言が可能であることを検証しなければならず、検証に失敗した場合にはエラーを返す必要があります。

例

```
<color_clear index="0">0.0 0.0 0.0 0.0</color_clear>
```

color_target

カテゴリ: レンダリング

プロファイル: CG、GLES、GLES2、GLSL

概要

特定のパスで、カラーの情報を出力から受け取る<image>を指定します。

コンセプト

複数レンダリングターゲット (MRT) を利用すると、フラグメントシェーダに対して、パスごとに複数の値を出力させたり、標準のデプスユニットやステンシルユニットをリダイレクトして任意のオフスクリーンバッファを読み書きさせることができます。これらの要素は、FX ランタイムに、使用すべき定義済み画像や、画像の取得元となるパラメータを教えます。

属性

GLES スコープ内にある <color_target> 要素には、属性はありません。

この要素が Cg、GLES2、GLSL スコープ内にある場合には、以下のような属性があります。

index	xs:nonNegativeInteger	マルチレンダリングターゲットのいずれか1つを指し示します。デフォルトは、0 です。オプション。
slice	xs:nonNegativeInteger	単一の MIP マップレベル、ユニークな立方体の面、3次元テクスチャのレイヤなどの、対象とする <surface> 中のサブイメージを指し示します。デフォルト値は0です。オプション。
mip	xs:nonNegativeInteger	デフォルト値は0です。オプション。
face	列挙	有効な値は、POSITIVE_X、NEGATIVE_X、POSITIVE_Y、NEGATIVE_Y、POSITIVE_Z、およびNEGATIVE_Z です。デフォルト値は、POSITIVE_X です。オプション。

関連要素

<color_target>要素は、以下の要素と関連性があります。

親要素	evaluate
子要素	下記サブセクションを参照してください。
その他	newparam, image

子要素

注:子要素として `<param>` または `<instance_image>` のいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<code><param></code> (reference)	どの画像を使用すべきかを判断するには、sampler パラメータを参照してください。主見出し項目を参照してください。	なし	0 または 1
<code><instance_image></code>	レンダリング可能な画像を直接インスタンス化します。主見出し項目を参照してください。	なし	0 または 1

詳細

現時点のプラットフォームでは、MRT を設定するルールがかなり制限されています。たとえば、Direct3D[®] 9 ハードウェアのほとんどでは、同じサイズとピクセルフォーマットのカラーバッファ 4 つ、およびすべてのカラーバッファに対してアクティブなデプスバッファ 1 つとステンシルバッファ 1 つだけがサポートされます。COLLADA FX の宣言は、この制限を緩めるように設計されており、FX ランタイムは、実際に適用する前に `<pass>` の中に指定されている特定の MRT 宣言が可能であることを検証しなければならず、検証に失敗した場合にはエラーを返す必要があります。

この要素には、`<sampler_*>` を含む `<newparam>` を参照している `<param>`、もしくは、画像に直接リンクする `<instance_image>` のいずれかが含まれます。`<color_target>` が指定されていない場合、FX ランタイムは、対象とするプラットフォーム用のデフォルトのバックバッファセットを利用します。

例

```
<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>
```


compiler

カテゴリ: シェーダ

プロファイル: CG、GLES2

概要

シェーダコンパイラ用のコマンドラインもしくはランタイム呼出しオプションが含まれます。

コンセプト

シェーダコンパイラは、シェーダプログラムソースコードを (`<sources>` を参照) を受け取って、マシンで実行可能なオブジェクトコードにコンパイルします。シェーダコンパイラは、特定の操作を実行するようにそのオブジェクトコードを設定するコマンドラインオプションを、受け入れます。

属性

`<compiler>` 要素には、以下の属性があります。

<code>platform</code>	<code>xs:string</code>	必須。複数のコンパイラ設定を識別するためのサブプラットフォーム名。
<code>target</code>	<code>xs:string</code>	オプション。ターゲットバイナリプロファイル。arbvp1、arbfpl、glslv、glself、hlslv、hlself、vs_3_0、ps_3_0 など。
<code>options</code>	<code>xs:string</code>	オプション。コンパイラオプション。

関連要素

`<compiler>` 要素は、以下の要素と関連性があります。

親要素	shader
子要素	下記サブセクションを参照してください。
その他	sources , linker

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><binary></code>	主見出し項目を参照してください。	なし	0 または 1

詳細

この要素には、ツールにテキスト文字列として渡されるコンパイルオプションのテキストが含まれます。この要素には、オプションで、コンパイル結果のバイナリ表現を含むこともできます。

例

```
<compiler platform="PC" target="arbvp1" options="--debug"/>
```

constant

(FX)

カテゴリ: レンダリング

プロファイル: 共通

概要

ライティングに影響されない、固定的な陰影のある面を生成します。

注: テクスチャコンパイナに関連する<constant>については、<texenv>や<texcombiner>を参照してください。

コンセプト

<profile_COMMON>効果の中で利用して、光源とは関係なく、シェーディング処理したサーフェスを常に生成する固定機能パイプラインを宣言します。

反射カラーは、以下のように計算されます。

$$color = <emission> + <ambient> * al$$

ただし、

- *al* – シーンからの環境光の寄与分を表す定数。共通プロファイルでは、<visual_scene>の中の<light><technique_common><ambient><color>の値の合計になります。

属性

<constant>要素には属性はありません。

関連要素

<constant>要素は、以下の要素と関連性があります。

親要素	technique (FX) (profile_COMMON 内)
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<emission>	このオブジェクトのサーフェスから放射される光源の量を宣言します。 「fx_common_color_or_texture_type」を参照してください。	なし	0 または 1
<reflective>	完全鏡面反射のカラーを宣言します。 「fx_common_color_or_texture_type」を参照してください。	なし	0 または 1

名前/例	解説	デフォルト値	出現回数
<code><reflectivity></code>	反射光源に追加する完全鏡面反射の量 (0.0~1.0) を宣言します。詳しくは、 fx_common_float_or_param_type を参照してください。	なし	0または1
<code><transparent></code>	完全鏡面反射光源のカラーを宣言します。詳しくは、 fx_common_color_or_texture_type および7章「FX入門」の「透明度(不透明度)を判定する」を参照してください。	なし	0または1
<code><transparency></code>	反射カラーに追加する完全鏡面反射光源の量(0.0~1.0)をスカラー値として宣言します。詳しくは、 fx_common_float_or_param_type 、および7章「FX入門」の「透明度(不透明度)を判定する」を参照してください。	なし	0または1
<code><index_of_refraction></code>	単一スカラーインデックスとして完全鏡面反射光源の反射インデックスを宣言します。 「 fx_common_float_or_param_type 」を参照してください。	なし	0または1

例

以下の例は、完全に透明な(不可視の)立方体を表示します。

```
<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</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>
```

先の例で、立方体を透明から不透明な黒に変えるには、以下を

```
<transparent opaque="RGB_ZERO"
```

次のように変更してください。

```
<transparent opaque="A_ONE"
```

以下の例では、constant を単純な赤に設定します。

```
<profile_COMMON>
<technique sid="T1">
  <constant>
    <emission><color>1.0 0.0 0.0 1.0</color></emission>
  </constant>
</technique>
</profile_COMMON>
```

この例では、色をパラメータから取得しています。

```
<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>
```

ラスターライザやレイトレーサは、屈折のような機能をサポートするかどうか、あるいは、サポートする方法によって、大幅に違った結果を生成する可能性があることに注意してください。

create_2d

カテゴリ: テクスチャリング

プロファイル: 外部

概要

2次元<image>アセットの手作業での作成を支援します。

コンセプト

ユーザは、画像を作成する前に、画像のサイズと構造を定義することができます。レンダーターゲットはデータで初期化されていないことが多いので、この要素はレンダーターゲットでよく使われます。この要素は、画像作成の方法としては、かならずしもお勧めの方法ではありませんが、<init_from>よりもその過程を細かく制御することができます。この要素は、ユーザが生成したい2次元構造を記述してから、その構造の各部分にロードすべきデータを記述します。

属性

<create_2d>要素には属性はありません。

関連要素

<create_2d>要素は、以下の要素と関連性があります。

親要素	image
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素を指定する場合には、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><size_exact width="512" height="512" ></code>	表面を厳密にこの寸法に変更する必要があることを指定します。 <code>xs:unsignedInt</code> 型の属性が 2 つ必要です。 <code><size_exact></code> と <code><size_ratio></code> のいずれか一方を指定する必要がありますが、両方指定しないようにしてください。	なし	0 または 1
<code><size_ratio width="1.0" height="1.0"></code>	画像のサイズが、ビューポートのサイズとの相対サイズであることを示します。たとえば、(1, 1) なら、ビューポートと同じサイズ、(0.5, 0.5) なら、面積がビューポートの 1/4 で、各方向の長さが半分です。 <code>float_type</code> 型の属性が 2 つ必要です。 <code><size_exact></code> と <code><size_ratio></code> のいずれか一方を指定する必要がありますが、両方指定しないようにしてください。	なし	0 または 1
<code><mips levels="8" auto_generate ="true"></code>	MIP 情報。 <code><mips></code> と <code><unnormalized></code> のいずれか一方を指定する必要がありますが、両方指定しないようにしてください。引数は両方必要です。 <ul style="list-style-type: none"> <code>levels</code> 属性 <code>xs:unsignedInt</code> 型で、1 なら MIP なし、0 なら、OpenGL および DirectX で以下のように定義される最大レベルになります。 $1 + \text{floor}(\log_2(\max(w, h, d)))$ アプリケーションやそのグラフィックス API によって自動的に生成される、より高い MIP レベルを初期化するには、<code>auto_generate</code> 属性を利用します。 	なし	0 または 1
<code><unnormalized></code>	テクセルの標準化されていないアドレッシング(0-W、0-H)。アドレッシングはレベル間で様ではないので、 <code><mips></code> と <code><unnormalized></code> のいずれか一方を指定する必要がありますが、両方指定しないようにしてください。これは、OpenGL の <code>textureRECT</code> 拡張に相当します。この要素には属性はありません。 DirectX には、これに相当する機能は存在しません。	なし	0 または 1
<code><array length="32" ></code>	2D 配列の長さを指定します。 <code>length</code> 属性は、 <code>xs:positiveInteger</code> 型の必須属性です。	なし	0 または 1
<code><format></code>	画像のピクセル形式や圧縮形式を指定します。指定しない場合には、形式は R8G8B8A8 リニアであると仮定されます。主見出し項目を参照してください。	なし	0 または 1
<code><init_from></code>	どの 2D 画像を初期化すべきか、およびどの MIP レベルを初期化すべきかを指定します。主見出し項目を参照してください。	なし	0 以上

詳細

`<create_2d>` を利用すると、2次元テクスチャの初期化をカスタマイズすることができます。カスタム 2D 画像を初期化するには、サイズ、ビューポートの比率、MIP レベル、正規化、ピクセル形式、データソースを指定します。また、この要素は、2次元画像の配列もサポートしています。

2次元画像の寸法は、`<size_exact>` 要素もしくは `<size_ratio>` 要素のいずれかによって指定されます。これは、かならずどちらか一方は指定する必要があります。

2次元画像型は、`<unnormalized>`要素と`<array>`要素のどちらが存在するかを考慮に入れて作成されます。どちらも存在しない場合、作成されるのは、サンプリング座標が正規化された通常の2次元画像であって、配列ではありません。

1つ以上の`<init_from>`によって画像の各部分を初期化します。ただし、この要素は必須ではなく、レンダリングのような他の操作によってデータを提供できるように、画像を空のままにしておくこともあります。

例

以下の例は、ソースファイルによる2次元画像のカスタム初期化を示しています。この初期化には、6つのMIPレベルの指定が含まれているので、アプリケーションはこれを生成する必要があります。

```
<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>
```

create_3d

カテゴリ: テクスチャリング

プロファイル: 外部

概要

3次元`<image>`アセットの手作業での作成を支援します。

コンセプト

ユーザは、画像を作成する前に、画像のサイズと構造を定義することができます。この要素は、画像作成の方法としては、かならずしもお勧めの方法ではありませんが、`<init_from>`よりもその過程を細かく制御することができます。この要素は、ユーザが生成したい3D構造を記述してから、その構造の各部分にロードすべきデータを記述します。

属性

`<create_3d>`要素には属性はありません。

関連要素

`<create_3d>`要素は、以下の要素と関連性があります。

親要素	<code>image</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<code><size width="256" height="256" depth="256" ></code>	表面を厳密にこの寸法に変更する必要があることを指定します。 <code>xs:unsignedInt</code> 型の属性が3つ必要です。	なし	1
<code><mips levels="7" auto_generate ="true"></code>	MIP 情報。属性は両方必要です。 <ul style="list-style-type: none"> levels 属性 <code>xs:unsignedInt</code> 型で、1 なら MIP なし、0 なら、OpenGL および DirectX で以下のように定義される最大レベルになります。 $1 + \text{floor}(\log_2(\max(w, h, d)))$ アプリケーションやそのグラフィックス API によって自動的に生成される、より高い MIP レベルを初期化するには、<code>auto_generate</code> 属性を利用します。 	なし	1
<code><array length="8"></code>	3次元配列の長さを指定します。length 属性は、 <code>xs:positiveInteger</code> 型の必須属性です。注:現時点では、3次元配列をサポートしている API はほとんどありません。	なし	0 または 1
<code><format></code>	画像のピクセル形式や圧縮形式を指定します。指定しない場合には、形式は R8G8B8A8 リニアであると仮定されます。主見出し項目を参照してください。	なし	0 または 1
<code><init_from></code>	どの3次元画像を初期化するべきか、およびどの MIP レベルを初期化するべきかを指定します。主見出し項目を参照してください。	なし	0 以上

詳細

サイズ、MIP レベル、ピクセル形式、データソースを指定することにより、カスタム 3 次元画像（立体画像）を初期化します。また、この要素は、3 次元画像の配列もサポートしています。

3 次元画像の寸法は、`<size>` 要素によって指定されます。

3 次元画像型は、`<array>` 要素が存在するかどうかを考慮に入れて作成されます。この要素が存在しない場合には、配列ではない、通常の 3 次元画像が作成されます。

1 つ以上の `<init_from>` によって画像の各部分を初期化します。ただし、この要素は必須ではなく、レンダリングのような他の操作によってデータを提供できるように、画像を空のままにしておくこともあります。

例

以下の例では、独立した 2 次元画像を 3 次元画像のデプススライスにロードすることによる、3 次元ノイズ立方体の初期化を示しています。フォーマットとしては、8 ビットソースを指定します。

```
<library_images>
  <image name="Noise3D">
    <create_3d>
      <size width="128" height="128" depth="128"/>
      <format>
        <exact>A8</exact>
      </format>
    </create_3d>
  </image>
</library_images>
```

```

<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>

```

create_cube

カテゴリ: テクスチャリング

プロファイル: 外部

概要

立方体<image>アセットを初期化します。

コンセプト

ユーザは、画像を作成する前に、画像のサイズと構造を定義することができます。サイズ、MIP レベル、ピクセル形式、データソースを指定することによって、立方体の 6 つの面を初期化します。また、この要素では、立方体の各面ごとに、画像の配列をサポートしています。

この要素は、立方体形の<image>アセットの作成を支援するためのものです。この要素は、画像作成の方法としては、かならずしもお勧めの方法ではありませんが、<init_from>よりもその過程を細かく制御することができます。この要素は、ユーザが生成したい立方体形の構造を記述してから、その構造の各部分にロードすべきデータを記述します。

属性

<create_cube>要素には属性はありません。

関連要素

<create_cube>要素は、以下の要素と関連性があります。

親要素	image
子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<code><size width="256"></code>	立方体を厳密にこの寸法に変更する必要があることを指定します。 <code>xs:unsignedInt</code> 型の <code>width</code> 属性は必須です。	なし	1
<code><mips levels="7" auto_generate="true"></code>	MIP 情報。属性は両方必要です。 <ul style="list-style-type: none"> <code>levels</code> 属性 <code>xs:unsignedInt</code> 型で、1 なら MIP なし、0 なら、OpenGL および DirectX で以下のように定義される最大レベルになります。 $1 + \text{floor}(\log_2(\max(w, h, d)))$ アプリケーションやそのグラフィックス API によって自動的に生成される、より高い MIP レベルを初期化するには、<code>auto_generate</code> 属性を利用します。 	なし	1
<code><array length=16></code>	立方体配列の長さを指定します。 <code>length</code> 属性は、 <code>xs:positiveInteger</code> 型の必須属性です。注:現時点では、3次元配列をサポートしている API はほとんどありません。	なし	0 または 1
<code><format></code>	画像のピクセル形式や圧縮形式を指定します。指定しない場合には、形式は R8G8B8A8 リニアであると仮定されます。主見出し項目を参照してください。	なし	0 または 1
<code><init_from></code>	どの立方体画像を初期化するべきか、どの MIP レベルを初期化するべきか、そして、MIP 中のどの立方体面を初期化すべきか指定します。主見出し項目を参照してください。	なし	0 以上

詳細

立方体画像の寸法は、`<size>`要素によって指定されます。

立方体画像型は、`<array>`要素が存在するかどうかを考慮に入れて作成されます。この要素が存在しない場合には、配列ではない、通常の立方体画像が作成されます。

1つ以上の`<init_from>`によって画像の各部分を初期化します。ただし、この要素は必須ではなく、レンダリングのような他の操作によってデータを提供できるように、画像を空のままにしておくこともあります。

例

以下の例は、キューブマップの6つの面の初期化を示しています。

```
<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>
    </create_cube>
  </image>
</library_images>
```

```

<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_X">
  <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_X">
  <ref>./images/sky_z_neg.tga</ref>
</init_from>
</create_cube>
</image>
</library_images>

```

depth_clear

カテゴリ: レンダリング

プロファイル: CG、GLES、GLES2、GLSL

概要

レンダリングターゲット画像をクリアするかどうか、クリアする場合にはどんな値を利用するのかを指定します。

コンセプト

描画を行う前に、レンダリングターゲットの画像をブランクのキャンバスまたはデフォルト値にリセットしなければならない場合もあります。これらの<depth_clear>宣言は、リセットする際にどんな値を利用するのかを指定します。クリアする指示が含まれていない場合には、ターゲットの画像はレンダリングが始まっても変更されません。

属性

GLES スコープ内にある <depth_clear> 要素には、属性はありません。

この要素が Cg、GLES2、GLSL スコープ内にある場合には、以下のような属性があります。

の	xs:nonNegativeInteger	設定するマルチレンダリングターゲット。デフォルト値は 0 です。オプション。
---	-----------------------	--

関連要素

<depth_clear>要素は、以下の要素と関連性があります。

親要素	evaluate
子要素	なし
その他	なし

詳細

この要素には、リソースを消去するのに使われる単一の浮動小数点値が含まれます。

この要素がパスの中に存在するということは、特定のバックバッファやレンダーターゲットリソースを消去する必要があることをランタイムに知らせる合図になっています。つまり、リソース中の既存の画像データのすべてを、指定された浮動小数点値で置き換える必要があるということです。この要素は、リソースを何も描かれていない既知の状態にして、以後このリソースを使った操作が予測どおりに行われるようにします。

`index` 属性は、消去したいリソースを特定します。インデックス 0 は、プライマリリソースを表します。プライマリリソースは、バックバッファ、もしくは、適当な `<*_target>` 要素 (`<color_target>`、`<depth_target>`、`<stencil_target>` など) によって提供されるオーバーライドです。

Direct3D® 9k クラスプラットフォームでは、MRT を設定するルールがかなり制限されています。たとえば、MRT が持つことができるのは、同じサイズとピクセルフォーマットのカラーバッファ 4 つと、全カラーバッファに対してアクティブなデプスバッファ 1 つとステンシルバッファ 1 つだけです。COLLADA FX の宣言は、この制限を緩めるように設計されており、FX ランタイムは、実際に適用する前に `<evaluate>` の中に指定されている特定の MRT 宣言が可能であることを検証しなければならず、検証に失敗した場合にはエラーを返す必要があります。

例

```
<depth_clear index="0">0.0</depth_clear>
```

depth_target

カテゴリ: レンダリング

プロファイル: CG、GLES、GLES2、GLSL

概要

深度情報をパス出力から受け取る `<image>` を指定します。

コンセプト

複数レンダリングターゲット (MRT) を利用すると、フラグメントシェーダに対して、パスごとに複数の値を出力させたり、標準のデプスユニットやステンシルユニットをリダイレクトして任意のオフスクリーンバッファを読み書きさせることができます。これらの要素は、FX ランタイムに、使用するべき定義済み画像や、画像を特定するパラメータを教えます。

属性

GLES スコープ内にある `<depth_target>` 要素には、属性はありません。

この要素が Cg、GLES2、GLSL スコープ内にある場合には、以下のような属性があります。

<code>index</code>	<code>xs:nonNegativeInteger</code>	マルチレンダリングターゲットの 1 つのインデックス。デフォルト値は 0 です。オプション。
<code>slice</code>	<code>xs:nonNegativeInteger</code>	単一の MIP マップレベル、ユニークな立方体の面、3 次元テクスチャのレイヤなどの、対象とする <code><surface></code> の中のサブイメージを指し示します。デフォルト値は 0 です。オプション。
<code>mip</code>	<code>xs:nonNegativeInteger</code>	デフォルト値は 0 です。オプション。

face	列挙	有効な値は、POSITIVE_X、NEGATIVE_X、POSITIVE_Y、NEGATIVE_Y、POSITIVE_Z、およびNEGATIVE_Zです。デフォルト値は、POSITIVE_Xです。オプション。
------	----	--

関連要素

<depth_target>要素は、以下の要素と関連性があります。

親要素	evaluate
子要素	下記サブセクションを参照
その他	なし

子要素

注:子要素として <param>または <instance_image> のいずれか1つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<param> (reference)	どの画像を使用すべきかを判断するには、samplerパラメータを参照してください。主見出し項目を参照してください。	なし	0または1
<instance_image>	レンダリング可能な画像を直接インスタンス化します。主見出し項目を参照してください。	なし	0または1

詳細

現時点のプラットフォームでは、MRTを設定するルールがかなり制限されています。たとえば、どれも同じサイズとピクセルフォーマットでなければならない4つのカラーバッファと、すべてのカラーバッファに対して1つの深度バッファとステンシルバッファだけがアクティブとなります。COLLADA FXの宣言は、この制限を緩めるように設計されており、FXランタイムは、実際に適用する前に<pass>の中に指定されている特定のMRT宣言が可能であることを検証しなければならず、検証に失敗した場合にはエラーを返す必要があります。

<depth_target>が指定されていない場合、FXランタイムは、対象とするプラットフォーム用のデフォルトのデプスバッファセットを利用します。

例

```
<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>
```

draw

カテゴリ: レンダリング

プロファイル: CG、GLES、GLES2、GLSL

概要

どんな種類のジオメトリを送り出すのかを FX ランタイムに対して指示します。

コンセプト

複数パスのテクニックを実行する場合、それぞれのパスで異なる種類のジオメトリを送り出さなければなりません。あるパスは、モデルを受け取る必要があるかもしれないし、別のパスは、フルスクリーンウッドによってオフスクリーンバッファの各ピクセルにフラグメントシェーダを適用する必要があるかもしれないし、また別のパスは表向きのポリゴンしか必要としないかもしれません。`<draw>` は、このパスが想定するジオメトリを FX ランタイムに対して記述するセマンティックとして利用できる、ユーザ定義の文字列を宣言します。

属性

`<draw>`要素には属性はありません。

関連要素

`<draw>`要素は、以下の要素と関連性があります。

親要素	<code>evaluate</code>
子要素	なし
その他	なし

詳細

`<draw>`要素には、`xs:string` が含まれます。以下のリストには、`<draw>`で使用する共通の文字列が含まれています。ただし、使用する文字列は、これらの文字列に限定されているわけではありません。

- **GEOMETRY**: この`<instance_geometry>`または`<instance_material>` (ジオメトリ)に関連付けられているジオメトリ。
- **SCENE_GEOMETRY**: この効果を使って、このジオメトリに既に関連付けられている効果や材料にではなく、シーンのジオメトリ全体を描画します。これは、シャドウバッファ生成などのテクニックのためのものです。これにより、Z 値を光源から抽出することのみ集中できます。これは、ZBuffer が順番を処理するという前提のもとでの順序付けとは関係なく行われます。
- **SCENE_IMAGE**: シーン全体を自分のターゲットに描画します。各オブジェクトにとって適切な効果や材料を使用します。これは、後処理のぼかしなどの効果のために、作業するシーンの正確な画像を必要とする効果のためのものです。これは、デプスバッファが順序を処理するという前提のもとでの順序付けとは関係なく行われます。
- **FULL_SCREEN_QUAD**: 位置は 0,0 ~ 1,1 であり、UV 同士は一致しています。
- **FULL_SCREEN_QUAD_PLUS_HALF_PIXEL**: 位置は 0,0 ~ 1,1 であり、UV はピクセルの UV サイズの 1/2 だけ正にずれています。

例

```

<!-- シーンを MyRenderTarget 画像に描画 -->
<pass>
  <evaluate>
    <color_target><param ref="MyRenderTarget" /></color_target>
    <draw>SCENE_IMAGE</draw>
  </evaluate>
</pass>

<!-- エンジンは、マテリアルがどのようなジオメトリに関連付けられているかにかかわらず、
画面にアラインメントされたクワッドを出力するはずなので、それを画面に描画-->
<pass>
  <evaluate>
    <draw>FULL_SCREEN_QUAD</draw>
  </evaluate>
</pass>

<!-- マテリアルが関連付けられているジオメトリを描画。
画面に描画。 -->
<pass>
  <evaluate>
    <draw>GEOMETRY</draw>
  </evaluate>
</pass>

```

effect

カテゴリ: 効果

プロファイル: エフェクト

概要

COLLADA 効果の記述を表します。

コンセプト

エフェクトは、ジオメトリの外観、および画面領域の画像処理に必要な数式を定義します。

プログラマブルパイプラインでは、3次元パイプラインの各段階を、高水準言語を利用してプログラミングすることができます。これらのシェーダを正しく機能させるためには、特定のデータを渡し、なおかつ、また3次元パイプラインの残りの部分を特定の方法で設定しなければならないことが珍しくありません。シェーダ効果は、シェーダを表すだけでなく、内部で処理を行う環境を表す方法でもあります。環境では、イメージ、サンブラ、シェーダ、入出力パラメータ、ユニフォームパラメータ、レンダリングステートの設定を記述する必要があります。

さらに、一部のアルゴリズムは、効果をレンダリングするために複数のパスを必要とします。これは、パイプラインの記述を順番付けされた<pass>オブジェクトのコレクションに分割することでサポートされています。これらは、効果を生成するための複数の方法の1つを記述した <technique>にグループ化されます。

<effect>宣言の中の要素は、シェーダの作成/利用/管理、ソースコード、パラメータなどを処理する基盤となるライブラリコードを利用するものと想定しています。この基盤となるライブラリは「FXランタイム」と呼ばれています。

<>要素の中、しかし<profile_*>要素の外側で宣言されているパラメータは、「<effect>範囲内に位置する」と呼ばれます。<effect>範囲内のパラメータは、基本データ型の制約リストからのみ描画することが可能で、宣言した後、すべてのプロファイルの <shader>と宣言で利用できます。<effect>範囲は、多くのプロファイルとテクニックを単一のパラメータにパラメータ化する簡単な方法です。

属性

<effect>要素には、以下の属性があります。

id	xs:ID	オブジェクトのグローバルな識別子。必須。
name	xs:token	効果の名称。オプション。

関連要素

<effect>要素は、以下の要素と関連性があります。

親要素	library_effects
子要素	下記サブセクションを参照してください。
その他	instance_effect

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<asset>	コアの主見出し項目を参照してください。	なし	0 または 1
<annotate>	主見出し項目を参照してください。	なし	0 以上
newparam	主見出し項目を参照してください。	なし	0 以上
profile	プロファイルは最低でも1つは存在する必要がありますが、以下のプロファイルのうちから任意のものを任意の数だけ含めることができます。 <ul style="list-style-type: none"> <profile_BRIDGE> <profile_CG> <profile_GLES> <profile_GLES2> <profile_GLSL> <profile_COMMON> 主見出し項目を参照してください。	なし	1 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

例

```
<effect id="fx-simple-uid71243231231" name="simple_diffuse_with_one_light">
  <profile_CG>
    <!-- profile_CG example の例を参照 -->
  </profile_CG>
</effect>
```

evaluate

カテゴリ: レンダリング

プロファイル: CG、GLES、GLES2、GLSL

概要

レンダリングパス用の評価要素が含まれます。

コンセプト

この要素には、何をどこに描くかといった、描画に関連するアクションが含まれます。この要素の子要素は、パスの使い方や呼び出し方、シーンに必要なデータなどの重要な細部を記述します。この要素には、状態情報やシェーダ情報を収集するが、描画可能な面を直接操作しない他のグループとは対照的に、データの変更につながるコマンド志向の情報が含まれます。

属性

<evaluate>要素には属性はありません。

関連要素

<evaluate>要素は、以下の要素と関連性があります。

親要素	pass
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<color_target>	主見出し項目を参照してください。	なし	0または1
<depth_target>	主見出し項目を参照してください。	なし	0または1
<stencil_target>	主見出し項目を参照してください。	なし	0または1
<color_clear>	主見出し項目を参照してください。	なし	0または1
<depth_clear>	主見出し項目を参照してください。	なし	0または1
<stencil_clear>	主見出し項目を参照してください。	なし	0または1
<draw>	主見出し項目を参照してください。	なし	0または1

詳細

この要素の用途は、主に情報の分類整理です。この要素は、パスを評価したり呼び出したりするために必要な要素を論理的にグループ化して、API 状態情報やシェーダプログラム作成情報のような他のグループから切り離します。

例

```
<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>

```

format

カテゴリ: テクスチャリング

プロファイル: 外部

概要

`<image>`アセットに期待されるフォーマットやメモリレイアウトを記述します。

コンセプト

`<image>`アセットでは、色彩情報をさまざまな異なる方法で配置することができます。format 要素は、そのテクセル情報のエンコード方法の記述を支援します。この要素は、厳密なエンコード、および汎用のヒンティングフォールバックメカニズムをサポートします

属性

`<format>`要素には属性はありません。

関連要素

`<format>`要素は、以下の要素と関連性があります。

親要素	create_2d , create_3d , create_cube
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<pre><hint channels=... range=... precision=... space=... ></pre>	<p>この要素もしくはより優先順位の高い要素が存在しない場合には、sRGBではなく、線形色勾配の一般的な R8G8B8A8 フォーマットを使うようにしてください。この要素にはデータはありません。</p> <p>属性は以下の通りです。</p> <ul style="list-style-type: none"> channels: 必須の列挙型。詳しくは、「詳細」を参照してください。 range: 必須の列挙型。詳しくは、「詳細」を参照してください。 precision: 列挙型(オプション)。デフォルトは、DEFAULT です。詳しくは、「詳細」を参照してください。 space: オプション。xs:token 型。 	なし	1
<pre><exact></pre>	<p>このサーフェスに使用したいプロファイル固有およびプラットフォーム固有のテクセル形式を表している文字列を、含んでいます。この要素が指定されなかった場合、または、指定されたけれど指定されたフォーマットをアプリケーションが処理できない場合には、アプリケーションはヒントを利用します。この要素には属性はありません。</p>	なし	0 または 1

詳細

デジタルコンテンツ作成 (DCC) ツールは、何も書かないか、もしくは<hint>だけを書き込みます。

ゲームエンジンツールは、通常、DirectX 40C テクスチャコードのようなデザインケースのために<format>を追加するでしょう。つまり、正確性や、メモリスペースの最小化や、パフォーマンスやクオリティの最大化のために、DXT3 や DXT5 や API 固有のフォーマットコードである GL_RGBAなどを、きわめて具体的に指定することになるはずですが、また、より一般的なコードを指定することもできます。Direct3D® 10 には、R8G8B8A8 のように、接頭辞なしで組合せることが推奨される、きわめて汎用的なよく定式化された列挙型が用意されています。

画像を作成するアプリケーションは、以下のものを使用すべきです。

- format が存在しない場合には、sRGBではなく、線形色勾配をもつ共通フォーマット R8G8B8A8 を仮定とします。
- <exact>が存在し、その文字列を認識できる場合には、それを使用します。
- そうでなく、<hint>が存在する場合は、そこに記述されている機能や特性を使用して、API にふさわしいフォーマットを選択します。

channels 属性は、フォーマットのテクセルごとのレイアウトを記述します。列挙された文字列の長さは、存在するチャンネルの数を示し、各文字はチャンネルの名前を表します。通常は、1~4 つのチャンネルがあります。有効な列挙値は次のとおりです。

- RGB - RGB カラーマップ。
- RGBA - アルファマップ付きの RGB カラー。色に加えて透明度や鏡面反射の指数などをチャンネル A にパックする場合に使われます。
- RGBE - HDR 用の共有指数を持つ RGB カラー。

- L - 光マッピングに頻繁に使用される輝度マップ
- LA - 光マッピングに頻繁に使用されるアルファマップ付きの輝度マップ
- D - 変位、視差、レリーフ、シャドウマッピングなどによく使われるデプスマップ

range 属性は、テクセルチャンネル値の範囲を記述します。各チャンネルは、値の範囲を表します。いくつかの範囲例には、符号付き整数または符号なし整数のものや、0.0f ~ 1.0f などの固定範囲内のもの、浮動小数点を利用した高位の動的範囲などがあります。有効な列挙値は次のとおりです。

- SNORM - -1 ~ 1 の範囲内にある 10 進値を表すフォーマット。実装は、固定小数点数、浮動小数点数のいずれであってもかまいません。
- UNORM - 0 ~ 1 の範囲内にある 10 進値を表すフォーマット。実装は、固定小数点数、浮動小数点数のいずれであってもかまいません。
- SINT - 符号付き整数を表すフォーマット。たとえば、8 ビットなら、-128 ~ 127 です。
- UINT - 符号なしの整数を表すフォーマット。たとえば、8 ビットなら、0 ~ 255 です。
- FLOAT - あらゆる範囲の浮動小数点数をサポートするフォーマット。精度が HIGH (高) の場合、32 ビットであることが期待されます。精度が MID (中) の場合、16 ~ 32 ビットでかまいません。精度が LOW (低) の場合、16 ビットであることが期待されます。

precision 属性は、テクセルチャンネル値の精度を特定します。テクセルの各チャンネルはそれぞれ精度を持っています。通常は、どのチャンネルも同じ精度です。exact フォーマットは、各チャンネルの精度を下げる可能性があります。チャンネルをひとまとめにしてより高い精度を適用すれば、同等の情報を伝えることができます。有効な列挙値は次のとおりです。

- DEFAULT - 「適切な」精度やパフォーマンスを提供できさえすれば、デザイナーは精度にこだわらないことを表します。
- LOW - 整数の場合、これは通常 8 ビットを表します。浮動小数点の場合、これは通常 16 ビットです。
- MID - 整数の場合、これは通常 8 ~ 24 ビットを表します。浮動小数点の場合、これは通常 16 ~ 32 ビットです。
- HIGH - 整数の場合、これは通常 16 ~ 32 ビットを表します。浮動小数点の場合、これは通常 24 ~ 32 ビットです。
- MAX - 利用できる場合、これは通常 32 ビットか 64 ビットです。64 ビットは、パフォーマンスに重大な影響を与え、CAD 以外のソフトウェアで高精度と見なされる精度を超えているので、HIGH とは別の専用のカテゴリに分類されています。

例

```
<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>

```

fx_common_color_or_texture_type

カテゴリ: レンダリング

プロファイル: 共通

概要

`<profile_COMMON>`効果内の固定機能シェーダ要素のカラー属性を記述するタイプ。

コンセプト

この型は、以下の要素の属性、および関連する要素を記述します。

- `<ambient>` (FX)
- `<diffuse>`
- `<emission>`
- `<reflective>`
- `<specular>`
- `<transparent>`

属性

属性があるのは`<transparent>`だけです。それ以外の `fx_common_color_or_texture_type` 型 `fx_common_color_or_texture_type` の要素には、属性はありません。

opaque	列挙	<p>透明度情報を取得するチャンネルを指定します。オプション。有効な値は、以下の通りです。</p> <ul style="list-style-type: none"> • A_ONE (デフォルト) : 透明度情報はカラーのアルファチャンネルから取得します。ここで値 1.0 は不透明を表します。 • RGB_ZERO : 各チャンネルが独立して調整される、カラーの R (赤)、G (緑)、B (青) チャンネルから透明度情報を取り出します。ここで値 0.0 は不透明を表します。 • A_ZERO (デフォルト) : 透明度情報はカラーのアルファチャンネルから取得します。ここで値 0.0 は不透明を表します。 • RGB_ONE : 各チャンネルが独立して調整される、カラーの R (赤)、G (緑)、B (青) チャンネルから透明度情報を取り出します。ここで値 1.0 は不透明を表します。 <p>詳しくは、7章「FX 入門」の「透明度 (不透明度) を判定する」を参照してください。</p>
--------	----	---

関連要素

`fx_common_color_or_texture_type` 型の要素は以下の要素と関連性があります。

親要素	<code>constant</code> (FX)、 <code>lambert</code> 、 <code>phong</code> 、 <code>blinn</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

注:子要素 `<color>`、`<param>`、または`<texture>`のいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<code><color></code>	値はリテラルのカラーであり、RGBA の順番に並んだ 4 つの浮動小数点数で指定します。主見出し項目を参照してください。	なし	「注」を参照してください。
<code><param></code> (reference)	値は、 <code><float4></code> に直接キャスト可能な、現在のスコープ内で事前に定義されたパラメータを参照して指定します。主見出し項目を参照してください。	なし	「注」を参照してください。
<code><texture texture="myParam" texcoord="myUVs"> <extra.../> </texture></code>	値は、事前に定義された <code><sampler2D></code> オブジェクトを参照して指定します。texcoord 属性は、セマンティックトークンを提供します。このトークンは、 <code><bind_material></code> の中で、 <code><geometry></code> インスタンスからのtexcoordの配列をサンブラにバインドするため参照されます。 属性はどちらも <code>xs:NCName</code> 型の必須属性です。 <code><extra></code> 子要素は、ゼロを含む任意の回数だけ出現することができます。詳しくは、コアの主見出し項目を参照してください。	なし	「注」を参照してください。

詳細

スキーマでは、`<ambient>`や`<diffuse>`、および `<blinn>`、`<constant>`、`<lambert>`、`<phong>`などのシェーダの子要素に対して、デフォルト色を指定していません。指定されていない子要素がある場合、それを除いて指定されたシェーダの式を適用してください。これは、その子要素に対して、明示的に黒を指定したのと同じ結果をもたらします。たとえば、`<diffuse>`のない`<phong>`の式は、次のようになります。

$$color = <emission> + <ambient> * al + <specular> * \max(R \cdot I, 0)^{<shininess>}$$

透明度の判定時における`<transparent>`や`<transparency>`の動作についての説明は、7章「FX 入門」の「透明度（不透明度）を判定する」を参照してください。

fx_common_float_or_param_type

カテゴリ: レンダリング

プロファイル: 共通

概要

`<profile_COMMON>`効果内の固定機能シェーダ要素のスカラー属性を記述するタイプ。

コンセプト

この型は、以下の要素の属性、およびそれらに関連する要素を記述します。

- `<index_of_refraction>`
- `<reflectivity>`
- `<shininess>`
- `<transparency>`

属性

`fx_common_float_or_param_type` 型の要素には属性はありません。

関連要素

`fx_common_float_or_param_type` 型の要素は以下の要素と関連性があります。

親要素	<code>constant (FX)</code> 、 <code>lambert</code> 、 <code>phong</code> 、 <code>blinn</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

注:子要素`<float>`または `<param>` のいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<code><float sid="..."></code>	値は、リテラルの浮動小数点スカラーで指定します。例： <code><float> 3.14 </float></code> <code>sid</code> 属性はオプションです。	なし	「注」を参照してください。
<code><param></code> (reference)	値は、浮動小数点スカラーに直接キャスト可能な事前に定義されたパラメータを参照して指定します主見出し項目を参照してください。	なし	「注」を参照してください。

詳細

透明度の判定時における`<transparent>`や`<transparency>`の動作についての説明は、7章「FX 入門」の「透明度（不透明度）を判定する」を参照してください。

fx_sampler_common

カテゴリ: テクスチャリング

プロファイル: `External`、`Effect`、`CG`、`COMMON`、`GLSL`、`GLES`、`GLES2`

概要

`<sampler*>`要素のサンプリング状態を記述する型。

コンセプト

この型は、以下の要素の属性、およびそれらに関連する要素を記述します。

- `<sampler1D>`
- `<sampler2D>`
- `<sampler3D>`
- `<samplerCUBE>`
- `<samplerDEPTH>`
- `<samplerRECT>`
- `<samplerStates>`

これから継承するスキーマ型は、上記の状態のサンプリング時における使われ方の詳細を記述します。

属性

この型の要素には属性はありません。

関連要素

`<sampler*>`要素は、以下の要素と関連性があります。

親要素	コア要素: <code>newparam</code> , <code>setparam</code> FX 要素: <code>array</code>
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><instance_image></code>	<code><setparam></code> マテリアルがないときにサンブラが消費するデフォルト画像をインスタンス化します。主見出し項目を参照してください。	なし	0 または 1
<code><texcoord semantic=... ></code>	GLES の <code><newparam></code> / <code><sampler2D></code> においてのみ有効です。テクスチャユニットがメッシュの中で読み込み元として使用する必要のあるテクスチャ座標チャンネルに、セマンティック名を提供するセマンティック属性が含まれます。チャンネル (<code>array</code>) は、ここで <code><bind_material></code> を使ってマッピングされます。シェーダベースのプログラミングの場合には、 <code><texcoord></code> はシェーダの中で計算できますが、OpenGL ES 1.x のような固定機能 API の場合には、テクスチャ座標はメッシュを使ってパラメータ化する必要があります。この要素にはデータはありません。	なし	0 または 1 回
<code><wrap_s></code>	テクスチャの S 座標方向の繰り返しやクランピングを制御します。列挙型について、詳しくは、「詳細」を参照してください。	WRAP	0 または 1 回
<code><wrap_t></code>	テクスチャの T 座標方向の繰り返しやクランピングを制御します。列挙型について、詳しくは、「詳細」を参照してください。	WRAP	0 または 1 回
<code><wrap_p></code>	テクスチャの P 座標方向の繰り返しやクランピングを制御します。列挙型について、詳しくは、「詳細」を参照してください。GLES の <code><sampler2D></code> では無効です	WRAP	0 または 1 回
<code><minfilter></code>	テクスチャ最小化。列挙型。詳しくは、「詳細」を参照してください。プリミティブにテクスチャを適用することは、テクスチャ画像空間からフレームバッファ画像空間への写像を意味します。一般に、この写像の際には、サンプリングされたテクスチャ画像の再構成、それに続くフレームバッファ空間への写像に伴うようなワーピング、そしてフィルタリング、およびフィルタリングされワーピングされ復元された画像がフラグメントに適用される前サンプリングが必要です。	LINEAR	0 または 1
<code><magfilter></code>	テクスチャ拡大。列挙型。詳しくは、「詳細」を参照してください。この値は、ガンマが拡大を示している場合に、テクスチャ値を取得する方法を決めます。	LINEAR	0 または 1

名前/例	解説	デフォルト値	出現回数
<code><mipfilter></code>	ミップマップフィルタ。列挙型。詳しくは、「詳細」を参照してください。	LINEAR	0 または 1
<code><border_color></code>	クランプのあるラップモードで、テクスチャのアドレス空間の端を越えて読み取りを行うと、この色が読み取られます。 <code>fx_color_common</code> 型 (RGBA 順の 4 つの浮動小数点数)。GL ES の <code><sampler2D></code> では無効です	なし	0 または 1
<code><mip_max_level></code>	<code>xs:unsignedByte</code> 型。サンブラが評価するプログレッシブレベルの最大数。	0	0 または 1
<code><mip_min_level></code>	評価を開始する最小プログレッシブレベルを表す <code>xs:unsignedByte</code> 型。GL ES の <code><sampler2D></code> では無効です	0	0 または 1
<code><mip_bias></code>	<code>float_type</code> 型。サンブラがミップマップチェーンを評価するために使うガンマ (詳細度パラメータ) を補正する。	0.0	0 または 1
<code><max_anisotropy></code>	<code>xs:unsignedInt</code> 型。異方性フィルタリングの際に利用できるサンプル数。GL ES の <code><sampler2D></code> では無効です	1	0 または 1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

`<sampler*>` 子要素すべてについての詳細は、OpenGL 仕様を参照してください。

以下のラップモードは、特定の `<sampler*>` 設定の使い方に基づいて、 $[0.0 \sim 1.0]$ の範囲外にある s , t , p テクスチャ座標の解釈に影響を与えます。以下の表では、理解を助けるため、ラップモード列挙型について説明するとともに、それを OpenGL の識別名に対応させています。

ラップモード	OpenGL 識別名	解説
WRAP	<code>GL_REPEAT</code>	テクスチャ座標の整数部を無視し、小数部だけ使います。テクスチャを整数単位でタイル表示します。たとえば、 u 値が $0 \sim 3$ に変化する間に、テクスチャは 3 回繰り返されます。ミラーリングは実行されません。
MIRROR	<code>GL_MIRRORED_REPEAT</code>	まず、テクスチャ座標を鏡像反転させます。反転された座標は、 <code>CLAMP_TO_EDGE</code> の説明にしたがって、強制的に補正されます。テクスチャを整数単位で反転表示します。たとえば、 u 値が $0 \sim 1$ の間は、テクスチャは通常通りに処理されます。1~2の間、テクスチャは反転されます (鏡像)。2~3の間、テクスチャは再び通常表示になります。以下同様です。
CLAMP	<code>GL_CLAMP_TO_EDGE</code>	全ミップマップレベルのテクスチャ座標を強制的に補正して、テクスチャフィルタが境界テクセルをサンプリングしないようにします。 注: <code>GL_CLAMP</code> は、サンプリング境界を越えるあらゆるテクセルを、境界色に置き換えます。したがって、 <code>CLAMP_TO_EDGE</code> の方が適切です。OpenGL ES 仕様からは <code>GL_CLAMP</code> 識別名が削除されたので、OpenGL ES ではなおのこと改善されます。境界がサンプリングされないように、 $[0.0, 1.0]$ の範囲に到達もしくはそれを越えたテクスチャ座標は、 $0.0 \sim 1.0$ の範囲内に設定されます。
BORDER	<code>GL_CLAMP_TO_BORDER</code>	あらゆるミップマップにおけるテクスチャ座標を強制的に補正して、対応するテクスチャ座標が $[0, 1]$ の範囲を大幅に外れるフラグメントについて、テクスチャフィルタが常に境界テクセル

ラップモード	OpenGL 識別名	解説
<code>MIRROR_ONCE</code>		ルをサンプリングするようにします。 [0.0、1.0]の範囲外にあるテクスチャ座標が境界色に設定されることを除けば、 <code>CLAMP</code> とほとんど同じです。 テクスチャ座標の絶対値をとって（つまり、0を中心に鏡像反転させて）から、最大値にクランプします。

GLES の場合、`<newparam>/<sampler2D>`の中で有効なのは、以下の値だけです。

ラップモード	OpenGL 識別名	解説
<code>REPEAT</code>		
<code>CLAMP</code>		
<code>CLAMP_TO_EDGE</code> <code>MIRRORED_REPEAT</code>		GLES 1.1 だけでサポートされます。

プリミティブにテクスチャを適用するという事は、テクスチャ画像空間からフレームバッファ画像空間への写像を意味します。一般に、この写像の際には、サンプリングされたテクスチャ画像の再構成、フレームバッファ空間への写像に伴う一様なワーピング、フィルタリング、およびフィルタリングされワーピングされ復元された画像がフラグメントに適用される前リサンプリングが必要です。以下の表は、フィルタリング要素で有効な値を示しています。

列挙値	解説	有効な場所
<code>NONE</code>	縮小しない。	<code><mipfilter></code>
<code>NEAREST</code>	バイリニア	<code><minfilter></code> 、 <code><mipfilter></code> 、 <code><magfilter></code>
<code>LINEAR</code>	トライリニア。	<code><minfilter></code> 、 <code><mipfilter></code> 、 <code><magfilter></code>
<code>ANISOTROPIC</code>	ポリゴンと画面の間の角度の違いに起因する歪みを補償します。 <code>max_anisotropy</code> に依存します。	<code><minfilter></code>

image

カテゴリ: テクスチャリング

プロファイル: 外部

概要

オブジェクトのグラフィカルな表現の保存場所を宣言します。

コンセプト

デジタル画像データは、大きく分けて、ラスタ、ベクトル、ハイブリッドの3種類の形式があります。ラスタ画像は、輝度やカラー値を表す画素（ピクセル）の並びから構成され、このピクセルが集まって1つの画像を構成しています。またベクトル画像では、曲線、直線、図形などを表す数学的な式を利用して、図を記述します。さらにハイブリッド画像は、ラスタ情報とベクトル情報のそれぞれの長所を組み合わせることで画像を記述します。

`<image>`要素は、ラスタ画像データの記述に最も適していますが、それ以外の画像形式を処理することもできます。ラスタ画像データは、通常 n 次元の配列によって構成されます。テクスチャ参照を行う関数では、この配列構造をディスプレイメント、法線、高さといったフィールド値を格納し利用することもできます。

属性

`<image>`要素には、以下の属性があります。

id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<image>`要素は、以下の要素と関連性があります。

親要素	<code>library_images</code>
子要素	下記サブセクションを参照してください。
その他	<code>instance_image</code>

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。また、`<init_from>`要素や`<create_*>`要素はどちらか片方だけしか利用できず、複数個利用することはできません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><renderable share=... ></code>	画像をレンダーターゲットとして定義します。この要素が存在する場合には、そこに画像をレンダリングすることができます。この要素にはデータはありません。インスタンス化の際に、レンダーターゲットを複製しないで、全インスタンスで共有したい場合には、Boolean型の必須属性 <code>share</code> を <code>true</code> に設定します。	なし	0 または 1
<code><init_from mips_generate=... ></code>	画像を URL (ファイルなど) もしくは 16 進数値のリストから初期化します。画像全体の構造やデータを初期化するには、DDS のようなフォーマットを使います。MIP レベルのより高いデータがファイルの中に存在しない場合には、Boolean 型の <code>mips_generate</code> 属性を使って初期化します。主見出し項目を参照してください。	なし	0 または 1
<code><create_2d></code>	カスタム 2D 画像を初期化する際には、サイズ、ビューポートの比率、MIP レベル、正規化、ピクセル形式、データソースを指定します。また、この要素は、2次元画像の配列もサポートしています。主見出し項目を参照してください。	なし	
<code><create_3d></code>	サイズ、MIP レベル、ピクセル形式、データソースを指定することにより、カスタム 3次元画像(立体画像)を初期化します。また、この要素は、3次元画像の配列もサポートしています。主見出し項目を参照してください。	なし	
<code><create_cube></code>	サイズ、MIP レベル、ピクセル形式、データソースを指定することによって、立方体の 6つの面を初期化します。また、この要素では、立方体の各面ごとに、画像の配列をサポートしています。また、この要素は、立方体画像の配列もサポートして	なし	

名前/例	解説	デフォルト値	出現回数
	います。主見出し項目を参照してください。		
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

画像アセットは、`<init_from>`を使ってファイルから直接初期化することもできるし、`<create_*>`を使って手続き的により複雑な生成を行うこともできます。`<create_*>`は、特にレンダーターゲットを生成する際に重要です。

アプリケーションの多くは、ファイルローディングのメカニズムとしては`<init_from>`だけをサポートし、より複雑な構造は DDS ファイルフォーマットによってサポートしています。けれども、一部のアプリケーション、特に DDS を無料で利用できないプラットフォーム上で動いているものは、`<create_*>`要素によって、テクスチャのサブ画像を一度に1つずつロードする必要があります。

`<image>`オブジェクト。

- フィールドのサイズやレイアウトをそれぞれのピクセルで記述するデータ形式を持っています。
- サイズ設定は、`<size_exact>`を使ってピクセルの絶対数で設定することもできるし、`<size_ratio>`を使ってビューポートサイズとの比率で設定することもできます。
- `<mips>`を使って、決まった数のミップマップレベルを宣言することができます。

例

以下は、外部の PNG アセットを参照する`<image>`要素の例です。

```
<library_images>
  <image name="WoodFloor">
    <init_from><ref>./Textures/WoodFloor-01.png</ref></init_from>
  </image>
</library_images>
```

これ以外の例については、`<create_*>`要素を参照してください。

include

カテゴリ: シェーダ

プロファイル: CG、GLSL

概要

外部リソースを参照して、ソースコードまたは事前にコンパイルされたバイナリシェーダを FX ランタイムにインポートします。

コンセプト

属性

`<include>`要素には、以下の属性があります。

sid	sid_type	
		ソースコードブロックまたはバイナリシェーダの識別子。必須。詳しくは、アドレス構文 3 章の「スキーマのコンセプト」を参照してください。
url	xs:anyURI	リソースがある場所。必須。

関連要素

`<include>`要素は、以下の要素と関連性があります。

親要素	<code>profile_CG, profile_GLES2, profile_GLSL</code>
子要素	なし
その他	なし

詳細

`<include>`要素自体には、データは含まれません。その代わりに、`url` 属性を使ってデータを参照します。

例

```
<include sid="ShinyShader" url="file://assets/source/shader.glsl"/>
```

init_from

カテゴリ: テクスチャリング

プロファイル: 外部

概要

画像全体もしくは画像の一部を、参照されたもしくは埋め込まれたデータで初期化します。

コンセプト

この要素の正しい使い方は、親要素によって異なります。

この要素が`<image>`の子要素である場合には、画像を完全に初期化することを目的とします。その画像のサイズや構造は、できるだけデータと一致させる必要があります。

要素が`<image>`の子要素である`<create_*>`の子要素である場合には、画像の一部を（一回ごとに構造の一部分を）初期化することを目的とします。

画像アセットのほとんどは、アーティストによって生成されたデータによって初期化されます。このようなデータは、通常、BMP、JPG、TGA、PSD、DDS など数多くの一般的なファイルフォーマットのいずれかを使った画像ファイルに格納されます。

ノイズテクスチャやライトマップのような一部の画像データは、アプリケーションによって手続的に生成されます。このようなデータは、オプションで、COLLADA ファイルに直接埋め込むこともできます。

この要素は、画像データ、およびそのデータを画像構造中のどこにロードすべきかの指示の、参照や埋め込みをサポートします。

属性

`<image>`の子要素である場合、`<init_from>`要素は以下のような属性を持ちます。

<code>mips_generate</code>	<code>xs:boolean</code>	オプション。データがファイルの中に存在しない場合、より高いMIPレベルを初期化します。デフォルトは true です。
----------------------------	-------------------------	--

`<create_*>`の子要素である場合、`<init_from>`要素は以下のような属性を持ちます。

<code>array_index</code>	<code>xs:unsignedInt</code>	オプション。画像中のどの配列要素を初期化(フィル)するかを指定します。デフォルト値は0です。
<code>mip_index</code>	<code>xs:unsignedInt</code>	必須。画像中のどの MIP レベルを初期化するかを指定します。
<code>depth</code>	<code>xs:unsignedInt</code>	<code><create_3d></code> では必須です。 <code><create_2d></code> や <code><create_cube></code> では無効です。 初期化する MIP 中のスライス(デプスレベル)を指定します。
<code>face</code>	列挙	<code><create_cube></code> では必須です。 <code><create_2d></code> や <code><create_3d></code> では無効です。 初期化する MIP 中の立方体面を指定します。有効な値は、以下の通りです。 <ul style="list-style-type: none"> • <code>POSITIVE_X</code> • <code>NEGATIVE_X</code> • <code>POSITIVE_Y</code> • <code>NEGATIVE_Y</code> • <code>POSITIVE_Z</code> • <code>NEGATIVE_Z</code>

関連要素

`<init_from>`要素は、以下の要素と関連性があります。

親要素	<code>image</code> , <code>create_2d</code> , <code>create_3d</code> , <code>create_cube</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

以下の要素のいずれかがちょうど1個存在する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><ref></code>	初期化データの取得元になるファイルのURL(<code>xs:anyURI</code>)が含まれます。該当するファイルの特性に依存します。DDSのような複雑なフォーマットのファイルである場合、初期化データには、キューブマップ、ボリューム、MIPなどが含まれることもあります。	なし	0または1
<code><hex format=... ></code>	画像データが、16進でエンコードされたバイナリオクテットの並びとして含まれます。このデータには、通常、データの幅や高さのようなヘッダ情報を含む、必要な情報のすべてが含まれています。画像の記述やデータをデコードする CODEC を指定するには、必須属性である <code>format</code> 属性(<code>xs:token</code> 型)を使います。この属性の値は、通常、一般的なファイル拡張子(BMP、JPG、DDS、TGAなど)です。	なし	0または1

詳細

アプリケーションの多くは、`<init_from>`を単に`<image>`の子要素として使って、ファイルを画像としてロードします。画像自体の特性は、ほとんどの場合、ファイルによって決まります。`mip_generate`属性は、MIPレベル0の画像からミップマップを生成することにより、より良いサンプリング動作のサポートに役立ちます。ただし、DDSのようなフォーマットは全MIPレベルの格納をサポートしています。

それ以外の親を持つ<init_from>要素は、ディスク上のさまざまなファイルや、COLLADA 文書に埋め込まれたデータから、より複雑な構造の画像の手続き上の構築をサポートします。

例

以下の例では、Rag_OpenGL という名前のサブディレクトリにある DDS ファイルから、3D ボリュームテクスチャをロードします。

```
<library_images>
  <image name="Noise">
    <init_from >
      <ref>./Rag_OpenGL/NoiseVolume.dds</ref>
    </init_from>
  </image>
</library_images>
```

これ以外の例については、<create_2d>、<create_3d>、<create_cube>を参照してください。

instance_effect

カテゴリ: 効果

プロファイル: 外部

概要

COLLADA エフェクトをインスタンス化します。

コンセプト

エフェクトは、ジオメトリの外観、および画面領域の画像処理に必要な数式を定義します。

COLLADA のインスタンス要素の詳細は、3 章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

<instance_effect>は、<library_effects>のエフェクト定義をインスタンス化して、そのパラメータをカスタマイズします。

url 属性は、エフェクトを参照します。

<setparam>の、特定のエフェクトやプロファイルパラメータに、そのインスタンス固有の値を割り当てます。

<technique_hint>は、エフェクトプロファイル中の、推奨の、もしくは、最後に使われたテクニックを表します。この要素を利用すると、エフェクトインスタンスのルックアンドフィールを、ユーザが最後に使ったときと同じ状態に維持することができます。ランタイムレンダリングエンジンによっては、その場で自動的に新しいテクニックを選択するものもありますが、オーサリングの間、視覚的な外観を同一に保つことは、一部のエフェクトにとって、あるいは、デジタルコンテンツ作成の一貫性にとって重要です。

属性

<instance_effect>要素には、以下の属性があります。

sid	sid_type	説明
		この要素のスコープ付き識別子を含むテキスト文字列。この属性の値は、親要素のスコープ内で一意である必要があります。オプション。詳しくは、3 章「スキーマのコンセプト」の「アドレス構文」を参照してください。

name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	インスタンス化する <effect> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。

関連要素

<instance_effect>要素は、以下の要素と関連性があります。

親要素	material
子要素	下記サブセクションを参照してください。
その他	effect

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<technique_hint>	主見出し項目を参照してください。	なし	0 以上
<setparam>	コアの主見出し項目を参照してください。	なし	0 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

例

```
<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>
```

instance_image

カテゴリ: テクスチャリング

プロファイル: External、Effect、CG、COMMON、GLES、GLES2、GLSL

概要

シェーダの中で使う画像をインスタンス化します。

コンセプト

通常は、エフェクトの中で、ジオメトリ表面のシェーディングを行うために使われます。ただし、この画像は、レンダリングのターゲットとしても利用できます。この方法を使うと、画像内の写真やデータを、高度な FX シェーディングテクニックで動的に更新することができます。ただし、レンダリングのターゲットになる画像は、**<renderable>**要素を含んでいる必要があります。

属性

`<instance_image>`要素には、以下の属性があります。

<code>url</code>	<code>xs:anyURI</code>	必須。画像アセットの URI。
<code>sid</code>	<code>sid_type</code>	オプション。この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
<code>name</code>	<code>xs:token</code>	オプション。この要素の文字列の名。

関連要素

`<instance_image>`要素は、以下の要素と関連性があります。

親要素	<code><sampler*></code> 、 <code>color_target</code> 、 <code>depth_target</code> 、 <code>stencil_target</code>
子要素	下記サブセクションを参照してください。
その他	<code>image</code> 、 <code>library_images</code> 、 <code>sampler_image</code>

子要素

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	COLLADA で定義されない追加情報の格納用。5章 コア要素のリファレンスの主見出し項目を参照してください。	なし	0 以上

詳細

画像をインスタンス化する際の動作は、レンダリングできない画像を除けば、通常はきわめて単純です。レンダリング可能な画像には、2つの動作オプションがあります。レンダリング可能な画像が「共有」とマークされている場合、その画像の写真やデータは全インスタンスで共有されます。この画像は共有されているので、レンダリング時には、全インスタンスが更新データを受け取ります。レンダリング可能な画像が共有されていない場合、その画像インスタンスへのレンダリングが他の画像インスタンスに影響を与えないように、インスタンスごとに画像の一意のコピーが生成されます。

例

これ以外の例については、`<color_target>`、`<depth_target>`、`<stencil_target>`を参照してください。

```
<newparam sid="surfaceTex">
  <sampler2D>
    <instance_image url="noise1"/>
  </sampler2D>
</newparam>
```


instance_material

(ジオメトリ)

カテゴリ: マテリアル

プロファイル: 外部

概要

COLLADA マテリアルリソースをインスタンス化します。

コンセプト

エフェクトは、ジオメトリの外観、および画面領域の画像処理に必要な数式を定義します。マテリアルは、エフェクトをインスタンス化して、パラメータ値を選択し、さらに、テクニクを選択します。マテリアルインスタンスは、マテリアルをジオメトリやシーンアイテムに関連付けます。

COLLADA のインスタンス要素の詳細は、3 章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

マテリアルを使用する際には、インスタンス化して、ジオメトリに関連付けます。`<instance_material>` の `symbol` 属性は、そのマテリアルがどのジオメトリに関連付けられるかを表し、`target` 属性は、その要素がインスタンス化しているマテリアルを参照します。

また、この要素は、関連付けるジオメトリのセクション (`symbol`) を特定するだけでなく、頂点ストリームのリマップの方法や、シーンオブジェクトがマテリアルエフェクトパラメータにバインドされる方法をも定義します。このような関連付けは、ずっと後でしかできない処理であり、関連付けるシーンジオメトリに依存します。

`<bind>` は、マテリアルのエフェクト中のパラメータを、シーン中のターゲットに、セマンティックによって関連付けます。

`<bind_vertex_input>` は、頂点シェーダの頂点ストリームセマンティックス (`TEXCOORD2` など) を、`input_semantic` および `input_set` 属性によって指定されるジオメトリの頂点入力ストリームに関連付けます。

属性

`<instance_material>` 要素には、以下の属性があります。

sid	sid_type	
<code>sid</code>	<code>xs:token</code>	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3 章「スキーマのコンセプト」のアドレス構文を参照してください。
<code>name</code>	<code>xs:token</code>	この要素の文字列の名。オプション。
<code>target</code>	<code>xs:anyURI</code>	インスタンス化する <code><material></code> 要素の場所の URL。必須。ローカルなインスタンスや外部参照を参照することができます。 ローカルなインスタンスの場合、この URL は、「#」の文字で始まる相対 URI フラグメント識別子です。このフラグメント識別子は、インスタンス化する要素の ID から構成される、XPointer の短縮ポインタです。 外部参照の場合、この URL は、絶対 URL もしくは相対 URL です。
<code>symbol</code>	<code>xs:NCName</code>	このマテリアルでバインドしているジオメトリ内から定義されているシンボル。必須。

関連要素

`<instance_material>`要素は、以下の要素と関連性があります。

親要素	<code>technique_common</code> 場所は、 <code>bind_material</code>
子要素	下記サブセクションを参照してください。
その他	<code>material</code>

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><bind></code> (FX)	主見出し項目を参照してください。	なし	0 以上
<code><bind_vertex_input></code>	インスタンス化時に、頂点入力をエフェクトパラメータにバインドします。主見出し項目を参照してください (<code><bind_material></code> 内のみ)	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

```
<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>
```

instance_material

(レンダリング)

カテゴリ: レンダリング

プロファイル: 外部

概要

画面エフェクト用の COLLADA マテリアルリソースをインスタンス化します。

コンセプト

COLLADA のインスタンス要素の詳細は、3 章 スキーマのコンセプトの「インスタンス化と外部参照」を参照してください。

マテリアルを利用するには、インスタンス化する必要があります。インスタンス化されたマテリアルのほとんどは、ジオメトリに関連付けられます。けれども、この要素の場合、画像レベルやレンズレベルの処理のために、マテリアルをシーン自体に関連付けます。

url 属性は、インスタンス化対象のマテリアルを参照します。

また、この属性は、シーンオブジェクトをマテリアルエフェクトパラメータにどのようにバインドするかを特定します。このような関連付けは、ずっと後でしかできない処理であり、関連付けるシーンジオメトリに依存します。

<bind> (FX) は、マテリアルのエフェクト中のパラメータを、シーン中のターゲットに、セマンティックによって関連付けます。

<technique_override> (オプション) を使うと、マテリアルの **<technique_hint>** を使って各パスをレンダリングする典型的なパターンとは異なり、マテリアルの technique や pass サブ要素を極めて具体的に利用することができます。この要素が利用できるのは、親要素が **<evaluate_scene>** の中にある **<render>** 要素であるときだけです。シーン評価エフェクトの評価は、ジオメトリ表面シェーダの多くに比べてはるかに手続き的で複雑な可能性があるため、この要素は、シーンエフェクトを実現するために、必要に応じて、マテリアル一部だけを呼び出すようなコントロールを追加します。

これを COLLADA の旧バージョンで実現しようとする、ユーザは、エフェクトを多数の小さいエフェクトやマテリアルに分解し、さらに、その分解したマテリアル用のそれぞれ異なるパラメータテーブルを管理する必要がありました。この新しいコントロールのおかげで、このような処理はもはや不要になりました。

属性

<instance_material> 要素には、以下の属性があります。

url	xs:anyURI	マテリアルがある場所。必須。
-----	-----------	----------------

関連要素

<instance_material> 要素は、以下の要素と関連性があります。

親要素	evaluate_scene/render
子要素	下記サブセクションを参照してください。
その他	material

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<bind>	エフェクトのテクニックやパスを複数のエフェクトに分解する代わりに、マテリアル中の特定のテクニックやパスをターゲットにします。 ref 属性は必須属性で、 <technique> の SID を指定します。 pass 属性はオプションで、実行する特定のパスの SID を指定します。指定されなかった (空) 場合には、テクニックの <pass> の全てが使われます。	なし	0 または 1
<bind_vertex_input> (FX)	インスタンス化時にエフェクトパラメータに値をバインドします。主見出し項目を参照してください。	なし	0 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

`<evaluate_scene><render>`のレンダリングは、特定のシーンの後処理エフェクトのパラメータ設定を複数回もしくは複数のシーンで利用できるようにするために、エフェクトを直接インスタンス化する代わりに、マテリアルをインスタンス化することによって実現されます。ブラーのようなエフェクトは、同じシーンに複数回適用したり、異なるシーンの中で共有したりすることが容易です。

例

```
<visual_scene>
<node>
  <!--以下、すばらしいシーンがあります-->
</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>
```

lambert

カテゴリ: レンダリング

プロファイル: 共通

概要

ライティングに影響されない、拡散シェーディングのある面を生成します。

コンセプト

`<profile_COMMON>`エフェクトの中で使われ、ライティングに影響されない拡散シェーディング処理したサーフェスを生成する固定機能パイプラインを宣言します。

結果は、粗面に当たった光は、全方向に等しく反射されるというランバートの法則に基づきます。反射カラーは、以下のように計算します。

$$color = <emission> + <ambient> * al + <diffuse> * \max(N \cdot L, 0)$$

- *al* – シーンからの環境光の寄与分を表す定数。共通プロファイルでは、<visual_scene>の中の<light><technique_common><ambient><color>の値の合計になります。
- *N* – 法線ベクトル（正規化済み）
- *L* – 光ベクトル

属性

<lambert>要素には属性はありません。

関連要素

<lambert>要素は、以下の要素と関連性があります。

親要素	technique (FX) (profile_COMMON 内)
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<emission>	このオブジェクトのサーフェスから放射される光源の量を宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<ambient> (FX)	この物体の表面で反射される環境光の量を宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<diffuse>	このオブジェクトのサーフェスから反射される拡散光の量を宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<reflective>	完全鏡面反射のカラーを宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<reflectivity>	反射光源に追加する完全鏡面反射の量 (0.0~1.0) を宣言します。詳しくは、 fx_common_float_or_param_type を参照してください。	なし	0 または 1
<transparent>	完全鏡面反射光源のカラーを宣言します。 詳しくは、 fx_common_color_or_texture_type および 7 章「FX 入門」の「透明度 (不透明度) を判定する」を参照してください。	なし	0 または 1
<transparency>	反射カラーに追加する完全鏡面反射光源の量 (0.0~1.0) をスカラー値として宣言します。詳しくは、 fx_common_float_or_param_type 、および 7 章「FX 入門」の「透明度 (不透明度) を判定する」を参照してください。	なし	0 または 1

名前/例	解説	デフォルト値	出現回数
<code><index_of_refraction></code>	単一スカラーインデックスとして完全鏡面反射光源の反射インデックスを宣言します。 「 <code>fx_common_float_or_param_type</code> 」を参照してください。	なし	0または1

例

```

<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>

```

library_effects

カテゴリ: 効果

プロファイル: 外部

概要

`<effect>` アセットを格納するためのライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存し可能です。

属性

`<library_effects>` 要素には、以下の属性があります。

id	xs:ID	<code><library_effects></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_effects>` 要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。

その他	なし
-----	----

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<asset>	コアの主見出し項目を参照してください。	なし	0 または 1
<effect>	主見出し項目を参照してください。	なし	1 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、<library_effects>要素の例です。

```
<library_effects>
  <effect id="fullscreen_effect1">
    ...
  </effect>
</library_effects>
```

library_images

カテゴリ: テクスチャリング

概要

<image>アセットを格納するためのライブラリを提供します。

コンセプト

<image>要素は、画像データ、もしくは、そのデータを後で受信するレンダリング可能なオブジェクトを表します。

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

<library_images>要素には、以下の属性があります。

id	xs:ID	<library_images> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

<library_images>要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><image></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<library_images>`要素の例です。

```
<library_images>
  <image name="Rose">
    <init_from>../flowers/rose01.jpg</init_from>
  </image>
</library_images>
```

library_materials

カテゴリ: マテリアル

プロファイル: 外部

概要

`<material>`アセットを格納するためのライブラリを提供します。

コンセプト

エフェクトは、ジオメトリの外観、および画面領域の画像処理に必要な数式を定義します。マテリアルは、エフェクトをインスタンス化して、パラメータ値を選択し、さらに、テクニクを選択します。

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_materials>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_materials>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><material></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<library_materials>`要素の例です。

```

<library_materials>
  <material id="mat1">
    ...
  </material>

  <material id="mat2">
    ...
  </material>

</library_materials>

```

linker

カテゴリ: シェーダ

プロファイル: GLES2

概要

シェーダをプログラムに組み込むためのシェーダリンカ用の、コマンドラインもしくはランタイム呼出しオプションが含まれます。

コンセプト

コンパイルおよびリンクは、プログラマフレンドリーな高水準コードを、マシンで実行可能な問題に変換する、複雑な処理の一部です。この処理の詳細をここで説明することはできません。このような処理は、通常、ターゲットプラットフォームやプロファイルに依存しています。

一般に、GLSL や GLES2 が提供する API 関数からリンカを呼び出す際には、API オプションは不要です。また、GLES2 のベースライン API は、リンク済みのバイナリを明示的にサポートしていません。けれども、プラットフォームの中には、必要なシェーダのあらゆる組合せ管理するという代償を払って、このようなさらなる最適化の機会を提供しています。

プラットフォームがバイナリをサポートしていて、再コンパイルの必要がない場合、この要素に、コンパイラのバイナリ結果を格納できることもあります。

属性

`<linker>`要素には、以下の属性があります。

<code>platform</code>	<code>xs:string</code>	必須。複数のリンカ設定を識別するためのサブプラットフォーム名。
<code>target</code>	<code>xs:string</code>	オプション。ターゲットバイナリプロファイル。

options	xs:string	オプション。リンクオプション。詳しくは、プラットフォームプロバイダのドキュメントを参照してください。
---------	-----------	--

関連要素

<linker>要素は、以下の要素と関連性があります。

親要素	program
子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<binary>	主見出し項目を参照してください。	なし	0 以上

詳細

この要素には、ツールにテキスト文字列として渡されるリンクオプションのテキストが含まれます。この要素には、オプションで、コンパイルおよびリンク結果のバイナリ表現を含むこともできます。

例

```
<linker platform="PC" target="assemblyProfile" options="--debug"/>
```

material

カテゴリ: **マテリアル**

プロファイル: **外部**

概要

ジオメトリの外観、および画面領域の画像処理に必要な数式を定義します

コンセプト

マテリアルは、エフェクトをインスタンス化して、パラメータ値を選択し、さらに、テクニクを選択します。マテリアルは、ジオメトリオブジェクトの外観を記述するだけでなく、ブラー、ブルーム、カラーフィルタといったカメラレンズ的なエフェクトを生み出すための画面領域処理を行うこともできます。

コンピュータグラフィックスでは、ジオメトリオブジェクトに対して、そのマテリアル(素材)の特性を記述する各種のパラメータを持つことができます。このマテリアルの特性がレンダリング計算時のパラメータとして使用され、最終出力においてオブジェクトの外観を決定します。同様に、画面空間処理や合成を行う際にも、計算を実行するために、さまざまなパラメータが必要になることがあります。

実際のマテリアルのパラメータは、利用するグラフィックスレンダリングシステムに依存します。固定機能グラフィックスパイプラインでは、Phong シェーディングのような定義済みの照明モデルを解くためのパラメータが必要となります。これらのパラメータには、環境、拡散、鏡面の各反射率などが含まれます。

これに対して、プログラマブルグラフィックスパイプラインでは、マテリアルのパラメータセットをプログラマが定義します。これらの定義されたパラメータが、頂点プログラムやピクセルプログラムで定義されているレンダリングアルゴリズムで使用されます。

属性

`<material>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<material>`要素は、以下の要素と関連性があります。

親要素	library_materials
子要素	下記サブセクションを参照してください。
その他	instance_material (ジオメトリ)

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><instance_effect></code>	主見出し項目を参照してください。	なし	1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、簡単な`<material>`要素の例です。このマテリアルは、マテリアル `<library_materials>` 要素に含まれています。

```
<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</float>
    </setparam>
  </instance_effect>
</material>
</library_materials>
```

modifier

カテゴリ: パラメータ

プロファイル: External、Effect、CG、COMMON、GLES、GLES2、GLSL

概要

宣言の可変性またはリンク<newparam>に関する追加情報を指定します。

コンセプト

COLLADA FX パラメータ宣言で、定数、外部またはユニフォームパラメータを指定することができます。

属性

<modifier>要素には属性はありません。

関連要素

<modifier>要素は、以下の要素と関連性があります。

親要素	newparam
子要素	なし
その他	なし

詳細

リンク修飾子が含まれます。どの FX ランタイムでもすべてのリンク修飾子がサポートされているわけではありません。有効な修飾子は、以下のとおりです。

- CONST
- UNIFORM
- VARYING
- STATIC
- VOLATILE
- EXTERN
- SHARED

例

```
<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>
```

pass

カテゴリ: レンダリング

プロファイル: CG、GLES、GLES2、GLSL

概要

1つのレンダリングパイプラインのすべてのレンダリングステートとシェーダと設定を静的に宣言します。

コンセプト

`<pass>`は、レンダリングパイプラインのすべてのレンダリングステートとシェーダについて記述するためのもので、プログラムでジオメトリを送り出す前に、FX ランタイムに対して、現在のグラフィックスのステートを「適用」するように指示するための要素です。

「静的な」宣言というのは、グラフィックスステートに適用するためにスクリプトエンジンやランタイムシステムで評価を行う必要がない宣言を意味します。`<pass>`が適用される時点では、すべてのレンダリングステートの設定とユニフォームパラメータは事前に計算されて値がわかっています。

属性

`<pass>`要素には、以下の属性があります。

sid	sid_type	オプションのラベルで、名前が <code><pass></code> を指定できるようにします。また必要であれば、テクニックを評価しながらアプリケーションで並べ替えられるようにします。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
-----	----------	--

関連要素

`<pass>`要素は、以下の要素と関連性があります。

親要素	<code>technique</code> (FX) (<code>profile_CG</code> 、 <code>profile_GLES</code> 、 <code>profile_GLSL</code> 、 <code>profile_GLES2</code> 内)
子要素	下のサブセクションを参照
その他	なし

GLES スコープ内の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><annotate></code>	主見出し項目を参照してください。	なし	0 または 1
<code><states></code>	主見出し項目を参照してください。	なし	0 または 1
<code><evaluate></code>	主見出し項目を参照してください。	なし	0 または 1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

CG、GLES2、または GLSL スコープ内の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><annotate></code>	主見出し項目を参照してください。	なし	0 または 1
<code><states></code>	主見出し項目を参照してください。	なし	0 または 1
<code><program></code>	主見出し項目を参照してください。	なし	0 または 1
<code><evaluate></code>	主見出し項目を参照してください。	なし	0 または 1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

`<pass>`の並べ替えは、1つの`<pass>`を繰り返し適用するような場合に役立ちます。たとえば、目的の効果を生み出すには、「ぼかし」のローパスコンボリューションをオフスクリーンのテクスチャに何度も適用しなければならない場合もあります。

例

以下は、`<profile_CG>`に含まれている`<pass>`の例です。

```
<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>
      <compiler platform="PC" target="GLSLF"/>
    </shader>
  </program>
</pass>
```

phong

カテゴリ: レンダリング

プロファイル: 共通

概要

Phong BRDF 近似に従って鏡面反射をシェーディングした、陰影のある面を生成します。

コンセプト

<profile_COMMON>効果の中で利用し、環境、拡散、鏡面反射を行うシェーディング処理されたサーフェスを生成する固定機能パイプラインを宣言します。その際、鏡面反射は Phong BRDF 近似にしたがってシェーディング処理されます。

<phong>シェーダは、共通の Phong シェーディング公式を利用します。

$$color = \langle emission \rangle + \langle ambient \rangle * al + \langle diffuse \rangle * \max(N \cdot L, 0) + \langle specular \rangle * \max(R \cdot I, 0)^{\langle shininess \rangle}$$

- al – シーンからの環境光の寄与分を表す定数。共通プロファイルでは、<visual_scene>の中の<light><technique_common><ambient><color>の値の合計になります。
- N – 法線ベクトル (正規化済み)
- L – 光ベクトル
- I – 視線ベクトル
- R – 完全反射ベクトル (L の N に関する鏡映)

属性

<phong>要素には属性はありません。

関連要素

親要素	technique (FX) (profile_COMMON 内)
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<emission>	このオブジェクトのサーフェスから放射される光源の量を宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<ambient> (FX)	このオブジェクトのサーフェスから放射される環境光の量を宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<diffuse>	このオブジェクトのサーフェスから反射される拡散光の量を宣言します。	なし	0 または 1

名前/例	解説	デフォルト値	出現回数
	「 fx_common_float_or_param_type 」を参照してください。		
<code><specular></code>	このオブジェクトのサーフェスから反射される鏡面反射光の量を宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<code><shininess></code>	このオブジェクトのサーフェスから鏡面反射突起部分の鏡面度または荒さを宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<code><reflective></code>	完全鏡面反射のカラーを宣言します。 「 fx_common_color_or_texture_type 」を参照してください。	なし	0 または 1
<code><reflectivity></code>	反射光源に追加する完全鏡面反射の量 (0.0~1.0) を宣言します。 「 fx_common_float_or_param_type 」を参照してください。	なし	0 または 1
<code><transparent></code>	完全鏡面反射光源のカラーを宣言します。 詳しくは、 fx_common_color_or_texture_type および 7 章「FX 入門」の「透明度 (不透明度) を判定する」を参照してください。	なし	0 または 1
<code><transparency></code>	反射カラーに追加する完全鏡面反射光源の量 (0.0 ~1.0) をスカラー値として宣言します。 詳しくは、 fx_common_float_or_param_type および 7 章「FX 入門」の「透明度 (不透明度) を判定する」を参照してください。	なし	0 または 1
<code><index_of_refraction></code>	単一スカラーインデックスとして完全鏡面反射光源の反射インデックスを宣言します。 「 fx_common_float_or_param_type 」を参照してください。	なし	0 または 1

例

この例には、以下のような性質があります。

- これは、パラメータとして拡散色を受け取るエフェクトです。拡散色は、デフォルトで(0.2 0.56 0.35) に設定されますが、マテリアルの中でオーバーライドすることができます。
- これは、いかなる光も放射しないし、いかなる間接照明 (環境光) も吸収しません。
- これは、光沢のある小さな点があります。50 という値は、光沢指数の項としてはやや大きいので、この光沢はかなり鮮明になるはずですが。
- これは反射性があり、標準の表面色計算に加えて 5%の強度で環境光を反射します。
- これは、透明ではありません。詳しくは、7 章「FX 入門」の「透明度 (不透明度) を判定する」を参照してください。

```

<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>

```

profile_BRIDGE

カテゴリ: プロファイル

プロファイル: BRIDGE

概要

外部規格で書かれたエフェクトプロファイルの参照に対するサポートを提供します。

コンセプト

この要素を使うと、現在 COLLADA が直接サポートしていないシステムで作業を行ったり、COLLADA 以前に書かれた既存のエフェクトライブラリを参照したり、COLLADA を使わないことを選んだ人によって書かれたエフェクトを利用したりすることができます。

この要素は、COLLADA FX スキーマの一部でない追加の表現、API、プラットフォームに対するサポートを維持しつつ、COLLADA FX エフェクトを複数の API、プラットフォーム、共通プロファイルで表現することを可能にします。

FX にブリッジできる規格の例としては、Microsoft/HLSL、CgFX (NVIDIA®)、SushiFX (AMD) などがあります。

この機能は、

- 既存の COLLADA 標準およびスキーマを拡張して、単一エフェクト複数プロファイルのパラダイムを可能にします。
- 現在 COLLADA FX ではサポートされておらず、COLLADA スキーマのリビジョン時に導入される、実行時の手続き的なエフェクト構築のない、COLLADA FX のシェーダ言語、エフェクト言語、API の陳腐化を防ぐ。

使用の際には、エフェクトファイルはプロファイルとしてインポートされます。エフェクトファイルのパラメータは、プロファイルレベルのパラメータになります。パラメータのスコープ付き識別子 (SID) や名前は、エフェクトファイルのパラメータ名と同じです。テクニックやパスも同じルールに従います。つまり、そのファイル中の名前を `<material>` の `<setparam>` や `<technique_hint>`、あるいは、COLLADA 文書の他の場所から参照する際には、SID で参照します。

インポートされた `<profile_BRIDGE>` 要素は、特定のプロファイル用にプラットフォーム固有の値と宣言をすべてカプセル化します。 `<profile_BRIDGE>` ブロックによってインポートされたパラメータは、他のプロファイルでは利用できません。

`<profile_BRIDGE>` 要素は、具体化されたプラットフォーム固有のデータ型と、ドキュメント中で利用されている COLLADA の抽象データ型との明確なインタフェースを定義します。この範囲の外側で宣言

されたパラメータの場合、`<profile_BRIDGE>`ブロックの中で利用する際にキャストしなければならない場合があります。

詳しくは、7章「FX 入門」の「プラットフォーム固有のエフェクトにプロファイルを利用する」を参照してください。

属性

`<profile_BRIDGE>`要素には、以下の属性があります。

<code>id</code>	<code>xs:ID</code>	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
<code>platform</code>	<code>xs:NCName</code>	プラットフォームの種類。これはベンダ定義された文字列で、プラットフォームや機能ターゲット（通常は OpenGL ES 2.0 プラットフォーム）を表します。ターゲットは、特定のハードウェアであることも、ハードウェアのファミリーであることもあります。オプション。
<code>url</code>	<code>xs:anyURI</code>	橋渡しをするファイルの URI。必須。

関連要素

`<profile_BRIDGE>`要素は、以下の要素と関連性があります。

親要素	<code>effect</code>
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	リソース管理のトラッキング用です。コアの主見出し項目を参照してください。	なし	0 または 1
<code><extra</code>	COLLADA スキーマ仕様外の拡張データを格納する手段です。コアの主見出し項目を参照してください。	なし	0 以上

詳細

例

```
<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>
```

profile_CG

カテゴリ: プロファイル

プロファイル: CG

概要

NVIDIA® Cg 言語で書かれたエフェクトのプラットフォーム固有の表現を宣言します。

コンセプト

`<profile_CG>`要素は、プラットフォーム固有の値および宣言のすべてをカプセル化して特定の外観を実現する、エフェクト中のプロファイルです。`<effect>`範囲内では、すべてのプラットフォームでパラメータが利用できますが、`<profile_CG>`ブロック内で宣言されたパラメータは、そのプロファイル中のシェーダだけが利用できます。

`<profile_CG>`要素は、具体化されたプラットフォーム固有のデータ型と、ドキュメント中で利用されている COLLADA の抽象データ型との明確なインタフェースを定義します。この範囲の外側で宣言されたパラメータの場合、`<profile_CG>`ブロックの中で利用する際にキャストしなければならない場合があります。

詳しくは、7章「FX 入門」の「プラットフォーム固有のエフェクトにプロファイルを利用する」を参照してください。

属性

`<profile_CG>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
platform	xs:NCName	プラットフォームの種類。ベンダ定義された文字列で、プラットフォームまたは technique の機能ターゲットを表します。デフォルトは"PC"。オプション。

関連要素

`<profile_CG>`要素は、以下の要素と関連性があります。

親要素	<code>effect</code>
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。ただし、以下は例外です。

- `<include>`と`<code>`は、入れ替えることができます。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><code></code>	主見出し項目を参照してください。	なし	0 以上
<code><include></code>	主見出し項目を参照してください。	なし	0 以上
<code><newparam></code>	主見出し項目を参照してください。	なし	0 以上
<code><technique></code> (FX)	詳しくは、属性の主見出し項目と説明、および以下の	なし	1 以上

名前/例	解説	デフォルト値	出現回数
	子要素詳細のサブセクションを参照してください。		
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

<profile_CG> / <technique>の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>	コアの主見出し項目を参照してください。	なし	0 または 1
<annotate>	主見出し項目を参照してください。	なし	0 以上
<pass>	主見出し項目を参照してください。	なし	1 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

例

```

<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>
      </shader>
      <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>
    </program>
    <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>
  </pass>
</technique>
</profile_CG>

```

profile_COMMON

カテゴリ: プロファイル

プロファイル: 共通

概要

プラットフォームに関係のない共通の固定機能シェーダの宣言ブロックをオープンします。

コンセプト

`<profile_COMMON>`要素は、プラットフォームに依存しない固定機能シェーダのためのあらゆる値および宣言をカプセル化します。`<profile_COMMON>`は、どのプラットフォームでもサポートする必要があります。

`<profile_COMMON>`効果は、現在の効果ランタイムで他のプロファイルを認識できない際に、信頼性の高いフォールバックとして利用するように設計されています。詳しくは、7章「FX入門」の「プラットフォーム固有のエフェクトにプロファイルを利用する」を参照してください。

属性

`<profile_COMMON>`要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
----	-------	---

関連要素

`<profile_COMMON>`要素は、以下の要素と関連性があります。

親要素	<code>effect</code>
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><newparam></code>	すべてのプラットフォームで認識可能な制約された型セット(補足のセマンティックを伴った <code><float></code> 、 <code><float2></code> 、 <code><float3></code> 、 <code><float4></code> 、 <code><sampler2D></code>)から新しいパラメータを作成します。主見出し項目を参照してください。 例： <pre><newparam sid="mySID"> <semantic> DIFFUSECOLOR </semantic> <float3> 1 2 3 </float3> </newparam></pre>	なし	0 以上
<code><technique></code> (FX)	この効果用の <code>technique</code> を1つだけ宣言します。詳しくは、属性の主見出し項目と説明、および以下の子要素詳細のサブセクションを参照してください。	なし	1

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

`<profile_COMMON>` / `<technique>`の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code>shader_element</code>	<code><constant></code> (FX)、 <code><lambert></code> 、 <code><phong></code> 、 <code><blinn></code> のいずれか。 主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

```

<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>

```

profile_GLES

カテゴリ: プロファイル

プロファイル: GLES

概要

OpenGL ES のためのプラットフォーム固有のデータ型および`<technique>`を宣言します。

コンセプト

`<profile_GLES>`要素は、特定のプロファイル用にプラットフォーム固有の値と宣言をすべてカプセル化します。`<effect>`範囲内では、すべてのプラットフォームでパラメータが利用できますが、`<profile_GLES>`ブロック内で宣言されたパラメータは、そのプロファイル中のシェーダだけが利用できます。

`<profile_GLES>`要素は、具体化されたプラットフォーム固有のデータ型と、ドキュメント中で利用されている COLLADA の抽象データ型との明確なインタフェースを定義します。この範囲の外側で宣言さ

れたパラメータの場合、<profile_GLES>ブロックの中で利用する際にキャストしなければならない場合があります。

詳しくは、7章「FX 入門」の「プラットフォーム固有のエフェクトにプロファイルを利用する」を参照してください。

属性

<profile_GLES>には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
platform	xs:NMTOKEN	プラットフォームの種類。ベンダ定義された文字列で、プラットフォームまたは technique の機能ターゲットを表します。オプション。

関連要素

<profile_GLES>要素は、以下の要素と関連性があります。

親要素	effect
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<asset>	コアの主見出し項目を参照してください。	なし	0 または 1
<newparam>	すべてのプラットフォームで認識可能な制約された型セット(補足のセマンティックを伴った<float>、<float2>、<float3>、<float4>、<surface>、<sampler2D>)から新しいパラメータを作成します。主見出し項目を参照してください。 例： <pre><newparam sid="mySID"> <semantic> DIFFUSECOLOR </semantic> <float3> 1 2 3 </float3> </newparam></pre>	なし	0 以上
<technique> (FX)	このエフェクト用のテクニックを宣言します。詳しくは、属性の主見出し項目と説明、およびそれに続く子要素詳細のサブセクションを参照してください。	なし	1 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

<profile_GLES> / <technique>の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>	コアの主見出し項目を参照してください。	なし	0 または 1

名前/例	解説	デフォルト値	出現回数
<annotate>	主見出し項目を参照してください。	なし	0 以上
<pass>	主見出し項目を参照してください。	なし	1 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下の例は、地形レンダリングによる2つの異なる地面テクスチャの間の移行を示しています。この例では、各テクセルのブレンドパーセンテージを指定するアルファトランジションテクスチャを使って、砂利テクスチャと草地テクスチャを組み合わせています。

```

<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>

```

profile_GLES2

カテゴリ: プロファイル

プロファイル: GLES2

概要

OpenGL ES 2.0 のためのプラットフォーム固有のデータ型および<technique>を宣言します。

コンセプト

<profile_GLES2>は、OpenGL ES 2.0 (GLES2) に対するサポートを提供します。このプロファイルの構造は、Cg や GLSL のような他のシェーダをベースにしたプロファイルと似ていますが、OpenGL ES 2.0 の対象や細部に重点をおいています。

属性

<profile_GLES2>要素には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
language	xs:NCName	使用するシェーディング言語。現時点で有効な言語は、GLSL-ES および Cg です。必須。
platforms	list_of_names_type	プラットフォームの種類。これらは、テクニクのプラットフォームや機能ターゲットを示す、ベンダ定義の文字列です。複数の OpenGL ES 2.0 プラットフォームをサポートすることが可能になっています。ターゲットは、特定のハードウェアであることも、ハードウェアのファミリーであることもあります。オプション。

関連要素

<profile_GLES2>要素は、以下の要素と関連性があります。

親要素	effect
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。ただし、以下は例外です。

- <code>と<include>は、入れ替えることができます。

名前/例	解説	デフォルト値	出現回数
<asset>	リソース管理のトラッキング用です。コアの主見出し項目を参照してください。	なし	0 または 1
<code>	ソースコードの埋め込みブロック。主見出し項目を参照してください。	なし	0 以上
<include>	URL によって参照されるソースコードのブロック。主見出し項目を参照してください。	なし	0 以上
<newparam>	シェーダを渡す新しいパラメータの宣言。主見出し項目を参照してください。	なし	0 以上
<technique> (FX)	プロファイルをレンダリングする主な方法や代替の方法。	なし	1 以上

名前/例	解説	デフォルト値	出現回数
	通常は LOD です。主見出し項目を参照してください。詳しくは、属性の主見出し項目と説明、およびそれに続く子要素詳細のサブセクションを参照してください。		
<code><extra></code>	COLLADA スキーマ定義に含まれない拡張データを格納するための方法。コアの主見出し項目を参照してください。	なし	0 または 1

`<profile_GLES2>` / `<technique>`の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><annotate></code>	主見出し項目を参照してください。	なし	0 以上
<code><pass></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

このプロファイルには、以下のように、GLS2 API 固有の特性の一部が反映されています。

- コンパイル後に直接利用される Cg API や Direct3D9 シェーダオブジェクトとは異なり、GLS2 API シェーダオブジェクトは、ユーザによってコンパイルされ、さらに、プログラムオブジェクトにリンクされます。
- API は、ソースコードとバイナリシェーダの両方をサポートします。
 - バイナリシェーダをサポートするためには、かならずしも、ソースコードが GLSL ES である必要はありません。
- GLS2 が PC 上でサポートするのは、OpenGL 2.x API の限定されたサブセットだけです。この API は、シェーダベースのレンダリング、およびシェーダソースコードによって制御されない対応するレンダーステートだけを供給するように限定されています。
- GLS2 の詳細については、http://www.khronos.org/opengles/2_X/ にアクセスするか、もしくは、対象とするプラットフォームの実際のベンダによる文書を参照してください。
- GLS2 を他の COLLADA FX プロファイルと比べたときの最も重要な違いは、シェーダとプログラムを結びつける方法です。シェーダソースコードは、ソースのリストから構成されています。ソースコードのセグメントとしては、以下を任意の順序で組み合わせることができます。
 - 共用可能な埋め込まれた `<code>`
 - 共用可能な参照である `<include>`
 - uber-シェーダを設定する `#define` コマンドのような、リスト中のコードインライン。これは、ローカルな `#define` インラインコマンドによって uber-シェーダを特殊化することにより、共用可能なソースコードセグメントを再利用することを可能にします。

例

追加の例については、「付録 B：プロファイル GLSL および GLS2 のサンプル」を参照してください。

```
<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"/>
    </states>
    </pass>
</technique>

```

```

    <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_GLES>

```

profile_GLSL

カテゴリ: プロファイル

プロファイル: GLSL

概要

OpenGL シェーディング言語のためのプラットフォーム固有のデータ型および<technique>を宣言します。

コンセプト

<profile_GLSL>要素は、特定のプロファイル用のプラットフォーム固有の値と宣言をすべてカプセル化します。<effect>範囲内では、すべてのプラットフォームでパラメータが利用できますが、<profile_GLSL>ブロック内で宣言されたパラメータは、そのプロファイル中のシェーダだけが利用できます。

<profile_GLSL>要素は、具体化されたプラットフォーム固有のデータ型と、ドキュメント中で利用されている COLLADA の抽象データ型との明確なインタフェースを定義します。この範囲の外側で宣言さ

れたパラメータの場合、<profile_GLSL>ブロックの中で利用する際にキャストしなければならない場合があります。

詳しくは、7章「FX 入門」の「プラットフォーム固有のエフェクトにプロファイルを利用する」を参照してください。

属性

<profile_GLSL>には、以下の属性があります。

id	xs:ID	要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。オプション。
platform	xs:NMTOKEN	プラットフォームの種類。ベンダ定義された文字列で、プラットフォームまたは technique の機能ターゲットを表します。オプション。デフォルトは"PC"。

関連要素

<profile_GLSL>要素は、以下の要素と関連性があります。

親要素	effect
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。ただし、以下は例外です。

- <include>と<code>は、入れ替えることができます。

名前/例	解説	デフォルト値	出現回数
<asset>	コアの主見出し項目を参照してください。	なし	0 または 1
<code>	主見出し項目を参照してください。	なし	0 以上
<include>	主見出し項目を参照してください。	なし	0 回以上
<newparam>	すべてのプラットフォームで認識可能な制約された型セット(補足のセマンティックを伴った<float>、<float2>、<float3>、<float4>、<surface>、<sampler2D>)から新しいパラメータを作成します。主見出し項目を参照してください。 例： <pre><newparam sid="mySID"> <semantic> DIFFUSECOLOR </semantic> <float3> 1 2 3 </float3> </newparam></pre>	なし	0 以上
<technique> (FX)	このエフェクト用のテクニックを宣言します。詳しくは、属性の主見出し項目と説明、およびそれに続く子要素詳細のサブセクションを参照してください。	なし	1 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

<profile_GLSL> / <technique>の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<asset>	コアの主見出し項目を参照してください。	なし	0 または 1
<annotate>	主見出し項目を参照してください。	なし	0 以上
<pass>	主見出し項目を参照してください。	なし	1 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

例

「付録B：プロファイル GLSL および GLES2 のサンプル」を参照してください。

program

カテゴリ: シェーダ

プロファイル: CG、GLSL、GLES2

概要

複数のシェーダをリンクして、ジオメトリ処理用のパイプラインを生成します。

コンセプト

頂点シェーダやフラグメントシェーダなどのシェーダ作成する方法を記述します。さらに、この要素は、それらをリンクし、プログラムを生成して、それを GLES2 および GLSL 用のエフェクトパラメータとバインドする方法を記述します (Cg シェーダの場合には、プログラムではなくエフェクトパラメータにバインドします)。

属性

<program>要素には属性はありません。

関連要素

<program>要素は、以下の要素と関連性があります。

親要素	pass
子要素	下のサブセクションを参照
その他	なし

CG スコープ内の子要素

`<profile_CG>`のスコープ内に子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><shader></code>	頂点シェーダやピクセルシェーダのようなシェーダ用のセットアップ情報やコンパイル情報。主見出し項目を参照してください。	なし	0 以上

GLSL スコープ内の子要素

`<profile_GLSL>`のスコープ内に子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><shader></code>	頂点シェーダやピクセルシェーダのようなシェーダ用のセットアップ情報やコンパイル情報。主見出し項目を参照してください。	なし	0 以上
<code><bind_attribute></code>	シェーダ変数をエフェクトパラメータにバインドするための情報。主見出し項目を参照してください。	なし	0 以上
<code><bind_uniform></code>	シェーダの uniform 変数をパラメータや値にバインドします。主見出し項目を参照してください。	なし	0 以上

GLES2 スコープ内の子要素

`<profile_GLES2>`のスコープ内に子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><shader></code>	上記を参照してください。	なし	0 以上
<code><linker></code>	シェーダ全体もしくはリンクの結果をキャプチャした情報。主見出し項目を参照してください。	なし	0 以上
<code><bind_attribute></code>	シェーダ変数をエフェクトパラメータにバインドするための情報。主見出し項目を参照してください。	なし	0 以上
<code><bind_uniform></code>	シェーダの uniform 変数をパラメータや値にバインドします。主見出し項目を参照してください。	なし	0 以上

例

```

    <program>
    <shader stage="VERTEX"> ... </shader>
    <shader stage="FRAGMENT"> ... </shader>
    </program>

```


render

カテゴリ: レンダリング

プロファイル: 外部

概要

シーンを評価するための単一のエフェクトパスを記述します。

コンセプト

この要素は、カメラレンズや画面後処理のための、単一のレンダリングパスを表します。レンダリングは、特定のマテリアルやエフェクトのないレイヤレンダリングのような簡単なものでもよいし、一般に、後処理エフェクト、レンズエフェクト、シーンエフェクトなどと呼ばれるブラー、ブルーム、被写界深度などの特殊なエフェクトを追加することもできます。

また、この要素の中では、各パスでレンダリングを行うシーンの、カメラやレイヤを変更することもできます。

属性

`<render>`要素には、以下の属性があります。

name	xs:token	この要素の文字列の名。オプション。
sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
camera_node	xs:anyURI	この合成手順をレンダリングする視点を記述するカメラを含むノードを参照します。オプション。

関連要素

`<render>`要素は、以下の要素と関連性があります。

親要素	<code>evaluate_scene</code>
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><layer></code>	この合成手順が、シーンの評価中にレンダリングするレイヤ（複数も可）を指定します。 <code>xs:NCName</code> 型のレイヤ名が含まれます。この要素には属性はありません。	なし	0 以上
<code><instance_material></code> (レンダリング)	この合成手順が、シーンの評価中にレンダリングするエフェクトを指定します。主見出し項目を参照してください。	なし	0 または 1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

[<visual_scene>](#)を参照してください。

RGB

カテゴリ: テクスチャリング

プロファイル: GLES

概要

合成モードテクスチャリング操作の[<texture_pipeline>](#)コマンドのRGB部分を定義します。

コンセプト

割り当てと全体的なコンセプトの詳細に関しては[<texcombiner>](#)の解説を参照してください。

属性

[<RGB>](#)要素には、以下の属性があります。

operator	列挙	glTexEnv(TEXTURE_ENV, COMBINE_RGB, operator) の利用を想定。詳細は <texcombiner> を参照してください。有効な値は、以下の通りです。 REPLACE MODULATE ADD ADD_SIGNED INTERPOLATE SUBTRACT DOT3_RGB DOT3_RGBA
scale	float_type	glTexEnv(TEXTURE_ENV, RGB_SCALE, scale) の利用を想定。詳細は <texcombiner> を参照してください。

関連要素

[<RGB>](#)要素は以下の要素と関連があります。

親要素	texcombiner
子要素	下のサブセクションを参照
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<argument>	実行する特定の操作に必要な引数を設定します。主見出し項目を参照してください。	なし	1 から 3

詳細

詳細は[<texcombiner>](#)を参照してください。

例

[<texture_pipeline>](#)を参照してください。

sampler1D

カテゴリ: テクスチャリング

プロファイル: COMMON、CG、GLSL

概要

1次元のテクスチャサンプラを宣言します。

コンセプト

属性

`<sampler1D>`要素には属性はありません。

関連要素

`<sampler1D>`要素は、以下の要素と関連性があります。

親要素	コア要素: <code>newparam</code> , <code>setparam</code> FX 要素: <code>array</code>
子要素	「 <code>fx_sampler_common</code> 」を参照してください。
その他	なし

詳細

一次元サンブラは t、p 座標軸を使わないので、`<wrap_t>`、`<wrap_p>`を使っても結果に影響はありません。

例

この例は、UV が 0~1 の範囲を越えるかどうかに関わらず、テクスチャを面全体に繰り返します。テクスチャを拡大する必要がある場合には、線形に拡大します。ラスタライズするピクセルよりテクセルの方が小さい場合には、トライリニアフィルタリングを行います。この要素は、一次元の面、つまり $N \times 1$ (height=1) の面から読み込みを行います。

```
<sampler1D>
  <wrap_s>WRAP</wrap_s>
  <minfilter>LINEAR</minfilter>
  <magfilter>LINEAR</magfilter>
</sampler1D>
```

sampler2D

カテゴリ: テクスチャリング

プロファイル: External、Effect、CG、COMMON、GLES2、GLSL

概要

2次元のテクスチャサンプラを宣言します。

属性

<sampler2D>要素には属性はありません。

関連要素

<sampler2D>要素は、以下の要素と関連性があります。

親要素	コア要素: newparam , setparam FX 要素: array
子要素	「 fx_sampler_common 」を参照してください。
その他	なし

詳細

1次元サンプラは p 座標軸を使わないので、<wrap_p>を使っても結果に影響はありません。

例

これは、最も一般的なサンプラ型の例です。この例は、UV が 0~1 の範囲を越えるかどうかに関わらず、テクスチャを面全体に繰り返します。テクスチャを拡大する必要がある場合には、線形に拡大します。ラスタ化するピクセルよりテクセルの方が小さい場合には、トライリニアフィルタリングを行います。

```
<sampler2D>
  <wrap_s>WRAP</wrap_s>
  <wrap_t>WRAP</wrap_t>
  <minfilter>LINEAR</minfilter>
  <magfilter>LINEAR</magfilter>
</sampler2D>
```

sampler3D

カテゴリ: テクスチャリング

プロファイル: External、Effect、CG、COMMON、GLES2、GLSL

概要

3次元のテクスチャサンプラを宣言します。

属性

<sampler3D>要素には属性はありません。

関連要素

`<sampler3D>`要素は、以下の要素と関連性があります。

親要素	コア要素: <code>newparam</code> , <code>setparam</code> FX 要素: <code>array</code>
子要素	「 <code>fx_sampler_common</code> 」を参照してください。
その他	なし

例

この例は、UV が 0~1 の範囲を越えるかどうかに関わらず、テクスチャを面全体に繰り返します。テクスチャを拡大する必要がある場合には、線形に拡大します。ラスタライズするピクセルよりテクセルの方が小さい場合には、トライリニアフィルタリングを行います。

この例では、3次元テクスチャ、つまり、立体から典型的なサンプリング操作を行います。これは、ノイズ、医学画像、木材などのようなパターンからの読み取りを行う際に一般的です。

```
<sampler3D>
  <wrap_s>WRAP</wrap_s>
  <wrap_t>WRAP</wrap_t>
  <wrap_p>WRAP</wrap_p>
  <minfilter>LINEAR</minfilter>
  <magfilter>LINEAR</magfilter>
</sampler3D>
```

samplerCUBE

カテゴリ: テクスチャリング

プロファイル: External、Effect、CG、COMMON、GLES2、GLSL

概要

キューブマップ用のテクスチャサンプラを宣言します。

属性

`<samplerCUBE>`要素には属性はありません。

関連要素

`<samplerCUBE>`要素は、以下の要素と関連性があります。

親要素	コア要素: <code>newparam</code> , <code>setparam</code> FX 要素: <code>array</code>
子要素	「 <code>fx_sampler_common</code> 」を参照してください。
その他	なし

詳細

一次元サンプラは p 座標軸を使わないので、`<wrap_p>`を使っても結果に影響はありません。

例

この例では、キューブマップの面から読み込みを行います。シェーダは法線の3次元ベクトルを渡します。この法線は、キューブマップの6面のうちの1面上の位置を指しています。この法線の示す座標の周囲のサンプルが、フィルタリングされて返されます。

```
<samplerCUBE>
  <wrap_s>WRAP</wrap_s>
  <wrap_t>WRAP</wrap_t>
  <minfilter>LINEAR</minfilter>
  <magfilter>LINEAR</magfilter>
</samplerCUBE>
```

samplerDEPTH

カテゴリ: テクスチャリング

プロファイル: External、Effect、CG、COMMON、GLSL

概要

奥行きマップのテクスチャサンプラを宣言します。

属性

`<samplerDEPTH>`要素には属性はありません。

関連要素

`<samplerDEPTH>`要素は、以下の要素と関連性があります。

親要素	コア要素: <code>newparam</code> , <code>setparam</code> FX 要素: <code>array</code>
子要素	「 <code>fx_sampler_common</code> 」を参照してください。
その他	なし

詳細

一次元サンプラは `p` 座標軸を使わないので、`<wrap_p>` を使っても結果に影響はありません。

例

この例は、UV が 0~1 の範囲を越えるかどうかに関わらず、テクスチャを面全体に繰り返します。テクスチャを拡大する必要がある場合には、線形に拡大します。ラスタライズするピクセルよりテクセルの方が小さい場合には、トライリニアフィルタリングを行います。表面がデプスデータである場合には、近傍比率フィルタリングを実行します。このテクニックは、シャドウマップなどとして使うデプスマップをサンプリングする際に、よい結果が得られます。

```
<samplerDEPTH>
  <wrap_s>WRAP</wrap_s>
  <wrap_t>WRAP</wrap_t>
  <minfilter>LINEAR</minfilter>
  <magfilter>LINEAR</magfilter>
</samplerDEPTH>
```

samplerRECT

カテゴリ: テクスチャリング

プロファイル: External、Effect、CG、COMMON、GLSL

概要

RECT テクスチャサンプラを宣言します。

コンセプト

RECT テクスチャは、OpenGL 拡張の 1 つです。RECT テクスチャは、2 次元の不等辺四角形テクスチャとは違います。この要素は、<sampler2D>の一般的な不等辺四角形の代わりではなく、通常はレンダーターゲットもしくは画面空間の処理として使われます。詳しくは、www.opengl.org/registry/specs/ARB/texture_rectangle.txt を参照してください。

属性

<samplerRECT>要素には属性はありません。

関連要素

<samplerRECT>要素は、以下の要素と関連性があります。

親要素	コア要素: <code>newparam</code> , <code>setparam</code> FX 要素: <code>array</code>
子要素	「 <code>fx_sampler_common</code> 」を参照してください。
その他	なし

詳細

RECT は、OpenGL RECT サンプラを反映しています。このサンプラは、DirectX ではサポートされません。RECT は 2 次元です。RECT は、MIP マッピングをサポートしません。サンプルは、0~1 の 2 次元範囲ではなく、[0~幅、0~高さ]の範囲にある `float2_type` を使います。

例

RECT サンプラは、非常に制限されています。このサンプラは MIP マッピングをサポートしていないので、この簡単な例が実は最も一般的な使い方です。

```
<samplerRECT>
  <instance_image url="myRenderableSurface"/>
</samplerRECT>
```

sampler_image

カテゴリ: パラメータ

プロファイル: 外部

概要

サンブラの対象となる画像をインスタンス化します。

コンセプト

これは、サンブラの一種ではなく、むしろ、既存のサンブラを変更するのに使われる要素です。親の `<setparam>` によって特定されるサンブラ `<newparam>` は、インスタンス化された画像を受信します。

詳しくは `<instance_image>` を参照してください。この派生型には、具体的な拡張はありませんが、このような状況を明確にするために名前を変更しています。

属性

`<instance_image>` を参照してください。

関連要素

`<sampler_image>` 要素は、以下の要素と関連性があります。

親要素	<code><instance_effect>/<setparam></code>
子要素	詳しくは <code>instance_image</code> を参照してください。
その他	なし

例

```
<material id="foo-smiley">
  <instance_effect url="foo">
    <setparam ref="bar">
      <sampler_image url="smiley-1"/>
    </setparam>
  </instance_effect>
</material>
```

sampler_states

カテゴリ: マテリアル

プロファイル: 該当なし

概要

マテリアルからエフェクトのサンブラ状態を変更することを可能にします。

コンセプト

この要素は、サンブラの基底型である (`fx_sampler_states`) の派生クラスです。有効な状態の一覧については、「`fx_sampler_common`」を参照してください。これには、`<instance_image>` 以外の

あらゆる要素が含まれます。マテリアルの<setparam>の ref 属性は、<sampler*>を含むエフェクトの<newparam>を参照します。<sampler_image>が、サンブラの<instance_image>を変更する(より一般的な操作)のに使われるのに対して、この要素は、サンブラのサンプリング状態だけを変更します。

以前の FX フレームワークでは、サンブラの状態をエフェクトの外で変更することはできず、このような機能は一般的ではないので、エフェクトオーサリングツールの多くはこの機能をサポートする予定はありません。この機能が COLLADA に含まれているのは、将来を考えて、柔軟性のあるゲームエンジン技術をベースにしたいからで、また、GL では歴史的にサンブラ状態がテクスチャオブジェクトに付随しているからです。

属性

<sampler_states>要素には属性はありません。

関連要素

<sampler_states>要素は、以下の要素と関連性があります。

親要素	setparam
子要素	「fx_sampler_common」を参照してください。
その他	なし

例

```
<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>
```

semantic

カテゴリ: **パラメータ**

プロファイル: **External、Effect、CG、COMMON、GLES、GLES2、GLSL**

概要

パラメータ宣言の目的を記述するメタデータを指定します。

コンセプト

セマンティックスは、オーバーロードの考えを利用して、効果中のパラメータ宣言の意図や目的を表します。セマンティックスは、これまで以下の3つの異なる種類のメタデータを表すのに利用されてきています。

- パラメータに割り当てられたハードウェアのリソース (例: **TEXCOORD2、NORMAL**)

- このパラメータで表現されているシーングラフまたはグラフィックス API からの値（例：
MODELVIEWMATRIX、CAMERAPOS、VIEWPORTSIZE）
- ランタイム時に効果を初期化する際に、アプリケーションで設定するユーザ定義された値（例：
DAMAGE_PERCENT、MAGIC_LEVEL）

セマンティックスは、マッピングを明確にするために<bind_material>メカニズムを利用して、シーングラフ中にある値とデータソースを効果パラメータにバインドするために、<node>中の<instance_geometry>宣言で利用されます。

属性

<semantic>要素には属性はありません。

関連要素

<semantic>要素は、以下の要素と関連性があります。

親要素	newparam
子要素	なし
その他	なし

詳細

現時点では、セマンティックスの標準セットはありません。この要素は、アプリケーションによって定義された任意の `xs:NCName` を含むことができます。

詳しくは、3章「スキーマのコンセプト」の「共通プロファイル」を参照してください。

例

```
<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>
```

shader

カテゴリ: シェーダ

プロファイル: CG、GLES2、GLSL

概要

<pass>のレンダリングパイプラインで実行するシェーダを宣言して準備を整えます。

コンセプト

実行可能形式のシェーダは、特定の段階でレンダリングパイプラインを行う小さな関数またはプログラムです。こういったシェーダは、事前にロードされコンパイルされたバイナリから、もしくは埋め込まれたソースコードからランタイム時に動的に生成させて組み立てることができます。<shader>宣言は、シェーダをコンパイルして、値または事前に定義されたパラメータをユニフォーム入力にバインドするために必要なすべての情報を保持します。

COLLADA FX では、FX ランタイムのサポートに応じて、ソースコードシェーダと事前にコンパイルされたバイナリシェーダの両方が宣言できます。事前にコンパイルされたバイナリシェーダの場合、すでにコンパイル時にターゲットプロファイルが指定されていますが、バイナリヘッダをロードして解析しなくても、事前にコンパイルされたシェーダに関係した宣言を COLLADA リーダで検証できるようにするために、やはりプロファイル宣言が必要となります。

事前に定義されたパラメータ、ソースシェーダ、バイナリシェーダは、同じ名前空間/シンボルテーブル/ソースコード文字列にまとめられるとみなされるため、すべてのシンボルと関数がシェーダの宣言で利用でき、共通の関数、たとえば、共通のライティングコードを `<technique>` 中の複数のシェーダで利用することが可能です。「変換ユニット」の考えを利用した FX ランタイムでは、それぞれのソースコードブロックに名前を付けて、名前空間をそういったユニットに分割することができます。

ユニフォーム入力パラメータを持つシェーダは、シェーダの宣言時に、事前に定義されたパラメータもしくはリテラル値をこれらの値にバインドすることができ、必要であれば、コンパイラに対してリテラルや定数の値をインライン化させることが可能です。

属性

`<shader>` 要素には、以下の属性があります。

<code>stage</code>	列挙	必須。プログラマブルシェーダの実行対象となるパイプラインの段階を指定します。有効な値は、以下の通りです。TESSELATION、VERTEX、GEOMETRY、FRAGMENT。
--------------------	----	---

関連要素

`<shader>` 要素は、以下の要素と関連性があります。

親要素	<code>program</code>
子要素	下のサブセクションを参照
その他	なし

CG スコープ内の子要素。

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><sources></code>	1つまたは複数のソースからのシェーダ用ソースコードを連結します。主見出し項目を参照してください。	なし	1
<code><compiler></code>	複数のプラットフォーム用のコンパイラ情報。主見出し項目を参照してください。	なし	0 以上
<code><bind_uniform></code>	主見出し項目を参照してください。	なし	0 以上

GLS2 スコープ内の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><sources></code>	複数のソースからのシェーダ用ソースコードを連結します。主見出し項目を参照してください。	なし	1
<code><compiler></code>	複数のプラットフォーム用のコンパイラ情報。主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

GLSL スコープ内の子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><sources></code>	複数のソースからのシェーダ用ソースコードを連結します。主見出し項目を参照してください。	なし	1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<profile_CG>`の例です。

```

<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>

```

sources

カテゴリ: シェーダ

プロファイル: CG、GLES2、GLSL

概要

1つまたは複数のソースからのシェーダ用ソースコードを連結します。

コンセプト

ときどき、シェーダのソースコードが、単一のインクルードファイルや単一の埋め込みコードブロックだけに入りきらないことがあります。また、複数のコードブロックの共通部分を組み合わせたいと考える人もいるかもしれません。

1つの例として、万能シェーダを書いて、それを`<import>`を使って`<sources>`の中に取り込んでから、その`<import>`の上に任意の数の`<inline>`ブロックを追加して、`#defines`を使って万能シェーダをカスタマイズするといった方法があります。

また別の例としては、基本式を定義するプラグイン可能な主要機能シェーダを書いて、拡張性については関数呼出しに依存するといった方法もあります。このような場合、`source`の複数の子要素を利用して、メイン関数に関数ブロックを組み込むことができます。

属性

`<sources>`要素には、以下の属性があります。

<code>entry</code>	<code>xs:token</code>	Cg スコープ内では必須です。GLES2 スコープ内ではオプションです。それ以外では無効です。このシェーダのエントリ関数名。この属性は、子要素が連結されたあとの、エントリポイントの名前を特定します。GLES2 では、デフォルトは「main」です。
--------------------	-----------------------	---

関連要素

`<sources>`要素は、以下の要素と関連性があります。

親要素	<code>shader</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素は、任意の順序、任意の組合せで出現することができます。

名前/例	解説	デフォルト値	出現回数
<code><inline></code>	<code>xs:string</code> 型で、インポートされるシェーダ用の <code>#define</code> のようなコードが含まれます。	なし	1 以上
<code><import ref=""></code>	<code><import></code> 要素自体には、データは含まれません。 <code>ref</code> 属性は必須属性で、プロファイルやエフェクトレベルの <code><code></code> 要素や <code><include></code> 要素のSIDが含まれます。 <code>ref</code> 属性は必須です。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。	なし	0 以上

例

```

<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>

```

states

カテゴリ: レンダリング

プロファイル: CG、GLES、GLES2、GLSL

概要

親パスのために設定されるあらゆるレンダリング状態が含まれます。

コンセプト

異なるFX プロファイルは`<pass>`要素の中で利用できる別々のレンダリングステートセットを持ちます。

属性

`<states>`要素には属性はありません。

関連要素

`<states>`要素は、以下の要素と関連性があります。

親要素	<code>pass</code> (<code>profile_CG</code> 、 <code>profile_GLES</code> 、 <code>profile_GLES2</code> 、 <code>profile_GLSL</code> 内)
子要素	下記サブセクションを参照してください。
その他	なし

子要素

レンダー状態を表す子要素は、任意の順序と組合せで出現することができます。

各レンダー状態、または（レンダー状態の表に示すように子要素がある場合は）その子要素には、以下のような属性があります。

value	以下の表で指定された型	レンダー状態固有の値を与えます。特に表に書かれていなければ、 value と param のいずれか（両方ではない）が必要です。
param	sidref_type	レンダー状態が value の代わりに使うパラメータの SID を参照します。 value が指定されない場合には無効です。SIDREF の詳細は、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
index	以下の表で指定された型	これは通常数値です。この属性はどのレンダー状態にもあるわけではなく、その意味もレンダー状態によって異なります。詳しくはレンダー状態表を参照してください。必須かオプションかもそこに指定されています。

以下に例を示します。

```
<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>
```

以下のレンダー状態の詳細に関しては、OpenGL の仕様書を参照してください。参考資料。

- <http://www.opengl.org/documentation/specs/>
- <http://www.opengl.org/registry/>

以下の表に、<profile_CG>、<profile_GLSL>、<profile_GLES>、<profile_GLES2>のレンダリング状態を示しておきます。レンダー状態は、GLES、GLES2 プロファイルについて注記した違いを除けば、どのプロファイルでも同じです。

レンダー状態および子要素	有効な値もしくは型、およびインデックス属性	GLES	GLES2
alpha_func func value	NEVER、LESS、LEQUAL、EQUAL、GREATER、NOTEQUAL、GEQUAL、ALWAYS float 値の 0.0 ~ 1.0 (0.0 と 1.0 を含む)	はい	いいえ
alpha_test_enable	論理値	はい	いいえ
auto_normal_enable	論理値	いいえ	いいえ
blend_color	float4_type	いいえ	はい
blend_enable	論理値	はい	はい
blend_equation	FUNC_ADD、FUNC_SUBTRACT、FUNC_REVERSE_SUBTRACT、MIN、MAX	いいえ	はい
blend_equation_separate rgb alpha	blend_equation の値と同じ	いいえ	はい
blend_func src dest	(src と dest の両方) ZERO、ONE、SRC_COLOR、ONE_MINUS_SRC_COLOR、	はい	はい

レンダーステートおよび子要素	有効な値もしくは型、およびインデックス属性	GL ES	GL ES2
	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		
blend_func_separate src_rgb dest_rgb src_alpha dest_alpha	blend_func の値と同じ	いいえ	はい
clip_plane	float4_type index 属性は、クリッピングプレーンを指定します。必須。	はい bool4_type	いいえ
clip_plane_enable	論理値 index 属性は、クリッピングプレーンを指定します。オプション。	はい	いいえ
color_logic_op_enable	論理値	はい	いいえ
color_mask	bool4_type	はい	はい
color_material face mode	FRONT、BACK、FRONT_AND_BACK EMISSION、AMBIENT、DIFFUSE、 SPECULAR、 AMBIENT_AND_DIFFUSE	いいえ	いいえ
color_material_enable <color_material>の利用を有効もしくは無効にします。つまり、ランタイムがいつ glEnable (GL_COLOR_MATERIAL) や glDisable (GL_COLOR_MATERIAL) (または同等) を実行するべきかを示します。	論理値	はい	いいえ
cull_face	FRONT、BACK、FRONT_AND_BACK	はい	はい
cull_face_enable	論理値	はい	はい
depth_bounds depth_bounds_enable	float2_type 論理値	いいえ いいえ	いいえ いいえ
depth_clamp_enable	論理値	いいえ	いいえ
depth_func	NEVER、LESS、LEQUAL、EQUAL、 GREATER、NOTEQUAL、GEQUAL、 ALWAYS	はい	はい
depth_mask	論理値	はい	はい
depth_range	float2_type	はい	はい
depth_test_enable	論理値	はい	はい
dither_enable	論理値	はい	はい
fog_color	float4_type	はい	いいえ
fog_coord_src	FOG_COORDINATE、 FRAGMENT_DEPTH	いいえ	いいえ
fog_density	float_type	はい	いいえ
fog_enable	論理値	はい	いいえ

レンダーステートおよび子要素	有効な値もしくは型、およびインデックス属性	GL ES	GL ES2
<code>fog_end</code>	<code>float_type</code>	はい	いいえ
<code>fog_mode</code>	<code>LINEAR</code> 、 <code>EXP</code> 、 <code>EXP2</code>	はい	いいえ
<code>fog_start</code>	<code>float_type</code>	はい	いいえ
<code>front_face</code>	<code>CW</code> 、 <code>CCW</code>	はい	はい
<code>light_ambient</code>	<code>float4_type</code> index 属性は、光源を指定します。必須。	はい	いいえ
<code>light_constant_attenuation</code>	<code>float_type</code> index 属性は、光源を指定します。必須。	はい	いいえ
<code>light_diffuse</code>	<code>float4_type</code> index 属性は、光源を指定します。必須。	はい	いいえ
<code>light_enable</code>	論理値 index 属性は、光源を指定します。必須。	はい	いいえ
<code>light_linear_attenuation</code>	<code>float_type</code> index 属性は、光源を指定します。必須。	はい	いいえ
<code>light_model_ambient</code>	<code>float4_type</code>	はい	いいえ
<code>light_model_color_control</code>	<code>SINGLE_COLOR</code> 、 <code>SEPARATE_SPECULAR_COLOR</code>	いいえ	いいえ
<code>light_model_local_viewer_enable</code>	論理値	いいえ	いいえ
<code>light_model_two_side_enable</code>	論理値	はい	いいえ
<code>light_position</code>	<code>float4_type</code> index 属性は、光源を指定します。必須。	はい	いいえ
<code>light_quadratic_attenuation</code>	<code>float_type</code> index 属性は、光源を指定します。必須。	はい	いいえ
<code>light_specular</code>	<code>float4_type</code> index 属性は、光源を指定します。必須。	はい	いいえ
<code>light_spot_cutoff</code>	<code>float_type</code> index 属性は、光源を指定します。必須。	はい	いいえ
<code>light_spot_direction</code>	<code>float3_type</code> index 属性は、光源を指定します。必須。	はい	いいえ
<code>light_spot_exponent</code>	<code>float_type</code> index 属性は、光源を指定します。必須。	はい	いいえ
<code>lighting_enable</code>	論理値	はい	いいえ
<code>line_smooth_enable</code>	論理値	いいえ	いいえ
<code>line_stipple</code>	<code>int2_type</code>	いいえ	いいえ
<code>line_stipple_enable</code>	論理値	いいえ	いいえ
<code>line_width</code>	<code>float_type</code>	はい	はい
<code>logic_op</code>	<code>CLEAR</code> 、 <code>AND</code> 、 <code>AND_REVERSE</code> 、 <code>COPY</code> 、 <code>AND_INVERTED</code> 、 <code>NOOP</code> 、 <code>XOR</code> 、 <code>OR</code> 、 <code>NOR</code> 、 <code>EQUIV</code> 、 <code>INVERT</code> 、 <code>OR_REVERSE</code> 、 <code>COPY_INVERTED</code> 、 <code>NAND</code> 、 <code>SET</code>	はい	いいえ

レンダーステートおよび子要素	有効な値もしくは型、 およびインデックス属性	GL ES	GL ES2
logic_op_enable	論理値	いいえ	いいえ
material_ambient	float4_type	はい	いいえ
material_diffuse	float4_type	はい	いいえ
material_emission	float4_type	はい	いいえ
material_shininess	float_type	はい	いいえ
material_specular	float4_type	はい	いいえ
model_view_matrix	float4x4_type	はい	いいえ
multisample_enable	論理値	はい	いいえ
normalize_enable	論理値	はい	いいえ
point_distance_attenuation	float3_type	はい	いいえ
point_fade_threshold_size	float_type	はい	いいえ
point_size	float_type	はい	はい
point_size_enable	論理値	いいえ	はい - GL ES2 のみ
point_size_max	float_type	はい	いいえ
point_size_min	float_type	はい	いいえ
point_smooth_enable	論理値	いいえ	いいえ
polygon_mode face mode	FRONT、BACK、FRONT_AND_BACK POINT、LINE、FILL	いいえ	いいえ
polygon_offset	float2_type	はい	はい
polygon_offset_fill_enable	論理値	はい	はい
polygon_offset_line_enable	論理値	いいえ	いいえ
polygon_offset_point_enable	論理値	いいえ	いいえ
polygon_smooth_enable	論理値	いいえ	いいえ
polygon_stipple_enable	論理値	いいえ	いいえ
projection_matrix	float4x4_type	はい	いいえ
rescale_normal_enable	論理値	はい	いいえ
sample_alpha_to_coverage_enable	論理値	はい	はい
sample_alpha_to_one_enable	論理値	はい	いいえ
sample_coverage value invert	float_type 論理値	いいえ	はい - GL ES2 のみ
sample_coverage_enable	論理値	はい	いいえ
scissor	int4_type	はい	はい
scissor_test_enable	論理値	はい	はい
shade_model	FLAT、SMOOTH	はい	いいえ
stencil_func func ref mask	NEVER、LESS、LEQUAL、EQUAL、 GREATER、NOTEQUAL、GEQUAL、 ALWAYS 符号なし byte 符号なし byte	はい	はい
stencil_func_separate front back ref mask	(front と back) NEVER、LESS、LEQUAL、EQUAL、 GREATER、NOTEQUAL、GEQUAL、 ALWAYS 符号なし byte 符号なし byte	いいえ	はい
stencil_mask	int_type	はい	はい
stencil_mask_separate		いいえ	はい

レンダーステートおよび子要素	有効な値もしくは型、およびインデックス属性	GL ES	GL ES2
face mask	FRONT、BACK、FRONT_AND_BACK 符号なし byte		
stencil_op fail zfail zpass	(fail、zfail、zpass の場合) KEEP、ZERO、REPLACE、INCR、 DECR、INVERT、INCR_WRAP、 DECR_WRAP	はい	はい
stencil_op_separate face fail zfail zpass	FRONT、BACK、FRONT_AND_BACK (fail、zfail、zpass の場合) KEEP、ZERO、REPLACE、INCR、 DECR、INVERT、INCR_WRAP、 DECT_WRAP	いいえ	はい
stencil_test_enable	論理値	はい	はい
texture_env_color	float4_type index 属性は、テクスチャユニットを 指定します。オプション。	いいえ (<texture_pipeline>を参照)	いいえ
texture_env_mode	xs:string index 属性は、テクスチャユニットを 指定します。オプション。	いいえ (<texture_pipeline>を参照)	いいえ
texture_pipeline	文字列。 <texture_pipeline>パ ラメータの名前。	はい - GL ES のみ	いいえ
texture1D	sampler1D 型 index 属性は、テクスチャユニットを 指定します。必須。	いいえ	いいえ
texture1D_enable	論理値 index 属性は、テクスチャユニットを 指定します。オプション。	いいえ	いいえ
texture2D	sampler2D 型 index 属性は、テクスチャユニットを 指定します。必須。	いいえ (<texture_pipeline>を参照)	いいえ
texture2D_enable	論理値 index 属性は、テクスチャユニットを 指定します。オプション。	いいえ (<texture_pipeline>を参照)	いいえ
texture3D	sampler3D 型 index 属性は、テクスチャユニットを 指定します。必須。	いいえ	いいえ
texture3D_enable	論理値 index 属性は、テクスチャユニットを 指定します。オプション。	いいえ	いいえ
textureCUBE	samplerCUBE 型 index 属性は、テクスチャユニットを 指定します。必須。	いいえ	いいえ
textureCUBE_enable	論理値 index 属性は、テクスチャユニットを 指定します。オプション。	いいえ	いいえ
textureDEPTH	samplerDEPTH 型 index 属性は、テクスチャユニットを 指定します。必須。	いいえ	いいえ

レンダーステートおよび子要素	有効な値もしくは型、およびインデックス属性	GL ES	GL ES2
<code>textureDEPTH</code>	<code>samplerDEPTH</code> 型 index 属性は、テクスチャユニットを指定します。必須。	いいえ	いいえ
<code>textureDEPTH_enable</code>	論理値 index 属性は、テクスチャユニットを指定します。オプション。	いいえ	いいえ
<code>textureRECT</code>	<code>samplerRECT</code> 型 index 属性は、テクスチャユニットを指定します。必須。	いいえ	いいえ
<code>textureRECT_enable</code>	論理値 index 属性は、テクスチャユニットを指定します。オプション。	いいえ	いいえ

例

```
<states>
  <depth_test_enable value="true"/>
  <depth_func value="Less"/>
  <cull_face_enable value="true"/>
  <cull_face value="Back"/>
  <front_face value="CCW"/>
</states>
```

stencil_clear

カテゴリ: レンダリング

プロファイル: CG、GL ES、GL ES2、GL SL

概要

レンダリングターゲット画像をクリアするかどうか、クリアする場合にはどんな値を利用するのかを指定します。

コンセプト

レンダリングターゲット画像に描画を行う前に、空のキャンバス、つまりデフォルトにリセットしなければならない場合があります。一連の`<stencil_clear>`宣言では、どの値を利用するのか指定します。クリアする指示が含まれていない場合には、ターゲットの画像はレンダリングが始まってでも変更されません。

属性

`<stencil_clear>`要素には、以下の属性があります。

index	xs:nonNegativeInteger	マルチレンダリングターゲットのどれを設定するのか指定します。デフォルト値は 0 です。オプション。

関連要素

`<stencil_clear>`要素は、以下の要素と関連性があります。

親要素	evaluate

子要素	なし
その他	なし

詳細

この要素には、リソースの消去に使われる `xs:byte` 型の値が含まれます。

この要素がパスの中に存在するということは、特定のバックバッファやレンダーターゲットリソースを消去する必要があることをランタイムに知らせる合図になっています。つまり、リソース中の既存の画像データのすべてを、指定された値で書きかえる必要があるということです。この要素は、リソースを何も描かれていない既知の状態にして、以後このリソースを使った操作が予測どおりに行われるようにします。

`index` 属性は、消去したいリソースを特定します。インデックス 0 は、プライマリリソースを表します。プライマリリソースは、バックバッファ、もしくは、適当な `<*_target>` 要素 (`<color_target>`、`<depth_target>`、`<stencil_target>` など) によって提供されるオーバーライドです。

Direct3D[®]9 クラスのプラットフォームでは、MRT を設定するルールがかなり制限されています。たとえば、可能なのは、同じサイズとピクセルフォーマットのカラーバッファ 4 つと、全カラーバッファに対してアクティブなデプスバッファ 1 つとステンシルバッファ 1 つだけです。COLLADA FX の宣言は、この制限を緩めるように設計されており、FX ランタイムは、実際に適用する前に `<pass>` の中に指定されている特定の MRT 宣言が可能であることを検証しなければならず、検証に失敗した場合にはエラーを返す必要があります。

例

```
<stencil_clear index="0">0</stencil_clear>
```

stencil_target

カテゴリ: レンダリング

プロファイル: CG、GLES、GLES2、GLSL

概要

このパスの出力からステンシル情報を受け取る `<image>` を指定します。

コンセプト

複数レンダリングターゲット (MRT) を利用すると、フラグメントシェーダに対して、パスごとに複数の値を出力させたり、標準のデプスユニットやステンシルユニットをリダイレクトして任意のオフスクリーンバッファを読み書きさせることができます。これらの要素は、どの定義済みレンダーターゲットを利用するのかを FX ランタイムに教えます。

属性

`<stencil_target>` 要素には、以下の属性があります。

<code>index</code>	<code>xs:nonNegativeInteger</code>	マルチレンダリングターゲットのいずれか 1 つを指し示します。デフォルト値は 1 です。オプション。
<code>slice</code>	<code>xs:nonNegativeInteger</code>	単一の MIP マップレベル、ユニークな立方体の面、3 次元テクスチャのレイヤなどの、対象とする <code><image></code> の中のサブイメージを指し示します。デフォルト値は 0 です。オプション。
<code>mip</code>	<code>xs:nonNegativeInteger</code>	対象とする MIP レベル。デフォルト値は 0 です。オプション。

face	列挙	対象とする立方体面。有効な値は、POSITIVE_X、NEGATIVE_X、POSITIVE_Y、NEGATIVE_Y、POSITIVE_Z、およびNEGATIVE_Zです。デフォルト値は、POSITIVE_Xです。オプション。
------	----	--

関連要素

<stencil_target>要素は、以下の要素と関連性があります。

親要素	evaluate
子要素	下記サブセクションを参照
その他	なし

子要素

以下の要素のいずれかがちょうど1個存在する必要があります。

名前/例	解説	デフォルト値	出現回数
<param>(reference)	どの画像を使用すべきかを判断するには、samplerパラメータを参照してください。主見出し項目を参照してください。	なし	0または1
<instance_image>	レンダリング可能な画像を直接インスタンス化します。主見出し項目を参照してください。	なし	0または1

詳細

Direct3D® 9クラスのプラットフォームでは、MRTを設定するルールがかなり制限されています。たとえば、可能なのは、同じサイズとピクセルフォーマットのカラーバッファ4つと、全カラーバッファに対してアクティブなデプスバッファ1つとステンシルバッファ1つだけです。COLLADA FXの宣言は、この制限を緩めるように設計されており、FXランタイムは、実際に適用する前に<pass>の中に指定されている特定のMRT宣言が可能であることを検証しなければならず、検証に失敗した場合にはエラーを返す必要があります。

<stencil_target>が指定されていない場合、FXランタイムは、対象とするプラットフォーム用のデフォルトのステンシルバッファセットを利用します。

例

```
<newparam sid="surfaceTex">
  <sampler2D><instance_image url="renderTarget1"/></sampler2D>
</newparam>
<technique>
<pass>
  <evaluate>
    <stencil_target>
      <param ref="surfaceTex"/>
    </stencil_target>
  </evaluate>
</pass>
</technique>
```

technique

(FX)

カテゴリ: 効果

プロファイル: CG、COMMON、GLES、GLES2、GLSL

概要

ある方法を利用して効果をレンダリングするために必要なテクスチャやサンブラ、シェーダ、パラメータ、パスに関する記述を保持します。

<profile_*>以外の要素の中の<technique>については、「<technique>(core)」を参照してください。

コンセプト

テクニックは、効果のレンダリングに必要なすべての必須要素を保持します。それぞれの効果には複数のテクニックを含めることができ、それぞれのテクニックで効果をレンダリングする別々の方法を表します。一般にテクニックが利用される状況としては、以下の3つがあります。

- あるテクニックで高いLODバージョンの効果を記述し、2番目のテクニックで同じ効果の低いLODバージョンを記述するような場合。
- 同じ効果を別々の方法で記述しておき、標準のAPIを利用した未知のデバイスに対して最も効率的なバージョンの効果を見つけるためにFXランタイムの検証ツールを利用する場合。
- 異なるゲームのステート、たとえば、昼間のテクニックと夜間のテクニックなどの効果を記述し、通常のテクニックと「魔法が有効」なテクニックを記述する場合。

属性

<technique>要素には、以下の属性があります。

id	xs:ID	オプション。要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。
sid	sid_type	必須。この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。SIDの詳細は、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。

関連要素

<technique>要素は、以下の要素と関連性があります。

親要素	profile CG , profile Common , profile GLSL , profile GLES , profile GLES2
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素は、プロファイルによって異なります。詳しくは、親要素の主見出し項目を参照してください。以下の一覧は、有効な子要素をまとめたものです。子要素が存在する場合、その並びは以下の順番でなければなりません。ただし、以下は例外です。<blinn>、<constant>、<lambert>、<phong>は例外で、その場所に任意の順序で出現するかできる選択肢です。

名前	profile_CG	profile_COMMON	profile_GLES	profile_GLES2	profile_GLSL	出現回数
<asset>	はい	はい	はい	はい	はい	0 または 1
<annotate>	はい	-	はい	はい	はい	0 以上
<blinn>	-	はい	-	-	-	1
<constant> (FX)	-	はい	-	-	-	
<lambert>	-	はい	-	-	-	
<phong>	-	はい	-	-	-	
<pass>	はい	-	はい	はい	はい	1 以上
<extra>	はい	はい	はい	はい	はい	0 以上

詳細

ツールを利用して効果のテクニックを自動的に生成することができ、テクニックを最善の<asset>として管理する(作成時間、利用可能期間、親子関係、生成に利用するツールを管理する)ことが可能です。

例

```
<effect id="BumpyDragonSkin">
  <profile_GLSL>
    <technique sid="HighLOD">
      ...
    </technique>
    <technique sid="LowLOD">
      ...
    </technique>
  </profile_GLSL>
</effect>
```

technique_hint

カテゴリ: 効果

プロファイル: 外部

概要

効果で利用するテクニックのプラットフォームのヒントを追加します。

コンセプト

シェーダエディタは、効果がインスタンス化された際にデフォルトで利用するテクニックの情報が必要とします。検証の前提として、FX ランタイムがプラットフォームの文字列を認識できた場合には、推奨されているテクニックを利用すべきです。

属性

<technique_hint>要素には、以下の属性があります。

platform	xs:Name	ヒントが対象としているプラットフォームを指定した文字列を定義します。オプション。
ref	xs:NCName	プラットフォームの名前を参照します。必須。

profile	xs:NCName	このヒントがどのAPI プロファイルを対象としているかを指定する文字列。テクニックを含んでいる効果内のプロファイルの名前です。プロファイルは、この属性値を「profile_」に付加することで作成されます。たとえば、profile_CG を選択するには、profile="CG" を指定します。オプション。
---------	-----------	--

関連要素

`<technique_hint>`要素は、以下の要素と関連性があります。

親要素	instance_effect
子要素	なし
その他	なし

詳細

例

```
<technique_hint platform="PS3" ref="HighLOD"/>
<technique_hint platform="OpenGL|ES" ref="twopass"/>
<technique_hint profile="CG" platform="GL" ref="HighLOD"/>
<technique_hint profile="GLES" platform="NOKIA_SW" ref="OneLight"/>
```

texcombiner

カテゴリ: テクスチャリング

プロファイル: GLES

概要

合成モードテクスチャリング用の`<texture_pipeline>`コマンドを定義します。

コンセプト

この要素は、割り当てられているサンプラの合成状態を設定します。また、通常モードと合成モードで `glTexEnv` を利用することにより、マルチテクスチャ操作に変換されるテクスチャコマンドセットを定義します。

割り当てと全体的なコンセプトの詳細に関しては`<texture_pipeline>`の解説を参照してください。

属性

`<texcombiner>`要素には属性はありません。

関連要素

`<texcombiner>`要素は、以下の要素と関連性があります。

親要素	texture_pipeline
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><constant value= ... param= ... ></code> (コンパイナ)	OpenGL ES テクスチャリングユニットに渡すことのできる <code>float4_type</code> 型の静的なパラメータ。この値は、テクスチャリングユニットから最終的な色出力を生成するために、テクスチャからサンプリングされた色と合成されます。合成の式は、 <code><texture_pipeline></code> の設定によって異なります。 この要素にはデータはありません。引数はオプションです。1つだけを使ってください。 value: <code>glTexEnv(TEXTURE_ENV, TEXTURE_ENV_COLOR, value)</code> 用の <code>float4_type</code> 型の値を指定します。 param: <code>glTexEnv</code> 関数用の <code>float4_type</code> 型の値を含むパラメータの SID を指定します。	なし	0 または 1
<code><RGB></code>	テクスチャコンパイナコマンドの RGB コンポーネントを設定します。主見出し項目を参照してください。	なし	0 または 1
<code><alpha></code>	テクスチャコンパイナコマンドのアルファコンポーネントを設定します。主見出し項目を参照してください。	なし	0 または 1

詳細

この要素は、`<texture_pipeline>`の内でも複雑なステージ用のコマンドです。この要素は、`<texenv>`から利用できる操作よりカスタマイズされた操作のために使われます。

とりあえず、`<texenv>`の説明を読んでください。以下の情報は、その基本的な知識を前提にして書かれています。

`<RGB>`および`<alpha>`子要素は、ほとんど同じで、`<alpha>`は、単なる`<RGB>`のサブセットです。

`<texenv>`では、あらゆる状態で使われる演算式を1つだけ指定できますが、`<texcombiner>`では、`<RGB>`チャンネルと`<alpha>`チャンネルに異なる方程式を指定できるので、さらなる柔軟性が加わります。

指定される式は、最高で3つの引数から構成されます。この引数は、 $(Arg0, Arg1, Arg2)$ という系列で指定されます。各チャンネルは、式に対して独自の`<argument>`を指定することができます。各`<argument>`は、ソース、オペランド、およびサンブラを指定します。`<argument>`の `source` 属性は、その式の引数値の取得元を決めます。

- **TEXTURE:** `sampler` 属性で指定されるサンブラから。
- **CONSTANT:** また、`<texcombiner>`スキーマは、各ステージが、オペレータが利用できる固有の`<constant>`を持てるようにします。
- **PRIMARY:** マテリアルから受け取ったフラグメント色。
- **PREVIOUS:** 前のテクスチャパイプラインステージから受け取った色。

`<argument>`の `operand` 属性は、ソースによって選択された値のどの部分を式で利用するか決めます。

- **SRC_COLOR:** ソースの RGB 部分。
- **ONE_MINUS_SRC_COLOR:** `SRC_COLOR` の成分単位の逆 (1 マイナス)。
- **SRC_ALPHA:** ソースのアルファ部分。
- **ONE_MINUS_SRC_ALPHA:** `SRC_ALPHA` の逆 (1 マイナス)。

texcombiners では、以下の演算子式を利用できます。

- **REPLACE**: $Arg0$
- **MODULATE**: $Arg0 + Arg1$
- **ADD**: $Arg0 + Arg1$
- **ADD_SIGNED**: $Arg0 + Arg1 - 0.5$
- **INTERPOLATE**: $Arg0 * Arg2 + Arg1 * (1 - Arg2)$
- **SUBTRACT**: $Arg0 - Arg1$
- **DOT3** (<RGB>のみ):

$$4 \times ((Arg0.r - 0.5) * (Arg1.r - 0.5) + (Arg0.g - 0.5) * (Arg1.g - 0.5) + (Arg0.b - 0.5) * (Arg1.b - 0.5))$$

最後に、各チャンネルはスケールリングすることができます。式の結果のRGB やアルファには、scale 属性が (指定された場合には) 掛け算されて、各チャンネルの最終的な値が計算されます。

そして、RGB およびアルファチャンネルは、4 つの成分を持つ色としてまとめられ、次のステージに PREVIOUS ソースとして渡されます。

これらの列挙型の詳しい情報は、OpenGL および OpenGL ES の仕様を参照してください。

コマンドは、最終的に OpenGL ES ハードウェアのテクスチャユニットに割り当てられます。このコマンドタイプでは、以下のコマンドで、それぞれのテクスチャユニットをテクスチャ合成モードに変更しなければなりません。

```
glTexEnv(TEXTURE_ENV, TEXTURE_ENV_MODE, COMBINE)
```

OpenGL ES ハードウェアのテクスチャユニット代入の詳細に関しては<texture_pipeline>の解説を参照してください。

例

<texture_pipeline>を参照してください。

texenv

カテゴリ: テクスチャリング

プロファイル: GLES

概要

非合成モードテクスチャリング用の<texture_pipeline>コマンドを定義します。

コンセプト

この要素は、割り当てられているサンプラのステートを設定します。

割り当てと全体的なコンセプトの詳細に関しては<texture_pipeline>の解説を参照してください。

属性

<texenv>要素には、以下の属性があります。

operator	列挙	オプション。入力フラグメントに対して実行する操作。有効な値は、以下の通りです。 REPLACE MODULATE DECAL BLEND ADD
----------	----	---

sampler	sidref_type	オプション。サンブラのSIDへの参照。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
---------	-------------	---

関連要素

<texenv>要素は、以下の要素と関連性があります。

親要素	texture_pipeline
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<pre><constant value= ... param= ... ></pre> (コンパイナ)	<p>OpenGL ES サンブラに渡すことのできる <code>float4_type</code> 型の静的、またはパラメータ。この値は、サンブラから最終的な色出力を生成するために、テクスチャからサンプリングされた色と合成されます。合成の式は、テクスチャパイプラインの設定によって異なります。</p> <p>この要素にはデータはありません。引数はオプションです。1つだけを使ってください。</p> <p><code>value</code>: <code>glTexEnv(TEXTURE_ENV, TEXTURE_ENV_COLOR, value)</code>用の静的な <code>float4_type</code> 型の値を指定します。</p> <p><code>param</code>: <code>glTexEnv</code> コマンド用の <code>float4_type</code> 型の値を含むパラメータのSIDを指定します。</p>	なし	0 または 1

詳細

この要素は、<texture_pipeline>の内でも単純なステージ用のコマンドです。この要素は、一般的な演算のうちより設定の少ないものに使われます。

割り当てられたサンブラに対して `glTexEnv(TEXTURE_ENV, TEXTURE_ENV_MODE, operator)` の呼び出しを行うことを想定しています。

この演算に使われる式は、`operator` 属性によって指定されます。以下の式を利用できます。

- **REPLACE**: 出力は、アルファ値にかかわらず、`sampler` 属性で指定されたサンブラによってサンプリングされた値になります。
- **MODULATE**: 出力は、入力値と、`sampler` 属性で指定されたサンブラによってサンプリングされた値との積になります。
- **DECAL**: 出力は、`sampler` 属性で指定されたサンブラからサンプリングされた色と、前のステージもしくはマテリアルの入力との、(アルファによる)ブレンドになります。
- **BLEND**: 出力は、`sampler` 属性で指定されたサンブラからサンプリングされた色と、前のステージもしくはマテリアルの入力との、(各色成分による)ブレンドになります。
- **ADD**: 出力は、先のステージもしくはマテリアルの入力と、`sampler` 属性で指定されたサンブラからサンプリングされた色との、和になります。

これらの列挙型の詳しい情報は、OpenGL および OpenGL ES の仕様を参照してください。

例

[<texture_pipeline>](#)を参照してください。

texture_pipeline

カテゴリ: テクスチャリング

プロファイル: GLES

概要

通常モードと合成モードで `glTexEnv` を利用してマルチテクスチャ操作に変換される、一連のテクスチャコマンドセットを定義します。

コンセプト

この要素には、すべての GLES マルチテクスチャステートを定義する順序付けられたコマンドシーケンスが含まれます。

それぞれのコマンドは、最終的にテクスチャユニットに割り当てられます。

- [<texcombiner>](#) は、合成モードでテクスチャユニットの設定を定義します。
- [<texenv>](#) 要素は、非合成モードでテクスチャユニットの設定を定義します。

それぞれのコマンドは、テクスチャユニットの名前とコマンドの利用特性を基にして、後のバインド処理の際にテクスチャユニットに割り当てられます。

パスは、[<texture_pipeline>](#) 状態を利用してフラグメントシェーダをアクティブにします。

それぞれのコマンドの順番は 1:1 で、ハードウェアのテクスチャユニットが割り当てられます。テクスチャクロスバーがサポートされているかどうかに応じて (GLES 1.1)、個々のコマンドから名前付きのテクスチャユニットオブジェクト ([<sampler2D>](#)) が適切なハードウェアテクスチャユニットに割り当てられます。 GLES 1.0 では、サンプリャは既存のユニットである必要があるので、 `source="sampler"` である 2 つの引数は、同じ [<sampler2D>](#) 要素を参照しないと、有効ではありません (「例」サブセクションの `<RGB>` と `<alpha>` 要素を参照してください)。

属性

[<texture_pipeline>](#) 要素には、以下の属性があります。

<code>sid</code>	<code>sid_type</code>	オプション。この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
------------------	-----------------------	---

関連要素

[<texture_pipeline>](#) 要素は、以下の要素と関連性があります。

親要素	states
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素を任意の順で記述できます。

名前/例	解説	デフォルト値	出現回数
<code><texcombiner></code>	主見出し項目を参照してください。	なし	0 以上
<code><texenv></code>	主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	OpenGL ES 拡張のようなアプリケーション固有の追加情報を含みます。コアの主見出し項目を参照してください。	なし	0 以上

詳細

`<texture_pipeline>`は、マテリアル色、テクスチャ、デスティネーション(バックバッファ)のデータを合成する方法を記述するコマンドシーケンスを作成します。テクスチャパイプラインの各ステージは、単一のコマンドから構成されます。利用できるコマンドには2つの種類があります。

- `<texcombiner>`
- `<texenv>`

パイプラインの各ステージの目的は、既存もしくは新しい入力データを合成して、新しい出力色を生成することです。各ステージから出力された色は、その後、次のステージの入力データとして利用できません。

`<texture_pipeline>`がGL呼び出しに変換される方法を理解するには、`<texenv>`および`<texcombiner>`の主見出し項目をお読みください。

API中のステージコマンドは、通常、`glTexEnv`関数呼び出しに翻訳されます。このようなAPIとCOLLADA関数の主な違いは、`glTexEnv`呼び出しが特定のサンプラと対になっていることです。COLLADAの設計では、演算の設計をより簡単にしたり、インポータやコンディショナーが解決操作にインデックスを割り当てたりできるように、演算をサンプラから解放しています。テクスチャリングパイプラインのステージのインデックスは、`glTexEnv`呼び出しが割り当てられるテクスチャユニットのインデックスになります。OpenGL ES 1.0の場合、unit属性によって参照されるテクスチャは、同じテクスチャユニットに配置される必要があります。OpenGL ES 1.1の場合、テクスチャは好きな場所に配置してクロスパーを活用できますが、ハードウェアによっては、1.0と同じようにテクスチャユニットと対にした方が、パフォーマンスがよくなる場合があります。

注:テクスチャリングクロスパーのサポートにより、一部の`<texture_pipeline>`は、OpenGL ES 1.1では直接解決できても、OpenGL ES 1.0では解決できないことがあります。さらに、ハードウェアのテクスチャ数制限のため、一部のハードウェアでは、テクスチャがたくさんあるパイプラインを解決できないことがあります。

例

```

<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>
<texenv sampler="debug-decal-unit" operator="DECAL"/>
</value>
</texture_pipeline>

```

usertype

カテゴリ: **パラメータ**

プロファイル: **CG、GLES2**

概要

パラメータ用の構造化されたクラスのインスタンスを作成します。

コンセプト

インタフェースオブジェクトは、オブジェクトクラス用の抽象インタフェースを宣言します。インタフェースオブジェクトは、必要な関数シグネチャだけを宣言し、特定のメンバデータのために必要なものは何も宣言しません。

ユーザタイプは、必要となる任意のメンバデータとともにインタフェース中に宣言された各関数の実装を提供するための関数宣言を含む、そういったインタフェースや構造体の具体化されたインスタンスです。

ユーザタイプは、ソースコードまたはインクルードされたシェード中でだけ宣言でき、`<usertype>`宣言は、テクニク用にすべてのソースコードが宣言された後にだけ指定することができます。

属性

`<usertype>`要素には、以下の属性があります。

<code>typename</code>	<code>xs:token</code>	必須。現在のソースコード変換ユニット中にある構造体宣言のための識別子です。
<code>source</code>	<code>xs:NCName</code>	Cg スコープ内では必須です。GLES2 では無効です。コードを参照するか、またはユーザタイプを定義する要素を含みます。

関連要素

`<usertype>`要素は、以下の要素と関連性があります。

親要素	コア要素: <code>newparam</code> , <code>setparam</code> FX 要素: <code>array</code> , <code>bind_uniform</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<code><setparam></code>	一連のこの要素を用いて、名前によってメンバを設定します。 ref 属性は、親 usertype からの相対位置です。コアの主見出し項目を参照してください。	なし	0 以上

詳細

`<usertype>`の要素は、一連の`<setparam>`宣言を使って、各葉ノードに名前でアクセスすることにより、`<newparam>`内で作成時に初期化することができます。

いくつかのユーザタイプにはデータがありません。それらは、インタフェース関数を実装するためののみ使用できます。

例

```
<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>
```

このページは空白です。

9章 B-Rep リファレンス

概要

この章では、COLLADA アニメーションの境界表現である「B-rep」部分を構成する要素を扱います。

要素のカテゴリ分類

この章では、要素をアルファベット順に記載します。関連する要素を見つけやすくするため、以下の表では、要素をカテゴリ別に記載しています。

ジオメトリ

<code>brep</code>	境界表現 (B-rep) の構造を記述します。
-------------------	-------------------------

曲線

<code>circle</code>	3次元空間中の円を記述します。
<code>curve</code>	特定の曲線を記述します。
<code>curves</code>	B-rep 構造で使われるあらゆる曲線が含まれます。
<code>ellipse</code>	3次元空間中の楕円を記述します。
<code>hyperbola</code>	3次元空間中の双曲線を記述します。
<code>line</code>	3次元空間中の直線を記述します。
<code>nurbs</code>	3次元空間中の NURBS 曲線を記述します。
<code>parabola</code>	3次元空間中の放物線を記述します。
<code>surface_curves</code>	B-rep 構造で使われるあらゆるパラメトリック曲線 (pcurve) が含まれます。

位相要素

<code>edges</code>	B-rep 構造の辺を記述します。
<code>faces</code>	B-rep 構造の面を記述します。
<code>pcurves</code>	面のパラメータ空間の中で辺を表す方法を指定します。
<code>shells</code>	B-rep 構造のシェルを記述します。
<code>solids</code>	B-rep 構造のソリッドを記述します。
<code>wires</code>	B-rep 構造のワイヤを記述します。

曲面

<code>cone</code>	円錐面を記述します。
<code>cylinder</code>	無限の円柱面を記述します。
<code>nurbs_surface</code>	3次元空間中の NURBS 曲面を記述します。
<code>plane</code> (フィジックス)	無限の平面を記述します。
<code>sphere</code> (フィジックス)	中央揃えされた球体プリミティブを記述します。
<code>surface</code>	特定の曲面を記述します。
<code>surfaces</code>	B-rep 構造で使われるあらゆる曲面が含まれます。

<code>swept_surface</code>	曲線の押し出しや回転によって曲面を記述します。
<code>torus</code>	3次元空間中のトーラス（円環面）を記述します。

変換

<code>orient</code>	物体座標系の方向を記述します。
<code>origin</code>	物体座標系の原点を記述します。

COLLADA の B-rep について

概要

`<brep>`要素は、`<mesh>`や`<convex_mesh>`や`<spline>`の代わりに`<geometry>`要素の下に配置することができます。

境界表現（B-rep）モデルは、位相要素と幾何要素の2つの部分から構成されます。

位相要素は、対応する制約のないジオメトリを限定するさまざまなエンティティ（頂点、辺など）を指定します。位相要素には、以下のようなものがあります。

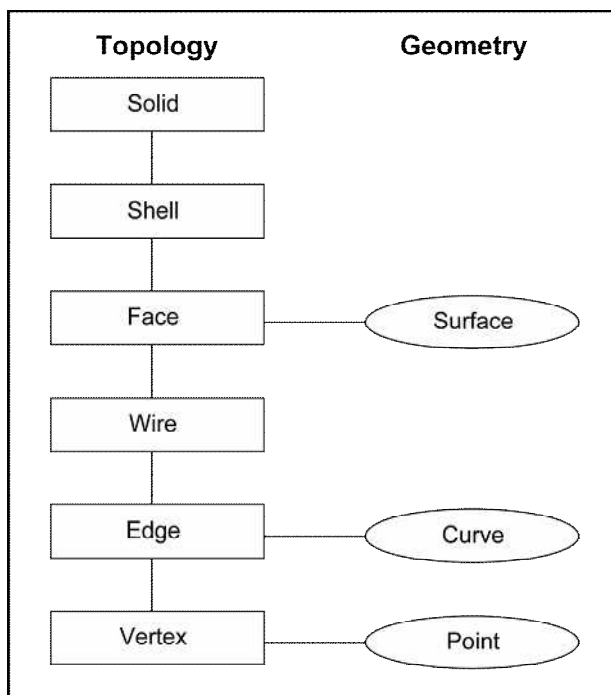
- 頂点: 最も低レベルの位相エンティティ。
- 辺: 2つの頂点によって限定されます。
- ワイヤ: 1つ以上の辺によって限定もしくは構築されます。
- 面: 1つ以上のワイヤに囲まれています。
- シェル: 1つ以上の面によって限定もしくは構築されます。
- ソリッド: 1つ以上のシェルによって限定されます。

幾何学要素には、以下のようなものがあります。

- 点: 3次元空間の点
- 曲線: 3次元空間中の直線、円、NURBS など。
- パラメトリック 曲線: 円のパラメトリック空間中の直線や円など。
- 曲面: 3次元空間中の円柱、球、平面など。

境界表現では、位相要素と幾何要素が組合せられます。

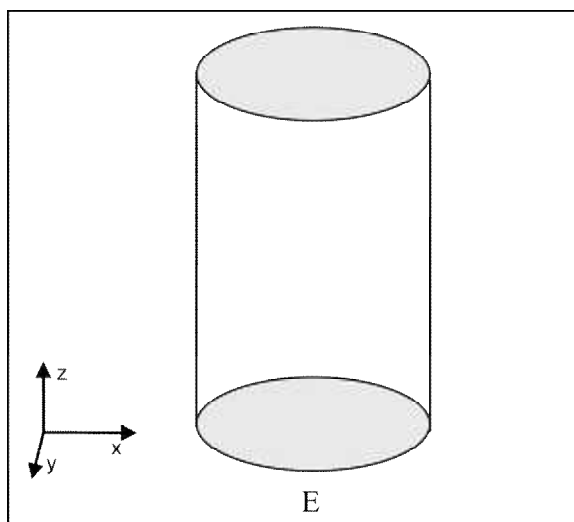
- 頂点は、点によって表現されます。
- 辺は、曲線によって表現されます。
- 面は、曲面によって表現されます。



方向

すべての位相エンティティには、幾何学的な表現や、サブエンティティから構築する際の論理的な順序によって定義される、ベース方向があります。

エンティティ	定義されるベース方向
頂点	頂点には方向はありません。
辺	その頂点の順序。開始頂点と終了頂点と同じ（円など）場合、方向は曲線のベース方向によって定義されます。
ワイヤ	その辺の方向。
面	その曲面の方向。
シェル	その面の方向。



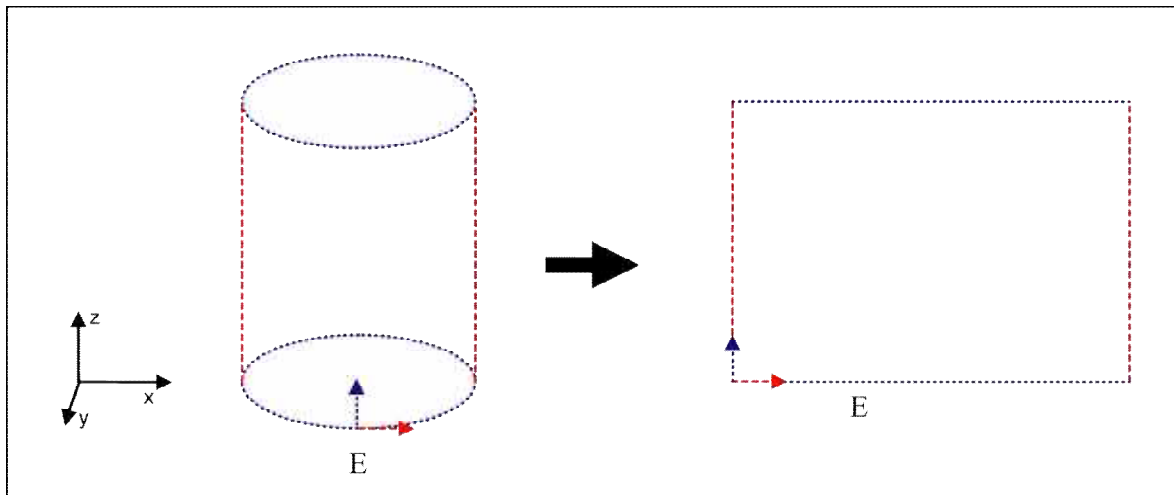
位相エンティティを次のより高いエンティティで使う際には、そのベース方向 (FORWARD) をそのまま使うこともできるし、その反対方向 (REVERSED) を使うこともできます。

パラメトリック曲線

B-rep モデルは、通常、位相エンティティだけでも十分に記述できます。けれども、STEP、IGES、Parasolid など CAD ファイルフォーマットのほとんどでは、辺の幾何学的表現を、その辺に囲まれた曲面のパラメータ空間の中で記述しています。これらの曲線は、曲面の限界を厳密に指定するために必要であり、面をレンダリングする際に役に立ちます。

パラメトリック曲線の例

閉じた円柱には、3次元空間中の円として表現される辺 E があります。この辺は、円柱の面と、底部の平面と、2つの面を限定しています。したがって、この辺の E パラメトリック曲線表現には、平面上での表現と円柱面上での表現の2つがあります。平面のパラメータ空間の中では、この曲線はやはり円です。けれども、円柱のパラメータ空間の中では、この曲線は直線になります。このことは、以下の図のように、円柱に切り込みを入れて、長方形として展開したところを想像すれば、よりわかりやすくなります。

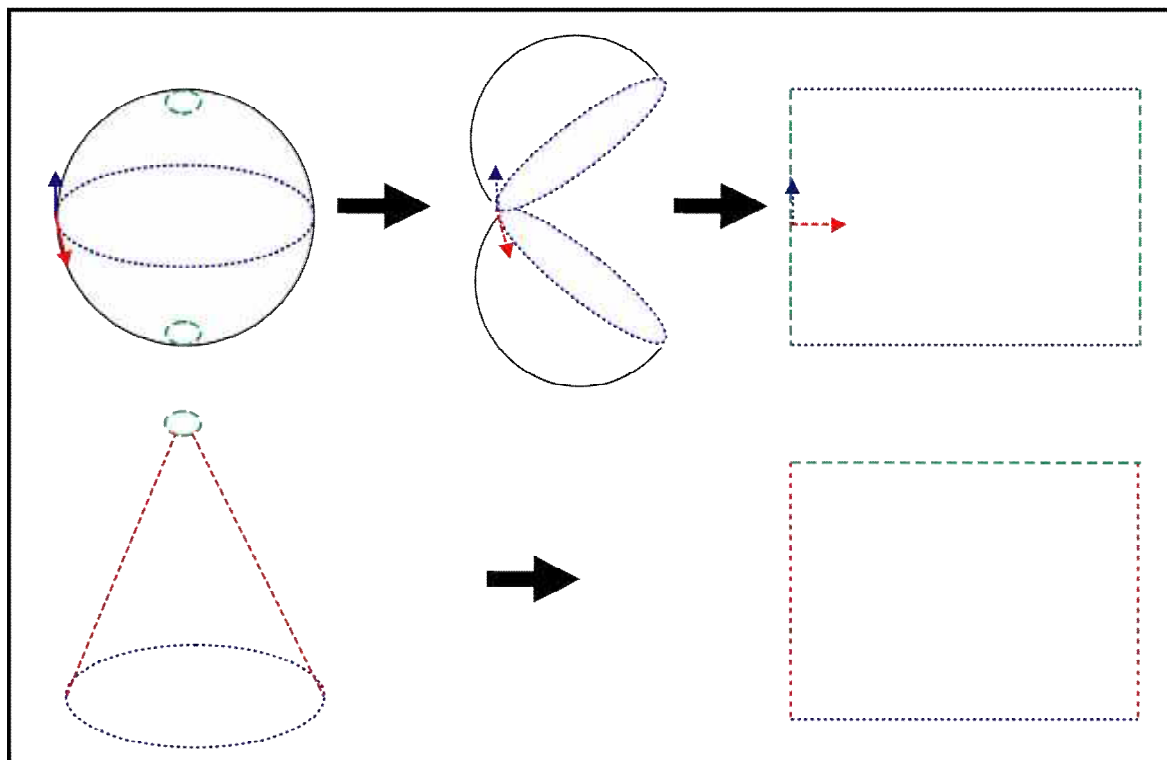


退化した辺

前の例でわかるように、辺は、3次元表現を1つと、2次元表現を複数持つことができます。2次元表現しかない辺は、*退化した辺*と呼ばれます。

退化した辺のあるモデルの例としては、球や円錐があります。円錐の場合は、退化した辺は頂点、球の場合は、極になります。

下の図は、退化した辺の生成過程を説明しようとしたものです。球および円錐のパラメータ空間における緑色の辺は、3次元空間には現れません。この辺は、球や円錐の極であり、3次元空間においては1つの点にすぎません。



ノンマニホールド B-rep

ノンマニホールド B-rep は（立体としては）製造できない B-rep です。しかしながら、B-rep としては依然として有効です。

ほとんどの場合、構築プロセスの中でノンマニホールド B-rep が生じる場合には、次の2つがあります。

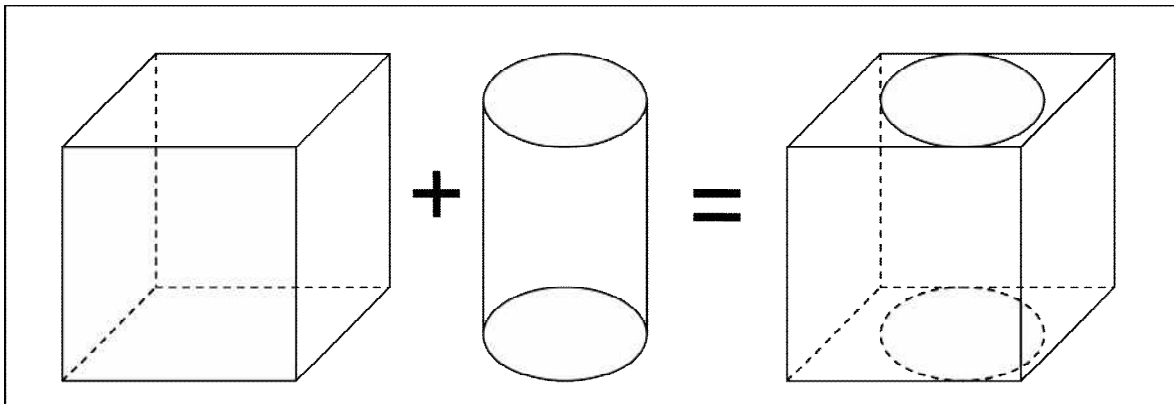
- 構築過程のなかで状況をはっきりさせる。
- プール演算の結果として生じる。

以下はこの2つの例です。

以下の図は、鉄筋コンクリートの柱を表しています。この柱は、鉄とそれを囲むコンクリートの、2種類の素材からできています。見えているのはコンクリートだけなので、この柱は、B-repでは、3次元ブロックとして作成することができません。けれども、ブロック中のどこに鉄があるかを示すために、その場所を長い円柱として表現しています。



下の図は、立方体と円柱の論理和を示しています。これはやはり立方体になります。けれども、このようなブール演算を行った後に、上面や底面の交差部分が残ることがあります。このような交差部分は、立方体用の長方形のワイヤと円柱用の円形のワイヤの2本のワイヤを外周に持つので、ノンマニホールド面として表現されます。



brep

カテゴリ: ジオメトリ

概要

境界表現 (B-rep) の構造を記述します。

コンセプト

B-rep は、単一のソリッドであることも、単一の面や頂点であることもあります。B-rep は、ノンマニホールドであることも、複数のソリッドから構成されていることもあります。`<brep>` 要素には、静的な構造の完全な位相的記述と、対応する頂点の幾何学的な記述が含まれます。

属性

`<brep>` 要素には属性はありません。

関連要素

<brep>要素は、以下の要素と関連性があります。

親要素	<code>geometry</code> (5章 コア要素のリファレンスを参照)
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<curves>	このB-repで使われる曲線のすべてが含まれます。曲線は、B-rep 構造において必須です。この中には、辺の種類を記述する曲線も含まれます。この要素は、<edges>要素が存在する場合には必須です。主見出し項目を参照してください。	なし	0 または 1
<surface_curves>	この B-rep で使われる 2 次元曲線のすべてが含まれます。この中には、面の種類を記述する曲面も含まれます。この要素は、<faces>要素が存在する場合には必須です。主見出し項目を参照してください。	なし	0 または 1
<surfaces>	この B-rep で使われる曲面のすべてが含まれます。主見出し項目を参照してください。	なし	0 または 1
<source>	位相エンティティに必要なデータのすべてが含まれます。幾何学的エンティティに頂点、辺、面を提供します。頂点には、<source>要素が最低でも 1 つ必要です。<edges>が指定された場合、 <code>SSIDREF_array</code> によって<curve>要素の中の曲線にアクセスするために、追加の<source>要素が必要です (コアの主見出し項目を参照)。<faces>が指定された場合、 <code>SIDREF_array</code> によって<surface>要素の中の曲面にアクセスするために、追加の<source>要素が必要です (コアの主見出し項目を参照)。コアの主見出し項目を参照してください。	なし	1 以上
<vertices>	B-repの頂点のすべてを記述します。頂点はあらゆるB-rep構造の基礎となる位相エンティティなので、この要素は必須です。コアの主見出し項目を参照してください。	なし	1
<edges>	B-repの辺のすべてを記述します。主見出し項目を参照してください。	なし	0 または 1
<wires>	B-repのワイヤのすべてを記述します。主見出し項目を参照してください。	なし	0 または 1
<faces>	B-repの面のすべてを記述します。主見出し項目を参照してください。	なし	0 または 1
<pcurves>	B-repのパラメトリック曲線のすべてを記述します。主見出し項目を参照してください。	なし	0 または 1
<shells>	B-repのシェルのすべてを記述します。主見出し項目を参照してください。	なし	0 または 1
<solids>	B-repのソリッドのすべてを記述します。主見出し項目を参照してください。	なし	0 または 1
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<brep>`要素の例です。

```

<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>

```

circle

カテゴリ: 曲線

概要

3次元空間中の円を記述します。

コンセプト

円は、ローカル座標系の原点を中心として定義されます。円は、 (x, y) 平面上に配置されます。

属性

`<circle>`要素には属性はありません。

関連要素

`<circle>`要素は、以下の要素と関連性があります。

親要素	<code>curve</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><radius></code>	円の半径を指定する浮動小数点値が含まれます。この要素には属性はありません。	なし	1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

円は半径によって定義され、あらゆる円錐曲線と同じように、右手座標系を持つ空間に配置されます。この座標系では、

- 原点は、円の中心です
- 原点、 x 方向、 y 方向によって、円のある平面が定義されます。

この座標系は、円のローカル座標系です。この座標系の主方向は、円のある平面の法線となるベクトルです。

この円の軸（主軸）は、その原点や単位ベクトルが、それぞれ、ローカル座標系の原点や主方向になるような軸です。ローカル座標系の主方向は、円に明示的な方向（三角法の意味での定義）を与え、この円に沿ったパラメータの増加方向を決めます。

円は、角度によってパラメータ化されます。

$$P(u) = O + R * \cos(u) * XDir + R * \sin(u) * YDir$$

- P は、パラメータ u の点です。
- O 、 $XDir$ 、 $YDir$ は、それぞれ、ローカル座標系の原点、 x 方向、 y 方向です。
- R は、円の半径です。

したがって、ローカル座標系の x 軸は、円のパラメータの原点を定義します。パラメータは、この x 方向に対する角度になります。円は閉じた周期曲線です。周期は $2 * \pi$ 、パラメータの範囲は $[0, 2 * \pi[$ です。

例

以下は、`<circle>`要素の例です。

```
<curve sid="curve">
  <circle>
    <radius>3</radius>
  </circle>
  <origin>0 0 10</origin>
</curve>
```

cone

カテゴリ: 曲面

概要

円錐面を記述します。

コンセプト

円錐は、以下のように、その頂点における半角によって定義され、座標系および基準半径によって空間中の位置が決めます。

- ローカル座標系の z 軸が、円錐の回転軸です。
- ローカル座標系の原点 $(0,0,0)$ 、 x 軸、 y 軸によって定義される平面が、円錐の基準面です。この基準面と円錐の交線が、基準半径と同じ半径の円です。

属性

<cone>要素には属性はありません。

関連要素

<cone>要素は、以下の要素と関連性があります。

親要素	surfaces/surface
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<radius>	円錐の半径を指定する浮動小数点値が含まれます。この要素には属性はありません。	なし	1
<angle>	円錐面の半角 ($[0, \pi/2]$) を指定する浮動小数点数が含まれます。この要素には属性はありません。	なし	1
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

この座標系は、円錐のローカル座標系です。この座標系には、以下が該当します。

- z 軸のまわりの回転は、 x 軸、 y 軸によって決まる三角法にしたがって、パラメータ u の方向を定義します。
- x 軸は、パラメータ u の原点を与えます。
- z 軸は、円錐のパラメータ v の方向です。
- 原点は、パラメータ v の原点です。

パラメータ u 、 v のパラメータの範囲は、以下の通りです。

- $[0, 2 * \pi]$ 。 u の場合

- $]-\infty, +\infty[$ v の場合

円錐のパラメータ方程式は、次の通りです。

$$P(u, v) = O + (R + v * \tan(\text{Ang})) * (\cos(u) * X\text{Dir} + \sin(u) * Y\text{Dir}) + v * Z\text{Dir}$$

- 0 は、原点です。
- XDir、YDir、ZDir は、 x 方向、 y 方向、 z 方向です。
- Ang は、円錐の頂点の半角です。
- R は、基準半径です。

例

以下は、`<cone>`要素の例です。

```
<cone>
  <radius>1.92574</radius>
  <angle>0.785398</angle>
</cone>
```

curve

カテゴリ: 曲線

概要

特定の曲線を記述します。

コンセプト

この要素は、曲線の属性を定義します。曲面は、`<orient>`と`<origin>`によって、適切な位置に配置することができます。

属性

`<curve>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この文字列は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。

関連要素

`<curve>`要素は、以下の要素と関連性があります。

親要素	<code>curves</code> , <code>surface_curves</code> , <code>swept_surface</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
曲線要素	曲線要素には、以下の要素のいずれかをちょうど1個含む必要があります。 <code><line></code> <code><circle></code> <code><ellipse></code> <code><parabola></code> <code><hyperbola></code> <code><nurbs></code> 主見出し項目を参照してください。	なし	1
<code><orient></code>	物体座標系の方向を記述します。主見出し項目を参照してください。	なし	0 以上
<code><origin></code>	物体座標系の原点を記述します。主見出し項目を参照してください。	0 0 0	0 または 1

例

以下は、`<curve>`要素の例です。

```
<curve sid="curve">
  <line>
    <origin>5 0 0</origin>
    <direction>0 0 1</direction>
  </line>
</curve>
```

curves

カテゴリ: 曲線

概要

B-rep 構造で使われるあらゆる曲線が含まれます。

コンセプト

この要素は、B-rep 構造の辺に使われる全 3 次元曲線のためのコンテナです。

属性

`<curves>`要素には属性はありません。

関連要素

`<curves>`要素は、以下の要素と関連性があります。

親要素	<code>brep</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><curve></code>	単一の曲線を記述します。主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

この要素は、B-rep 構造に必要な曲線のすべてを保持します。この中には、辺の種類を記述する曲線や、押し出し曲面を作成するために必要な曲線も含まれます。

この要素は、位相エンティティ `<edges>` に使われる 3 次元曲線すべてのためのコンテナです。

例

以下は、`<curves>` 要素の例です。

```
<curves>
  <curve sid="curve-1" />
  <curve sid="curve-2" />
</curves>
```

cylinder

(B-Rep)

カテゴリ: 曲面

概要

無限の円柱面を記述します。

注: `<shape>` 中の `<cylinder>` 要素については「`<cylinder>` (フィジックス)」を参照してください。

コンセプト

無限円柱には半径はありますが、長さは無限であると仮定されます。

属性

`<cylinder>` 要素には属性はありません。

関連要素

`<cylinder>` 要素は、以下の要素と関連性があります。

親要素	<code>surface</code> (B-Rep)
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><radius></code>	円柱（楕円形も可能）の半径を表す 2 つの浮動小数点値が含まれます。この要素には属性はありません。		1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<cylinder>`要素の例です。

```
<cylinder>
  <radius>5.0</radius>
</cylinder>
```

edges

カテゴリ: 位相要素

概要

B-rep 構造の辺を記述します。

コンセプト

辺は 2 つの頂点によって限定され、その幾何学的表現は曲線になります。また、曲線のセグメントは、開始、終了パラメータによって限定されます。

属性

`<edges>`要素には、以下の属性があります。

id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。必須。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。
count	xs:unsignedLong	辺の数。必須。

関連要素

`<edges>`要素は、以下の要素と関連性があります。

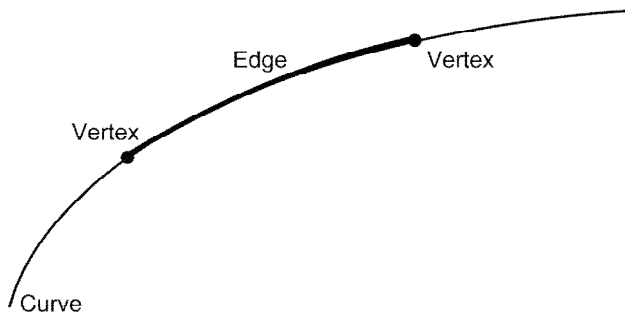
親要素	<code>brep</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<code><input></code>	<p>辺を定義するには、4つの<code><input></code>要素が必要です。</p> <ul style="list-style-type: none"> そのうち1つは、<code>semantic="CURVE"</code>という属性を持ち、辺に対応する幾何学的要素を参照します。 そのうち2つは、<code>semantic="VERTEX"</code>という属性を持ち、各辺を限定する2つの頂点を参照します。 そのうち1つは、<code>semantic="PARAM"</code>という属性を持ち、曲線のパラメータ値(開始および終了パラメータ)を設定します。 <p>コアの主見出し項目を参照してください。</p>	なし	4 以上
<code><p></code>	<p>入力のインデックスを参照します。この要素は、B-repのあらゆる辺の属性を記述します。この要素には属性はありません。</p>	なし	0 または 1
<code><extra></code>	<p>コアの主見出し項目を参照してください。</p>	なし	0 以上

詳細

辺は常に `FORWARD` として宣言されるので、1番目の頂点が開始頂点、2番目の頂点が終了頂点です。



例

以下は、`<edges>`要素の例です。

```

<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>

```

ellipse

カテゴリ: 曲線

概要

3次元空間中の楕円を記述します。

コンセプト

円は長径と短径によって定義され、あらゆる円錐曲線と同じように、右手座標系を持つ空間に配置されます。この座標系では、

- 原点は、楕円の中心です。
- x 軸は、長軸を定義します。
- y 軸は、短軸を定義します。

属性

<ellipse>要素には属性はありません。

関連要素

<ellipse>要素は、以下の要素と関連性があります。

親要素	curve
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<radius>	楕円の半径を指定する2つの浮動小数点数が含まれます。この要素には属性はありません。	なし	1
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

楕円は、ローカル座標系の原点を中心として定義されます。楕円は、 (x, y) 平面上に配置されます。1番目の半径は長径（パラメータ空間の u 方向）、2番目の半径は短径（ v 方向）です。

この座標系の原点、 x 方向、 y 方向は、楕円のある平面を定義します。この座標系は、楕円のローカル座標系です。

この座標系の主方向は、楕円のある平面の法線となるベクトルです。この楕円の軸（主軸）は、その原点や単位ベクトルが、それぞれ、ローカル座標系の原点や主方向になるような軸です。ローカル座標系の主方向は、楕円に明示的な方向（三角法にしたがった定義）を与え、この楕円に沿ったパラメータの増加方向を決めます。楕円は、角度によってパラメータ化されます。

$$P(u) = O + \text{MajorRad} * \cos(u) * XDir + \text{MinorRad} * \sin(u) * YDir$$

- P は、パラメータ u の点です。
- O 、 $XDir$ 、 $YDir$ は、それぞれ、ローカル座標系の原点、 x 方向、 y 方向です。

- MajorRad、MinorRad は、楕円の長径および短径です。

したがって、ローカル座標系の x 軸は、楕円のパラメータの原点を定義します。楕円は、閉じた周期曲線です。周期は $2 * \pi$ 、パラメータの範囲は $[0, 2 * \pi[$ です。

例

以下は、`<ellipse>`要素の例です。

```
<curve sid="curve">
  <ellipse>
    <radius>3 5</radius>
  </ellipse>
</curve>
```

faces

カテゴリ: 位相要素

概要

B-rep 構造の面を記述します。

コンセプト

面は、1つ以上のワイヤによって限定されます。一般に、面は外側のワイヤ1本と、内側のワイヤ0本以上に囲まれています。この場合、その面はマニホールド面です。けれども、COLLADA B-rep は、ノンマニホールドB-repもサポートしているので、面を囲む外側のワイヤは1本以上であることができます。

属性

`<faces>`要素には、以下の属性があります。

id	xs:ID	<code><faces></code> 要素のユニークな識別子を含んだテキスト文字列。この値は、そのインスタンス文書中で一意である必要があります。必須。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。
count	xs:unsignedLong	面の数。必須。

関連要素

`<faces>`要素は、以下の要素と関連性があります。

親要素	<code>brep</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><input></code>	<code><input></code> は、最低でも3つ以上必要です。そのうち1つは、 <code>semantic= "SURFACE"</code> という属性を持ち、面に対応する幾何学的要素を参照します。	なし	3 以上

名前/例	解説	デフォルト値	出現回数
<code><vcount></code>	<p>そのうち1つは、<code>semantic="WIRE"</code>という属性を持ち、各面のワイヤを参照します。</p> <p>そのうち1つは、<code>semantic="ORIENTATION"</code>という属性を持ち、面の中で参照されているワイヤの方向を定義します。</p> <p>さらに、別の<code><input></code>では <code>semantic="MATERIAL"</code>を指定して、特定の面を、B-rep をインスタンス化する際にマテリアルにバインドできるシンボル名とリンクすることができます。</p> <p>コアの主見出し項目を参照してください。</p> <p>各面のワイヤの数を記述している整数のリストが含まれます。この要素には属性はありません。</p>	なし	1
<code><p></code>	<p>入力のインデックスを参照します。この要素は、B-repのあらゆる面の属性を記述します。この要素には属性はありません。</p>	なし	0または1
<code><extra></code>	<p>コアの主見出し項目を参照してください。</p>	なし	0以上

詳細

面は、1つ以上のワイヤに囲まれています。ワイヤは、面に対する方向を必要とします。

面の外側のワイヤが、対応する曲面の法線と同じ方向を向いている場合、その面は可視です。この面は両面から可視です。

外側のワイヤは、曲面を限定します。

内側のワイヤは、囲まれた曲面に穴を開けます（方向は、曲面の法線とは反対です）。

マテリアルの記号名は、トークンソースと`<input semantic="MATERIAL">`を使って各面にバインドされます。メッシュプリミティブは、面の代わりに頂点頻度データをアセンブルします。面は、曲面および1つ以上のワイヤから構成されており、`<faces>`要素はちょうど1つしかないので、面は頂点データをアセンブルしません。

例

以下は、`<faces>`要素の例です。

```
<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>
```

hyperbola

カテゴリ: 曲線

概要

3次元空間中の双曲線を記述します。

コンセプト

双曲線は長径と短径によって定義され、あらゆる円錐曲線と同じように、右手座標系を持つ空間に配置されます。この座標系では、

- 原点は、双曲線の中心です。
- x 軸は、長軸を定義します。
- y 軸は、短軸を定義します。

属性

<hyperbola>要素には属性はありません。

関連要素

<hyperbola>要素は、以下の要素と関連性があります。

親要素	curve
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<radius>	双曲線の半径を指定する2つの浮動小数点数が含まれます。この要素には属性はありません。	なし	1
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

双曲線のある平面は、この座標系の原点、 x 方向、 y 方向によって定義されます。この座標系は、双曲線のローカル座標系です。

例

以下は、<hyperbola>要素の例です。

```
<curve sid="curve">
  <hyperbola>
    <radius>3 5</radius>
  </hyperbola>
</curve>
```

line

カテゴリ: 曲線

概要

3次元空間中の直線を記述します。

コンセプト

直線は、原点と方向を示す単位ベクトルによって、定義され、空間の中に配置されます。

属性

<line>要素には属性はありません。

関連要素

<line>要素は、以下の要素と関連性があります。

親要素	curve
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<origin>	直線の原点を記述する3つの浮動小数点数が含まれます。主見出し項目を参照してください。	なし	1
<direction>	直線の方向を単位ベクトルとして記述する3つの浮動小数点数が含まれます。この要素には属性はありません。	なし	1
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、<line>要素の例です。

```
<curve sid="curve">
  <line>
    <origin>5 0 0</origin>
    <direction>0 0 1</direction>
  </line>
</curve>
```

nurbs

カテゴリ: 曲線

概要

3次元空間中の NURBS 曲線を記述します。

コンセプト

NURBS 曲線は、以下のよって定義されます。

- 次数。
- 周期的か非周期的か。
- 極(制御点とも呼ばれる)のテーブル、および NURBS 曲線が有理曲線の場合には、対応する重み。曲線の極は、曲線の変形に使われる制御点です。曲線が非周期的な場合、最初の極は曲線の開始点、最後の極は曲線の終了点です。最初の極と2番目の極を結ぶセグメントは、曲線の開始点における接線であり、最後の極と最後から2番目の極を結ぶセグメントは、曲線の終了点における接線です。曲線が周期的な場合には、このような幾何学的性質は認められません。重みに幾何学的な意味を与えるのは困難ですが、円や楕円の弧を正確に表現する際に役立ちます。さらに、極の重みがすべて等しい場合、その曲線は多項式曲線になります。つまり、有理曲線ではなくなります。
- ノットとその多重度のテーブル。NURBS の場合、ノットのテーブルは、繰り返しのない実数の増加数列です。

属性

<nurbs>要素には、以下の属性があります。

degree	uint_type	NURBS 曲線の次数を指定します。必須。
closed	xs:boolean	NURBS 曲線が閉じているかどうか指定します。デフォルトは、false です。オプション。

関連要素

<nurbs>要素は、以下の要素と関連性があります。

親要素	curve
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<source>	コアの主見出し項目を参照してください。	なし	1 以上
<control_vertices>	コアの主見出し項目を参照してください。	なし	1
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

次数の値は p と呼ばれます。NURBS セグメントの階数は o で表され、 $p + 1$ として定義されます。

NURBS 曲線は、`<source>/<input>` の以下の semantic 値によって表される、3 つの情報源を使って評価されます。

- POLE: 制御頂点 (極とも呼ばれる)。1 つのセグメントの制御頂点の数は、 n で表します。
- WEIGHT: 極の重み。これは、極ソースの最後の成分で、制御頂点を同次座標で記述するのに使われます。
- KNOT: NURBS のノットベクトル。非減少浮動小数点ベクトルのリスト (1 セグメント当たり 1 ベクトル)。特定のベクトル (k で表される) のサイズは、次の関係によって制約されます。

$$k = (n + p + 1) = (n + o)$$

WEIGHT 入力セマンティックが存在する場合、そのサイズは POLE 入力サイズと一致する必要があります。

POLE ソースは、3 次元の位置なので、`float3_type` 型でも `float4_type` 型でもかまいません。float4 を使った場合、4 番目の成分は制御頂点に適用される重みとして使われるので、WEIGHT ソースは不要になります。WEIGHT と両方を指定した場合、WEIGHT は無視されます。

KNOT ソースの値の数は、次の関係式に従う必要があります。

$$\text{Count}(\text{KNOTS}) = \text{SUM}(n + p + 1)$$

ただし、 n はセグメントの制御頂点の数、 p は次数です。

例

以下は、`<nurbs>` 要素の例です。

```
<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>
  </source>
</nurbs>
```

```

        </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>

```

nurbs_surface

カテゴリ: 曲面

概要

3次元空間中のNURBS曲面を記述します。

コンセプト

NURBS曲面では、パラメータ方向ごとに以下を設定できます。

- 一様か非一様か
- 有理か非有理か
- 周期的か非周期的か。

NURBS曲面は、以下によって定義されます。

- パラメータ u 、 v 各方向の次数
- パラメータ u 、 v 各方向の周期性
- 極（制御点とも呼ばれる）のテーブル、（曲面が有理曲面の場合には、対応する重み）
- ノットのテーブル

属性

`<nurbs_surface>`要素には、以下の属性があります。

<code>degree_u</code>	<code>uint_type</code>	NURBS 曲線の u 方向の次数を指定します。必須。
<code>closed_u</code>	<code>xs:boolean</code>	NURBS 曲線が u 方向に閉じているかどうかを指定します。デフォルトは、 <code>false</code> です。オプション。
<code>degree_v</code>	<code>uint_type</code>	NURBS 曲線の v 方向の次数を指定します。必須。
<code>closed_v</code>	<code>xs:boolean</code>	NURBS 曲線が v 方向に閉じているかどうかを指定します。デフォルトは、 <code>false</code> です。オプション。

関連要素

`<nurbs_surface>`要素は、以下の要素と関連性があります。

親要素	<code>surface</code> (B-Rep)
子要素	下記サブセクションを参照してください。

その他	なし
-----	----

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><source></code>	コアの主見出し項目を参照してください。	なし	1 以上
<code><control_vertices></code>	コアの主見出し項目を参照してください。	なし	1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

次数値は、 p_u および p_v で表されます。NURBS セグメントの階数は、 o_u および o_v で表され、それぞれ、 $p_u + 1$ および $p_v + 1$ によって定義されます。

NURBS 曲線は、`<source>/<input>` の以下の semantic 値によって表される、3 つの情報源を使って評価されます。

- POLE: 制御頂点。1 つのセグメントの制御頂点の数は、 n で表します。ただし、

$$n = n_u * n_v$$
- WEIGHT: 極ソースの最後の成分で、制御頂点を同次座標で記述するのに使われます。
- KNOT: 非減少浮動小数点ベクトルのリスト (1 セグメント当たり 1 ベクトル)。特定のベクトル (k で表される) のサイズは、次の関係によって制約されます。

$$k_u = (n_u + p_u + 1) = (n_u + o_u)$$

$$k_v = (n_v + p_v + 1) = (n_v + o_v)$$

WEIGHT 入力セマンティックが存在する場合、そのサイズは POLE 入力サイズと一致する必要があります。

POLE ソースは、3 次元の位置なので、`float3_type` 型でも `float4_type` 型でもかまいません。`float4_type` を使った場合、4 番目の成分は制御頂点に適用される重みとして使われるので、WEIGHT ソースは不要になります。WEIGHT と両方を指定した場合、WEIGHT は無視されます。

KNOT_U ソースの値の数は、次の関係式に従う必要があります。

$$\text{Count}(\text{KNOT_U}) = \text{SUM}(n_u + p_u + 1)$$

ただし、 n_u は u 方向の曲線の制御頂点の数、 p_u はその次数です。

KNOT_V ソースの値の数は、次の関係式に従う必要があります。

$$\text{Count}(\text{KNOT_V}) = \text{SUM}(n_v + p_v + 1)$$

ただし、 n_v は v 方向の曲線の制御頂点の数、 p_v はその次数です。

例

```
<nurbs_surface degree_u="3" degree_v="3" closed_u="false" closed_v="false">
  <source id="nurbs-lib-knots_u">
    <float_array id="nurbs-lib-knots_u-array" count="9">
      0 0 0 0 0.5 1 1 1 1
    </float_array>
  </source>
  <technique_common>
    <accessor source="..." count="9" stride="1">
      <param name="KNOT" type="float"/>
    </accessor>
  </technique_common>
</nurbs_surface>
```



```

    </technique_common>
</source>

<source id="nurbs-lib-knots_v">
  <float_array id="nurbs-lib-knots_v-array" count="9">
    0 0 0 0 0.5 1 1 1 1
  </float_array>
  <technique_common>
    <accessor source="..." count="9" stride="1">
      <param name="KNOT" type="float"/>
    </accessor>
  </technique_common>
</source>

<source id="nurbs-lib-pos">
  <float_array id="nurbs-lib-pos-array" count="...">
  </float_array>
  <technique_common>
    <accessor source="..." count=".." stride="3">
      <param name="X" type="float"/>
      <param name="Y" type="float"/>
      <param name="Z" type="float"/>
    </accessor>
  </technique_common>
</source>

<source id="nurbs-lib-weights">
  <float_array id="nurbs-lib-weights-array" count="..">
  </float_array>
  <technique_common>
    <accessor source="..." count=".." stride="1">
      <param name="WEIGHT" type="float"/>
    </accessor>
  </technique_common>
</source>

<control_vertices>
  <input semantic="KNOT_U" source="nurbs-lib-knots_u"/>
  <input semantic="KNOT_V" source="nurbs-lib-knots_v"/>
  <input semantic="POLE" source="nurbs-lib-pos"/>
  <input semantic="WEIGHT" source="nurbs-lib-weights"/>
</control_vertices>
</nurbs_surface>

```

orient

カテゴリ:変換

概要

物体座標系の方向を記述します。

コンセプト

方向は、任意の軸と角度によって指定されます。

属性

<orient>要素には属性はありません。

関連要素

<orient>要素は、以下の要素と関連性があります。

親要素	curve , surface (B-Rep)
子要素	なし
その他	なし

詳細

4つの浮動小数点数が含まれます。これらの値は、回転軸を指定する列ベクトル [X, Y, Z]と、度数で表された角度を表します。

<orient>要素は、<rotate>要素と似ていますが、<rotate>を使った要素 (<node>など) は、要素を操作することによってアニメーションを行うことができるのに対し、B-rep ではアニメーションはできません。

例

以下は、<orient>要素の例です。

```
<curve sid="curve">
  <line/>
  <orient>0 0 1 45</orient>
</curve>
```

origin

カテゴリ:変換

概要

物体座標系の原点を記述します。

コンセプト

原点は、3次元空間中の位置によって指定されます。

属性

`<origin>`要素には属性はありません。

関連要素

`<origin>`要素は、以下の要素と関連性があります。

親要素	<code>curve</code> 、 <code>line</code> 、 <code>surface</code> (B-rep)、 <code>swept_surface</code>
子要素	なし
その他	なし

詳細

3つの浮動小数点数が含まれます。これらの値は、列ベクトル[X, Y, Z]として扱われ、作業している系における物体座標系の原点を指定します。(0, 0, 0)という値は、物体座標系の原点が、作業座標系(ワールド座標系など)の原点であることを意味します。

例

以下は、`<origin>`要素の例です。

```
<curve sid="curve">
  <line/>
  <origin>10 0 0</origin>
</curve>
```

parabola

カテゴリ: 曲線

概要

3次元空間中の放物線を記述します。

コンセプト

放物線は、焦点距離、つまり、焦点と頂点の間の距離によって定義され、以下のような座標系にしたがって空間に配置されます。

- 原点は、放物線の頂点です。
- x 軸は、対称軸を定義します。放物線は、この軸の正方向に存在します。
- 原点、 x 方向、 y 方向によって、放物線のある平面が定義されます。

属性

`<parabola>`要素には属性はありません。

関連要素

`<parabola>`要素は、以下の要素と関連性があります。

親要素	<code>curve</code>
子要素	下記サブセクションを参照してください。

その他	なし
-----	----

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><focal></code>	放物線の焦点距離を記述する浮動小数点数。この要素には属性はありません。	なし	1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

この座標系は、放物線のローカル座標系です。

例

以下は、`<parabola>`要素の例です。

```
<curve sid="curve">
  <parabola>
    <focal>3.6</focal>
  </parabola>
</curve>
```

pcurves

カテゴリ: 位相要素

概要

面のパラメータ空間の中で辺を表す方法を指定します。

コンセプト

各辺は、その辺が限定する面ごとに、1つの pcurve を持ちます。

`<pcurves>`要素は、辺が存在する面の曲面のパラメータ空間の中で、その辺がどのように表現されるかを指定します。

属性

`<pcurves>`要素には、以下の属性があります。

id	xs:ID	<code><pcurves></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。必須。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。
count	xs:unsignedLong	pcurves の数。必須。

関連要素

`<pcurves>`要素は、以下の要素と関連性があります。

親要素	<code>brep</code>
-----	-------------------

子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<code><input></code>	<p><code><input></code>は、最低でも3つ以上必要です。</p> <p>最初の2つは、<code>semantic="EDGE"</code> および <code>semantic="FACE"</code> という属性で、辺と面の間の結びつきを指定します。</p> <p>3番目は <code>semantic="CURVE2D"</code> という属性で、<code>pcurve</code> の基準を指定します。</p> <p>コアの主見出し項目を参照してください。</p>	なし	3 以上
<code><vcount></code>	各辺と面との接続に使われる <code>pcurves</code> の数を記述する整数のリストが含まれます。この要素には属性はありません。	なし	1
<code><p></code>	入力のインデックスを参照します。この要素には属性はありません。	なし	0 または 1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

辺は、通常、その辺が限定する面ごとに、`pcurve` をちょうど1つずつ持ちます。ただし、円柱や円錐のような閉曲面の場合、1つの辺が2回使われます。例については、「B-repの完全な例」のセクションを参照してください。

例

以下は、`<pcurves>`要素の例です。

```
<pcurves id="pcurves" count="10">
  <input semantic="EDGE" source="#edges" offset="0"/>
  <input semantic="FACE" source="#faces" offset="1"/>
  <input semantic="CURVE2D" offset="2" source="#geom-curves2d"/>
  <vcount>2 1 1 1 1 1 1 1 2 </vcount>
  <p>0 0 0 0 0 2 1 0 1</p>
</pcurves>
```

shells

カテゴリ: 位相要素

概要

B-rep 構造のシェルを記述します。

コンセプト

シェルは、複数の面の和集合です。閉じたシェルは、ソリッドを限定することができます。

属性

<shells>要素には、以下の属性があります。

id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。必須。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。
count	xs:unsignedLong	シェルの数。必須。

関連要素

<shells>要素は、以下の要素と関連性があります。

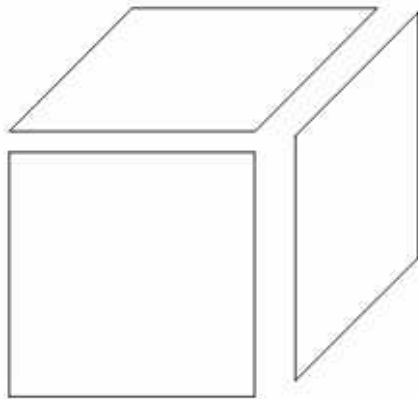
親要素	brep
子要素	下記サブセクションを参照してください。
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<input>	<input>は、最低でも2つ以上必要です。 そのうち1つは、semantic="FACE"という属性を持ち、各面のシェルを参照するます。 そのうち1つは、semantic="ORIENTATION"という属性を持ち、シェルの中で参照されている面の方向を定義します。 コアの主見出し項目を参照してください。	N./A	2以上
<vcount>	各シェルの面の数を記述している整数のリストが含まれます。この要素には属性はありません。	なし	1
<p>	入力のインデックスを参照します。この要素には属性はありません。	なし	0または1
<extra>	コアの主見出し項目を参照してください。	なし	0以上

詳細

面の方向は、閉じたシェルの面の外側を定義します。面が forward に宣言された場合、曲面の法線が面の外側を定義します。それ以外の場合には、法線は内側を定義します。



例

以下は、`<shells>`要素の例です。

```
<shells id="shells" count="1">
  <input semantic="FACE" source="#faces" offset="0"/>
  <input semantic="ORIENTATION" offset="1" source="#orientations"/>
  <vcount>4</vcount>
  <p>0 1 1 0 2 1 3 0</p>
</shells>
```

solids

カテゴリ: **位相要素**

概要

B-rep 構造のソリッドを記述します。

コンセプト

ソリッドは、1つ以上のシェルによって限定されます。一般に、ソリッドは外側のシェル1個と、内側のシェル0個以上に囲まれています。この場合、そのソリッドはマニホールドソリッドです。けれども、COLLADA B-rep は、ノンマニホールド B-rep もサポートしているので、ソリッドを囲む外側のシェルは1個以上であることができます。

属性

`<solids>`要素には、以下の属性があります。

属性名	型	説明
id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。必須。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。
count	xs:unsignedLong	ソリッドの数。必須。

関連要素

`<solids>`要素は、以下の要素と関連性があります。

親要素	brep
子要素	下記サブセクションを参照してください。
その他	なし

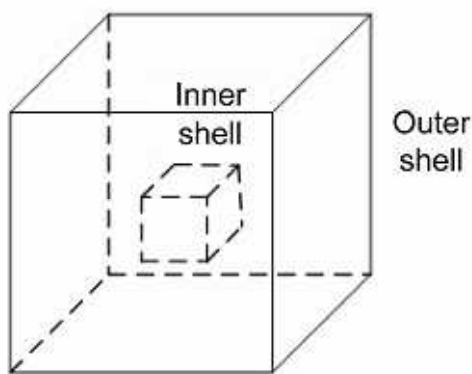
子要素

名前/例	解説	デフォルト値	出現回数
<code><input></code>	<p><code><input></code>は、最低でも2つ以上必要です。</p> <p>そのうち1つは、<code>semantic="SHELL"</code>という属性を持ち、各ソリッドのシェルを参照します。</p> <p>そのうち1つは、<code>semantic="ORIENTATION"</code>という属性を持ち、ソリッドの中で参照されているシェルの方向を定義します。</p> <p>コアの主見出し項目を参照してください。</p>		2 以上

名前/例	解説	デフォルト値	出現回数
<code><vcount></code>	各ソリッドのシェルの数を記述している整数のリストが含まれます。この要素には属性はありません。		1
<code><p></code>	入力のインデックスを参照します。この要素には属性はありません。		0 または 1
<code><extra></code>	コアの主見出し項目を参照してください。		0 以上

詳細

以下の図では、ソリッドは2つのシェルによって限定されています。このソリッドは、立方体の空洞を持つ立方体を記述しています。比喩的に言えば、物質が存在するのは、外側のシェルの内側でかつ内側のシェルの外側の部分だけです。



閉じたシェルが forward として宣言された場合、物質はシェルの内側にあります。それ以外の場合には、物質は外側にあります。

例

以下は、`<solids>`要素の例です。

```
<solids count="1" id="solids">
  <input semantic="SHELL" offset="0" source="#shells"/>
  <input semantic="ORIENTATION" offset="1" source="#orientations"/>
  <vcount>1 </vcount>
  <p>0 1</p>
</solids>
```

surface

カテゴリ: 曲面

概要

特定の曲面を記述します。

コンセプト

曲面の属性を定義します。曲面は、`<origin>`と`<orient>`によって、適切な位置に配置することができます。

B-rep では、無限の曲面や曲線を使ってジオメトリを記述します (NURBS および NURBS 曲面を除く)。限定は、位相要素によって行われます。

属性

<surface>要素には、以下の属性があります。

sid	sid_type	<solids> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。

関連要素

<surface>要素は、以下の要素と関連性があります。

親要素	surfaces
子要素	下記サブセクションを参照してください。
その他	なし

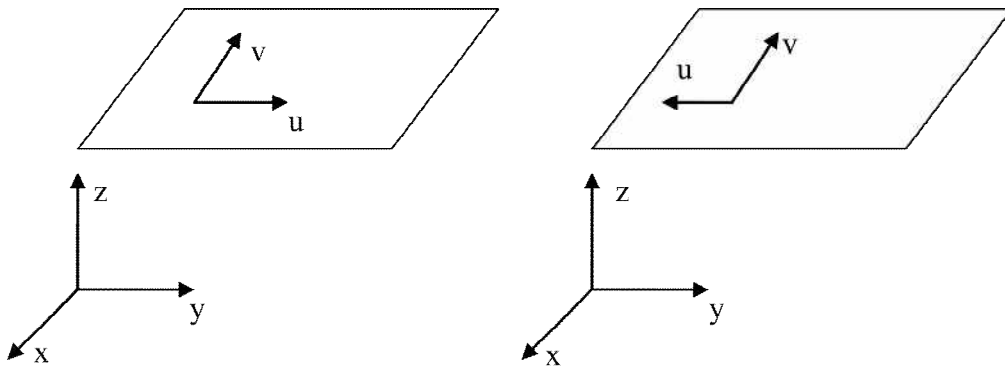
子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
曲面要素	曲面要素には、以下の要素のいずれかをちょうど1個含む必要があります。 <cone>: 円錐を記述します。主見出し項目を参照してください。 <plane>: 平面を記述します。フィジックスの主見出し項目を参照してください。 <cylinder>: 円筒面を記述します。主見出し項目を参照してください。 <nurbs_surface>: NURBS 曲面を記述します。主見出し項目を参照してください。 <sphere>: 球を記述します。フィジックスの主見出し項目を参照してください。 <torus>: トーラス (円環面) を記述します。主見出し項目を参照してください。 <swept_surface>: 曲線を押し出ししたり回転させたりすることによって作成される曲面を記述します。主見出し項目を参照してください。	なし	1
<orient>	物体座標系の方向を記述します。主見出し項目を参照してください。	なし	0 以上
<origin>	物体座標系の原点を記述します。主見出し項目を参照してください。	なし	0 または 1

詳細

pcurves の位置を決めるためには、曲面の座標系が必要です。平面の方程式では、座標系はなくなっています。したがって、平面は常に決まった状態で指定されます。位置決めは、<origin>と<orient>によって行われます。



この例では、どちらの平面も方程式は同じですが、ローカル座標系が違います。

```
<plane>
  <equation>0 0 1 -10</equation>
</plane>
```

この曖昧さを避けるために、1番目の平面は、以下のように指定します。

```
<plane>
  <equation>0 0 1 0</equation>
</plane>
<orient>0 0 1 90</orient>
<origin>0 0 10</origin>
```

そして、2番目の平面は、

```
<plane>
  <equation>0 0 1 0</equation>
</plane>
<orient>0 0 1 90</orient>
<orient>1 0 0 180</orient>
<origin>0 0 10</origin>
```

したがって、ベース平面は、常に、原点が $(0,0,0)$ 、 u 方向が $(1,0,0)$ の平面です。

例

以下は、`<surface>`要素の例です。

```
<surface sid="surface">
  <plane/>
</surface>
```

surfaces

カテゴリ: 曲面

概要

B-rep 構造で使われるあらゆる曲面が含まれます。

コンセプト

この要素は、B-rep 構造の面に使われる全曲面のためのコンテナです。

属性

`<surfaces>` 要素に属性はありません。

関連要素

`<surfaces>`要素は、以下の要素と関連性があります。

親要素	<code>brep</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><surface></code>	単一の曲面を記述します。主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<surfaces>`要素の例です。

```
<surfaces>
  <surface sid="surface-1"/>
  <surface sid="surface-2"/>
</surfaces>
```

surface_curves

カテゴリ: 曲線

概要

B-rep 構造で使われるあらゆるパラメトリック曲線 (pcurve) が含まれます。

コンセプト

Pcurves は、その曲線が存在する曲面のパラメータ空間の曲線です。これらの曲線は、通常、B-rep モデル自体を記述するためには必要ありませんが、B-rep ファイルフォーマットのほとんどは計算が不正確になるのを回避するためにこれを利用します。実際には、pcurve は、それをバインドするワイヤの一部である面の曲面の (u, v) 空間における辺の曲線表現です。曲線の z 座標がすべて 0 なのはそのためです。

属性

`<surface_curves>`要素には属性はありません。

関連要素

`<surface_curves>`要素は、以下の要素と関連性があります。

親要素	<code>brep</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><curve></code>	単一の2次元曲線を記述します。主見出し項目を参照してください。		1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

曲線の方向は、その曲線が存在する曲面の方向に基づいています。参照された曲面は、`<pcurves>`要素によって、`pcurve`に関連付けられます。

例

以下は、`<surface_curves>`要素の例です。

```
<surface_curves>
  <curve sid="curve-1"/>
  <curve sid="curve-2"/>
</surface_curves>
```

swept_surface

カテゴリ: 曲面

概要

曲線の押し出しや回転によって曲面を記述します。

コンセプト

曲線を別の曲線にそって掃引することによって構築される曲面に、共通する挙動を記述します。具体的には、2種類の掃引表面がサポートされます。

- 回転面 (回転された曲面)
- 線形押し出し面 (押し出された曲面)

属性

`<swept_surface>`要素には属性はありません。

関連要素

`<swept_surface>`要素は、以下の要素と関連性があります。

親要素	<code>surface</code> (B-Rep)
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。`<direction>`、もしくは、`<origin>`と`<axis>`の両方の、どちらかを指定する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><curve></code>	ベースカーブを記述します。主見出し項目を参照してください。	なし	1
<code><direction></code>	押し出しの方向を記述する3つの浮動小数点数が含まれます。 <code><origin></code> と <code><axis></code> が指定された場合には、無効です。この要素には属性はありません。	なし	1
<code><origin></code>	回転軸の原点を指定する3つの浮動小数点数が含まれます。これを指定した場合には、 <code><axis></code> も指定する必要があります。主見出し項目を参照してください。	なし	1
<code><axis></code>	回転軸の方向を指定する3つの浮動小数点数が含まれます。これを指定した場合には、 <code><origin></code> も指定する必要があります。この要素には属性はありません。	なし	1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

掃引表面は、曲線の押し出しもしくは回転によって記述されます。

例

以下は、`<swept_surface>`要素の例です。

```
<swept_surface>
  <curve/>
  <direction>1.0 0 0</direction>
</swept_surface>

<swept_surface>
  <curve/>
  <origin>0 0 0</origin>
  <axis>0 0 1</axis>
</swept_surface>
```

torus

カテゴリ: 曲面

概要

3次元空間中のトーラス（円環面）を記述します。

コンセプト

トーラスは、長径と短径によって定義され、以下の座標系にしたがって空間に配置されます。

- 原点は、トーラスの中心です。
- この曲面は、ローカルな z 軸のまわりに円を回転させることによって取得されます。この円の半径は短径と同じで、原点、 x 軸、 z 軸によって定義される平面上に存在します。この円は、 x 軸の正方向に、原点から長径と同じ距離だけ離れた位置に中心を置いています。この円は、トーラスの基準円です。

属性

`<torus>`要素には属性はありません。

関連要素

`<torus>`要素は、以下の要素と関連性があります。

親要素	<code>surface</code> (B-Rep)
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><radius></code>	トーラスの半径を指定する2つの浮動小数点数が含まれます。1番目の値は長径、2番目の値は短径です。この要素には属性はありません。	なし	1
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0以上

例

以下は、`<torus>`要素の例です。

```
<torus>
  <radius>10.0 6.0</radius>
</torus>
```

wires

カテゴリ: 位相要素

概要

B-rep 構造のワイヤを記述します。

コンセプト

ワイヤは、1つ以上の辺の組合せです。閉じたワイヤは、面を限定することができます。

属性

<wires>要素には、以下の属性があります。

id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。必須。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。
count	xs:unsignedLong	ワイヤの数。必須。

関連要素

<wires>要素は、以下の要素と関連性があります。

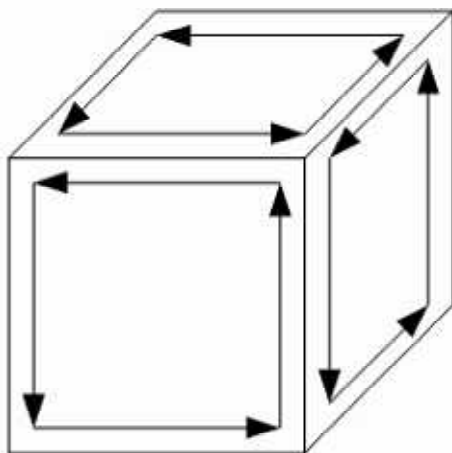
親要素	brep
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<input>	<input>は、最低でも2つ以上必要です。 そのうち1つは、semantic="EDGE"という属性を持ち、各ワイヤの辺を参照します。 そのうち1つは、semantic="ORIENTATION"という属性を持ち、ワイヤの中で参照されている辺の方向を定義します。 コアの主見出し項目を参照してください。	なし	2 以上
<vcount>	各ワイヤの辺の数を記述している整数のリストが含まれます。この要素には属性はありません。	なし	1
<p>	入力のインデックスを参照します。この要素は、全ワイヤの属性を記述します。この要素には属性はありません。	なし	0 または 1
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

詳細



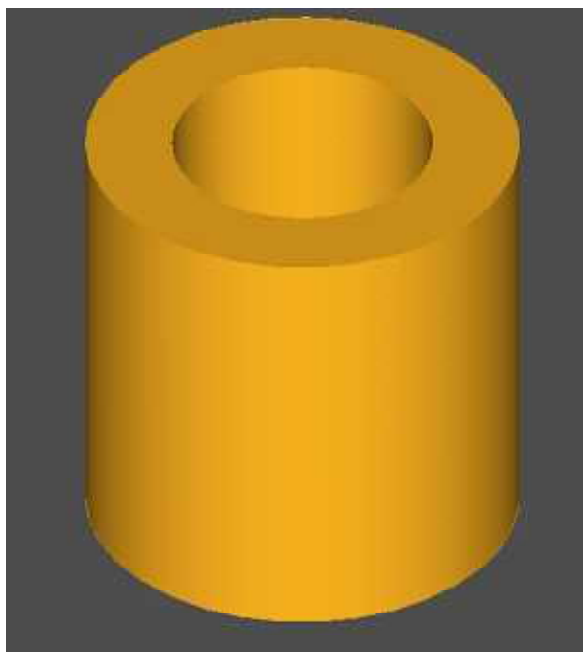
例

以下は、`<wires>`要素の例です。

```
<wires id="wires" count="6">  
  <input semantic="EDGE" source="#edges" offset="0"/>  
  <input semantic="ORIENTATION" source="#orientations" offset="1"/>  
  <vcount>4 1 1 1 1 4</vcount>  
  <p>0 1 1 0 0 0 2 1 3 1 2 0 4 0 1 1 5 0 3 0 5 1 4 1</p>  
</wires>
```

B-rep の完全な例

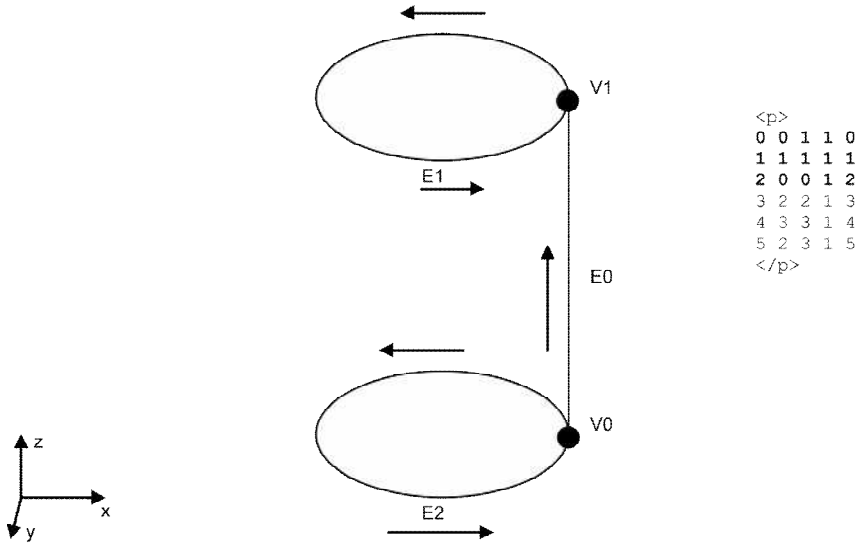
このセクションの例は、下のオブジェクトを指定する方法を示しています。



コードの説明

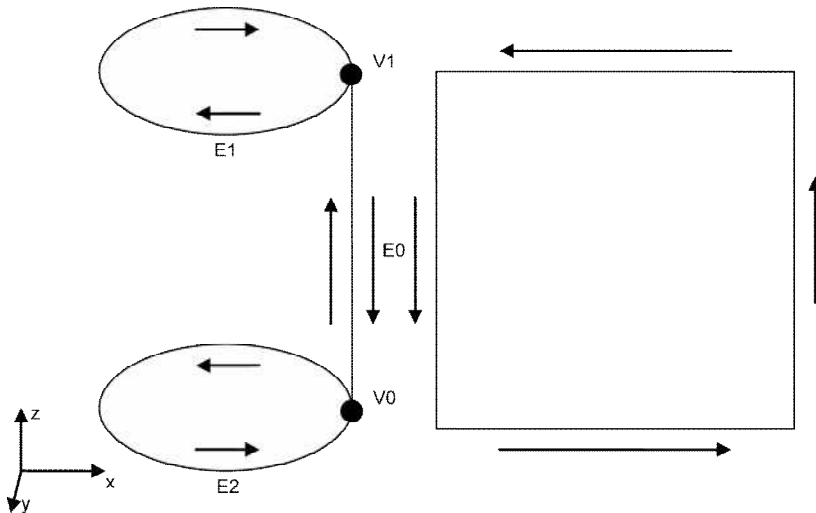
このセクションでは、サンプルコードの内側と外側のワイヤについて説明します。

下図は、1番目のワイヤの最初の3辺の定義を示しています。



```
<p>
0 0 1 1 0
1 1 1 1 1
2 0 0 1 2
3 2 2 1 3
4 3 3 1 4
5 2 3 1 5
</p>
```

下図は、1番目のワイヤの辺の方向を示しています。



```
<vcount>
4 1 1 1 1 4
</vcount>
<p>
0 1 1 0 0 2 1
3 0
2 1
4 0
1 1
5 1 4 0 5 0 3 1
</p>
```

上記のコードでは、

- E0 は、最初 FORWARD に定義されます (V0 から V1)
- E1 は、REVERSED に定義されます (時計回り)
- E0 は、今度は REVERSED に定義されます (V1 から V0)
- E2 は、FORWARD に定義されます

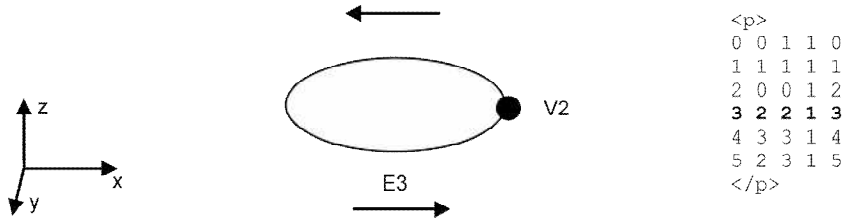
何が起きているかをよりよく理解するために、円柱を E0 に沿って切って、曲面を展開してみましょう。

- ワイヤは、FORWARD に定義されます

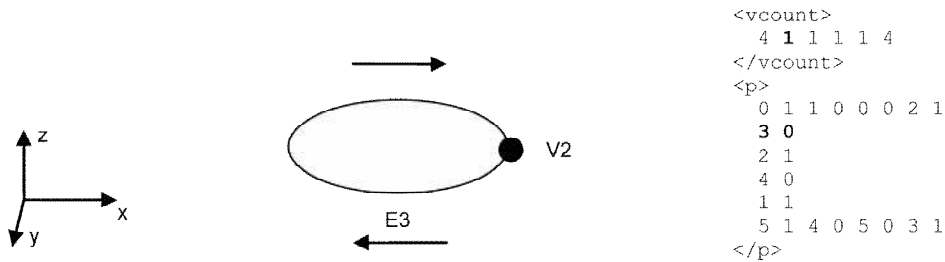
ワイヤの方向は、曲面（外側の円柱）の法線と同じ向きで、外側を向いています。

- このワイヤは、外側のワイヤです。

下図は、3番目の辺の定義を示しています。これは2番目のワイヤになります。



下図は、2番目のワイヤの辺の方向を示しています。



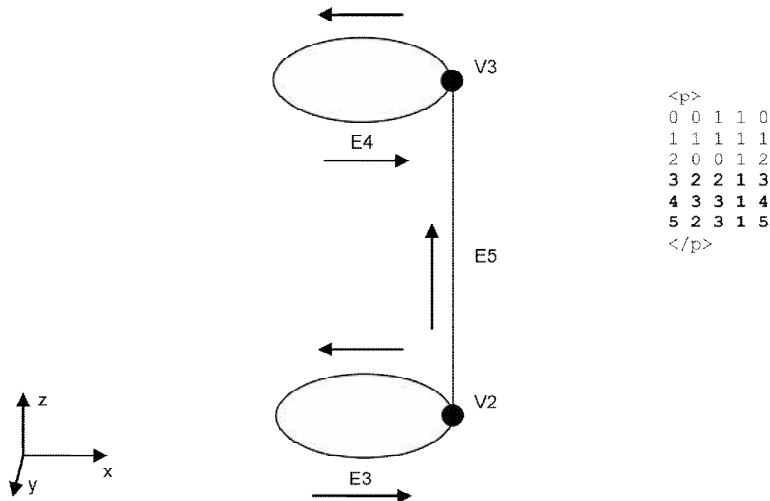
- E3 は、REVERSED に定義されます（時計回り）

このワイヤは FORWARD に定義されるので、ワイヤの方向は曲面（平面）の法線と反対方向になります。

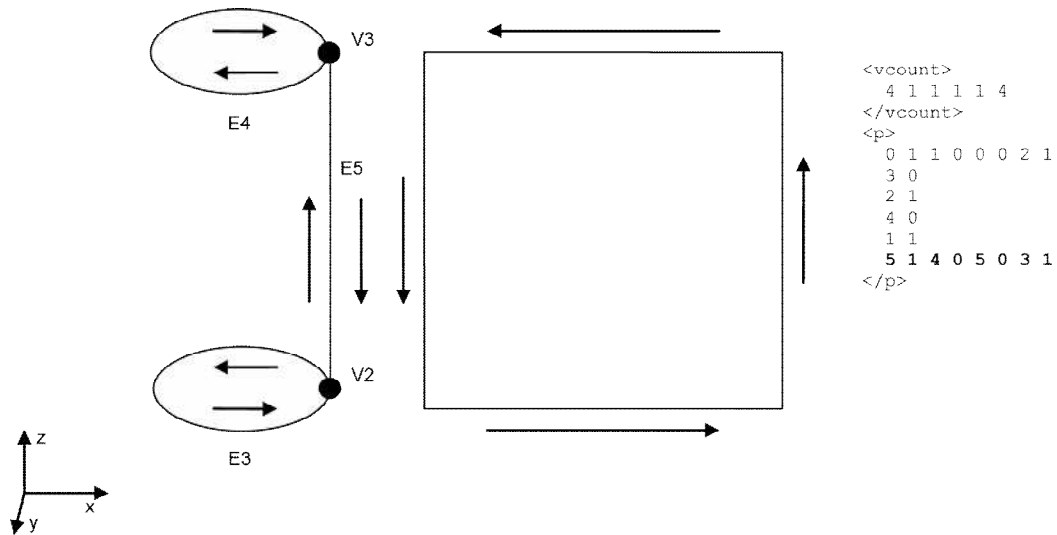
- これは、内側のワイヤです。

3番目、4番目、5番目のワイヤは、2番目のワイヤと同様です。3番目ワイヤは外側のワイヤ、4番目は内側のワイヤ、5番目は外側のワイヤです。

下図は、6番目のワイヤ中の3辺の定義を示しています。



下図は、6番目のワイヤの辺の方向を示しています。



上記のコードでは、

- E5 は、最初 FORWARD に定義されます (V0 から V1)
- E4 は、REVERSED に定義されます (時計回り)
- E5 は、今度は REVERSED に定義されます (V1 から V0)
- E3 は、FORWARD に定義されます

ワイヤは FORWARD に定義されるので、ワイヤの方向は、曲面 (内側の円柱) の法線と同じ向きで、外側を向いています。

- このワイヤは、外側のワイヤです。

サンプルコード

```
<?xml version="1.0" encoding="UTF-8"?>
<COLLADA xmlns="http://www.collada.org/2008/03/COLLADASchema" version="1.5.0">
  <asset>
    <created>2007-09-12T12:00:00</created>
    <modified>2007-09-12T12:00:00</modified>
  </asset>
  <library_geometries id="ring.BREP.lib">
    <geometry id="ring.BREP.lib.geo">
      <brep>
        <!-- 全曲線を定義 -->
      </brep>
      <curves>
        <curve sid="curve-1">
          <line>
            <origin>5 -1.22461e-015 0</origin>
            <direction>0 0 1</direction>
          </line>
        </curve>
        <curve sid="curve-2">
          <circle>
            <radius>5</radius>
          </circle>
          <origin>0 0 10</origin>
        </curve>
      </curves>
    </geometry>
  </library_geometries>
</COLLADA>
```

```

        <curve sid="curve-3">
          <circle>
            <radius>5</radius>
          </circle>
        </curve>
        <curve sid="curve-4">
          <circle>
            <radius>3</radius>
          </circle>
        </curve>
        <curve sid="curve-5">
          <circle>
            <radius>3</radius>
          </circle>
          <origin>0 0 10</origin>
        </curve>
        <curve sid="curve-6">
          <line>
            <origin>3 -7.34764e-016 0</origin>
            <direction>0 0 1</direction>
          </line>
        </curve>
      </curves>
      <!-- 曲面上の全曲線を定義 -->
      <surface_curves>
        <curve sid="curve2d-1">
          <line>
            <origin>6.28319 0 0</origin>
            <direction>0 1 0</direction>
          </line>
        </curve>
        <curve sid="curve2d-2">
          <line>
            <origin>0 10 0</origin>
            <direction>1 0 0</direction>
          </line>
        </curve>
        <curve sid="curve2d-3">
          <line>
            <origin>4.13891e-013 0 0</origin>
            <direction>0 1 0</direction>
          </line>
        </curve>
        <curve sid="curve2d-4">
          <line>
            <origin>0 0 0</origin>
            <direction>1 0 0</direction>
          </line>
        </curve>
        <curve sid="curve2d-5">
          <circle>
            <radius>3</radius>
          </circle>
        </curve>
        <curve sid="curve2d-6">
          <circle>
            <radius>5</radius>
          </circle>
        </curve>
        <curve sid="curve2d-7">

```

```

    <circle>
      <radius>3</radius>
    </circle>
  </curve>
    <curve sid="curve2d-8">
      <circle>
        <radius>5</radius>
      </circle>
    </curve>
    <curve sid="curve2d-9">
      <line>
        <origin>6.28319 0 0</origin>
        <direction>0 1 0</direction>
      </line>
    </curve>
    <curve sid="curve2d-10">
      <line>
        <origin>0 10 0</origin>
        <direction>1 0 0</direction>
      </line>
    </curve>
    <curve sid="curve2d-11">
      <line>
        <origin>4.13891e-013 0 0</origin>
        <direction>0 1 0</direction>
      </line>
    </curve>
    <curve sid="curve2d-12">
      <line>
        <origin>0 0 0</origin>
        <direction>1 0 0</direction>
      </line>
    </curve>
  </surface_curves>
  <!-- 全曲面を定義 -->
  <surfaces>
    <surface sid="surface-1">
      <cylinder>
        <radius>5</radius>
      </cylinder>
    </surface>
    <surface sid="surface-2">
      <plane>
        <equation>0 0 1 0</equation>
      </plane>
    </surface>
    <surface sid="surface-3">
      <plane>
        <equation>0 0 1 0</equation>
      </plane>
      <origin>0 0 10</origin>
    </surface>
    <surface sid="surface-4">
      <cylinder>
        <radius>3</radius>
      </cylinder>
    </surface>
  </surfaces>
  <!-- 位置のソース -->
  <source id="ring.brep.lib.geo.brep.geom-points">

```

```

    <float_array id="ring.brep.lib.geo.brep.geom-points-array" count="12">
      5 -1.22461e-015 0
      5 -1.22461e-015 10
      3 -7.34764e-016 0
      3 -7.34764e-016 10
    </float_array>
    <technique_common>
      <accessor count="4" offset="0"
source="#ring.brep.lib.geo.brep.geom-points-array" stride="3">
        <param name="X" type="float"/>
        <param name="Y" type="float"/>
        <param name="Z" type="float"/>
      </accessor>
    </technique_common>
  </source>

  <!-- 2次元曲線を参照するソース -->
  <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>

  <!-- 曲線を参照するソース -->
  <source id="ring.brep.lib.geo.brep.geom-curves">
    <SIDREF_array count="6" id="ring.brep.lib.geo.brep.geom-curves-array">
      curve-1 curve-2 curve-3 curve-4 curve-5 curve-6
    </SIDREF_array>
  </source>

  <!-- 曲面を参照するソース -->
  <source id="ring.brep.lib.geo.brep.geom-surfaces">
    <SIDREF_array count="4"
id="ring.brep.lib.geo.brep.geom-surfaces-array">
      surface-1 surface-2 surface-3 surface-4
    </SIDREF_array>
  </source>
  <!-- 可能な方向のソース -->
  <source id="ring.brep.lib.geo.brep.orientations">
    <Name_array id="ring.brep.lib.geo.brep.orientations-array" count="2">
      REVERSED FORWARD
    </Name_array>
    <technique_common>
      <accessor count="2" offset="0"
source="#ring.brep.lib.geo.brep.orientations-array" stride="1">
        <param name="ORIENTATION" type="Name"/>
      </accessor>
    </technique_common>
  </source>

  <!-- 辺の曲線用のパラメータを含むソース -->
  <source id="ring.brep.lib.geo.brep.curve-params">
    <float_array id="ring.brep.lib.geo.brep.curve-params-array" count="12">
      0 10
      0 6.28319
      0 6.28319
      0 6.28319
  </float_array>
  </source>

```

```

    0 6.28319
    0 10
  </float_array>
  <technique_common>
    <accessor count="6" offset="0"
source="#ring.brep.lib.geo.brep.curve-params-array" stride="2">
      <param name="START" type="float"/>
      <param name="END" type="float"/>
    </accessor>
  </technique_common>
</source>

<!-- 面のマテリアルシンボルを含むソース -->
<source id="ring.brep.lib.geo.brep.materials">
  <Name_array id="ring.brep.lib.geo.brep.materials-array" count="1">
    WHITE
  </Name_array>
  <technique_common>
    <accessor count="1" offset="0"
source="#ring.brep.lib.geo.brep.materials-array" stride="1">
      <param name="MATERIAL" type="Name"/>
    </accessor>
  </technique_common>
</source>

<!-- 頂点 -->
<vertices id="ring.brep.lib.geo.brep.vertices">
  <input semantic="POSITION"
source="#ring.brep.lib.geo.brep.geom-points"/>
</vertices>

<!-- 辺 -->
<edges id="ring.brep.lib.geo.brep.edges" count="6">
  <input semantic="CURVE" source="#ring.brep.lib.geo.brep.geom-curves"
offset="0"/>
  <input semantic="VERTEX" source="#ring.brep.lib.geo.brep.vertices"
offset="1"/>
  <input semantic="VERTEX" source="#ring.brep.lib.geo.brep.vertices"
offset="2"/>
  <input semantic="PARAM" source="#ring.brep.lib.geo.brep.curve-params"
offset="3"/>
  <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>

<!-- ワイヤ -->
<wires count="6" id="ring.brep.lib.geo.brep.wires">
  <input semantic="EDGE" source="#ring.brep.lib.geo.brep.edges"
offset="0"/>
  <input semantic="ORIENTATION" offset="1"
source="#ring.brep.lib.geo.brep.orientations"/>
  <vcount>4 1 1 1 1 4 </vcount>
  <p>0 1 1 0 0 0 2 1 3 0 2 1 4 0 1 1 5 1 4 0 5 0 3 1</p>
</wires>

<!-- 面 -->

```

```

        <faces count="4" id="ring.brep.lib.geo.brep.faces">
            <input semantic="SURFACE"
source="#ring.brep.lib.geo.brep.geom-surfaces" offset="0"/>
            <input semantic="WIRE" source="#ring.brep.lib.geo.brep.wires"
offset="1"/>
            <input semantic="ORIENTATION" offset="2"
source="#ring.brep.lib.geo.brep.orientations"/>
            <input semantic="MATERIAL" offset="3"
source="#ring.brep.lib.geo.brep.materials"/>
            <vcount>1 2 2 1</vcount>
            <p>0 0 1 0 1 1 1 0 1 2 1 0 2 3 1 0 2 4 1 0 3 5 1 0</p>
        </faces>

        <!-- パラメトリック曲線 -->
        <pcurves id="ring.brep.lib.geo.brep.pcurves" count="12">
            <input semantic="EDGE" source="#ring.brep.lib.geo.brep.edges"
offset="0"/>
            <input semantic="FACE" source="#ring.brep.lib.geo.brep.faces"
offset="1"/>
            <input semantic="CURVE2D" offset="2"
source="#ring.brep.lib.geo.brep.geom-curves2d"/>
            <vcount>2 1 1 1 1 1 1 1 2 </vcount>
            <p>0 0 0 0 0 2 1 0 1 1 2 7 2 0 3 2 1 5 3 1 4 3 3 11 4 2 6 4 3 9 5 3 8 5
3 10</p>
        </pcurves>

        <!-- シェル -->
        <shells count="1" id="ring.brep.lib.geo.brep.shells">
            <input semantic="FACE" source="#ring.brep.lib.geo.brep.faces"
offset="0"/>
            <input semantic="ORIENTATION" offset="1"
source="#ring.brep.lib.geo.brep.orientations"/>
            <vcount>4 </vcount>
            <p>0 1 1 0 2 1 3 0</p>
        </shells>

        <!-- ソリッド -->
        <solids count="1" id="ring.brep.lib.geo.brep.solids">
            <input semantic="SHELL" offset="0"
source="#ring.brep.lib.geo.brep.shells"/>
            <input semantic="ORIENTATION" offset="1"
source="#ring.brep.lib.geo.brep.orientations"/>
            <vcount>1 </vcount>
            <p>0 1</p>
        </solids>
    </brep>
</geometry>
</library_geometries>

<!-- エフェクトの定義 -->
<library_effects>
    <effect id="whitePhong">
        <profile_COMMON>
            <technique sid="phong1">
                <phong>
                    <emission>
                        <color>1.0 1.0 1.0 1.0</color>
                    </emission>
                    <ambient>
                        <color>1.0 1.0 1.0 1.0</color>
                    </ambient>
                </phong>
            </technique>
        </profile_COMMON>
    </effect>
</library_effects>

```



```

    <diffuse>
      <color>1.0 1.0 1.0 1.0</color>
    </diffuse>
    <specular>
      <color>1.0 1.0 1.0 1.0</color>
    </specular>
    <shininess>
      <float>20.0</float>
    </shininess>
    <reflective>
      <color>1.0 1.0 1.0 1.0</color>
    </reflective>
    <reflectivity>
      <float>0.5</float>
    </reflectivity>
    <transparent>
      <color>1.0 1.0 1.0 1.0</color>
    </transparent>
    <transparency>
      <float>1.0</float>
    </transparency>
  </phong>
</technique>
</profile_COMMON>
</effect>
</library_effects>

<!-- マテリアルの定義 -->
<library_materials>
  <material id="whiteMaterial">
    <instance_effect url="#whitePhong" />
  </material>
</library_materials>

<!-- ジオメトリをビジュアルシーンの中にインスタンス化する -->
<library_visual_scenes>
  <visual_scene id="DefaultScene">
    <node id="Ring" name="Ring">
      <translate> 0 0 0</translate>
      <rotate> 0 0 1 0</rotate>
    <rotate>0 1 0 0</rotate>
      <rotate> 1 0 0 0</rotate>
      <scale> 1 1 1</scale>
      <instance_geometry url="#ring.BREP.lib.geo">
        <bind_material>
          <technique_common>
            <instance_material symbol="WHITE" target="#whiteMaterial"/>
          </technique_common>
        </bind_material>
      </instance_geometry>
    </node>
  </visual_scene>
</library_visual_scenes>
<scene>
  <instance_visual_scene url="#DefaultScene"/>
</scene>
</COLLADA>

```

このページは空白です。

10章 キネマティックスリファレンス

概要

この章は、COLLADA アニメーションのキネマティックス部分を構成する要素を扱います。

COLLADA キネマティックスを用いると、コンテンツ制作者は、ビジュアルシーンのオブジェクトに運動学的な性質を加えることが可能になります。

ビジュアルシーンのノードは、運動学的シミュレーションによって制御できます。これは、任意の数のキネマティックモデルによって実現されます。キネマティックモデルは、ジョイントとリンクから構成されます。

- ジョイントは、1つ以上のジョイントプリミティブ（回転、並進）、およびその任意の軸によって指定されます。
- リンクは、ジョイントによって連結される剛体です。

キネマティックモデルは、1つ以上の関節式システムによって制御できます。関節式システムは、キネマティックモデルを拡張して、運動学的な性質や動力学的な性質を加えます。

要素のカテゴリ分類

この章では、要素をアルファベット順に記載します。関連する要素を見つけやすくするため、以下のセクションでは、要素をカテゴリ別に記載しています。

ジョイント

ジョイント型の適切かつ完全な定義は、キネマティックモデル定義の必須条件です。

以下の COLLADA 要素を使うと、シーンのキネマティック記述で使われるジョイント（プリミティブおよび複合）を定義することができます。

これらの要素の主な用途は、以下の通りです。

- 単数および複数の軸や自由度をもつジョイントを定義するメカニズムを単純化する
- 物理に影響されないようにする
- 各軸（もしくはプリミティブジョイント）を処理できるようにする
- 複雑なジョイントを定義する

<code><joint></code>	1つ以上の自由度をもつ単一のジョイントを定義します。
<code><library_joints></code>	<code><joint></code> 要素のモジュールを宣言します。
<code><prismatic></code>	ジョイントに単一の並進自由度を定義します。
<code><revolute></code>	ジョイントに単一の回転自由度を定義します。

キネマティックモデル

キネマティック要素は、COLLADA の既存の機能と以下のような関係があります。

- いかなるジョイント宣言にも影響を受けない`<node>`階層の定義を利用します
- 物理モデルやキネマティックモデルのインスタンスは、これらのノードを参照できます
- 各ノードで宣言された形状を、視覚的な表現に使用します

- `<library_nodes>` (もしくは`<visual_scene>`を直接) 利用してノード階層を定義します

COLLADA には、キネマティクスチェーンの一般的な記述は含まれません。

<code><attachment_end></code>	アタッチメント中の閉じたループの一方の端を定義します。
<code><attachment_full></code>	2つのリンクを連結します。
<code><attachment_start></code>	2つのリンクを連結して、閉ループの一方の端を定義します。
<code><instance_joint></code>	COLLADA ジョイントリソースをインスタンス化します。
<code><instance_kinematics_model></code>	COLLADA <code><kinematics_model></code> リソースをインスタンス化します。
<code><kinematics_model></code>	キネマティクスモデルを記述します。
<code><library_kinematics_models></code>	<code><kinematics_model></code> 要素を保存するライブラリを提供します。
<code><link></code>	運動が1つ以上のジョイントによって拘束される、質量のない、運動学的な剛体を表現します。

関節式システム

関節式システムは、キネマティクスモデルの挙動を制御します。COLLADA キネマティクスでは、関節式システムを2つに区別しています。1つのシステムはキネマティクスチェーンの厳密に運動学的な側面を制御し、もう1つのシステムは動力的な側面(運動)を制御します。

<code><articulated_system></code>	キネマティクスシステムの汎用的な制御情報の宣言をカテゴリ化します。
<code><axis_info></code>	関節式モデルの運動学的な運動の挙動を記述するための、軸情報を含みます。
<code><bind></code> (キネマティクス)	インスタンス化時に、入力をキネマティクスパラメータにバインドします。
<code><connect_param></code>	事前に定義した2つのパラメータの間にシンボリック接続を作成します。
<code><effector_info></code>	エフェクタに追加の動力学情報を指定します。
<code><frame_object></code>	キネマティクスの計算に使われる座標系の情報が含まれます。
<code><frame_origin></code>	キネマティクスの計算に使われる座標系の情報が含まれます。
<code><frame_tcp></code>	キネマティクスの計算に使われる座標系の情報が含まれます。
<code><frame_tip></code>	キネマティクスの計算に使われる座標系の情報が含まれます。
<code><instance_articulated_system></code>	COLLADA <code><articulated_system></code> リソースをインスタンス化します。
<code><kinematics></code>	関節式モデルの運動学的挙動を記述するための、追加情報が含まれます。
<code><library_articulated_systems></code>	<code><articulated_system></code> 要素を保存するライブラリを提供します。
<code><motion></code>	関節式モデルの動力的な挙動を記述するための、追加情報が含まれます。

キネマティクスシーン

キネマティクスシーンは、具体的なシーン用にインスタンス化されたモデルです。キネマティクスシーンは、使われるリンクや現在のシーンの設定、特に、デフォルトや現在のジョイント値を定義します。

<code><bind_joint_axis></code>	キネマティクスモデルのジョイント軸を、特定のノード変換にバインドします。
<code><bind_kinematics_model></code>	キネマティクスモデルをノードにバインドします。
<code><instance_kinematics_scene></code>	COLLADA <code><kinematics_scene></code> リソースをインスタンス化します。
<code><kinematics_scene></code>	COLLADA リソースのコンテンツ中の、関節式モデルに変換可能な情報セット全体を表します。
<code><library_kinematics_scenes></code>	<code><kinematics_scene></code> 要素を保存するライブラリを提供します。

articulated_system

カテゴリ: 関節式システム

概要

キネマティクスシステムの汎用的な制御情報の宣言をカテゴリ化します。

コンセプト

キネマティクスシステムは、関節式モデル (`<kinematics_model>`) の操作を管理するデバイスもしくはメカニズムです。

属性

`<articulated_system>`要素には、以下の属性があります。

id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<articulated_system>`要素は、以下の要素と関連性があります。

親要素	<code>library_articulated_systems</code>
子要素	下記サブセクションを参照してください。
その他	<code>instance_articulated_system</code>

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

注: 子要素 `<kinematics>` または `<motion>` のいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code>control_category</code>	制御の種類を記述します。以下のいずれかである必要があります。 <ul style="list-style-type: none"> <code><kinematics></code> <code><motion></code> 主見出し項目を参照してください。		1
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

例

以下は、運動学および動力学的な側面を制御する 2 つの `<articulated_system>` 要素の例です。

```
<articulated_system id="Kinematics">
  <kinematics/>
</articulated_system>

<articulated_system id="Motion">
  <motion/>
```

```
</articulated_system>
```

attachment_end

カテゴリ:キネマティクスモデル

概要

アタッチメント中の閉じたループの一方の端を定義します。

コンセプト

アタッチメントは、2つのリンクを連結して、ループの終端を定義します。

キネマティクスモデルは、通常、ジョイントによって連結されるリンクのツリーベースの階層ですが、このモデルでは、閉じたループを利用することもできます。`<attachment_end>`要素は、閉じたループの一方の端を定義します。

属性

`<attachment_end>`要素には、以下の属性があります。

joint	xs:token	親を子リンクに連結するジョイントへの参照。必須。
-------	----------	--------------------------

関連要素

`<attachment_end>`要素は、以下の要素と関連性があります。

親要素	link
子要素	下記サブセクションを参照してください。
その他	attachment_start , attachment_full

子要素

子要素を任意の順で記述できます。

名前/例	解説	デフォルト値	出現回数
<code><translate></code>	コアの主見出し項目を参照してください。		0 以上
<code><rotate></code>	コアの主見出し項目を参照してください。		0 以上

例

`<attachment_start>`を参照してください。

attachment_full

カテゴリ:キネマティックモデル

概要

2つのリンクを連結します。

コンセプト

<attachment_full>要素は、2つのリンクの間の実際の親子依存関係を記述します。

属性

<attachment_full>要素には、以下の属性があります。

joint	xs:token	親を子リンクに連結するジョイントへの参照。必須。
-------	----------	--------------------------

関連要素

<attachment_full>要素は、以下の要素と関連性があります。

親要素	link
子要素	下記サブセクションを参照してください。
その他	attachment_start, attachment_end

子要素

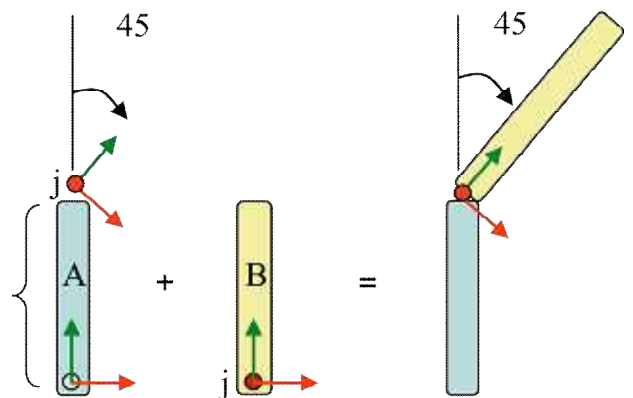
子要素が存在する場合には、以下の順序で出現する必要があります。ただし、<rotate>と<translate>は、<link>の前なら任意の順序で出現することができます。

名前/例	解説	デフォルト値	出現回数
<rotate>	コアの主見出し項目を参照してください。		0 以上
<translate>	コアの主見出し項目を参照してください。		0 以上
<link>	主見出し項目を参照してください。		1

例

以下は、<attachment_full>要素の例です。

```
<link sid="A">
  <translate>0 2 0</translate>
  <rotate>0 0 1 45</rotate>
  <attachment_full joint="model/joint">
    <link sid="B"/>
  </attachment_full>
</link>
```



attachment_start

カテゴリ:キネマティックモデル

概要

2つのリンクを連結して、閉ループの一方の端を定義します。

コンセプト

キネマティックモデルは、通常、ジョイントによって連結されるリンクのツリーベースの階層ですが、このモデルでは、閉じたループを利用することもできます。`<attachment_start>`要素は、閉じたループの一方の端を定義します。

属性

`<attachment_start>`要素には、以下の属性があります。

joint	xs:token	親を子リンクに連結するジョイントへの参照。必須。
-------	----------	--------------------------

関連要素

`<attachment_start>`要素は、以下の要素と関連性があります。

親要素	link
子要素	下記サブセクションを参照してください。
その他	attachment_end, attachment_full

子要素

子要素を任意の順で記述できます。子要素は、少なくとも1つは存在する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><translate></code>	コアの主見出し項目を参照してください。		0 以上
<code><rotate></code>	コアの主見出し項目を参照してください。		0 以上

例

以下は、`<attachment_start>`要素と`<attachment_end>`要素を利用した閉じたループの例です。

```
<link sid="A">
<attachment_full joint="model/j1">
  <translate>0 5 0</translate>
  <link sid="B">
    <translate>-6 0 0</translate>
    <attachment_start joint="model/j4">
      <translate>-3 0 0</translate>
    </attachment_start>
  </link>
</attachment_full>

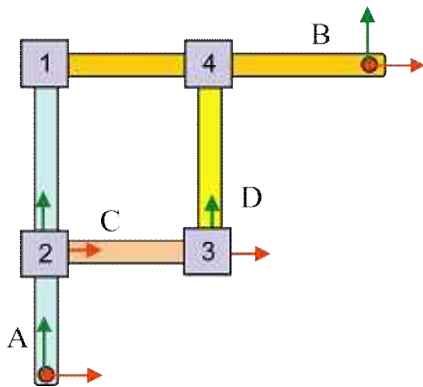
<attachment_full joint="model/j2">
  <translate>0 2 0</translate>
  <link sid="C">
    <attachment_full joint="model/j3">
      <translate>3 0 0</translate>
```



```

<link sid="D">
<attachment_end joint="model/j4">
  <translate>0 3 0</translate>
</attachment_end>
</link>
</attachment_full>
</link>
</attachment_full>
</link>

```



axis_info

カテゴリ: 関節式システム

概要

関節式モデルの運動力学的な運動の挙動を記述するための、軸情報を含みます。

コンセプト

この要素は、以下の機能を提供します。

- 各軸を拡張して、キネマティクスや運動に関する情報を追加する。
- 追加情報を直接設定する、もしくは、パラメータ定義により設定する。
- パラメータを利用して特定の値を後で変更する。

属性

<axis_info>要素には、以下の属性があります。

属性名	xs:token	説明
sid	xs:token	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
name	xs:token	この要素の文字列の名。オプション。
axis	xs:token	必須。 <kinematics>中: インスタンス化されたキネマティクスモデルのジョイント軸。 <motion>中: インスタンス化されたキネマティクスシステムのaxis_info。

関連要素

`<axis_info>`要素は、以下の要素と関連性があります。

親要素	<code>kinematics, motion</code>
子要素	下記サブセクションを参照してください。
その他	なし

`<kinematics>/<axis_info>`の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><newparam></code>	後で利用する新しいパラメータを指定します。コアの主見出し項目を参照してください。		0 以上
<code><active></code>	軸がアクティブかどうかを指定する <code>common_bool_or_param_type</code> が含まれます。この要素には属性はありません。	True	0 または 1
<code><locked></code>	軸がロックされているかどうかを指定する <code>common_bool_or_param_type</code> が含まれます。この要素には属性はありません。	False	0 または 1
<code><index semantic=" "></code>	ジョイントマップ (COLLADA では定義されていない) 中の軸のインデックスを指定する <code>common_int_or_param_type</code> が含まれます。設定しないと、この軸はジョイントマップに表示されません。 オプションの <code>semantic</code> 引数は、 <code>xs:NCName</code> 型で、そのインデックスの特殊な使い方を指定します。		0 以上
<code><limits></code> <code><min>...</min></code> <code><max>...</max></code> <code></limits></code>	ソフト制限を指定します。設定しなかった場合には、軸は物理的な制限だけに制約されます。 <code>min</code> , <code>max</code> は、 <code>common_float_or_param_type</code> 型の必須要素です。	なし	0 または 1
<code><formula></code>	<code>formula</code> は、パッシブリンクの挙動を 1 つ以上のアクティブな軸によって定義する際に役立つことがあります。主見出し項目を参照してください。	なし	0 以上
<code><instance_formula></code>	定義済みの <code>formula</code> をインスタンス化します。 <code>formula</code> は、たとえば、ソフト制限と別のジョイントの依存関係を定義する際などに利用できます。主見出し項目を参照してください。	なし	0 以上

`<motion>/<axis_info>`の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><bind></code> (キネマティックス)	パラメータを新しいパラメータシンボルにバインドします。主見出し項目を参照してください。	なし	0 以上
<code><newparam></code>	後で利用する新しいパラメータを指定します。コアの主見出し項目を参照してください。	なし	0 以上
<code><setparam></code>	指定されたパラメータに値を割り当てます。コアの主見出し項目を参照してください。	なし	0 以上

名前/例	解説	デフォルト値	出現回数
<code><speed></code>	軸の最大許容速度をメートル毎秒 (m/s) で指定する <code>common_float_or_param_type</code> が含まれます。この要素には属性はありません。	なし	0 または 1
<code><acceleration></code>	軸の最大許容加速度をメートル毎秒毎秒 (m/sec ²) で指定する <code>common_float_or_param_type</code> が含まれます。この要素には属性はありません。	なし	0 または 1
<code><deceleration></code>	軸の最大許容減速度を指定する <code>common_float_or_param_type</code> が含まれます。設定しない場合、加速度と減速度は同じ値 (m/sec ²) になります。この要素には属性はありません。	なし	0 または 1
<code><jerk></code>	軸の最大許容躍度 (ジャーク、加加速度) をメートル毎秒毎秒毎秒 (m/sec ³) で指定する <code>common_float_or_param_type</code> が含まれます。この要素には属性はありません。	なし	0 または 1

詳細

速度、加速度、減速度、躍度に関するより具体的な情報については、`<motion>`を参照してください。

例

以下は、キネマティクスシステム内の`<axis_info>`要素の例です。

```
<axis_info sid="a1" axis="man/jl/axis_x">
  <locked><bool>false</bool></locked>
  <active><bool>>true</bool></active>
  <index><int>1</int></index>
  <limits>
    <min><double>-90</double></min>
    <max><double>90</double></max>
  </limits>
</axis_info>
```

以下は、モーションシステム内の`<axis_info>`要素の例です。

```
<axis_info axis="kinematics_system/a1">
  <speed><double>0.5</double>/speed>
  <acceleration><double>0.3</double></acceleration>
  <deceleration><double>0.2</double></deceleration>
  <jerk><double>0.1</double></jerk>
</axis_info>
```

bind

(キネマティクス)

カテゴリ: 関節式システム

概要

インスタンス化時に、入力をキネマティクスパラメータにバインドします。

コンセプト

`<bind>`は、定義済みパラメータを新しいシンボル名にマップします。つまり、インスタンス化されたオブジェクト（関節式システムによってインスタンス化されたキネマティクスモデルなど）のパラメータは、それが定義されたスコープ内で有効な新しいシンボル名にバインドすることができます。

属性

`<bind>`要素には、以下の属性があります。

symbol	xs:NCName	新しいシンボル名にバインドするパラメータの識別子。必須。
--------	-----------	------------------------------

関連要素

`<bind>`要素は、以下の要素と関連性があります。

親要素	<code>instance_articulated_system</code> , <code>instance_kinematics_model</code> , <code>motion/axis_info</code> , <code>effector_info</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

以下の子要素のいずれか 1 つだけが出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><param></code> (reference)	コアの主見出し項目を参照してください。	なし	1
<code><float></code>		なし	1
<code><int></code>		なし	1
<code><bool></code>		なし	1
<code><SIDREF></code>	COLLADA スコープ付き識別子 (<code>sidref_type</code>) への参照が含まれます。SID および SIDREF の詳細は、3 章「スキーマのコンセプト」の「アドレス構文」を参照してください。	なし	1

例

以下は、`<bind>`要素の例です。

```

<instance_articulated_system sid="system" url="#MOTION">
  <bind symbol="motion.kinematics.model">
    <param ref="kinematics.model"/>
  </bind>
</instance_articulated_system>

```

bind_joint_axis

カテゴリ:キネマティックシーン

概要

キネマティックモデルのジョイント軸を、特定のノード変換にバインドします。

コンセプト

ジョイント軸をノードの変換にバインドすることによって、キネマティックシーンをビジュアルシーンに同期させることができます。

属性

`<bind_joint_axis>`要素には、以下の属性があります。

target	xs:token	ノードの変換への参照。オプション。
--------	----------	-------------------

関連要素

`<bind_joint_axis>`要素は、以下の要素と関連性があります。

親要素	<code>instance_kinematics_scene</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

以下の子要素のいずれか1つだけが出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<code><axis></code>	キネマティックモデルの軸を指定する <code>common_sidref_or_param_type</code> 。この要素には属性はありません。SIDの詳細は、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。	なし	1
<code><value></code>	軸の値を指定する <code>common_float_or_param_type</code> 。この要素には属性はありません。	なし	1

詳細

`<rotate>`は `float4_type` (値のある軸)、`<translate>`は `float3_type` (長さのある軸)なので、キネマティックスでは、アニメーションで使われるチャンネルメカニズムを利用することはできません。その代わりに、軸とその値の組合せを、`<rotate>`や`<translate>`要素にバインドします。参照されるSIDは、キネマティックシーンの中で定義されます。

例

以下は、`<bind_joint_axis>`要素の例です。

```
<instance_kinematics_scene url="#KINEMATICS_SCENE">

<bind_kinematics_model node="WORLD/POSITION_OF_THE_ARM">
  <param>scene.model</param>
</bind_kinematics_model>
```

```

<bind_joint_axis
target="WORLD/POSITION_OF_THE_ARM/instantiated_arm/ELBOW/FOREARM/rotate_x">
  <axis><param>scene.model.elbow.x.target</param></axis>
  <value><param>scene.model.elbow.x.value</param></value>
</bind_joint_axis>

</instance_kinematics_scene>

```

bind_kinematics_model

カテゴリ:キネマティックシーン

概要

キネマティックモデルをノードにバインドします。

コンセプト

キネマティックモデルの記述は、いかなる視覚情報にもまったく影響されませんが、位置は計算のために重要です。

属性

<bind_kinematics_model>要素には、以下の属性があります。

node	xs:token	ノードへの参照。オプション。
------	----------	----------------

関連要素

<bind_kinematics_model>要素は、以下の要素と関連性があります。

親要素	instance_kinematics_scene
子要素	下記サブセクションを参照してください。
その他	なし

子要素

以下の子要素のいずれか1つだけが出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<param>(reference)	インスタンス化されたキネマティックシーンの中で定義されたキネマティックモデルのパラメータ。コアの主見出し項目を参照してください。		1
<SIDREF>	ノードにバインドするキネマティックモデルへのSIDパス。COLLADA スコープ付き識別子が含まれます。SIDおよびSIDREFの詳細は、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。		1

詳細

例

以下は、`<bind_kinematics_model>`要素の例です。

```

<scene>

  <instance_visual_scene url="#VISUAL_SCENE" sid="vscene" />

  <instance_kinematics_scene url="#KINEMATICS_SCENE" sid="kscene">

    <bind_kinematics_model node="WORLD/POSITION_OF_THE_ARM">
      <param>scene.model</param>
    </bind_kinematics_model>

  </instance_kinematics_scene>

</scene>

```

connect_param

(キネマティックス)

カテゴリ: 関節式システム

概要

事前に定義した2つのパラメータの間にシンボリック接続を作成します。

コンセプト

パラメータ同士を接続すると、1つのパラメータを多数のキネマティックスオブジェクトの入力に結び付けることができます。親の値を設定することで、すべての子の参照が自動的に更新されます。

この接続メカニズムのおかげで、共通パラメータの値を一度に設定でき、また何度も再利用できます。さらに、抽象インターフェイスにアタッチする複数のクラスを具体化することも可能となります。

属性

`<connect_param>`要素には、以下の属性があります。

ref	xs:token	現在のパラメータに結びつけるターゲットパラメータへの参照。必須。
-----	----------	----------------------------------

関連要素

`<connect_param>`要素は、以下の要素と関連性があります。

親要素	setparam
子要素	なし。
その他	なし

例

以下は、`<connect_param>`要素の例です。

```

<instance_articulated_system url="#MOTION_SYSTEM" sid="model">
  <setparam ref="motion.model.elbow.x.locked">
    <bool>true</bool>
  </setparam>
</instance_articulated_system>

```

```

    </setparam>
    <setparam ref="motion.model.elbow.z.locked">
      <connect_param ref="motion.model.elbow.x.locked"/>
    </setparam>
  </instance_articulated_system>

```

effector_info

カテゴリ: 関節式システム

概要

エフェクタに追加の動力学情報を指定します。

コンセプト

速度や加速度を限定するための運動プロファイルを持つジョイントの場合、エンドエフェクタにも運動プロファイルがあることがあります。このエフェクタは、ジョイントとは違って、指定された軸に従いません。したがって、エフェクタの動力学的な属性は、2つの値によって定義されます。1番目の値は並進運動、2番目の値は回転運動を制限します。

属性

<effector_info>要素には、以下の属性があります。

sid	sid_type	説明
		この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」のアドレス構文を参照してください。
name	xs:token	この要素の文字列の名。オプション。

関連要素

<effector_info>要素は、以下の要素と関連性があります。

親要素	motion/technique_common
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<bind> (キネマティックス)	値やパラメータを新しいシンボル名にバインドします。主見出し項目を参照してください。	なし	0 以上
<newparam>	エフェクタ用の新しいパラメータを定義します。コアの主見出し項目を参照してください。	なし	0 以上
<setparam>	インスタンス化されたキネマティックスモデルの定義済みパラメータに、具体的な値を割り当てます。コアの主見出し項目を参照してください。	なし	0 以上
<speed> 1 0.8 </speed>	最高速度を指定します。1番目の値は並進速度 (m/s)、2番目の値は回転速度 (°/sec) です (common_float2_or_param_type 型です)。この	なし	0 または 1

名前/例	解説	デフォルト値	出現回数
	要素には属性はありません。		
<pre><acceleration> 0.6 0.8 </acceleration></pre>	最大加速度を指定します。1番目の値は並進加速度 (m/sec ²)、2番目の値は回転加速度 (°/sec ²) です (<code>common_float2_or_param_type</code> 型です)。この要素には属性はありません。	なし	0 または 1
<pre><deceleration> 0.6 0.8 </deceleration></pre>	最大減速度を指定します。1番目の値は並進加速度 (m/sec ²)、2番目の値は回転加速度 (°/sec ²) です (<code>common_float2_or_param_type</code> 型です)。この要素には属性はありません。	なし	0 または 1
<pre><jerk> 0.3 0.4 </jerk></pre>	最大躍度 (ジャーク、加加速度とも呼ばれる) を指定します。1番目の値は並進躍度 (m/sec ³)、2番目の値は回転減躍度 (°/sec ³) です (<code>common_float2_or_param_type</code> 型です)。この要素には属性はありません。	なし	0 または 1

詳細

速度、加速度、減速度、加加速度に関するより具体的な情報については、`<motion>`を参照してください。

例

以下は、`<effector_info>`要素の例です。

```
<effector_info>

  <speed><double>1.0 0.8</double></speed>
  <acceleration><double>0.8 0.6</double></acceleration>
  <deceleration><double>0.7 0.6</double></deceleration>
  <jerk><double>0.3 0.4</double></jerk>

</effector_info>
```

frame_object、frame_origin、frame_tcp、frame_tip

カテゴリ: 関節式システム

概要

キネマティックスの計算に使われる座標系の情報が含まれます。

コンセプト

キネマティックス座標系には、以下のような種類があります。

- Origin: キネマティックス計算用の基礎となる座標を定義します。
- Tip: キネマティックスチェーンの終端の座標を定義します。
- Tcp: キネマティックスの`<frame_tip>`からのオフセット座標を定義します。通常は、エンドエフェクタ (溶接ガンなど) の作業点を表現するのに使われます。
- Object: キネマティックス`<frame_origin>`からの、オフセット座標を定義します。このオフセットは、通常、工作物に対する変換を表現します。

属性

`<frame_*>`要素には、それぞれ、以下の属性があります。

link	xs:token	<code><kinematics_model></code> の中で定義された <code><link></code> のSIDを参照します。オプション。SIDの詳細は、3章「スキーマのコンセプト」のアドレス構文を参照してください。
------	----------	---

関連要素

`<frame_*>`要素は、それぞれ、以下の要素と関係があります。

親要素	kinematics/technique_common
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素を任意の順で記述できます。

名前/例	解説	デフォルト値	出現回数
<code><translate></code>	コアの主見出し項目を参照してください。	なし	0 以上
<code><rotate></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<frame_*>`要素の例です。

```
<frame_origin link="model/upper_arm"/>
<frame_tip link="model/upper_arm/fore_arm/hand">
  <translate>10 0 0</translate>
</frame_tip>
```

instance_articulated_system

カテゴリ: 関節式システム

概要

COLLADA`<articulated_system>`リソースをインスタンス化します。

コンセプト

関節式システムは、組み込まれたキネマティックモデルを拡張してプロパティを追加するために、別の関節式システムやキネマティックシーンの中でインスタンス化することができます。指定されたパラメータに対しては、具体的な値を設定することもできるし、指定しない場合にはデフォルト値が使われます。新しいパラメータを指定することもできます。

COLLADAのインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

`<instance_articulated_system>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURL	インスタンス化すべきキネマティクスモデルの場所の URL。必須。#文字で始まる、相対 URI の断片的な識別子を利用してローカルなインスタンスへの参照を表します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

`<instance_articulated_system>`要素は、以下の要素と関連性があります。

親要素	<code>kinematics_scene</code> , <code>motion</code>
子要素	下記サブセクションを参照してください。
その他	<code>articulated_system</code> , <code>library_articulated_systems</code>

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><bind></code> (キネマティクス)	主見出し項目を参照してください。		0 以上
<code><setparam></code>	コアの主見出し項目を参照してください。		0 以上
<code><newparam></code>	コアの主見出し項目を参照してください。		0 以上
<code><extra></code>	主見出し項目を参照してください。	なし	0 以上

例

以下は、`<instance_articulated_system>`要素の例です。

```
<instance_articulated_system sid="kinsys" url="#KinSys">
  <bind/>
</instance_articulated_system>
```

instance_joint

カテゴリ:キネマティクスモデル

概要

COLLADA ジョイントリソースをインスタンス化します。

コンセプト

ジョイントの実際のデータ表現は、一度しか格納されないこともあります。けれども、ジョイントは、複数のキネマティクスモデルの中に複数回出現することができます。キネマティクスモデル中に別々に出現したジョイントは、オブジェクトのインスタンスと呼ばれます。

ジョイントのインスタンスは、2つのリンク（物体）の間の1以上の自由度を表現します。ジョイントには、質量や慣性モーメントのような質量特性はありません。ジョイントプリミティブは、1並進自由度もしくは1回転自由度を表現します。並進プリミティブや回転プリミティブには、運動軸ベクトルがあります。

ジョイントには、ベースからフォロアへというリンク順序と、ジョイントプリミティブ軸の方向によって設定される方向があります。ジョイントデータの符号は、すべて、この方向によって決まります。

COLLADA のインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

<instance_joint>要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。
url	xs:anyURI	インスタンス化すべきオブジェクトの場所の URL。必須。#文字で始まる、相対 URI の断片的な識別子を利用してローカルなインスタンスへの参照を表します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<instance_joint>要素は、以下の要素と関連性があります。

親要素	kinematics_model/technique_common
子要素	下記サブセクションを参照してください。
その他	joint , library_joints

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<instance_joint>`要素の例です。

```
<kinematics_model id="model">
  <instance_joint sid="j1" url="#Joint"/>

  <link sid="...">
</link>

</kinematics_model>
```

instance_kinematics_model

カテゴリ:キネマティックモデル

概要

COLLADA`<kinematics_model>`リソースをインスタンス化します。

コンセプト

キネマティックモデルは、プロパティで拡張するために関節式システムの中でインスタンス化したり、キネマティックシーンの中でインスタンス化したりすることができます。このようなインスタンスに対しては、新しいパラメータを定義することができます。

COLLADA のインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

`<instance_kinematics_model>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	インスタンス化すべきキネマティックモデルの場所の URL。必須。#文字で始まる、相対 URI の断片的な識別子を利用してローカルなインスタンスへの参照を表します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

`<instance_kinematics_model>`要素は、以下の要素と関連性があります。

親要素	<code>kinematics_scene</code> , <code>kinematics</code>
子要素	下記サブセクションを参照してください。
その他	<code>kinematics_model</code> , <code>library_kinematics_models</code>

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><bind></code> (キネマティックス)	値やパラメータを新しいシンボル名にバインドします。主見出し項目を参照してください。	なし	0 以上
<code><newparam></code>	制約された型の集合から、新しいパラメータを作成します。コアの主見出し項目を参照してください。	なし	0 以上
<code><setparam></code>	インスタンス化されたキネマティックモデルの定義済みパラメータに、具体的な値を割り当てます。コアの主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<kinematics_scene>`要素の中の`<instance_kinematics_model>`要素の例です。

```
<instance_kinematics_model url="#KINEMATICS_MODEL_ARM" sid="model">
  <newparam sid="scene.model">
    <SIDREF>model</SIDREF>
  </newparam>
  <newparam sid="scene.model.elbow.x.target">
    <SIDREF>model/elbow/x</SIDREF>
  </newparam>
</instance_kinematics_model>
```

instance_kinematics_scene

カテゴリ: キネマティックシーン

概要

COLLADA`<kinematics_scene>`リソースをインスタンス化します。

コンセプト

シーンの中では、1つ以上のキネマティックシーンをインスタンス化されることができます。このようなインスタンスに対しては、新しいパラメータを定義することができます。キネマティックシーンの中で定義されたキネマティックモデルは、幾何学的外観とは完全に独立しているので、そのインスタンスはインスタンス化されたビジュアルシーンの要素にバインドすることができます。シーン中では、運動力学的な挙動は、幾何学的な表現に連動します。

COLLADA のインスタンス要素の詳細は、3章「スキーマのコンセプト」の「インスタンス化と外部参照」を参照してください。

属性

`<instance_kinematics_scene>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
name	xs:token	この要素の文字列の名。オプション。
url	xs:anyURI	インスタンス化すべきキネマティクスモデルの場所の URL。必須。#文字で始まる、相対 URI の断片的な識別子を利用してローカルなインスタンスへの参照を表します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

`<instance_kinematics_scene>`要素は、以下の要素と関連性があります。

親要素	<code>scene</code>
子要素	下記サブセクションを参照してください。
その他	<code>kinematics_scene</code> , <code>library_kinematics_scenes</code>

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	主見出し項目を参照してください。	なし	0 または 1
<code><newparam></code>	コアの主見出し項目を参照してください。	なし	0 以上
<code><setparam></code>	コアの主見出し項目を参照してください。	なし	0 以上
<code><bind_kinematics_model></code>	主見出し項目を参照してください。	なし	0 以上
<code><bind_joint_axis></code>	主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

例

以下は、2 つの `<instance_kinematics_scene>` 要素の例です。

```
<scene>
  <instance_visual_scene url="Simple_men_in_visual_scene"/>

  <instance_kinematics_scene sid="kinematics_scene"
    url="#Simple_man_in_kinematics_scene"/>

</scene>
```

joint

カテゴリ: ジョイント

概要

1つ以上の自由度をもつ単一のジョイントを定義します。

コンセプト

プリミティブジョイントは自由度 1 を持つジョイント（1つの軸が指定される）で、複数のプリミティブ（それぞれ1つの軸を表す）から構成されたより複雑な種類のジョイント（複合ジョイント）を構築するのに使われます。

属性

<joint>要素には、以下の属性があります。

id	xs:ID	<joint> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。
sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この文字列は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。

関連要素

<joint>要素は、以下の要素と関連性があります。

親要素	library_joints , kinematics_model/technique_common
子要素	下記サブセクションを参照してください。
その他	instance_joint

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<i>joint_type</i>	以下の種類のジョイントが最低でも1つは出現する必要があります。 <prismatic> <revolute> 主見出し項目を参照してください。	なし	1 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、さまざまなプリミティブの<joint>要素の例です。

```
<joint id="Joint1">
  <prismatic>
    ...
  </prismatic>
</joint>
```



```

<joint id="Joint2">
  <revolute>
    ...
  </revolute>
</joint>

```

以下は、複合<joint>要素の例です。

```

<joint id="universal">
  <revolute sid="axis1">
    <axis>1 0 0</axis>
  </revolute>
  <revolute sid="axis2">
    <axis>1 0 0</axis>
  </revolute>
</joint>

```

kinematics

カテゴリ: 関節式システム

概要

関節式モデルの運動学的挙動を記述するための、追加情報が含まれます。

コンセプト

このキネマティックシステムは、ジョイントのために追加のキネマティック情報を含んでおり、キネマティック座標と呼ばれるものを定義します。キネマティックシステムを拡張してキネマティック情報を追加するには、1つ以上のキネマティックモデルの1つ以上のインスタンスが必要です。

ジョイント軸のための追加のキネマティック情報には、以下のようなものがあります。

- アクティブ性: キネマティックモデルの中の軸は、アクティブかパッシブのいずれかです。アクティブとして宣言された軸は、すべて、(フォワードもしくはインバース)キネマティックの計算で考慮に入れる必要があります。インバースキネマティックを解決するアルゴリズムは、このキネマティック定義の一部ではありません。このアルゴリズムは、具体的なキネマティックアプリケーションが提供する必要があります。
パッシブ軸は、フォワードもしくはインバースキネマティックの計算には関与しません。パッシブ軸の値は、アクティブ軸をすべて計算した後で、アクティブ軸の構成から導出されます。(プリミティブ)ジョイント(軸)の式が指定された場合には、軸の値や位置・方向は、その式から計算することができます。それ以外の場合、パッシブ軸の正しい構成を導出するには、アプリケーションのアルゴリズムが必要です。
- ロックモード: 軸がロックモードである場合、インバースキネマティック計算では、この軸は固定値で動かせないとみなす必要があります。
- ジョイントマッピング: これは、2つの異なるキネマティックシステムから、複合システムに、ジョイントをマップします。たとえば、joint1は、どちらのキネマティックシステムにもあるので、関節式システムの中では、2つのジョイントにマップする必要があります。
- ソフト制限: この制限は、各ジョイントで定義された物理的な制限よりも優先されます。この制限は、ジョイントの物理的な制限の範囲にある必要があります。
- 式: 式は、特定のジョイントの制限と、別のジョイントもしくはその制限との間の依存関係を記述するのに利用できます。

属性

<kinematics>要素に属性はありません。

関連要素

<kinematics>要素は、以下の要素と関連性があります。

親要素	articulated_system
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<instance_kinematics_model>	主見出し項目を参照してください。	なし	1 以上
<technique_common>	すべてのCOLLADA実装がサポートしなければならない共通プロファイルのキネマティクス情報を、指定します。「共通プロファイル」の使い方のセクション、それに続く子要素詳細のサブセクション、およびコアの主見出し項目を参照してください。	なし	1
<technique>	各<technique>は、<technique>の profile 属性として指定された特定のプロファイルのためのキネマティクス情報を指定します。コアの主見出し項目を参照してください。	なし	0 以上
<extra>	<kinematics>に情報を追加する複数表現可能なユーザ定義データ (<technique>要素のように、ベースデータを切り替えるのとは反対の操作です)。コアの主見出し項目を参照してください。	なし	0 以上

<kinematics> / <technique_common>の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<axis_info>	主見出し項目を参照してください。	なし	0 以上
<frame_origin>	主見出し項目を参照してください。	なし	1
<frame_tip>	主見出し項目を参照してください。	なし	1
<frame_tcp>	主見出し項目を参照してください。	なし	0または1
<frame_object>	主見出し項目を参照してください。	なし	0または1

詳細

例

以下は、<kinematics>要素の例です。

```

<articulated_system id="KINEMATICS_SYSTEM_ARM">
  <kinematics>
    <instance_kinematics_model url="KINEMATICS_MODEL_ARM" sid="model">
      <newparam sid="kinematics.model">
        <SIDREF>model</SIDREF>
      </newparam>
      <newparam sid="kinematics.model.elbow.x.target">
        <SIDREF>model/elbow/x</SIDREF>
      </newparam>
    </instance_kinematics_model>

    <technique_common>
      <axis_info sid="a1" axis="model/elbow/x">
        <newparam sid="model.elbow.x.locked">
          <bool>>false</bool>
        </newparam>
        <active><bool>true</bool></active>
        <locked><param>model.elbow.x.locked</param></locked>
        <index><int>1</int></index>
        <limits>
          <min><float>-90</float></min>
          <max><float>90</float></max>
        </limits>
      </axis_info>

      <frame_origin link="model/upper_arm"/>
      <frame_tip link="model/upper_arm/fore_arm/hand">
        <translate>10 0 0</translate>
      </frame_tip>

    </technique_common>
  </kinematics>
</articulated_system>

```

kinematics_model

カテゴリ:キネマティックモデル

概要

キネマティックモデルを記述します。

コンセプト

`<kinematics_model>`要素は、キネマティック情報の宣言をカテゴリ化します。キネマティックスは、力学の1分野で、物体の運動を、運動中の質量や力を考慮に入れずに記述します。

`<kinematics_model>`要素には、ジョイント、リンク、接続点の宣言が含まれます。

キネマティックモデルは、厳密に運動学的な記述に重点を置いており、物理的記述はまったく付け加わっていません。この要素は、キネマティックモデルを零点で定義します。

属性

`<kinematics_model>`要素には、以下の属性があります。

id	xs:ID	<code><kinematics_model></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
----	-------	--

name	xs:token	この要素の名前を含むテキスト文字列。オプション。
------	----------	--------------------------

関連要素

<kinematics_model>要素は、以下の要素と関連性があります。

親要素	library_kinematics_models
子要素	下記サブセクションを参照してください。
その他	instance_kinematics_model

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>	主見出し項目を参照してください。	なし	0 または 1
<technique_common>	すべての COLLADA 実装がサポートしなければならない共通プロファイルのキネマティクスモデル情報を、指定します。「共通プロファイル」の使い方のセクション、以下の子要素詳細のサブセクション、およびコアの主見出し項目を参照してください。	なし	1
<technique>	各<technique>は、<technique>の profile 属性として指定された特定のプロファイルのためのキネマティクスモデル情報を指定します。コアの主見出し項目を参照してください。	なし	0 以上
<extra>	<kinematics_model>に情報を追加する複数表現可能なユーザ定義データ (<technique>要素のように、ベースデータを切り替えるのは反対の操作です)。コアの主見出し項目を参照してください。	なし	0 以上

<kinematics_model> / <technique_common>の子要素

名前/例	解説	デフォルト値	出現回数
<newparam>	コアの主見出し項目を参照してください。	なし	0 以上
<i>joint_reference</i>	キネマティクスチェーンで使われる単一のジョイント。オプションで以下のいずれかを指定できます。 <ul style="list-style-type: none"> <joint> <instance_joint> 主見出し項目を参照してください。	なし	0 以上
<link>	キネマティクスチェーンの開始を定義する剛体。主見出し項目を参照してください。	なし	1 以上
<formula>	ジョイント間の依存関係を指定します。コアの主見出し項目を参照してください。	なし	0 以上
<instance_formula>	コアの主見出し項目を参照してください。	なし	0 以上

詳細

キネマティクスモデルの一部となるジョイントは、そのモデル自体の中で定義することもできるし、<library_joints>の中で定義済みのジョイントのインスタンスであることもできます。

例

以下は、ジョイントインスタンスを持つ<kinematics_model>要素の例です。

```
<kinematics_model id="model">
  <instance_joint sid="j1" url="#Joint"/>

  <link sid="...">
</link>

</kinematics_model>
```

以下は、モデル自体でジョイントを定義している<kinematics_model>要素の例です。

```
<kinematics_model id="model">

<joint id="universal">
  <revolute sid="axis1">
    <axis>1 0 0</axis>
  </revolute>
</joint>

  <link sid="...">
    ...
</link>

</kinematics_model>
```

kinematics_scene

カテゴリ:キネマティックシーン

概要

COLLADA リソースのコンテンツ中の、関節式モデルに変換可能な情報セット全体を表します。

コンセプト

属性

<kinematics_scene>要素には、以下の属性があります。

id	xs:ID	<kinematics_scene> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

<kinematics_scene>要素は、以下の要素と関連性があります。

親要素	library_kinematics_scenes
子要素	下記サブセクションを参照してください。
その他	instance_kinematics_scene

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><instance_kinematics_model></code>	主見出し項目を参照してください。	なし	0 以上
<code><instance_articulated_system></code>	主見出し項目を参照してください。	なし	0 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、2つの`<kinematics_scene>`要素の例です。

```
<kinematics_scene>

  <instance_kinematics_model sid="model" url="#Model"/>

  <instance_articulated_system sid="system" url="#System"/>

</kinematics_scene>
```

library_articulated_systems

カテゴリ: 関節式システム

概要

`<articulated_system>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_articulated_systems>`要素には、以下の属性があります。

id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_articulated_systems>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	<code>instance_articulated_system</code>

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><articulated_system></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<library_articulated_systems>`要素の例です。

```
<library_articulated_system>
  <articulated_system id="system">
    ...
  </articulated_system>
  <articulated_system id="system2">
    ...
  </articulated_system>
</library_articulated_system>
```

library_joints

カテゴリ: ジョイント

概要

`<joint>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_joints>`要素には、以下の属性があります。

id	xs:ID	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。

関連要素

`<library_joints>`要素は、以下の要素と関連性があります。

親要素	<code>COLLADA</code>
子要素	下記サブセクションを参照してください。
その他	<code>instance_joint</code>

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><joint></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<library_joints>`要素の例です。

```
<library_joints>
  <joint id="Joint1">
    ...
  </joint>
  <joint id="Joint2">
    ...
  </joint>
</library_joints>
```

library_kinematics_models

カテゴリ: キネマティックモデル

概要

`<kinematics_model>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_kinematics_models>`要素には、以下の属性があります。

<code>id</code>	<code>xs:ID</code>	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
<code>name</code>	<code>xs:token</code>	この要素の名前を含むテキスト文字列。オプション。

関連要素

`<library_kinematics_models>`要素は、以下の要素と関連性があります。

親要素	<code>COLLADA</code>
子要素	下記サブセクションを参照してください。
その他	<code>instance_kinematics_model</code>

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><kinematics_model></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<library_kinematics_models>`要素の例です。

```
<library_kinematics_models>
  <kinematics_model id="Band">
    ...
  </kinematics_model>

  <kinematics_model id="Robot">
    ...
  </kinematics_model>
</library_kinematics_models>
```

library_kinematics_scenes

カテゴリ:キネマティックスシーン

概要

`<kinematics_scene>`要素を保存するライブラリを提供します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための方法の1つは、何らかの条件でデータをいくつかの小さな部分に分割することです。そういったモジュールセットは、ライブラリとして別々のリソース中に保存しておくことができます。

属性

`<library_kinematics_scenes>`要素には、以下の属性があります。

id	xs:ID	<code><library_kinematics_scenes></code> 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	xs:token	この要素の文字列の名。オプション。

関連要素

`<library_kinematics_scenes>`要素は、以下の要素と関連性があります。

親要素	COLLADA
子要素	下記サブセクションを参照してください。
その他	<code>instance_kinematics_scene</code>

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><asset></code>	コアの主見出し項目を参照してください。	なし	0 または 1
<code><kinematics_scene></code>	主見出し項目を参照してください。	なし	1 以上
<code><extra></code>	コアの主見出し項目を参照してください。	なし	0 以上

例

以下は、`<library_kinematics_scenes>`要素の例です。

```
<library_kinematics_scenes>
  <kinematics_scene id="scene1"/>
  <kinematics_scene id="scene2"/>
</library_kinematics_scenes>
```

link

カテゴリ:キネマティックモデル

概要

運動が1つ以上のジョイントによって拘束される、質量のない、運動学的な剛体を表現します。

コンセプト

属性

`<link>`要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この値は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
name	xs:token	この要素の名前を含むテキスト文字列。オプション。

関連要素

`<link>`要素は、以下の要素と関連性があります。

親要素	<code>kinematics_model/technique_common, attachment_full</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><rotate></code>	コアの主見出し項目を参照してください。	なし	0 以上
<code><translate></code>	コアの主見出し項目を参照してください。	なし	0 以上
<code><attachment_full></code>	主見出し項目を参照してください。	なし	0 以上
<code><attachment_start></code>	主見出し項目を参照してください。	なし	0 以上
<code><attachment_end></code>	主見出し項目を参照してください。	なし	0 以上

例

以下は、`<link>`要素の例です。

```
<link sid="A">
  <translate>0 2 0</translate>
  <rotate>0 0 1 45</rotate>

  <attachment_full>
    <link sid="B"/>
  </attachment_full>
</link>
```

motion

カテゴリ: 関節式システム

概要

関節式モデルの動力学的挙動を記述するための、追加情報が含まれます。

コンセプト

このキネマティックシステムには、あらゆる軸およびエンドエフェクタ用の追加の動力学情報が含まれます。

モーションシステムを動力学的な情報で拡張するには、キネマティックシステムのインスタンスが必要です。

追加の動力学的情報は、動力学的制限とも呼ばれ、以下のように分類されます。

- **速度:** 各ジョイント軸およびエンドエフェクタに対して、速度を指定できる必要があります。エンドエフェクタは、並進速度 (m/sec) と回転速度 (°/sec) を区別します。軸の場合には、並進運動と回転運動のどちらかしか行えないので、定義される速度の種類も一種類だけです。
- **加速度:** 各ジョイント軸およびエンドエフェクタに対して、加速度を指定できる必要があります。エンドエフェクタは、並進加速度 (m/sec²) と回転加速度 (°/sec²) を区別します。軸の場合には、並進運動と回転運動のどちらかしか行えないので、定義される加速度の種類も一種類だけです。
- **減速度:** 各ジョイント軸およびエンドエフェクタに対して、減速度を指定できる必要があります。これは、加速度と減速度が異なる場合に必要です。

- 躍度: 各ジョイント軸およびエンドエフェクタに対して、躍度を指定できる必要があります。エンドエフェクタは、並進躍度 (m/sec³) と回転躍度 (°/sec³) を区別します。軸の場合には、並進運動と回転運動のどちらかしか行えないので、定義される躍度の種類も一種類だけです。

属性

<motion>要素に属性はありません。

関連要素

<motion>要素は、以下の要素と関連性があります。

親要素	articulated_system
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<instance_articulated_system>	主見出し項目を参照してください。	なし	1
<technique_common>	コアの主見出し項目を参照してください。	なし	1
<technique>	コアの主見出し項目を参照してください。	なし	0 以上
<extra>	コアの主見出し項目を参照してください。	なし	0 以上

<motion> / <technique_common>の子要素

名前/例	解説	デフォルト値	出現回数
<axis_info>	主見出し項目を参照してください。	なし	0 以上
<effector_info>	主見出し項目を参照してください。	なし	0 または 1

例

以下は、<motion>要素の例です。

```
<articulated_system id="MOTION_SYSTEM_ARM">
  <motion>

    <instance_articulated_system sid="kinematics_system"
      url="#MOTION_SYSTEM_ARM">
    </instance_articulated_system>

      <technique_common>
        <axis_info sid="b1" axis="kinematics_system/a1">
          <bind symbol="motion.model.elbow.x.locked">
            <param ref="model.elbow.x.locked"/>
          </bind>
        </axis_info>

        <effector_info>
          <acceleration><float>0.4</float></acceleration>
        </effector_info>
      </technique_common>
    </instance_articulated_system>
  </motion>
</articulated_system>
```

```

</motion>
</articulated_system>

```

prismatic

カテゴリ: ジョイント

概要

ジョイントに単一の並進自由度を定義します。

コンセプト

必須の子要素<axis>は、対応する平行移動や回転の軸を定義します。この軸は、3次元空間中の任意の方向に、自由に指定することができます。つまり、<axis> 0 0 1 </axis>や<axis> 0.7071 0.7071 0 </axis>といった軸の定義ができるということです。

これらの要素では、オプションの sid 属性を使って空間識別子を定義することができます。したがって、各プリミティブジョイントは、{ジョイント型の ID}/{プリミティブの sid}、によって参照できます。

属性

<prismatic>要素には、以下の属性があります。

sid	sid_type	この要素のスコープ付き識別子を含むテキスト文字列。この文字列は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。
-----	----------	---

関連要素

<prismatic>要素は、以下の要素と関連性があります。

親要素	joint
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<axis sid="" name="" />	自由度の軸を指定する3つの浮動小数点値。sid および name 属性は、オプションです。	なし	1
<limits> <min sid="..." name="" /> <max sid="..." name="" /> </limits>	指定された軸を持つプリミティブジョイント型に制限を指定するには、静的な最小値や最大値を使った<limits>要素をオプションで追加できます。<limits>が省略された場合、そのジョイントには制限がないものとして扱われます。<min> および<max>: どちらもオプションで、浮動小数点数として1回ずつ指定することができます。<min>と<max>のどちらか片方だけが指定された場合には、そのジョイントは部分的に制限されたものとして扱われます。sid および name 属性は、オプションです。	なし	0 または 1

詳細

`<limits>`要素によって設定される制限は、物理的な制限です。

例

以下は、並進自由度をもつジョイントの例です。

```
<joint id="Joint">
  <prismatic>
    <axis sid="axis">1 0 0</axis>
  </prismatic>
</joint>
```

revolute

カテゴリ:ジョイント

概要

ジョイントに単一の回転自由度を定義します。

コンセプト

必須要素`<axis>`は、対応する平行移動や回転の軸を定義します。この軸は、3次元空間中の任意の方向に、自由に指定することができます。つまり、`<axis>0 0 1</axis>`や`<axis>0.7071 0.7071 0</axis>`といった軸の定義ができるということです。

これらの要素では、オプションの`sid`属性を使って空間識別子を定義することができます。したがって、各プリミティブジョイントは、{ジョイント型のID}/{プリミティブのsid}によって参照できます。

属性

`<revolute>`要素には、以下の属性があります。

sid	sid_type	
		この要素のスコープ付き識別子を含むテキスト文字列。この文字列は、親要素の範囲内でユニークでなければなりません。オプション。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。

関連要素

`<revolute>`要素は、以下の要素と関連性があります。

親要素	<code>joint</code>
子要素	下記サブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<code><axis sid="" name="" /></code>	自由度の軸を指定する3つの浮動小数点値。sidおよびnameは、オプションです。	なし	1
<code><limits></code>	指定された軸を持つプリミティブジョイント型に制限	なし	0または1

名前/例	解説	デフォルト値	出現回数
<pre><min sid="..." name=""/> <max sid="..." name=""/> </limits></pre>	<p>を指定するには、静的な最小値や最大値を使った <code><limits></code> 要素をオプションで追加できます。<code><limits></code> が省略された場合、そのジョイントには制限がないものとして扱われます。</p> <p><code><min></code> および <code><max></code>: どちらもオプションで、浮動小数点数として1回ずつ指定することができます。<code><min></code> と <code><max></code> のどちらか片方だけが指定された場合には、そのジョイントは部分的に制限されたものとして扱われます。 <code>sid</code> および <code>name</code> 属性は、オプションです。</p>		

詳細

`<limits>` 要素によって設定される制限は、物理的な制限です。

例

以下は、回転自由度をもつジョイントの例です。

```
<joint id="Joint">
  <revolute>
    <axis sid="axis">1 0 0</axis>
  </revolute>
</joint>
```

このページは空白です。

11章 型

概要

この章には、これまでの章で参照した単純な型や値の型の一覧が記載されています。

単純な値の型

以下は、COLLADA スキーマで定義された、役に立ちそうな単純な型の一部です。

種類	解説
bool2_type	2 個の論理値が含まれます
bool3_type	3 個の論理値が含まれます
bool4_type	4 個の論理値が含まれます
float_type	XML Schema の定義による、オプションで指数のついた浮動小数点数。 <code>xs:double</code> に基づいています。
float2_type	2 個の浮動小数点数が含まれます。
float2x2_type	2x2 の行列を表す 4 個の浮動小数点数が含まれます
float2x3_type	2x3 の行列を表す 6 個の浮動小数点数が含まれます
float2x4_type	2x4 の行列を表す 8 個の浮動小数点数が含まれます
float3_type	3 個の浮動小数点数が含まれます。
float3x2_type	3x2 の行列を表す 6 個の浮動小数点数が含まれます
float3x3_type	3x3 の行列を表す 9 個の浮動小数点数が含まれます
float3x4_type	3x4 の行列を表す 12 個の浮動小数点数が含まれます
float4_type	4 個の浮動小数点数が含まれます。
float4x2_type	4x2 の行列を表す 8 個の浮動小数点数が含まれます
float4x3_type	4x3 の行列を表す 12 個の浮動小数点数が含まれます
float4x4_type	4x4 の行列を表す 16 個の浮動小数点数が含まれます
float7_type	7 個の浮動小数点数が含まれます。
int_type	XML Schema で記述された整数が含まれます。 <code>xs:long</code> に基づいています。
int2_type	2 個の整数が含まれます
int2x2_type	2x2 行列を表す 4 個の整数が含まれます
int3_type	3 個の整数が含まれます
int3x3_type	3x3 行列を表す 9 個の整数が含まれます
int4_type	4 個の整数が含まれます
int4x4_type	4x4 行列を表す 16 個の整数が含まれます
list_of_bools_type	論理値を含む <code>xs:list</code> 型
list_of_floats_type	浮動小数点数を含む <code>xs:list</code> 型
list_of_hexBinary_type	<code>xs:hexBinary</code> の数を含む <code>xs:list</code> 型
list_of_ints_type	整数を含む <code>xs:list</code> 型
list_of_uints_type	<code>uint_type</code> 数値を含む <code>xs:list</code> 型

パラメータの型要素

COLLADA の中では、異なるスコープは、異なるセットの強く型付けされたパラメータ型要素を持っています。たとえば、FX プロファイルによっても、有効な型要素のセットは異なります。

このセクションに記載された名前は要素名ですが、この要素名は、通常、前述の「単純型」セクションで説明した同じ名前の型を反映しています。この要素のコンテンツは、指定された型の値です。要素名の末尾の数は、要素の中に出現する値の数を示しています。以下に例を挙げます。

注：アスタリスク（「*」）のついた要素の詳しい説明は、前述のリファレンス章の主見出し項目を参照してください。

```
<bool>true</bool>
<float2>2.0 3.0</float2>
<int2x3> 1 2 3 4 5 6</int2x3>
```

GLSL パラメータ要素 (glsl_value_group)

bool, bool2, bool3, bool4, int, int2, int3, int4, float, float2, float3, float4, float2x2, float3x3, float4x4, sampler1D*, sampler2D*, sampler3D*, samplerCUBE*, samplerRECT*, samplerDEPTH*, enum, array*

CG パラメータ要素 (cg_param_group)

bool, bool2, bool3, bool4, bool2x1, bool2x2, bool2x3, bool2x4, bool3x1, bool3x2, bool3x3, bool3x4, bool4x1, bool4x2, bool4x3, bool4x4, int, int2, int3, int4, int2x1, int2x2, int2x3, int2x4, int3x1, int3x2, int3x3, int3x4, int4x1, int4x2, int4x3, int4x4, float, float2, float3, float4, float2x1, float2x2, float2x3, float2x4, float3x1, float3x2, float3x3, float3x4, float4x1, float4x2, float4x3, float4x4, half, half2, half3, half4, half2x1, half2x2, half2x3, half2x4, half3x1, half3x2, half3x3, half3x4, half4x1, half4x2, half4x3, half4x4, fixed, fixed2, fixed3, fixed4, fixed1x1, fixed2x1, fixed2x2, fixed2x3, fixed2x4, fixed3x1, fixed3x2, fixed3x3, fixed3x4, fixed4x1, fixed4x2, fixed4x3, fixed4x4, sampler1D*, sampler2D*, sampler3D*, samplerCUBE*, samplerRECT*, samplerDEPTH*, enum, string, array*, usertype*

GLSL パラメータ要素 (gles_param_group)

bool, bool2, bool3, bool4, int, int2, int3, int4, float, float2, float3, float4, float1x1, float1x2, float1x3, float1x4, float2x1, float2x2, float2x3, float2x4, float3x1, float3x2, float3x3, float3x4, float4x1, float4x2, float4x3, float4x4, sampler2D*, enum

GLSL2 パラメータ要素 (gles2_value_group)

bool, bvec2, bvec3, bvec4, int, ivec2, ivec3, ivec4, float, vec2, vec3, vec4, mat2, mat3, mat4, sampler2D*, sampler3D*, samplerCUBE*, samplerDEPTH*, array*, usertype*

Effect パラメータ要素 (fx_newparam_group)

bool, bool2, bool3, bool4, int, int2, int3, int4, float, float2, float3, float4, float2x1, float2x2, float2x3, float2x4, float3x1, float3x2, float3x3, float3x4, float4x1, float4x2, float4x3, float4x4, sampler1D*, sampler2D*, sampler3D*, samplerCUBE*, samplerRECT*, samplerDEPTH*, enum

Instance_Effect パラメータ要素 (fx_setparam_group)

bool, bool2, bool3, bool4, int, int2, int3, int4, float, float2, float3, float4, float2x1, float2x2, float2x3, float2x4, float3x1, float3x2, float3x3, float3x4, float4x1, float4x2, float4x3, float4x4, enum, sampler_image*, sampler_states*

他の単純型

以下の表に記載されているのは、COLLADA スキーマの中で定義されたそれ以外の単純型の一部です。xs:に基づく型の定義および語彙の表現については、XML Schema (<http://www.w3.org/TR/xmlschema-2/#built-in-datatypes>) を参照してください。

種類	解説	定義
<code>list_of_names_type</code>		<code>xs:list itemType="xs:Name"</code>
<code>list_of_tokens_type</code>		<code>xs:list itemType="xs:token"</code>
<code>sid_type</code>	COLLADA スコープ付き識別子が含まれます。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。	<code>xs:extension base="xs:NCName"</code>
<code>sidref_type</code>	COLLADA スコープ付き識別子への参照。詳しくは、3章「スキーマのコンセプト」の「アドレス構文」を参照してください。	<code>xs:extension base="xs:token"</code>
<code>urifragment_type</code>	この型は、同じ文書の中で宣言されたリソースだけを参照できる URI 参照に使われます。有効な値は、以下の通りです。 <code>xs:pattern value="(#[.*)"</code>	<code>xs:restriction base="xs:string"</code>

値パラメータ選択型

COLLADA の要素の中には、特定の要素と、パラメータの型の要素とのいずれかを選択できるものが複数あります。

- `common_sidref_or_param_type`
- `common_float_or_param_type`
- `common_float2_or_param_type`
- `common_int_or_param_type`
- `common_bool_or_param_type`

これらはすべて、基本的に同じフォーマットに従います。

注: 以下の子要素のいずれか 1 つだけが出現する必要があります。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<code>type_element_name</code>	これは通常、値・パラメータ選択型の型名に示された型に一致します。たとえば、 <code>common_bool_or_param_type</code> の場合、有効な要素は <code><bool></code> だけです。	なし	「注」を参照してください。
<code><param></code> (reference)	主見出し項目を参照してください。	なし	「注」を参照してください。

以下については、8章 FX リファレンスの詳しいエントリを参照してください。

- `fx_common_color_or_texture_type`
- `fx_common_float_or_param_type`

このページは空白です。

付録 A : COLLADA のサンプル

例 : キューブ

この付録では、単純な白いキューブを記述する COLLADA インスタンス文書の簡単な例を示します。

```
<?xml version="1.0" encoding="utf-8"?>
<COLLADA xmlns="http://www.collada.org/2008/03/COLLADASchema" version="1.5.0">
  <asset>
    <created>2005-11-14T02:16:38Z</created>
    <modified>2005-11-15T11:36:38Z</modified>
    <revision>1.0</revision>
  </asset>
  <library_effects>
    <effect id="whitePhong">
      <profile_COMMON>
        <technique sid="phong1">
          <phong>
            <emission>
              <color>1.0 1.0 1.0 1.0</color>
            </emission>
            <ambient>
              <color>1.0 1.0 1.0 1.0</color>
            </ambient>
            <diffuse>
              <color>1.0 1.0 1.0 1.0</color>
            </diffuse>
            <specular>
              <color>1.0 1.0 1.0 1.0</color>
            </specular>
            <shininess>
              <float>20.0</float>
            </shininess>
            <reflective>
              <color>1.0 1.0 1.0 1.0</color>
            </reflective>
            <reflectivity>
              <float>0.5</float>
            </reflectivity>
            <transparent>
              <color>1.0 1.0 1.0 1.0</color>
            </transparent>
            <transparency>
              <float>1.0</float>
            </transparency>
          </phong>
        </technique>
      </profile_COMMON>
    </effect>
  </library_effects>
  <library_materials>
    <material id="whiteMaterial">
      <instance_effect url="#whitePhong" />
    </material>
  </library_materials>
  <library_geometries>
```

```

<geometry id="box" name="box">
  <mesh>
    <source id="box-Pos">
      <float_array id="box-Pos-array" count="24">
        -0.5 0.5 0.5
        0.5 0.5 0.5
        -0.5 -0.5 0.5
        0.5 -0.5 0.5
        -0.5 0.5 -0.5
        0.5 0.5 -0.5
        -0.5 -0.5 -0.5
        0.5 -0.5 -0.5
      </float_array>
      <technique_common>
        <accessor source="#box-Pos-array" count="8" stride="3">
          <param name="X" type="float" />
          <param name="Y" type="float" />
          <param name="Z" type="float" />
        </accessor>
      </technique_common>
    </source>
    <source id="box-0-Normal">
      <float_array id="box-0-Normal-array" count="18">
        1.0 0.0 0.0
        -1.0 0.0 0.0
        0.0 1.0 0.0
        0.0 -1.0 0.0
        0.0 0.0 1.0
        0.0 0.0 -1.0
      </float_array>
      <technique_common>
        <accessor source="#box-0-Normal-array" count="6" stride="3">
          <param name="X" type="float" />
          <param name="Y" type="float" />
          <param name="Z" type="float" />
        </accessor>
      </technique_common>
    </source>
    <vertices id="box-Vtx">
      <input semantic="POSITION" source="#box-Pos" />
    </vertices>
    <polygons count="6" material="WHITE">
      <input semantic="VERTEX" source="#box-Vtx" offset="0" />
      <input semantic="NORMAL" source="#box-0-Normal" offset="1" />
      <p>0 4 2 4 3 4 1 4</p>
      <p>0 2 1 2 5 2 4 2</p>
      <p>6 3 7 3 3 3 2 3</p>
      <p>0 1 4 1 6 1 2 1</p>
      <p>3 0 7 0 5 0 1 0</p>
      <p>5 5 7 5 6 5 4 5</p>
    </polygons>
  </mesh>
</geometry>
</library_geometries>
<library_visual_scenes>
  <visual_scene id="DefaultScene">
    <node id="Box" name="Box">
      <translate> 0 0 0</translate>
      <rotate> 0 0 1 0</rotate>
      <rotate> 0 1 0 0</rotate>
    </node>
  </visual_scene>
</library_visual_scenes>

```

```
<rotate> 1 0 0 0</rotate>
<scale> 1 1 1</scale>
<instance_geometry url="#box">
  <bind_material>
    <technique_common>
      <instance_material symbol="WHITE" target="#whiteMaterial"/>
    </technique_common>
  </bind_material>
</instance_geometry>
</node>
</visual_scene>
</library_visual_scenes>
<scene>
  <instance_visual_scene url="#DefaultScene"/>
</scene>
</COLLADA>
```

このページは空白です。

付録 B : プロファイル GLSL および GLES2 のサンプル

例 : <profile_GLSL>

これは、<profile_GLSL>を使った COLLADA インスタンス文書の簡単な例です。

```
<?xml version="1.0"?>
<COLLADA xmlns="http://www.collada.org/2008/03/COLLADASchema" version="1.5.0">
  <asset>
    <contributor>
      <author></author>
      <authoring_tool>RenderMonkey</authoring_tool>
      <comments>Output from RenderMonkey COLLADA Exporter</comments>
      <copyright></copyright>
      <source_data></source_data>
    </contributor>
    <created>2008-03-27T20:31:16Z</created>
    <modified>2008-03-27T20:31:16Z</modified>
    <unit meter="0.01" name="centimeter"></unit>
    <up_axis>Y_UP</up_axis>
  </asset>
  <library_visual_scenes>
    <visual_scene id="VisualSceneNode" name="untitled">
      <node id="Model_E0_MESH_0_REF_1" name="Model_E0_MESH_0_REF_1">
        <instance_geometry url="#Model_E0_MESH_0_REF_1_lib">
          <bind_material>
            <technique_common>
              <instance_material symbol="Textured_Bump_E0_MP_MAT"
target="#Textured_Bump_E0_MP_MAT">
                <bind_vertex_input semantic="rm_Binormal"
input_semantic="BINORMAL"></bind_vertex_input>
                <bind_vertex_input semantic="rm_Tangent"
input_semantic="TANGENT"></bind_vertex_input>
              </instance_material>
            </technique_common>
          </bind_material>
        </instance_geometry>
      </node>
    </visual_scene>
  </library_visual_scenes>
  <library_materials>
    <material id="Textured_Bump_E0_MP_MAT" name="Textured_Bump_E0_MP_MAT">
      <instance_effect url="#Textured_Bump_E0_MP_FX">
        <technique_hint platform="PC-OGL" profile="GLES2"
ref="Textured_Bump_E0_MP_TECH"></technique_hint>
        <setparam ref="fSpecularPower_E0_P0">
          <float>25</float>
        </setparam>
        <setparam ref="fvAmbient_E0_P0">
          <float4> 0.368627 0.368421 0.368421 1</float4>
        </setparam>
        <setparam ref="fvDiffuse_E0_P0">
          <float4>0.886275 0.885003 0.885003 1</float4>
        </setparam>
      </instance_effect>
    </material>
  </library_materials>
</COLLADA>
```

```

        <setparam ref="fvEyePosition_E0_P0">
            <float3>0 0 100</float3>
        </setparam>
        <setparam ref="fvLightPosition_E0_P0">
            <float3>-100 100 100</float3>
        </setparam>
        <setparam ref="fvSpecular_E0_P0">
            <float4> 0.490196 0.488722 0.488722 1</float4>
        </setparam>
    </instance_effect>
</material>
</library_materials>
<library_effects>
    <effect id="Textured_Bump_E0_MP_FX">
        <profile_GLSL>
            <code sid="Vertex_Program_E0_P0_VP">uniform vec3 fvLightPosition;
uniform vec3 fvEyePosition;

varying vec2 Texcoord;
varying vec3 ViewDirection;
varying vec3 LightDirection;

attribute vec3 rm_Binormal;
attribute vec3 rm_Tangent;

void main (void)
{
    gl_Position = ftransform();
    Texcoord = gl_MultiTexCoord0.xy;

    vec4 fvObjectPosition = gl_ModelViewMatrix * gl_Vertex;

    vec3 fvViewDirection = fvEyePosition - fvObjectPosition.xyz;
    vec3 fvLightDirection = fvLightPosition - fvObjectPosition.xyz;

    vec3 fvNormal = gl_NormalMatrix * gl_Normal;
    vec3 fvBinormal = gl_NormalMatrix * rm_Binormal;
    vec3 fvTangent = gl_NormalMatrix * rm_Tangent;

    ViewDirection.x = dot( fvTangent, fvViewDirection );
    ViewDirection.y = dot( fvBinormal, fvViewDirection );
    ViewDirection.z = dot( fvNormal, fvViewDirection );

    LightDirection.x = dot( fvTangent, fvLightDirection.xyz );
    LightDirection.y = dot( fvBinormal, fvLightDirection.xyz );
    LightDirection.z = dot( fvNormal, fvLightDirection.xyz );

}
</code>
            <code sid="Fragment_Program_E0_P0_FP">uniform vec4 fvAmbient;
uniform vec4 fvSpecular;
uniform vec4 fvDiffuse;
uniform float fSpecularPower;

uniform sampler2D baseMap;
uniform sampler2D bumpMap;

varying vec2 Texcoord;
varying vec3 ViewDirection;
varying vec3 LightDirection;

```

```

void main (void)
{
    vec3 fvLightDirection = normalize( LightDirection );
    vec3 fvNormal = normalize( ( texture2D( bumpMap, Texcoord ).xyz * 2.0 ) - 1.0 );
    float fNDotL = dot( fvNormal, fvLightDirection );

    vec3 fvReflection = normalize( ( ( 2.0 * fvNormal ) * fNDotL ) - fvLightDirection );
    vec3 fvViewDirection = normalize( ViewDirection );
    float fRDotV = max( 0.0, dot( fvReflection, fvViewDirection ) );

    vec4 fvBaseColor = texture2D( baseMap, Texcoord );

    vec4 fvTotalAmbient = fvAmbient * fvBaseColor;
    vec4 fvTotalDiffuse = fvDiffuse * fNDotL * fvBaseColor;
    vec4 fvTotalSpecular = fvSpecular * ( pow( fRDotV, fSpecularPower ) );

    gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
}
</code>
    <newparam sid="fSpecularPower_E0_P0">
        <float>25</float>
</newparam>
    <newparam sid="fvAmbient_E0_P0">
        <float4> 0.368627 0.368421 0.368421 1</float4>
</newparam>
    <newparam sid="fvDiffuse_E0_P0">
        <float4> 0.886275 0.885003 0.885003 1</float4>
</newparam>
    <newparam sid="fvEyePosition_E0_P0">
        <float3>0 0 100</float3>
</newparam>
    <newparam sid="fvLightPosition_E0_P0">
        <float3>-100 100 100</float3>
</newparam>
    <newparam sid="fvSpecular_E0_P0">
        <float4> 0.490196 0.488722 0.488722 1</float4>
</newparam>
    <newparam sid="baseMap_Sampler">
        <sampler2D>
            <instance_image url="base"></instance_image>
            <minfilter>LINEAR</minfilter>
            <magfilter>LINEAR</magfilter>
            <mipfilter>LINEAR</mipfilter>
        </sampler2D>
</newparam>
    <newparam sid="bumpMap_Sampler">
        <sampler2D>
            <instance_image url="bump"></instance_image>
            <minfilter>LINEAR</minfilter>
            <magfilter>LINEAR</magfilter>
            <mipfilter>LINEAR</mipfilter>
        </sampler2D>
</newparam>
    <technique sid="Textured_Bump_E0_MP_TECH">
        <pass sid="Pass_0">
            <program>
                <shader stage="VERTEX">
                    <sources>
                        <import ref="Vertex_Program_E0_P0_VP"></import>
                    </sources>

```

```

    </shader>
    <shader stage="FRAGMENT">
      <sources>
        <import ref="Fragment_Program_E0_P0_FP"></import>
      </sources>
    </shader>
    <bind_uniform symbol="fSpecularPower">
      <param ref="fSpecularPower_E0_P0"></param>
    </bind_uniform>
    <bind_uniform symbol="fvAmbient">
      <param ref="fvAmbient_E0_P0"></param>
    </bind_uniform>
    <bind_uniform symbol="fvDiffuse">
      <param ref="fvDiffuse_E0_P0"></param>
    </bind_uniform>
    <bind_uniform symbol="fvEyePosition">
      <param ref="fvEyePosition_E0_P0"></param>
    </bind_uniform>
    <bind_uniform symbol="fvLightPosition">
      <param ref="fvLightPosition_E0_P0"></param>
    </bind_uniform>
    <bind_uniform symbol="fvSpecular">
      <param ref="fvSpecular_E0_P0"></param>
    </bind_uniform>
    <bind_uniform symbol="baseMap">
      <param ref="baseMap_Sampler"></param>
    </bind_uniform>
    <bind_uniform symbol="bumpMap">
      <param ref="bumpMap_Sampler"></param>
    </bind_uniform>
  </program>
</pass>
</technique>
</profile_GLSL>
<extra>
  <technique profile="RenderMonkey">
    <RenderMonkey_TimeCycle>
      <param type="float">120.000000</param>
    </RenderMonkey_TimeCycle>
  </technique>
</extra>
</effect>
</library_effects>
<library_images>
  <image id="base" name="base">
    <init_from >
      <ref>./Textured_Bump_GLSL/Fieldstone.tga</ref>
    </init_from>
  </image>
  <image id="bump" name="bump">
    <init_from >
      <ref>./Textured_Bump_GLSL/FieldstoneBumpDOT3.tga</ref>
    </init_from>
  </image>
</library_images>
<library_geometries>
  <geometry id="Model_E0_MESH_0_REF_1_lib" name="Model_E0_MESH_0_REF_1">
<mesh>
  <source id="Model_E0_MESH_0_REF_1_lib_positions" name="position">

```

```

    <float_array id="Model_E0_MESH_0_REF_1_lib_positions_array"
count="9">-50 -50 0 0 50 0 50 -50 0</float_array>
    <technique_common>
        <accessor count="3"
source="#Model_E0_MESH_0_REF_1_lib_positions_array" stride="3">
            <param name="X" type="float"></param>
            <param name="Y" type="float"></param>
            <param name="Z" type="float"></param>
        </accessor>
    </technique_common>
</source>
    <source id="Model_E0_MESH_0_REF_1_lib_normals" name="normal">
        <float_array id="Model_E0_MESH_0_REF_1_lib_normals_array" count="9">0 0
-1 0 0 -1 0 0 -1</float_array>
        <technique_common>
            <accessor count="3" source="#Model_E0_MESH_0_REF_1_lib_normals_array"
stride="3">
                <param name="X" type="float"></param>
                <param name="Y" type="float"></param>
                <param name="Z" type="float"></param>
            </accessor>
        </technique_common>
    </source>
    <source id="Model_E0_MESH_0_REF_1_lib_texcoords" name="texcoords">
        <float_array id="Model_E0_MESH_0_REF_1_lib_texcoords_array" count="6">0
0 0.5 1 1 0</float_array>
        <technique_common>
            <accessor count="3"
source="#Model_E0_MESH_0_REF_1_lib_texcoords_array" stride="2">
                <param name="X" type="float"></param>
                <param name="Y" type="float"></param>
            </accessor>
        </technique_common>
    </source>
    <source id="Model_E0_MESH_0_REF_1_lib_tangents" name="tangent">
        <float_array id="Model_E0_MESH_0_REF_1_lib_tangents_array" count="9">1 0
0 1 0 0 1 0 0</float_array>
        <technique_common>
            <accessor count="3"
source="#Model_E0_MESH_0_REF_1_lib_tangents_array" stride="3">
                <param name="X" type="float"></param>
                <param name="Y" type="float"></param>
                <param name="Z" type="float"></param>
            </accessor>
        </technique_common>
    </source>
    <source id="Model_E0_MESH_0_REF_1_lib_binormals" name="binormal">
        <float_array id="Model_E0_MESH_0_REF_1_lib_binormals_array" count="9">0
1 0 0 0 0 1 0</float_array>
        <technique_common>
            <accessor count="3"
source="#Model_E0_MESH_0_REF_1_lib_binormals_array" stride="3">
                <param name="X" type="float"></param>
                <param name="Y" type="float"></param>
                <param name="Z" type="float"></param>
            </accessor>
        </technique_common>
    </source>
    <vertices id="Model_E0_MESH_0_REF_1_lib_vertices">
        <input semantic="POSITION"
source="#Model_E0_MESH_0_REF_1_lib_positions"></input>

```

```

        <input semantic="NORMAL"
source="#Model_E0_MESH_0_REF_1_lib_normals"></input>
        <input semantic="TEXCOORD"
source="#Model_E0_MESH_0_REF_1_lib_texcoords"></input>
        <input semantic="TANGENT"
source="#Model_E0_MESH_0_REF_1_lib_tangents"></input>
        <input semantic="BINORMAL"
source="#Model_E0_MESH_0_REF_1_lib_binormals"></input>
    </vertices>
    <triangles count="1" material="Textured_Bump_E0_MP_MAT">
        <input offset="0" semantic="VERTEX"
source="#Model_E0_MESH_0_REF_1_lib_vertices"></input>
        <p>0 1 2</p>
    </triangles>
</mesh>
</geometry>
</library_geometries>
<scene>
    <instance_visual_scene url="#VisualSceneNode"></instance_visual_scene>
</scene>
</COLLADA>

```

例 : <profile_GLES2>

これは、<profile_GLES2>を使った COLLADA インスタンス文書の簡単な例です。

```

<?xml version="1.0" encoding="UTF-8"?>
<COLLADA xmlns="http://www.collada.org/2008/03/COLLADASchema" version="1.5.0">
  <asset>
    <contributor>
      <author></author>
      <authoring_tool>RenderMonkey</authoring_tool>
      <comments>Output from RenderMonkey COLLADA Exporter</comments>
      <copyright></copyright>
      <source_data></source_data>
    </contributor>
    <created>2008-03-27T20:31:07Z</created>
    <modified>2008-03-27T20:31:07Z</modified>
    <unit meter="0.01" name="centimeter"></unit>
    <up_axis>Y_UP</up_axis>
  </asset>
  <library_visual_scenes>
    <visual_scene id="VisualSceneNode" name="untitled">
      <node id="Model_E0_MESH_0_REF_1" name="Model_E0_MESH_0_REF_1">
        <instance_geometry url="#Model_E0_MESH_0_REF_1_lib">
          <bind_material>
            <technique_common>
              <instance_material symbol="Textured_Bump_E0_MP_MAT"
target="#Textured_Bump_E0_MP_MAT">
                <bind_vertex_input semantic="rm_Binormal"
input_semantic="BINORMAL"></bind_vertex_input>
                <bind_vertex_input semantic="rm_Tangent"
input_semantic="TANGENT"></bind_vertex_input>
            </instance_material>
          </technique_common>
        </bind_material>
      </instance_geometry>
    </node>
  </library_visual_scenes>

```

```

</visual_scene>
</library_visual_scenes>
<library_materials>
  <material id="Textured_Bump_E0_MP_MAT" name="Textured_Bump_E0_MP_MAT">
    <instance_effect url="#Textured_Bump_E0_MP_FX">
      <technique_hint platform="PC-OpenGL" profile="GLES2"
ref="Textured_Bump_E0_MP_TECH"></technique_hint>
      <setparam ref="fSpecularPower_E0_P0">
        <float>25</float>
      </setparam>
      <setparam ref="fvAmbient_E0_P0">
<float4> 0.368627 0.368421 0.368421 1</float4>
      </setparam>
      <setparam ref="fvDiffuse_E0_P0">
<float4> 0.886275 0.885003 0.885003 1</float4>
      </setparam>
      <setparam ref="fvEyePosition_E0_P0">
        <float3>0 0 100</float3>
      </setparam>
      <setparam ref="fvLightPosition_E0_P0">
        <float3>-100 100 100</float3>
      </setparam>
      <setparam ref="fvSpecular_E0_P0">
<float4> 0.490196 0.488722 0.488722 1</float4>
      </setparam>
      <setparam ref="matViewProjection_E0_P0">
        <float4x4>-2.22782 -0.0171533 0.0525642 1.05927e-007 -0.0458611 2.04965
-1.27486 4.56546e-005 0.0159767 0.528878 0.849727 199.199 0.0159607 0.528349
0.848877 200</float4x4>
      </setparam>
      <setparam ref="matViewProjectionInverseTranspose_E0_P0">
        <float4x4>-0.448593 -0.00345406 0.0105843 1.11448e-010 -0.00786852
0.351669 -0.218728 -1.53279e-008 3.18896 105.564 169.606 -0.999002 -3.17619 -105.142
-168.927 1</float4x4>
      </setparam>
    </instance_effect>
  </material>
</library_materials>
<library_effects>
  <effect id="Textured_Bump_E0_MP_FX">
    <profile_GLES2 language="">
      <code sid="
Vertex_Program_E0_P0_VP">uniform mat4 matViewProjectionInverseTranspose;
uniform mat4 matViewProjection;
uniform vec3 fvLightPosition;
uniform vec3 fvEyePosition;

varying vec2 Texcoord;
varying vec3 ViewDirection;
varying vec3 LightDirection;

attribute vec4 rm_Vertex;
attribute vec4 rm_TexCoord0;
attribute vec4 rm_Normal;
attribute vec4 rm_Binormal;
attribute vec4 rm_Tangent;

void main (void)
{
  gl_Position = matViewProjection * rm_Vertex;

```

```

Texcoord = rm_TexCoord0.xy;

vec4 fvObjectPosition = matViewProjection * rm_Vertex;

vec3 fvViewDirection = fvEyePosition - fvObjectPosition.xyz;
vec3 fvLightDirection = fvLightPosition - fvObjectPosition.xyz;

vec3 fvNormal = (matViewProjectionInverseTranspose * rm_Normal).xyz;
vec3 fvBinormal = (matViewProjectionInverseTranspose * rm_Binormal).xyz;
vec3 fvTangent = (matViewProjectionInverseTranspose * rm_Tangent).xyz;

ViewDirection.x = dot( fvTangent, fvViewDirection );
ViewDirection.y = dot( fvBinormal, fvViewDirection );
ViewDirection.z = dot( fvNormal, fvViewDirection );

LightDirection.x = dot( fvTangent, fvLightDirection.xyz );
LightDirection.y = dot( fvBinormal, fvLightDirection.xyz );
LightDirection.z = dot( fvNormal, fvLightDirection.xyz );

}</code>
  <code sid="
Fragment_Program_E0_P0_FP">#ifdef GL_FRAGMENT_PRECISION_HIGH
  // デフォルトの精度
  precision highp float;
#else
  precision mediump float;
#endif

uniform vec4 fvAmbient;
uniform vec4 fvSpecular;
uniform vec4 fvDiffuse;
uniform float fSpecularPower;

uniform sampler2D baseMap;
uniform sampler2D bumpMap;

varying vec2 Texcoord;
varying vec3 ViewDirection;
varying vec3 LightDirection;

void main (void)
{
  vec3 fvLightDirection = normalize( LightDirection );
  vec3 fvNormal = normalize( ( texture2D( bumpMap, Texcoord ).xyz * 2.0 ) - 1.0 );
  float fNDotL = dot( fvNormal, fvLightDirection );

  vec3 fvReflection = normalize( ( ( 2.0 * fvNormal ) * fNDotL ) - fvLightDirection );
  vec3 fvViewDirection = normalize( ViewDirection );
  float fRDotV = max( 0.0, dot( fvReflection, fvViewDirection ) );

  vec4 fvBaseColor = texture2D( baseMap, Texcoord );

  vec4 fvTotalAmbient = fvAmbient * fvBaseColor;
  vec4 fvTotalDiffuse = fvDiffuse * fNDotL * fvBaseColor;
  vec4 fvTotalSpecular = fvSpecular * ( pow( fRDotV, fSpecularPower ) );

  gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
}</code>
  <newparam sid="fSpecularPower_E0_P0">

```



```

        <float>25</float>
</newparam>
    <newparam sid="fvAmbient_E0_P0">
        <vec4>0.368627 0.368421 0.368421 1</vec4>
</newparam>
    <newparam sid="fvDiffuse_E0_P0">
        <vec4>0.886275 0.885003 0.885003 1</vec4>
</newparam>
    <newparam sid="fvEyePosition_E0_P0">
        <vec3>0 0 100</vec3>
</newparam>
    <newparam sid="fvLightPosition_E0_P0">
        <vec3>-100 100 100</vec3>
</newparam>
    <newparam sid="fvSpecular_E0_P0">
        <vec4>0.490196 0.488722 0.488722 1</vec4>
</newparam>
    <newparam sid="matViewProjection_E0_P0">
        <semantic>ViewProjection</semantic>
        <mat4>-2.22782 -0.0458611 0.0159767 0.0159607 -0.0171533 2.04965 0.528878
0.528349 0.0525642 -1.27486 0.849727 0.848877 1.05927e-007 4.56546e-005 199.199
200</mat4>
</newparam>
    <newparam sid="matViewProjectionInverseTranspose_E0_P0">
        <semantic>ViewProjectionInverseTranspose</semantic>
        <mat4>-0.448593 -0.00786852 3.18896 -3.17619 -0.00345406 0.351669 105.564
-105.142 0.0105843 -0.218728 169.606 -168.927 1.11448e-010 -1.53279e-008 -0.999002
1</mat4>
</newparam>
    <newparam sid="baseMap_Sampler">
        <sampler2D>
            <instance_image url="base"></instance_image>
            <minfilter>LINEAR</minfilter>
            <magfilter>LINEAR</magfilter>
            <mipfilter>LINEAR</mipfilter>
        </sampler2D>
</newparam>
    <newparam sid="bumpMap_Sampler">
        <sampler2D>
            <instance_image url="bump"></instance_image>
            <minfilter>LINEAR</minfilter>
            <magfilter>LINEAR</magfilter>
            <mipfilter>LINEAR</mipfilter>
        </sampler2D>
</newparam>
    <technique sid="Textured_Bump_E0_MP_TECH">
        <pass sid="Pass_0">
            <program>
                <shader stage="VERTEX">
                    <sources>
                        <import ref="Vertex_Program_E0_P0_VP"></import>
                    </sources>
                </shader>
                <shader stage="FRAGMENT">
                    <sources>
                        <import ref="Fragment_Program_E0_P0_FP"></import>
                    </sources>
                </shader>
                <bind_uniform symbol="fSpecularPower">
                    <param ref="fSpecularPower_E0_P0"></param>

```

```

        </bind_uniform>
        <bind_uniform symbol="fvAmbient">
            <param ref="fvAmbient_E0_P0"></param>
        </bind_uniform>
        <bind_uniform symbol="fvDiffuse">
            <param ref="fvDiffuse_E0_P0"></param>
        </bind_uniform>
        <bind_uniform symbol="fvEyePosition">
            <param ref="fvEyePosition_E0_P0"></param>
        </bind_uniform>
        <bind_uniform symbol="fvLightPosition">
            <param ref="fvLightPosition_E0_P0"></param>
        </bind_uniform>
        <bind_uniform symbol="fvSpecular">
            <param ref="fvSpecular_E0_P0"></param>
        </bind_uniform>
        <bind_uniform symbol="matViewProjection">
            <param ref="matViewProjection_E0_P0"></param>
        </bind_uniform>
        <bind_uniform symbol="matViewProjectionInverseTranspose">
            <param ref="matViewProjectionInverseTranspose_E0_P0"></param>
        </bind_uniform>
        <bind_uniform symbol="baseMap">
            <param ref="baseMap_Sampler"></param>
        </bind_uniform>
        <bind_uniform symbol="bumpMap">
            <param ref="bumpMap_Sampler"></param>
        </bind_uniform>
    </program>
</pass>
</technique>
</profile_GLES>
<extra>
    <technique profile="RenderMonkey">
        <RenderMonkey_TimeCycle>
            <param type="float">120.000000</param>
        </RenderMonkey_TimeCycle>
    </technique>
</extra>
</effect>
</library_effects>
<library_images>
    <image id="base" name="base">
        <init_from>
            <ref>./Textured_Bump_GLES2/Fieldstone.tga</ref>
        </init_from>
    </image>
    <image id="bump" name="bump">
        <init_from >
            <ref>./Textured_Bump_GLES2/FieldstoneBumpDOT3.tga</ref>
        </init_from>
    </image>
</library_images>
<library_geometries>
    <geometry id="Model_E0_MESH_0_REF_1_lib" name="Model_E0_MESH_0_REF_1">
<mesh>
    <source id="Model_E0_MESH_0_REF_1_lib_positions" name="position">
        <float_array id="Model_E0_MESH_0_REF_1_lib_positions_array"
count="9">-50 -50 0 0 50 0 50 -50 0</float_array>
        <technique_common>

```

```

        <accessor count="3"
source="#Model_E0_MESH_0_REF_1_lib_positions_array" stride="3">
        <param name="X" type="float"></param>
        <param name="Y" type="float"></param>
        <param name="Z" type="float"></param>
    </accessor>
    </technique_common>
</source>
<source id="Model_E0_MESH_0_REF_1_lib_normals" name="normal">
    <float_array id="Model_E0_MESH_0_REF_1_lib_normals_array" count="9">0 0
-1 0 0 -1 0 0 -1</float_array>
    <technique_common>
        <accessor count="3" source="#Model_E0_MESH_0_REF_1_lib_normals_array"
stride="3">
            <param name="X" type="float"></param>
            <param name="Y" type="float"></param>
            <param name="Z" type="float"></param>
        </accessor>
    </technique_common>
</source>
<source id="Model_E0_MESH_0_REF_1_lib_texcoords" name="texcoords">
    <float_array id="Model_E0_MESH_0_REF_1_lib_texcoords_array" count="6">0
0 0.5 1 1 0</float_array>
    <technique_common>
        <accessor count="3"
source="#Model_E0_MESH_0_REF_1_lib_texcoords_array" stride="2">
            <param name="X" type="float"></param>
            <param name="Y" type="float"></param>
        </accessor>
    </technique_common>
</source>
<source id="Model_E0_MESH_0_REF_1_lib_tangents" name="tangent">
    <float_array id="Model_E0_MESH_0_REF_1_lib_tangents_array" count="9">1 0
0 1 0 0 1 0 0</float_array>
    <technique_common>
        <accessor count="3"
source="#Model_E0_MESH_0_REF_1_lib_tangents_array" stride="3">
            <param name="X" type="float"></param>
            <param name="Y" type="float"></param>
            <param name="Z" type="float"></param>
        </accessor>
    </technique_common>
</source>
<source id="Model_E0_MESH_0_REF_1_lib_binormals" name="binormal">
    <float_array id="Model_E0_MESH_0_REF_1_lib_binormals_array" count="9">0
1 0 0 0 0 1 0</float_array>
    <technique_common>
        <accessor count="3"
source="#Model_E0_MESH_0_REF_1_lib_binormals_array" stride="3">
            <param name="X" type="float"></param>
            <param name="Y" type="float"></param>
            <param name="Z" type="float"></param>
        </accessor>
    </technique_common>
</source>
<vertices id="Model_E0_MESH_0_REF_1_lib_vertices">
    <input semantic="POSITION"
source="#Model_E0_MESH_0_REF_1_lib_positions"></input>
    <input semantic="NORMAL"
source="#Model_E0_MESH_0_REF_1_lib_normals"></input>

```

```
        <input semantic="TEXCOORD"
source="#Model_E0_MESH_0_REF_1_lib_texcoords"></input>
        <input semantic="TANGENT"
source="#Model_E0_MESH_0_REF_1_lib_tangents"></input>
        <input semantic="BINORMAL"
source="#Model_E0_MESH_0_REF_1_lib_binormals"></input>
    </vertices>
    <triangles count="1" material="Textured_Bump_E0_MP_MAT">
        <input offset="0" semantic="VERTEX"
source="#Model_E0_MESH_0_REF_1_lib_vertices"></input>
        <p>0 1 2</p>
    </triangles>
</mesh>
</geometry>
</library_geometries>
<scene>
    <instance_visual_scene url="#VisualSceneNode"></instance_visual_scene>
</scene>
</COLLADA>
```

用語集

アニメーション曲線 - キーフレームの集合、およびその間の補間によって定義される 2 次元の関数。

孤 - ノードの間をつないだもの。

バックバッファ - ダブルバッファシステムにおいて、コンピュータが通常レンダリングを行うビューポートバッファ。

属性 - XML 要素は、0 個以上属性を持つことができます。属性は、開始タグの中のタグ名の直後に指定します。各属性は、名前と値の対によって構成されています。属性の値の部分は、かならず引用符 (") で囲みます。属性は、バインドされた要素についてのセマンティック情報を提供します。以下に例を示します。

```
<tagName attribute="value">
```

COLLADA - *Collaborative Design Activity* (共同設計活動) の略。

COLLADA 文書 - 特定のデジタルアセットを記述する COLLADA XML 要素を含むファイル。

COLLADA スキーマ - 有効な COLLADA 要素のすべてを定義する XML Schema 文書。

コメント - XML ファイルはコメント文字列を含むことができます。コメントは、以下に示す特殊な形のマークアップで識別されます。

```
<!-- これはXMLのコメントです -->
```

CV - Control vertex (制御頂点) の略。スプライン曲線の制御点。

DAE (.dae) - Digital Asset Exchange (デジタルアセット交換) の略で、COLLADA がデジタルアセット (COLLADA 文書) に関する情報を格納するフォーマットのこと。

DCC - デジタルコンテンツ作成。

エフェクトスコープ - `<effect>` 要素の中にあり、なおかつ特定の `<profile_*>` 要素の中にはない宣言空間。

要素 - XML 文書は、主に要素から構成されています。要素は、その前後をタグに囲まれた情報のブロックです。要素は、必要に応じて入れ子 (ネスト) にして、階層的なデータセットを作成することもできます。

関数曲線 - アニメーション曲線と同じ。

フラスタム - 視野錐台を参照。

FX ランタイム - シェーダ、ソースコード、パラメータ、その他のエフェクトプロパティの作成、使用、管理を扱うコードの基礎になるライブラリ。

HDR - High dynamic range (高ダイナミックレンジ) の略。

id - URI の一部として参照できる、インスタンス文書内で一意な要素の識別子。詳しくは、3 章「スキーマのコンセプト」の「アドレス構文」を参照してください。

IDREF - *id* への参照。詳しくは、3 章「スキーマのコンセプト」の「アドレス構文」を参照してください。

インスタンス - 実際に存在する特定のオブジェクト。オブジェクトのコピーやバージョンをインスタンス化した結果。

インスタンス文書 - COLLADA 文書のこと。

インスタンス化 - オブジェクトのコピー（インスタンス）を作成すること。

キーフレーム - アニメーションオブジェクトの開始点もしくは終了点。「入力」（通常は時間軸上の点）と「出力」（アニメーション対象の値）から成る 2D データサンプリングから構成されます。

MIP マップ - テクスチャ用に最適化されたビットマップ画像の集合。

モーフターゲット - 他のメッシュとブレンドできるメッシュ。

複数レンダーターゲット (MRT) - 複数の描画バッファに同時にレンダリングすること。

名前 - XML 属性の名前は、通常、その属性が属する要素との関連を意味しています。以下に例を示します。

```
<Array size="5" type="xs:float">
  1.0 2.0 3.0 4.0 5.0
</Array>
```

これは、size、type という 2 つの属性を持つ、Array という名前の要素を示しています。size 属性で配列のサイズを指定し、type 属性で配列に浮動小数点のデータが含まれることを表しています。

ノード - シーングラフ中の情報のある場所。COLLADA では、ノードという言葉は、外側ノード（葉）ではなく、内側ノード（枝）を表すために使います。

パス - 孤を参照。

プロファイル - 特定のプラットフォームや環境用のエフェクト情報を収集するための構造。

シーングラフ - シーンの階層構造。COLLADA では <scene> 要素のコンテンツによって表されます。具体的には、視覚情報および関連データをノードとして含む有向非巡回グラフ (DAG)、つまりツリー構造のデータです。

短縮ポインタ - インスタンス文書中の要素の id 属性の値。短縮ポインタは、XPointer 構文に準拠する URI フラグメント識別子です。

sid - 要素のスコープ付き識別子。インスタンス文書ではなく、特定のスコープの中においてのみ一意であるということを除けば、id と同じです。詳しくは、3 章「スキーマのコンセプト」の「アドレス構文」を参照してください。

タグ - XML 要素は、開始タグから始まります。開始タグの構文は、次のように、角括弧に囲まれた名前から構成されます。

```
<tagName>
```

それぞれの XML 要素は、終了タグで終わります。終了タグの構文は、次の通りです。

```
</tagName>
```

開始タグと終了タグの間は、任意の情報ブロックから構成されます。

トーンマッピング - 画像合成（レンダリング）の最後の手順として実行される、スペクトルサンプリングとダイナミックレンジ再マッピングの組合せ。

検証 - XML 規格それ自体では、いかなる文書構造やスキーマをも定義していません。その代わりに、XML では、XML 文書の検証を可能にするメカニズムを提供しています。対象文書（インスタンス文書）には、スキーマ文書へのリンクが含まれています。XML パーサは、このスキーマ文書を利用して、その中に指定された規則にしたがってインスタンス文書の構文と意味を検証できます。この処理は検証 (validation) と呼ばれています。

値 - XML において、属性値は、解析時に常にテキストデータとして扱われます。

視野錐台 - 空間中のカメラの視野に現れる領域。

XML - XML は、eXtensible Markup Language (拡張可能なマークアップ言語) の略です。XML は、文書、ファイル、データ集合などの構造や意味を記述するための、標準的な言語を提供します。XML 自体、要素、属性、コメント、およびテキストデータからなる構造的言語です。

XML Schema - XML Schema 言語は、構文、構造、意味において、同一の規則に準拠する XML 文書のファミリーの構造を記述する手段を提供します。XML Schema 自体も XML で記述されているため、他の XML ベースのフォーマットの設計に簡単に利用できます。

このページは空白です。

総合索引

#	
# (シャープ記号)	3-2
.	
.dae	3-1
<	
<argument>	8-6
B	
BINORMAL semantic	5-41, 5-93, 5-96
B-rep	
定義	9-2
要素	9-1
B スプライン曲線	
BSPLINE 値	5-26, 5-100
アニメーション内	5-101
スプライン	4-6
C	
COLLADA スキーマ	
B-rep 要素	9-1
FX 要素	8-1
キネマティクス要素	10-1
コア要素	5-1
フィジックス要素	6-1
前提条件と依存関係	1-1
目標とガイドライン	1-1
COLOR semantic	5-41
CONTINUITY semantic	4-1, 4-2, 5-26, 5-41
D	
DAE, 定義	3-1
deformers	5-77
F	
FX	
プロファイルリファレンス	8-1
プロファイル別要素	7-2
概要	7-1
I	
IMAGE semantic	5-41

IN_TANGENT semantic	4-1, 4-4, 5-26, 5-41, 5-100, 5-101
INPUT semantic	4-1, 5-41, 5-100, 5-101
INTERPOLATION semantic	4-1, 4-2, 4-6, 5-26, 5-41, 5-100
INV_BIND_MATRIX semantic	5-41, 5-113
J	
JOINT semantic	5-41, 5-57, 5-113, 5-128
L	
LINEAR_STEPS semantic	4-1, 4-2, 5-26, 5-41
M	
meter	5-15
MORPH_TARGET semantic	5-41, 5-78, 5-119
MORPH_WEIGHT semantic	5-41, 5-78, 5-119
N	
NORMAL semantic	5-41
O	
OUT_TANGENT semantic	4-1, 4-4, 5-26, 5-41, 5-100, 5-101
OUTPUT semantic	4-1, 5-41, 5-100, 5-101
P	
POSITION semantic	4-1, 4-3, 4-4, 4-5, 4-6, 5-26, 5-41, 5-76, 5-130
S	
semantic 属性	
<input> 属性 <input>	4-1
共通の値	3-8
曲線補間で使う	4-1
値	5-41
命名規則	3-6
SID	
定義	3-3
利用	3-3
T	
TANGENT semantic	5-41, 5-93, 5-96
TEXBINORMAL semantic	5-41, 5-93, 5-96
TEXCOORD semantic	5-41

例 8-17
 TEXTANGENT semantic..... 5-41, 5-93, 5-96

U

URI フラグメント識別子記法 3-2
 UV semantic 5-41

V

VERTEX semantic..... 5-41

W

WEIGHT semantic..... 5-41, 5-113

X

XML, 概要 3-1

あ

アドレス構文 3-1
 アニメーション
 インスタンス 5-44
 エクスポートによるサポート 2-2
 シーン利用と再生 5-13
 ライブラリ 5-59
 出力チャンネル 5-19
 要素 5-10
 アニメーション 曲線
 表現 5-98
 補間 5-98
 アニメーションクリップ
 ライブラリ 5-58
 要素 5-12
 アニメーション曲線
 定義 5-10
 分離・結合 5-12

い

インスタンス化
 COLLADA 内 3-5
 instance_... 要素のリスト 3-5
 インポータ 2-4

え

エクスポート 2-1
 エクスポートのユーザインタフェースオプション
 エクスポートによるサポート 2-3
 エルミート曲線
 HERMITE 値 5-26, 5-100
 アニメーション内 5-101
 スプライン 4-3

か

カーディナル曲線
 CARDINAL 値 5-26, 5-100
 アニメーション内 5-101
 スプライン 4-6
 カメラ
 インスタンス 5-45
 ライブラリ 5-60
 位置と方向 5-72
 画像センサー 5-38
 視野 5-90
 視野 5-85
 要素 5-17

き

キーフレーム, 定義 5-10
 キネマティクス要素 10-1
 キューブマップ 8-93

く

グループ
 cg_param_group 11-2
 fx_newparam_group 11-2
 fx_setparam_group 11-2
 gles_param_group 11-2
 gles2_value_group 11-2
 glsl_value_group 11-2

こ

コア要素 5-1

し

シーングラフトポロジ, のルート 5-131
 シーングラフのルート 5-82
 シーンデータ
 エクスポートによるサポート 2-3
 シェータ
 デフォルト色 8-45
 ジオメトリ 型 6-3
 ジョイント
 定義 (キネマティクス) 10-22
 定義 (コア) 5-113

す

スキニング
 記述と式 4-7
 計算と定義 5-113
 スコープ付き識別子: SID を参照
 ステップ 曲線
 STEP 値 5-100
 アニメーション内 5-100
 スペクトルサンプリング 5-38

た

ダイナミックレンジのリマップ 5-38

て

テクスチャ
 エクスポートによるサポート 2-2
 テクスチャリング 7-7

と

トーンマッピング 5-38

の

ノード, 定義 5-82

は

バインド形状, 定義 5-113
 バインド形状行列, 定義 5-113
 パラメータ
 bind や bind_vertex_input の中で特定する 7-5
 パラメータについて 7-4
 構造の定義 8-118
 値の設定 5-106
 配列型の定義 8-7
 名前と型の規則 3-6

ふ

フィジックス要素 6-1
 プラットフォーム 7-1
 プロファイリング 7-1

へ

ベースメッシュ, 定義 5-113
 ベクトル, 配列添字記法 3-9
 ベジエ曲線
 BEZIER 値 5-26, 5-100
 アニメーション内 5-101
 スプライン 4-3

ま

マテリアルエクスポートによるサポート 2-2

れ

レイヤー 5-131
 レンダリング, レンダリングについて 7-6

漢字

位相要素, 定義と要素 9-2
 可視性サポート 5-131
 階層

エクスポートによる サポート 2-1

関数曲線
 定義 5-10
 幾何学要素, 定義と要素 9-2

記法
 ベクトルおよび行列の配列添字 3-9
 逆バインド行列, 定義 5-113
 距離の尺度 5-15
 共通プロファイル 3-6
 テクスチャマッピング 7-7
 要素 3-6
 共通プロファイル: 要素:profile_COMMON を参照
 境界表現 B-Rep を参照
 曲線 特定の種類の曲線も参照
 補間 4-1
 曲線の補間 4-1, 5-98
 曲線型 5-100

型
 fx_common_color_or_texture_type 8-44
 fx_common_float_or_param_type 8-45
 fx_sampler_common 8-46
 値(パラメータ)型 11-2

検索
 bind や bind_vertex_input の中からパラメータを 7-5
 弧, 定義 5-82
 行列, 配列添字記法 3-9
 座標系
 軸方向の設定 5-16
 軸
 方向 5-16
 尺度, 単位, 設定 5-15
 重み定義 5-113
 線形曲線
 LINEAR 値 5-26, 5-100
 アニメーション内 5-101
 スプライン 4-2

属性
 id 3-3
 要素を特定する 3-2
 semantic semantic 属性を参照
 sid 3-3
 url 3-1, 3-2
 XML における 3-1
 ソース 3-2
 ターゲット
 メンバ選択値 3-8
 名前
 共通の値 3-8
 短縮ポイント 3-2
 値 型 11-2
 値, 参照 3-1
 頂点属性, エクスポートによるサポート 2-2
 配列添字記法 3-9
 配列添字記法の括弧 3-9
 反射要素 5-84
 部分グラフのルート 5-82

物理的な 単位	6-2	<bind_vertex_input>	8-16
変換		<bind> (FX)	8-10
エクスポータによるサポート	2-1	<bind> (キネマティックス)	10-10
補間の種類	5-26	<blend_color> (レンダリング ステート)	8-102
命名規則	3-6	<blend_enable> (レンダリング ステート)	8-102
用語集		<blend_equation_separate> (レンダリング ステート)	8-102
共通プロファイル名	3-7	<blend_equation> (レンダリング ステート) ...	8-102
要素		<blend_func_separate> (レンダリング ステート)	8-103
<acceleration>		<blend_func> (レンダリング ステート)	8-102
<axis_info>	10-9	<blinn>	8-18
<effector_info>	10-15	<bool_array>	5-16
<accessor>	5-4	<border_color>	8-48
<active>	10-8	<box>	6-5
<alpha_func> (レンダリング ステート)	8-102	<brep>	9-6
<alpha_test_enable> (レンダリング ステート)	8-102	<camera>	5-17
<alpha>	8-4	<capsule>	6-6
<altitude>	5-34	<channel>	5-19
<ambient> (FX)	8-44	ターゲットを参照	3-8
<ambient> (コア)	5-9	<circle>	9-8
<angle>	9-10	<clip_plane_enable> (レンダリング ステート)	8-103
<angular_velocity>	6-17	<clip_plane> (レンダリング ステート)	8-103
<angular>	6-39	<code>	8-21
<animation_clip>	5-12	<COLLADA>	5-20
<animation>	5-10	<color_clear>	8-22
<annotate>	8-5	<color_logic_op_enable> (レンダリング ステート)	8-103
<array>	8-7	<color_mask> (レンダリング ステート)	8-103
<create_cube>	8-33	<color_material_enable> (レンダリング ステート)	8-103
<create2d>	8-29	<color_material> (レンダリング ステート) ...	8-103
<create3d>	8-31	<color_target>	8-23
<articulated_system>	10-3	<color>	5-21
<aspect_ratio>		<comments>	5-23
<orthographic>	5-86	<compiler>	8-25
<perspective>	5-91	<cone>	9-10
<asset>	5-14	<connect_param> (キネマティックス)	10-13
外部ツールで使われる	2-4	<constant_attenuation>	
<attachment_end>	10-4	<point>	5-92
<attachment_full>	10-5	<spot>	5-117
<attachment_start>	10-6	<constant> (FX)	8-26
<author_email>	5-23	<constant> (コンパイナ)	8-113, 8-115
<author_website>	5-23	<contributor>	5-22
<author>	5-23	<control_vertices>	5-25
<authoring_tool>	5-23	<controller>	5-23
<auto_normal_enable> (レンダリング ステート)	8-102	<convex_mesh>	6-7
<axis_info>	10-7	<copyright>	5-23
<axis>		<coverage>	5-15
<prismatic>	10-35	<create_2d>	8-28
<revolute>	10-36	<create_3d>	8-30
<swept_surface>	9-37	<create_cube>	8-32
<binary>	8-9	<created>	5-15
<bind_attribute>	8-11	<cull_face_enable> (レンダリング ステート)	8-103
<bind_joint_axis>	10-11	<cull_face> (レンダリング ステート)	8-103
<bind_kinematics_model>	10-12	<curve>	9-11
<bind_material>	8-12		
<bind_shape_matrix>	5-112		
<bind_uniform>	8-15		

- <curves> 9-12
- <cylinder> 6-9
- <cylinder> (B-Rep) 9-13
- <damping> 6-39
- <deceleration>
 - <axis_info> 10-9
 - <effector_info> 10-15
- <density> 6-41
- <depth_bounds_enable> (レンダリング ステート) 8-103
- <depth_bounds> (レンダリング ステート) 8-103
- <depth_clamp_enable> (レンダリング ステート) 8-103
- <depth_clear> 8-34
- <depth_func> (レンダリング ステート) 8-103
- <depth_mask> (レンダリング ステート) 8-103
- <depth_range> (レンダリング ステート) 8-103
- <depth_target> 8-35
- <depth_test_enable> (レンダリング ステート) 8-103
- <diffuse> 8-44
- <direction>
 - <line> 9-20
 - <swept_surface> 9-37
- <directional> 5-27
- <dither_enable> (レンダリング ステート) 8-103
- <draw> 8-37
- <dynamic_friction> 6-25
- <dynamic>
 - <instance_rigid_body> 6-17
 - <rigid_body> 6-34
- <edges> 9-14
- <effect> 8-38
- <effector_info> 10-14
- <ellipse> 9-16
- <emission> 8-44
- <enabled> 6-38
- <equation> 6-31
- <evaluate_scene> 5-28
- <evaluate> 8-40
- <exact> 8-42
- <extra> 5-29
- <faces> 9-17
- <falloff_angle> 5-118
- <falloff_exponent> 5-118
- <float_array> 5-31
- <float> (シェーダ) 8-46
- <focal> 9-28
- <fog_color> (レンダリング ステート) 8-103
- <fog_coord_src> (レンダリング ステート) 8-103
- <fog_density> (レンダリング ステート) 8-103
- <fog_enable> (レンダリング ステート) 8-103
- <fog_end> (レンダリング ステート) 8-104
- <fog_mode> (レンダリング ステート) 8-104
- <fog_state> (レンダリング ステート) 8-104
- <force_field> 6-10
- <format> 8-41, 8-109
- <formula> 5-32
- <frame_object> 10-15
- <frame_origin> 10-15
- <frame_tcp> 10-15
- <frame_tip> 10-15
- <front_face> (レンダリング ステート) 8-104
- <func> (レンダリング ステート) 8-102
- <geographic_location> 5-34
- <geometry> 5-35
- <gravity> 6-29
- <h> 5-94
- <half_extents> 6-5
- <height>
 - <capsule> 6-6, 6-9
- <hex>
 - <binary> 8-9
 - <init_from> 8-53
- <hint> 8-42
- <hollow> 6-41
- <hyperbola> 9-19
- <IDREF_array> 5-37
- <image> 8-49
- <imager> 5-38
- <import> 8-101
- <include> 8-51
- <index_of_refraction> 8-45
- <index> 10-8
- <inertia>
 - <instance_rigid_body> 6-17
 - <rigid_body> 6-34
- <init_from> 8-52
- <inline> 8-101
- <input>
 - semantics 属性 <sampler> 5-100
 - semantics 属性 <skin> 5-113
 - semantics 属性 <triangles> 5-124
 - semantics 属性 <trifans> 5-126
 - semantics 属性 <tristrips> 5-127
 - semantics 属性 <vertex_weights> 5-128
 - semantics 属性 <vertices> 5-130
 - semantic を参照 entry 属性 semantic を参照
- <input> (共有) 5-40
- <input> (非共有) 5-42
- <instance_animation> 5-44
- <instance_articulated_system> 10-16
- <instance_camera> 5-45
- <instance_controller> 5-46
- <instance_effect> 8-54
- <instance_force_field> 6-11
- <instance_formula> 5-49
- <instance_geometry> 5-50
- <instance_image> 8-55
- <instance_joint> 10-18
- <instance_kinematics_model> 10-19
- <instance_kinematics_scene> 10-20
- <instance_light> 5-51

<instance_material> (ジオメトリ)	8-57
<instance_material> (レンダリング)	8-58
<instance_node>	5-53
<instance_physics_material>	6-12
<instance_physics_model>.....	6-13
<instance_physics_scene>.....	6-14
<instance_rigid_body>.....	6-15
<instance_rigid_constraint>	6-18
<instance_visual_scene>.....	5-55
<int_array>	5-56
<interpenetrate>	6-38
<jerk>	
<axis_info>	10-9
<effector_info>	10-15
<joint>	10-22
<joints>	5-57
<keywords>	5-15
<kinematics_model>	10-25
<kinematics_scene>	10-27
<kinematics>	10-23
<lambert>	8-60
<latitude>	5-34
<layer>	8-89
<library_animation_clips>.....	5-58
<library_animations>.....	5-59
<library_articulated_systems>	10-28
<library_cameras>.....	5-60
<library_controllers>.....	5-61
<library_effects>	8-62
<library_force_fields>.....	6-19
<library_formulas>	5-62
<library_geometries>.....	5-63
<library_images>	8-63
<library_joints>	10-29
<library_kinematics_models>	10-30
<library_kinematics_scenes>	10-31
<library_lights>	5-64
<library_materials>	8-64
<library_nodes>	5-65
<library_physics_materials>	6-20
<library_physics_models>.....	6-21
<library_physics_scenes>.....	6-23
<library_visual_scenes>.....	5-66
<light_ambient> (レンダリング ステート) ...	8-104
<light_constant_attenuation> (レンダリング ステート).....	8-104
<light_diffuse> (レンダリング ステート) ...	8-104
<light_enable> (レンダリング ステート)	8-104
<light_linear_attenuation> (レンダリング ステート)	8-104
<light_model_ambient> (レンダリング ステート)	8-104
<light_model_color_control> (レンダリング ステート).....	8-104
<light_model_local_viewer_enable> (レンダリング ステート).....	8-104
<light_model_two_side_enable> (レンダリング ステート)	8-104
<light_position> (レンダリング ステート) ...	8-104
<light_quadratic_attenuation> (レンダリング ステート)	8-104
<light_specular> (レンダリング ステート) ...	8-104
<light_spot_cutoff> (レンダリング ステート)	8-104
<light_spot_direction> (レンダリング ステート)	8-104
<light_spot_exponent> (レンダリング ステート)	8-104
<lighting_enable> (レンダリング ステート) ..	8-104
<lights>	5-67
<limits>	6-38
<axis_info>.....	10-8
<prismatic>.....	10-35
<revolute>.....	10-37
<line_smooth_enable> (レンダリング ステート)	8-104
<line_stipple_enable> (レンダリング ステート)	8-104
<line_stipple> (レンダリング ステート)	8-104
<line_width> (レンダリング ステート)	8-104
<line>.....	9-20
<linear_attenuation>	
<point>.....	5-92
<spot>.....	5-117
<linear>	6-38, 6-39
<lines>	5-69
<linestrips>	5-70
<link>.....	10-32
<linker>	8-65
<locked>	10-8
<logic_op_enable> (レンダリング ステート) ..	8-105
<logic_op> (レンダリング ステート)	8-104
<longitude>	5-34
<lookout>	5-72
<magfilter>	8-47
<mass_frame>	
<instance_rigid_body>/<technique_common> ..	6-17
<rigid_body>/<technique_common>	6-34
<mass>	
<instance_rigid_body>	6-17
<rigid_body>	6-34
<shape>	6-41
<material_ambient> (レンダリング ステート) ..	8-105
<material_diffuse> (レンダリング ステート) ..	8-105
<material_emission> (レンダリング ステート) ..	8-105
<material_shininess> (レンダリング ステート) ..	8-105
<material_specular> (レンダリング ステート) ..	8-105
<material>	8-66
<matrix>	5-73
<max_anisotropy>	8-48
<max>	
<limits>.....	10-8, 10-35, 10-37

- <mesh> 5-74
- <min>
 - <limits> 10-8, 10-35, 10-37
- <minfilter> 8-47
- <mip_bias> 8-48
- <mip_max_level> 8-48
- <mip_min_level> 8-48
- <mipfilter> 8-48
- <mips>
 - <create_cube> 8-33
 - <create2d> 8-29
 - <create3d> 8-31
- <model_view_matrix> (レンダリング ステート) 8-105
- <modified> 5-15
- <modifier> 8-68
- <morph> 5-77
- <motion> 10-33
- <multisample_enable> (レンダリング ステート)
 - 8-105
- <Name_array> 5-79
 - semantic 曲線属性 curves 4-1
- <newparam> 5-80
 - 共通 semantic を参照 3-8
- <node> 5-82
- <normalize_enable> (レンダリング ステート) 8-105
- <nurbs_surface> 9-23
- <nurbs> 9-21
- <optics> 5-84
- <orient> 9-26
- <origin> 9-26
- <orthographic> 5-85
- <p>
 - <edges> 9-15
 - <faces> 9-18
 - <lines> 5-70
 - <linestrips> 5-71
 - <pcurves> 9-29
 - <ph> 5-94
 - <polygons> 5-94
 - <polylist> 5-96
 - <shells> 9-30
 - <solids> 9-32
 - <triangles> 5-123
 - <trifans> 5-125
 - <tristrips> 5-127
 - <wires> 9-39
- <parabola> 9-27
- <param>
 - 名前を参照 3-8
- <param> (データ フロー) 5-87
- <param> (リファレンス) 5-88
- <pass> 8-69
- <pcurves> 9-28
- <perspective> 5-90
- <ph> 5-94
- <phong> 8-71
- <physics_material> 6-24
- <physics_model> 6-25
- <physics_scene> 6-28
- <plane> 6-31
- <point_distance_attenuation> (レンダリング ステート)
 - 8-105
- <point_fade_threshold_size> (レンダリング ステート)
 - 8-105
- <point_size_max> (レンダリング ステート) ... 8-105
- <point_size_min> (レンダリング ステート) ... 8-105
- <point_size> (レンダリング ステート) 8-105
- <point_smooth_enable> (レンダリング ステート)
 - 8-105
- <point> 5-91
- <polygon_mode> (レンダリング ステート) 8-105
- <polygon_offset_fill_enable> (レンダリング ステート)
 - 8-105
- <polygon_offset_line_enable> (レンダリング ステート)
 - 8-105
- <polygon_offset_point_enable> (レンダリング ステート)
 - 8-105
- <polygon_offset> (レンダリング ステート) ... 8-105
- <polygon_smooth_enable> (レンダリング ステート)
 - 8-105
- <polygon_stipple_enable> (レンダリング ステート)
 - 8-105
- <polygons> 5-92
- <polylist> 5-95
- <prismatic> 10-35
- <profile_BRIDGE> 8-73
- <profile_CG> 8-75
- <profile_COMMON> 8-78
 - 概要 3-6
- <profile_GLES> 8-79
- <profile_GLES2> 8-82
- <profile_GLSL> 8-85
- <program> 8-87
- <projection_matrix> (レンダリング ステート) 8-105
- <quadratic_attenuation>
 - <point> 5-92
 - <spot> 5-118
- <radius>
 - <capsule> 6-6, 6-9
 - <circle> 9-9
 - <cone> 9-10
 - <cylinder> 9-14
 - <ellipse> 9-16
 - <hyperbola> 9-19
 - <sphere> 6-42
 - <torus> 9-38
- <ref_attachment> 6-32
- <ref>
 - <binary> 8-9
 - <init_from> 8-53
- <reflective> 8-44
- <reflectivity> 8-45

<render>	8-89
<renderable>	8-50
<rescale_normal_enable> (レンダリング ステート)	8-105
<restitution>	6-25
<revision>	5-15
<revolute>	10-36
<RGB>	8-90
<rigid_body>	6-33
<rigid_constraint>	6-36
<rotate>	5-97
<sample_alpha_to_coverage_enable> (レンダリング ステート)	8-105
<sample_alpha_to_one_enable> (レンダリング ステート)	8-105
<sample_coverage_enable> (レンダリング ステート)	8-105
<sample_coverage> (レンダリング ステート) ..	8-105
<sampler_image>	8-96
<sampler_states>	8-96
<sampler>	5-98
補間.....	4-1
<sampler1D>	8-91
<sampler2D>	8-92
<sampler3D>	8-92
<samplerCUBE>	8-93
<samplerDEPTH>	8-94
<samplerRECT>	8-95
<scale>	5-104
<scene>	5-105
<scissor_test_enable> (レンダリング ステート)	8-105
<scissor> (レンダリング ステート)	8-105
<semantic>	8-97
<setparam>	5-106
<shade_model> (レンダリング ステート)	8-105
<shader>	8-98
<shape>	6-40
<shells>	9-29
<shininess>	8-45
<SIDREF_array>	5-108
<size_exact>	8-29
<size_ratio>	8-29
<size> <create_cube>	8-33
<create3d>	8-31
<skeleton>	5-109
<skew>	5-110
<skin>	5-111
<solids>	9-31
<source_data>	5-23
<source> (コア)	5-114
<sources>	8-100
<specular>	8-44
<speed> <axis_info>	10-9
<effector_info>	10-14
<sphere>	6-42
<spline>	5-116
補間.....	4-1
<spot>.....	5-117
<spring>	6-39
<states>	8-101
<static_friction>	6-25
<stencil_clear>	8-107
<stencil_func_separate> (レンダリング ステート)	8-105
<stencil_func> (レンダリング ステート)	8-105
<stencil_mask_separate> (レンダリング ステート)	8-105
<stencil_mask> (レンダリング ステート)	8-105
<stencil_op_separate> (レンダリング ステート)	8-106
<stencil_op> (レンダリング ステート)	8-106
<stencil_target>	8-108
<stencil_test_enable> (レンダリング ステート)	8-106
<stiffness>	6-39
<subject>	5-15
<surface_curves>	9-35
<surface>	9-32
<surfaces>	9-34
<swept_surface>	9-36
<swing_cone_and_twist>	6-38
<target_value>	6-39
<target>	5-32
<targets>	5-118
<technique_common>	5-121
<formula>.....	5-33
<instance_rigid_body>	6-17
<kinematics_model>	10-26
<kinematics>	10-24
<light>.....	5-68
<motion>.....	10-34
<optics>.....	5-85
<physics_material>	6-25
<physics_scene>	6-29
<rigid_body>	6-34
<rigid_constraint>	6-38
<source>.....	5-115
概要.....	3-6
<technique_hint>	8-111
<technique_override>	8-59
<technique> 概要.....	3-6
<technique> (FX)	8-110
<technique> (コア)	5-119
<texcombiner>	8-112
<texcoord>	8-47
<texenv>	8-114
<texture_env_color> (レンダリング ステート) ..	8-106
<texture_env_mode> (レンダリング ステート) ..	8-106

<texture_pipeline> 8-116
 <texture_pipeline> (レンダリング ステート) . 8-106
 <texture> (シェーダ) 8-45
 <texture1D_enable> (レンダリング ステート) . 8-106
 <texture1D> (レンダリング ステート) 8-106
 <texture2D_enable> (レンダリング ステート) . 8-106
 <texture2D> (レンダリング ステート) 8-106
 <texture3D_enable> (レンダリング ステート) . 8-106
 <texture3D> (レンダリング ステート) 8-106
 <textureCUBE_enable> (レンダリング ステート)
 8-106
 <textureCUBE> (レンダリング ステート) 8-106
 <textureDEPTH_enable> (レンダリング ステート)
 8-107
 <textureDEPTH> (レンダリング ステート) 8-106,
 8-107
 <textureRECT_enable> (レンダリング ステート)
 8-107
 <textureRECT> (レンダリング ステート) 8-107
 <time_step> 6-29
 <title> 5-15
 <torus> 9-38
 <translate> 5-122
 <transparency> 8-45
 <transparent> 8-44
 <triangles> 5-123
 <trifans> 5-124
 <tristrips> 5-126
 <unit> 5-15
 <unnormalized> 8-29
 <up_axis> 5-16
 <v> 5-128
 <value>

<bind_joint_axis> 10-11
 <value> (レンダリング ステート) 8-102
 <vcount>
 <faces> 9-18
 <pcurves> 9-29
 <polylist> 5-96
 <shells> 9-30
 <solids> 9-32
 <vertex_weights> 5-128
 <wires> 9-39
 <velocity> 6-17
 <vertex_weights> 5-128
 <vertices> 5-129
 <visual_scene> 5-130
 <wires> 9-39
 <wrap_p> 8-47
 <wrap_s> 8-47
 <wrap_t> 8-47
 <xfov> 5-90
 <xmag> 5-86
 <yfov> 5-91
 <ymag> 5-86
 <zfar>
 <orthographic> 5-86
 <perspective> 5-91
 <znear>
 <orthographic> 5-86
 <perspective> 5-91
 chnique_common
 <bind_material> 8-13
 XML 3-1
 参照 3-1
 路, 定義 5-82

このページは空白です。

COLLADA 要素の索引

<acceleration>		
<axis_info>	10-9
<effector_info>	10-15
<accessor>	5-4
<active>	10-8
<alpha_func> (レンダリング ステート)	8-102
<alpha_test_enable> (レンダリング ステート)	8-102
<alpha>	8-4
<altitude>	5-34
<ambient> (FX)	8-44
<ambient> (コア)	5-9
<angle>	9-10
<angular_velocity>	6-17
<angular>	6-39
<animation_clip>	5-12
<animation>	5-10
<annotate>	8-5
<array>	8-7
<create_cube>	8-33
<create2d>	8-29
<create3d>	8-31
<articulated_system>	10-3
<aspect_ratio>		
<orthographic>	5-86
<perspective>	5-91
<asset>	5-14
外部ツールで使われる	2-4
<attachment_end>	10-4
<attachment_full>	10-5
<attachment_start>	10-6
<author_email>	5-23
<author_website>	5-23
<author>	5-23
<authoring_tool>	5-23
<auto_normal_enable> (レンダリング ステート)	8-102
<axis_info>	10-7
<axis>		
<prismatic>	10-35
<revolute>	10-36
<swept_surface>	9-37
<binary>	8-9
<bind_attribute>	8-11
<bind_joint_axis>	10-11
<bind_kinematics_model>	10-12
<bind_material>	8-12
<bind_shape_matrix>	5-112
<bind_uniform>	8-15
<bind_vertex_input>	8-16
<bind> (FX)	8-10
<bind> (キネマティックス)	10-10
<blend_color> (レンダリング ステート)	8-102
<blend_enable> (レンダリング ステート)	8-102
<blend_equation_separate> (レンダリング ステート)	8-102
<blend_equation> (レンダリング ステート)	8-102
<blend_func_separate> (レンダリング ステート)	8-103
<blend_func> (レンダリング ステート)	8-102
<blinn>	8-18
<bool_array>	5-16
<border_color>	8-48
<box>	6-5
<brep>	9-6
<camera>	5-17
<capsule>	6-6
<channel>	5-19
ターゲットを参照	3-8
<circle>	9-8
<clip_plane_enable> (レンダリング ステート)	8-103
<clip_plane> (レンダリング ステート)	8-103
<code>	8-21
<COLLADA>	5-20
<color_clear>	8-22
<color_logic_op_enable> (レンダリング ステート)	8-103
<color_mask> (レンダリング ステート)	8-103
<color_material_enable> (レンダリング ステート)	8-103
<color_material> (レンダリング ステート)	8-103
<color_target>	8-23
<color>	5-21
<comments>	5-23
<compiler>	8-25
<cone>	9-10
<connect_param> (キネマティックス)	10-13
<constant_attenuation>		
<point>	5-92
<spot>	5-117
<constant> (FX)	8-26
<constant> (コンパイナ)	8-113, 8-115
<contributor>	5-22
<control_vertices>	5-25
<controller>	5-23
<convex_mesh>	6-7
<copyright>	5-23
<coverage>	5-15
<create_2d>	8-28
<create_3d>	8-30
<create_cube>	8-32
<created>	5-15
<cull_face_enable> (レンダリング ステート)	8-103
<cull_face> (レンダリング ステート)	8-103
<curve>	9-11

<curves>	9-12
<cylinder>	6-9
<cylinder> (B-Rep)	9-13
<damping>	6-39
<deceleration>	
<axis_info>	10-9
<effector_info>	10-15
<density>	6-41
<depth_bounds_enable> (レンダリング ステート)	
.....	8-103
<depth_bounds> (レンダリング ステート)	8-103
<depth_clamp_enable> (レンダリング ステート)	
.....	8-103
<depth_clear>	8-34
<depth_func> (レンダリング ステート)	8-103
<depth_mask> (レンダリング ステート)	8-103
<depth_range> (レンダリング ステート)	8-103
<depth_target>	8-35
<depth_test_enable> (レンダリング ステート)	8-103
<diffuse>	8-44
<direction>	
<line>	9-20
<swept_surface>	9-37
<directional>	5-27
<dither_enable> (レンダリング ステート)	8-103
<draw>	8-37
<dynamic_friction>	6-25
<dynamic>	
<instance_rigid_body>	6-17
<rigid_body>	6-34
<edges>	9-14
<effect>	8-38
<effector_info>	10-14
<ellipse>	9-16
<emission>	8-44
<enabled>	6-38
<equation>	6-31
<evaluate_scene>	5-28
<evaluate>	8-40
<exact>	8-42
<extra>	5-29
<faces>	9-17
<falloff_angle>	5-118
<falloff_exponent>	5-118
<float_array>	5-31
<float> (シェーダ)	8-46
<focal>	9-28
<fog_color> (レンダリング ステート)	8-103
<fog_coord_src> (レンダリング ステート)	8-103
<fog_density> (レンダリング ステート)	8-103
<fog_enable> (レンダリング ステート)	8-103
<fog_end> (レンダリング ステート)	8-104
<fog_mode> (レンダリング ステート)	8-104
<fog_state> (レンダリング ステート)	8-104
<force_field>	6-10
<format>	8-41, 8-109
<formula>	5-32
<frame_object>	10-15
<frame_origin>	10-15
<frame_tcp>	10-15
<frame_tip>	10-15
<front_face> (レンダリング ステート)	8-104
<func> (レンダリング ステート)	8-102
<geographic_location>	5-34
<geometry>	5-35
<gravity>	6-29
<h>	5-94
<half_extents>	6-5
<height>	
<capsule>	6-6, 6-9
<hex>	
<binary>	8-9
<init_from>	8-53
<hint>	8-42
<hollow>	6-41
<hyperbola>	9-19
<IDREF_array>	5-37
<image>	8-49
<imager>	5-38
<import>	8-101
<include>	8-51
<index_of_refraction>	8-45
<index>	10-8
<inertia>	
<instance_rigid_body>	6-17
<rigid_body>	6-34
<init_from>	8-52
<inline>	8-101
<input>	
semantics 属性 <sampler>	5-100
semantics 属性 <skin>	5-113
semantics 属性 <triangles>	5-124
semantics 属性 <trifans>	5-126
semantics 属性 <tristrips>	5-127
semantics 属性 <vertex_weights>	5-128
semantics 属性 <vertices>	5-130
semantic を参照 entry 属性 semantic を参照	
<input> (共有)	5-40
<input> (非共有)	5-42
<instance_animation>	5-44
<instance_articulated_system>	10-16
<instance_camera>	5-45
<instance_controller>	5-46
<instance_effect>	8-54
<instance_force_field>	6-11
<instance_formula>	5-49
<instance_geometry>	5-50
<instance_image>	8-55
<instance_joint>	10-18
<instance_kinematics_model>	10-19
<instance_kinematics_scene>	10-20
<instance_light>	5-51

- <instance_material> (ジオメトリ) 8-57
- <instance_material> (レンダリング) 8-58
- <instance_node> 5-53
- <instance_physics_material> 6-12
- <instance_physics_model> 6-13
- <instance_physics_scene> 6-14
- <instance_rigid_body> 6-15
- <instance_rigid_constraint> 6-18
- <instance_visual_scene> 5-55
- <int_array> 5-56
- <interpenetrate> 6-38
- <jerk>
 - <axis_info> 10-9
 - <effector_info> 10-15
- <joint> 10-22
- <joints> 5-57
- <keywords> 5-15
- <kinematics_model> 10-25
- <kinematics_scene> 10-27
- <kinematics> 10-23
- <lambert> 8-60
- <latitude> 5-34
- <layer> 8-89
- <library_animation_clips> 5-58
- <library_animations> 5-59
- <library_articulated_systems> 10-28
- <library_cameras> 5-60
- <library_controllers> 5-61
- <library_effects> 8-62
- <library_force_fields> 6-19
- <library_formulas> 5-62
- <library_geometries> 5-63
- <library_images> 8-63
- <library_joints> 10-29
- <library_kinematics_models> 10-30
- <library_kinematics_scenes> 10-31
- <library_lights> 5-64
- <library_materials> 8-64
- <library_nodes> 5-65
- <library_physics_materials> 6-20
- <library_physics_models> 6-21
- <library_physics_scenes> 6-23
- <library_visual_scenes> 5-66
- <light_ambient> (レンダリング ステート) ... 8-104
- <light_constant_attenuation> (レンダリング ステート) 8-104
- <light_diffuse> (レンダリング ステート) ... 8-104
- <light_enable> (レンダリング ステート) 8-104
- <light_linear_attenuation> (レンダリング ステート) 8-104
- <light_model_ambient> (レンダリング ステート) 8-104
- <light_model_color_control> (レンダリング ステート) 8-104
- <light_model_local_viewer_enable> (レンダリング ステート) 8-104
- <light_model_two_side_enable> (レンダリング ステート) 8-104
- <light_position> (レンダリング ステート) ... 8-104
- <light_quadratic_attenuation> (レンダリング ステート) 8-104
- <light_specular> (レンダリング ステート) ... 8-104
- <light_spot_cutoff> (レンダリング ステート) 8-104
- <light_spot_direction> (レンダリング ステート) 8-104
- <light_spot_exponent> (レンダリング ステート) 8-104
- <lighting_enable> (レンダリング ステート) .. 8-104
- <lights> 5-67
- <limits> 6-38
 - <axis_info> 10-8
 - <prismatic> 10-35
 - <revolute> 10-37
- <line_smooth_enable> (レンダリング ステート) 8-104
- <line_stipple_enable> (レンダリング ステート) 8-104
- <line_stipple> (レンダリング ステート) 8-104
- <line_width> (レンダリング ステート) 8-104
- <line> 9-20
- <linear_attenuation>
 - <point> 5-92
 - <spot> 5-117
- <linear> 6-38, 6-39
- <lines> 5-69
- <linestrips> 5-70
- <link> 10-32
- <linker> 8-65
- <locked> 10-8
- <logic_op_enable> (レンダリング ステート) .. 8-105
- <logic_op> (レンダリング ステート) 8-104
- <longitude> 5-34
- <lookout> 5-72
- <magfilter> 8-47
- <mass_frame>
 - <instance_rigid_body>/<technique_common> . 6-17
 - <rigid_body>/<technique_common> 6-34
- <mass>
 - <instance_rigid_body> 6-17
 - <rigid_body> 6-34
 - <shape> 6-41
- <material_ambient> (レンダリング ステート) . 8-105
- <material_diffuse> (レンダリング ステート) . 8-105
- <material_emission> (レンダリング ステート) 8-105
- <material shininess> (レンダリング ステート) 8-105
- <material_specular> (レンダリング ステート) 8-105
- <material> 8-66
- <matrix> 5-73
- <max_anisotropy> 8-48
- <max>
 - <limits> 10-8, 10-35, 10-37

<mesh>	5-74
<min>	
<limits>	10-8, 10-35, 10-37
<minfilter>	8-47
<mip_bias>	8-48
<mip_max_level>	8-48
<mip_min_level>	8-48
<mipfilter>	8-48
<mips>	
<create_cube>	8-33
<create2d>	8-29
<create3d>	8-31
<model_view_matrix> (レンダリング ステート)	8-105
<modified>	5-15
<modifier>	8-68
<morph>	5-77
<motion>	10-33
<multisample_enable> (レンダリング ステート)	
.....	8-105
<Name_array>	5-79
semantic 曲線属性 curves	4-1
<newparam>	5-80
共通 semantic を参照	3-8
<node>	5-82
<normalize_enable> (レンダリング ステート)	8-105
<nurbs_surface>	9-23
<nurbs>	9-21
<optics>	5-84
<orient>	9-26
<origin>	9-26
<orthographic>	5-85
<p>	
<edges>	9-15
<faces>	9-18
<lines>	5-70
<linestrips>	5-71
<pcurves>	9-29
<ph>	5-94
<polygons>	5-94
<polylist>	5-96
<shells>	9-30
<solids>	9-32
<triangles>	5-123
<trifans>	5-125
<tristrips>	5-127
<wires>	9-39
<parabola>	9-27
<param>	
名前を参照	3-8
<param> (データ フロー)	5-87
<param> (リファレンス)	5-88
<pass>	8-69
<pcurves>	9-28
<perspective>	5-90
<ph>	5-94
<phong>	8-71
<physics_material>	6-24
<physics_model>	6-25
<physics_scene>	6-28
<plane>	6-31
<point_distance_attenuation> (レンダリング ステート)	
.....	8-105
<point_fade_threshold_size> (レンダリング ステート)	
.....	8-105
<point_size_max> (レンダリング ステート) ...	8-105
<point_size_min> (レンダリング ステート) ...	8-105
<point_size> (レンダリング ステート)	8-105
<point_smooth_enable> (レンダリング ステート)	
.....	8-105
<point>	5-91
<polygon_mode> (レンダリング ステート)	8-105
<polygon_offset_fill_enable> (レンダリング ステート)	
.....	8-105
<polygon_offset_line_enable> (レンダリング ステート)	
.....	8-105
<polygon_offset_point_enable> (レンダリング ステート)	
.....	8-105
<polygon_offset> (レンダリング ステート) ...	8-105
<polygon_smooth_enable> (レンダリング ステート)	
.....	8-105
<polygon_stipple_enable> (レンダリング ステート)	
.....	8-105
<polygons>	5-92
<polylist>	5-95
<prismatic>	10-35
<profile_BRIDGE>	8-73
<profile_CG>	8-75
<profile_COMMON>	8-78
概要	3-6
<profile_GLES>	8-79
<profile_GLES2>	8-82
<profile_GLSL>	8-85
<program>	8-87
<projection_matrix> (レンダリング ステート)	8-105
<quadratic_attenuation>	
<point>	5-92
<spot>	5-118
<radius>	
<capsule>	6-6, 6-9
<circle>	9-9
<cone>	9-10
<cylinder>	9-14
<ellipse>	9-16
<hyperbola>	9-19
<sphere>	6-42
<torus>	9-38
<ref_attachment>	6-32
<ref>	
<binary>	8-9
<init_from>	8-53
<reflective>	8-44
<reflectivity>	8-45

<render>	8-89
<renderable>	8-50
<rescale_normal_enable> (レンダリング ステート)	8-105
<restitution>	6-25
<revision>	5-15
<revolute>	10-36
<RGB>	8-90
<rigid_body>	6-33
<rigid_constraint>	6-36
<rotate>	5-97
<sample_alpha_to_coverage_enable> (レンダリング ステート)	8-105
<sample_alpha_to_one_enable> (レンダリング ステート)	8-105
<sample_coverage_enable> (レンダリング ステート)	8-105
<sample_coverage> (レンダリング ステート)	8-105
<sampler_image>	8-96
<sampler_states>	8-96
<sampler>	5-98
補間	4-1
<sampler1D>	8-91
<sampler2D>	8-92
<sampler3D>	8-92
<samplerCUBE>	8-93
<samplerDEPTH>	8-94
<samplerRECT>	8-95
<scale>	5-104
<scene>	5-105
<scissor_test_enable> (レンダリング ステート)	8-105
<scissor> (レンダリング ステート)	8-105
<semantic>	8-97
<setparam>	5-106
<shade_model> (レンダリング ステート)	8-105
<shader>	8-98
<shape>	6-40
<shells>	9-29
<shininess>	8-45
<SIDREF_array>	5-108
<size_exact>	8-29
<size_ratio>	8-29
<size>	
<create_cube>	8-33
<create3d>	8-31
<skeleton>	5-109
<skew>	5-110
<skin>	5-111
<solids>	9-31
<source_data>	5-23
<source> (コア)	5-114
<sources>	8-100
<specular>	8-44
<speed>	
<axis_info>	10-9
<effector_info>	10-14
<sphere>	6-42
<spline>	5-116
補間	4-1
<spot>	5-117
<spring>	6-39
<states>	8-101
<static_friction>	6-25
<stencil_clear>	8-107
<stencil_func_separate> (レンダリング ステート)	8-105
<stencil_func> (レンダリング ステート)	8-105
<stencil_mask_separate> (レンダリング ステート)	8-105
<stencil_mask> (レンダリング ステート)	8-105
<stencil_op_separate> (レンダリング ステート)	8-106
<stencil_op> (レンダリング ステート)	8-106
<stencil_target>	8-108
<stencil_test_enable> (レンダリング ステート)	8-106
<stiffness>	6-39
<subject>	5-15
<surface_curves>	9-35
<surface>	9-32
<surfaces>	9-34
<swept_surface>	9-36
<swing_cone_and_twist>	6-38
<target_value>	6-39
<target>	5-32
<targets>	5-118
<technique_common>	5-121
<formula>	5-33
<instance_rigid_body>	6-17
<kinematics_model>	10-26
<kinematics>	10-24
<light>	5-68
<motion>	10-34
<optics>	5-85
<physics_material>	6-25
<physics_scene>	6-29
<rigid_body>	6-34
<rigid_constraint>	6-38
<source>	5-115
概要	3-6
<technique_hint>	8-111
<technique_override>	8-59
<technique>	
概要	3-6
<technique> (FX)	8-110
<technique> (コア)	5-119
<texcombiner>	8-112
<texcoord>	8-47
<texenv>	8-114
<texture_env_color> (レンダリング ステート)	8-106
<texture_env_mode> (レンダリング ステート)	8-106

<texture_pipeline>	8-116
<texture_pipeline> (レンダリング ステート)	8-106
<texture> (シェーダ)	8-45
<texture1D_enable> (レンダリング ステート)	8-106
<texture1D> (レンダリング ステート)	8-106
<texture2D_enable> (レンダリング ステート)	8-106
<texture2D> (レンダリング ステート)	8-106
<texture3D_enable> (レンダリング ステート)	8-106
<texture3D> (レンダリング ステート)	8-106
<textureCUBE_enable> (レンダリング ステート)	8-106
<textureCUBE> (レンダリング ステート)	8-106
<textureDEPTH_enable> (レンダリング ステート)	8-107
<textureDEPTH> (レンダリング ステート)	8-106, 8-107
<textureRECT_enable> (レンダリング ステート)	8-107
<textureRECT> (レンダリング ステート)	8-107
<time_step>	6-29
<title>	5-15
<torus>	9-38
<translate>	5-122
<transparency>	8-45
<transparent>	8-44
<triangles>	5-123
<trifans>	5-124
<tristrips>	5-126
<unit>	5-15
<unnormalized>	8-29
<up_axis>	5-16
<v>	5-128
<value>	
<bind_joint_axis>	10-11
<value> (レンダリング ステート)	8-102
<vcount>	
<faces>	9-18
<pcurves>	9-29
<polylist>	5-96
<shells>	9-30
<solids>	9-32
<vertex_weights>	5-128
<wires>	9-39
<velocity>	6-17
<vertex_weights>	5-128
<vertices>	5-129
<visual_scene>	5-130
<wires>	9-39
<wrap_p>	8-47
<wrap_s>	8-47
<wrap_t>	8-47
<xfov>	5-90
<xmag>	5-86
<yfov>	5-91
<ymag>	5-86
<zfar>	

<orthographic>	5-86
<perspective>	5-91
<znear>	
<orthographic>	5-86
<perspective>	5-91
chnique_common	
<bind_material>	8-13

このページは空白です。