



# **COLLADA – Digital Asset Schema**

## **リリース 1.4.0**

**仕様書**

**2006 年 1 月**

**Editor: Mark Barnes, Sony Computer Entertainment Inc.**

© 2005, 2006 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: January 2006**

Khronos Group  
P.O. Box 1019  
Clearlake Park, CA 95424, U.S.A.

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

本ドキュメントについて .....	vii
対象読者 .....	vii
本ドキュメントの構成 .....	vii
その他の情報源 .....	viii
表記法 .....	viii
<b>1 章 設計上の考慮事項 .....</b>	<b>1-1</b>
概要 .....	1-3
前提条件と依存関係 .....	1-3
目標とガイドライン .....	1-3
開発方法論 .....	1-7
<b>2 章 スキーマの概要 .....</b>	<b>2-1</b>
概要 .....	2-3
コンセプト .....	2-3
アドレス構文 .....	2-3
<b>3 章 スキーマのリファレンス .....</b>	<b>3-1</b>
はじめに .....	3-3
accessor .....	3-4
ambient .....	3-6
animation .....	3-7
animation_clip .....	3-9
asset .....	3-11
bool_array .....	3-13
camera .....	3-14
channel .....	3-16
COLLADA .....	3-17
contributor .....	3-19
controller .....	3-20
directional .....	3-21
extra .....	3-22
float_array .....	3-24
geometry .....	3-25
IDREF_array .....	3-27
image .....	3-28
imager .....	3-30
input .....	3-32
instance_animation .....	3-34
instance_camera .....	3-36
instance_controller .....	3-38
instance_geometry .....	3-41
instance_light .....	3-43
instance_node .....	3-45
instance_visual_scene .....	3-47
int_array .....	3-49
joints .....	3-50
library_animations .....	3-51

library_animation_clips .....	3-52
library_cameras .....	3-53
library_controllers .....	3-54
library_effects .....	3-55
library_force_fields .....	3-56
library_geometries .....	3-57
library_images .....	3-58
library_lights .....	3-59
library_materials .....	3-60
library_nodes .....	3-61
library_physics_models .....	3-62
library_physics_scenes .....	3-63
library_visual_scenes .....	3-64
light .....	3-65
lines .....	3-67
linestrips .....	3-69
lookat .....	3-71
material .....	3-73
matrix .....	3-75
mesh .....	3-76
morph .....	3-78
Name_array .....	3-80
node .....	3-81
optics .....	3-83
orthographic .....	3-84
param .....	3-85
perspective .....	3-86
point .....	3-87
polygons .....	3-88
polylist .....	3-91
rotate .....	3-93
sampler .....	3-94
scale .....	3-96
scene .....	3-97
skeleton .....	3-98
skew .....	3-100
skin .....	3-101
source .....	3-104
spline .....	3-106
spot .....	3-108
targets .....	3-109
technique .....	3-110
translate .....	3-111
triangles .....	3-112
trifans .....	3-114
tristrips .....	3-116
vertex_weights .....	3-118

	vertices .....	3-120
	visual_scene .....	3-121
<b>4 章</b>	<b>共通プロファイル .....</b>	<b>4-1</b>
	概要 .....	4-3
	命名規則 .....	4-3
	共通プロファイル .....	4-3
	インタフェースとしてのパラメータ .....	4-4
	共通用語集 .....	4-4
<b>5 章</b>	<b>ツールの要件とオプション .....</b>	<b>5-1</b>
	概要 .....	5-3
	エクスポート .....	5-4
	インポート .....	5-7
<b>6 章</b>	<b>COLLADA フィジックス .....</b>	<b>6-1</b>
	物理的な単位について .....	6-3
	慣性について .....	6-3
	新しいジオメトリの種類 .....	6-4
	box .....	6-6
	capsule .....	6-7
	convex_mesh .....	6-8
	cylinder .....	6-9
	force_field .....	6-10
	instance_physics_model .....	6-11
	instance_rigid_body .....	6-13
	physics_material .....	6-15
	physics_model .....	6-17
	physics_scene .....	6-20
	plane .....	6-23
	rigid_body .....	6-24
	rigid_constraint .....	6-28
	shape .....	6-34
	sphere .....	6-36
	tapered_capsule .....	6-37
	tapered_cylinder .....	6-38
<b>7 章</b>	<b>COLLADA FX .....</b>	<b>7-1</b>
	レンダリング状態 .....	7-3
	alpha .....	7-9
	annotate .....	7-10
	array .....	7-11
	argument .....	7-12
	bind .....	7-14
	bind_material .....	7-16
	blinn .....	7-18
	code .....	7-20
	color_clear .....	7-21
	color_target .....	7-22
	common_color_or_texture_type .....	7-23
	common_float_or_param_type .....	7-24

compiler_options .....	7-25
compiler_target .....	7-26
connect_param .....	7-27
constant .....	7-28
depth_clear .....	7-30
depth_target .....	7-31
draw .....	7-32
effect .....	7-33
generator .....	7-34
include .....	7-35
instance_effect .....	7-36
instance_material .....	7-37
lamBERT .....	7-38
modifier .....	7-40
name .....	7-41
newparam .....	7-42
param .....	7-43
pass .....	7-44
phong .....	7-46
profile_CG .....	7-48
profile_COMMON .....	7-50
profile_GLES .....	7-52
profile_GLSL .....	7-53
RGB .....	7-54
sampler1D .....	7-55
sampler2D .....	7-56
sampler3D .....	7-57
samplerCUBE .....	7-58
samplerRECT .....	7-59
sampler_state .....	7-60
semantic .....	7-61
setparam .....	7-62
shader .....	7-63
stencil_clear .....	7-65
stencil_target .....	7-66
surface .....	7-67
technique .....	7-68
technique_hint .....	7-70
texcombiner .....	7-71
texenv .....	7-72
texture_pipeline .....	7-73
texture_unit .....	7-75
usertype .....	7-76
VALUE_TYPES .....	7-77
<b>Appendix A キューブの例 .....</b>	<b>1</b>
例: キューブ .....	3
<b>用語集 .....</b>	<b>1</b>

## 本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 COLLABorative Design Activity の略で、複数のソフトウェアパッケージを組み合わせる非常に強力なツールチェーンとして使えるようにするため、情報を損なうことなく 3D オーサリングアプリケーション間で自由にデジタルアセットの交換を可能にする XML ベースのスキーマを定義しています。

本ドキュメントの趣旨は、ソフトウェアプログラマの方が 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。この仕様は、特にビデオゲームや映画業界などで利用されている 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、プロトタイプ化ツール、リアルタイム視覚化アプリケーションなどのインポートやエクスポートを行うために重要です。

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ます。また簡単な例として、COLLADA のインスタンス文書を付録 A に記載してあります。

COLLADA スキーマを利用した文書には、「.dae」（Digital Asset Exchange の頭字語）というファイル拡張子が割り当てられています。この拡張子は、インターネットで検索したところ、これまで利用された例はありません。

## 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。

本ドキュメントでは、以下の知識を持っている方を対象としています。

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

## 本ドキュメントの構成

本ドキュメントは、以下の章から構成されています。

章/節	説明
第 1 章：設計上の考慮事項	COLLADA の設計に関する課題の説明
第 2 章：スキーマの概要	スキーマと、その設計についての概要
第 3 章：スキーマのリファレンス	スキーマについての詳しいリファレンス
第 4 章：共通プロファイル	COLLADA の共通プロファイルの説明
第 5 章：ツールの要件とオプション	ツール作成者に向けた、COLLADA ツールに必要な条件の説明
第 6 章：COLLADA フィジックス	COLLADA フィジックスの要素についての詳しいリファレンス
第 7 章：COLLADA FX	COLLADA FX の要素についての詳しいリファレンス
付録 A: キューブの例	COLLADA 文書の例
用語集	本ドキュメントで利用されている用語の定義
索引	

## その他の情報源

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。

- [Extensible Markup Language \(XML\) 1.0, 2nd Edition](#)
- [XML Schema](#)
- [XML Base](#)
- [XML Path Language](#)
- [XML Pointer Language Framework](#)
- [Extensible 3D \(X3D™\) encodings ISO/IEC FCD 19776-1:200x](#)
- [Softimage® dotXSI™ FTK](#)
- [NVIDIA® Cg Toolkit](#)
- [Pixar's RenderMan®](#)

また、COLLADA の詳細に関しては [www.khronos.org/collada](http://www.khronos.org/collada) を参照してください。

## 表記法

本ドキュメントでは、解説文の意味を明確にするために、全体を通して以下のような表記法が使われています。

記法	説明
明朝体	説明文を示します。
Courier New	クラス名、メソッド名、変数名への参照を示します。
<b>Courier New bold</b>	ファイル名を示します。
<b>&gt; Courier New bold</b>	右角括弧 (>) を先頭に付けて、コマンドを示します。
<a href="#">青色</a>	ハイパーリンクを示します。



---

# 1章 設計上の考慮事項

---

このページは空白です。

---

## 概要

COLLADA デジタルアセットスキーマの開発には、協力して設計活動にあたる多くの会社の設計者やソフトウェア・エンジニアが関係しています。この章では、設計者が作成したより重要な設計目標と概念、さらに想定について再検討しておきましょう。

---

## 前提条件と依存関係

COLLADA アセットスキーマの設計の最初の段階で、参加者はさまざまな議論を行い、以下のような取り決めについて同意しました。

- COLLADA はゲームエンジンのフォーマットではなく、オーサリングツールのユーザと対話型アプリケーションコンテンツ作成用のパイプラインに有益なものと想定しています。また、対話型アプリケーションの多くで、COLLADA は、最終的な配信メカニズムではなく、むしろ製作パイプラインで利用されることを想定しています。たとえば、多くのゲームでは、サイズを最適化した専用のバイナリファイルが利用されています。
- エンドユーザは、頂点プログラムや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が含まれているけれども、比較的簡単なコンテンツやテストモデルを短期間で作成してテストできることを望んでいます。
- エンドユーザの制作環境では、Microsoft Windows®および Linux®オペレーティングシステムが利用すると想定しています。エンドユーザがディベロッパの場合には、ほとんどが C/C++言語でプログラミングを行うことを想定します。したがって、COLLADA のソースコードサンプルは、主にこれらの言語を利用して作成されています。

---

## 目標とガイドライン

COLLADA デジタルアセット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 ベースのオープンソース・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 COLLADA アセットが直接利用でき、ツールチェーンの統合を促進するような標準化された共通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。
- あらゆるデータが COLLADA を介して利用できるような簡単な統合メカニズムを提供すること。
- 3D アプリケーション同士でのデータ交換のための共通基盤となること。
- 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 DCC、ハードウェア、ミドルウェアベンダの間の共同作業の促進剤となること。

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

## 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：**専用のバイナリフォーマットから、正しく仕様化された XML ベースのオープンソースのフォーマットへとデジタルアセットを解放すること。

今日、ほとんどの 3D アプリケーションでは、デジタルアセットが大部分を占めています。

ディベロッパは、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は、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ことも少なくありません。つまり、このようなデータのエクスポートは、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な使命です。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、ゲームディベロッパは、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・ツールを使うことは不可能になります。

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、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 ファイルは、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・サイズの最適化などのために、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論点については、何度も議論が交わされました。けれども、多くの言語では、XML ファイル内部のバイナリデータを簡単にサポートすることもできなければ、バイナリデータ一般の操作もサポートしていないので、残念ながら、バイナリデータを使うことは、大多数のチームに便利であるべきという目標に反しています。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多くの共通データが含まれるように、常に拡張を続ける。

現在、COLLADA では、ポリゴンベースのモデルを共通フォーマットでサポートしています。けれども COLLADA では新たな課題が持ち上がるのに際して、シェーダー効果、物理的特性、パラメトリック曲面など、他の領域もカバーしていくようにしています。

## 標準化された共通フォーマットの提供

**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標準化された共通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、ということです。

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、スキーマや仕様ばかりではなく、COLLADA アセットを既存のツールチェーンに統合するのに役立つようにきちんと設計された API (COLLADA API) を提供する必要がある、という結論に達しました。この新しい開発の方向性は、ディベロッパの労力を省くばかりでなく、新たにさまざまな可能性を開きうるものです。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な、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フォーマットの標準としての COLLADA の地位を強化します。

## できるだけ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

**目標：**できるだけ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 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、彼らが 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。

- 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、ツール供給者がしたが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「ツールの要件とオプション」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。ゲーム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をもたらします。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う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：**あらゆるデータが COLLADA を介して利用できるような、簡単な統合メカニズムを提供すること。

COLLADA には、ディベロッパ固有のニーズに対応できるだけの十分な拡張性が備わっているので、これは、以下のような目標を導くことになります。

- 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 と今後の COLLADA 全体を設計する。
- 拡張が簡単にできる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共通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、ベンダが、そのエクスポートやインポートにベンダ独自の拡張としてそれらの機能を追加することを奨励する。
- これに相当するのは、たとえばアンドウ・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に、すぐにでも必要であるにもかかわらず、COLLADA への導入をまだ検討中であるようなコンセプトなどです。
- このような情報を収集して、次バージョンの COLLADA で共通プロファイル中の問題を解決する。

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できるようにする。
- 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 パッケージ間の共通データ交換の基盤となること。

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す。本書執筆時点では、COLLADA 共通プロファイルは、ポリゴン・ベースのモデル、マテリアルとシェーダ、複数のアニメーション、および DAG ベースのシーングラフをカバーしています。将来は、NURBS や細分割曲面などの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ことを予定しています。

## 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 DCC、ハードウェア、およびミドルウェア・ベンダの間の共同作業の促進剤となること。

DCC ベンダ、ハードウェア・ベンダ、ミドルウェア・ベンダ、ゲームディベロッパなどの各市場区分の内部、ならびに相互間では、激しい競争が行われています。けれども、デジタルコンテンツの問題を解決するためには、彼らすべてが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ことには、ひいては市場全体のソリューションの最適化に直接的な悪影響が出てきます。

- 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。

彼らは、他の関係者と相談して、営業上、もしくは技術上の利点を考慮に入れない限り、共通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者の地位にある Sony Computer Entertainment (SCE) が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。

私たちは、この運動を常に SCE が推進しようとは思っており、時機が来れば、主導者の役割をグループの他のメンバに譲り渡したいと思っています。

- 主導者の役割をあまり早く譲ってしまうと、SCE が COLLADA を見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 に対する管理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。

---

## 開発方法論

COLLADA スキーマの開発のためのアプローチと方法論としては、分析、設計、実装という標準的なウォーターフォールプロセスを採用しています。分析段階においては、現在、業界で使われているツールやフォーマットの比較分析に、かなりの労力が割かれました。

設計段階においては、Microsoft Visual Studio® XML Designer と Altova 社の XMLSPY®ツールを使って、フォーマットのスキーマを繰り返し開発・検証しました。さらに、Altova 社の XMLSPY®を使って、ファイルが COLLADA スキーマにしたがっているかを検証し、文書化用の図面を作成しました。

このページは空白です。



---

## 2章 スキーマの概要

---

このページは空白です。

## 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。

## コンセプト

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、開始タグと終了タグで囲まれた情報ブロックです。以下に例を示しておきます。

```
<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、<node>、<translate>、<rotate>、<matrix>という 4 つの要素が含まれており、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、いま示した例の中の<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”の別の<node>要素と区別することができるようになります。いまの場合だと、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。

- 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 属性によって特定する URI 参照方式を利用しています。
- **animation** 要素の target 属性では、インスタンス文書中の要素を id 属性と sid 属性によって特定する COLLADA 独自の参照方式を利用しています。この方式に C/C++スタイルの構造体メンバの選択構文を加えると、要素の値を参照することができます。

id 属性は、URI フラグメント識別子表記法を使って参照されます。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**ショートハンド・ポインタ**と呼ばれます。ショートハンド・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記号 (#) が付きます。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" />
<skin target="here" />
```

target 属性の構文は、以下のような複数の部分から構成されます。

- 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、1 つまたは複数のサブ識別子が続きます。各サブ識別子の先頭には、パスの区切り文字として、スラッシュ記号 (/) が付加されます。サブ識別子は、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。ネスト (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、複数のサブ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。
    - メンバ指定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。第 4 章の一般用語集には、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。
  - 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。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「)」。

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。以下に、target 属性構文の例をいくつか示します。

```
<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>
```

各<channel>要素は、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。

柔軟性と簡潔性のために、ターゲットのアドレッシング・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。実際、いくつかの要素では id 属性や sid 属性が指定できません。

また、対象となるテクニックごとにわざわざ余分にアニメーション・チャンネルを作成することなく、複数のテクニックの中のカメラの `yfov` を対象とすることも可能です（テクニックというのは「スイッチ」のことで、一方または他方がインポート時に選択され、両方が選択されることはありません。したがって、対象となるのは1つだけです）。

例：

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

それぞれのテクニッ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、ディベロッパが別個のアニメーション・チャンネルを使用できることは言うまでもありません。

このページは空白です。

---

## 3章 スキーマのリファレンス

---

このページは空白です。



---

## はじめに

本章では、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。スキーマ中の個々の 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	説明
概要	要素の名前と目的について簡単に述べてあります。
コンセプト	要素が定義された背景と意義について述べてあります。
属性	要素に適用可能な属性について述べてあります。
要素	要素の制約事項や相互関係について述べてあります。
備考	要素を使用する上で関係した情報を述べてあります。
例	要素の使い方を紹介してあります。

## accessor

### 概要

`<accessor>` 要素は、`<float_array>`、`<int_array>`、`<Name_array>`、`<bool_array>`、`<IDREF_array>`のいずれかの配列要素へのアクセスパターンを宣言するためのものです。`offset` と `stride` のどちらかの属性を指定して、インターリーブ方式または非インターリーブ方式で構成された配列へのアクセスを記述します。

### コンセプト

`<accessor>`要素では、配列データソースから値のストリームを記述します。このアクセッサの出力は、`<param>`子要素で記述されます。

### 属性

`<accessor>`要素には、以下の属性があります。

count	xs:nonNegativeLong
offset	xs:nonNegativeLong
source	xs:anyURL
stride	xs:nonNegativeLong

count は必須の属性で、配列へのアクセス回数を表します。

offset はオプションの属性で、配列から最初に読み取る値のインデックスを表します。デフォルト値は0です。

source は必須の属性で、URL 表記を利用して、アクセスする配列の場所を表します。

stride はオプションの属性で、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を表します。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。

### 関連要素

`<accessor>`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	technique_common
子要素	<code>&lt;param&gt;</code>
その他	なし

### 備考

`<param>`要素は任意の数だけ利用でき、まったく指定しなくてもかまいません。

`<param>`要素の数と順番で、`<accessor>`要素の出力が定義されることになります。

それぞれのパラメータは、指定されている順番で各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>`要素はバインドされず、その値は出力の一部ではないことを意味します。

`<param>`要素の type 属性は、`<accessor>`要素の子である場合、int、float、Name、bool、IDREF の型の配列セットに限定されます。

`<accessor>`要素の source 属性で、インスタンス文書の範囲外にある配列データソースを参照してもかまいません。

stride 属性は、`<param>`要素の数と等しいかまたは大きい値でなければなりません。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ておきます。

```
<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>
```

また、以下は 3 つの整数値のペアのストリームを表した<accessor>要素の例で、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 name="int" />
      <param name="C" type="int" />
    </accessor>
  </technique_common>
</source>
```

さらに以下の例では、stride 属性が 3 だけれども、3 番目の値を出力とバインドする<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="A" type="int" />
      <param name="B" type="int" />
    </accessor>
  </technique_common>
</source>
```

---

## ambient

### 概要

<ambient>要素は、環境光源を表すためのものです。

### コンセプト

<ambient>要素で、環境光源を表すために必要なパラメータを宣言します。環境光源というのは、位置や方向に関係なく、すべてに対して均一に照明が当たる光のことです。

### 属性

<ambient>要素に属性はありません。

### 関連要素

<ambient>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	technique_common
子要素	color
その他	なし

### 備考

<ambient>要素は、<light>要素の下の<technique\_common>の子としてだけ利用できます。<color>要素を 1 つだけ利用しなければなりません。<color>要素には、光源のカラーを表す 3 つの浮動小数点を含めておきます。sid 属性を指定することも可能です。

### 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>
  </light>
```

## animation

### 概要

`<animation>`要素は、アニメーション情報の宣言をカテゴリ化するためのものです。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ラ関数を記述するための要素が含まれます。これらの要素は、一緒に実行すべきアニメーションをグループ化して提供するように順番付けられています。

### コンセプト

アニメーションは、時間の経過にしたがって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は、キーフレームアニメーションです。

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、時間です。ただし、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。キーフレームのセットと補間アルゴリズムを利用してキーフレーム間の時間に対して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。これらのキーフレームセットとその間の補間によって、「アニメーション曲線」や「関数曲線」と呼ばれる2次元の関数を定義することになります。

### 属性

`<animation>`要素には、以下の属性があります。

id	xs:ID
name	xs:NCName

id はオプションの属性で、`<animation>`要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<animation>`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>library_animations</code> 、 <code>animation</code>
子要素	<code>asset</code> 、 <code>animation</code> 、 <code>source</code> 、 <code>sampler</code> 、 <code>channel</code> 、 <code>extra</code>
その他	なし

### 備考

`<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。データの実際の型と複雑度の詳細を表すのは、子要素に任せられています。

`<animation>`要素には、0個以上の`<source>`要素を記述することができます。

`<animation>`要素には、0個以上の`<sampler>`要素を記述することができます。

`<animation>`要素には、0個以上の`<channel>`要素を記述することができます。

`<asset>`要素はまったく記述しないか、一度だけ記述することができます。

`<extra>`要素はまったく記述しないか、任意の数だけ記述することができます。

それぞれの子要素は、`<asset>`、`<source>`、`<sampler>`、`<channel>`、`<animation>`、`<extra>`の順番でなければなりません。

## 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">
    <source />
    <source />
    <sampler />
    <channel />
  </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 id="left_hip_rotation">
      <source/><source/><sampler/><channel/>
    </animation>
    <animation id="left_knee_rotation">
      <source/><source/><sampler/><channel/>
    </animation>
    <animation id="right_hip_rotation">
      <source/><source/><sampler/><channel/>
    </animation>
    <animation id="right_knee_rotation">
      <source/><source/><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、もっと複雑なアニメーションツリーの例です。

```
<library_animations>
  <animation name=" elliot's animations" id="all_elliot">
    <animation name="elliot's spells" id="spells_elliot">
      <animation id="elliot_fire_blast"/>
      <animation id="elliot_freeze_down"/>
      <animation id="elliot_ferocity"/>
    </animation>
    <animation name="elliot's moves" id="moves_elliot">
      <animation id="elliot_walk"/>
      <animation id="elliot_run"/>
      <animation id="elliot_jump"/>
    </animation>
  </animation>
</library_animations>
```

## animation\_clip

### 概要

`<animation_clip>`要素は、アニメーションクリップとして一緒に利用するアニメーション曲線の一部を定義するためのものです。

### 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の1つのセットから異なる部分を切り分けるのに利用します。たとえば、キャラクターが最初は歩いており、途中から走り出すアニメーションがある場合、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を2つの別々のクリップとして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ができません。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
start	xs:double
end	xs:double

id はオプションの属性で、`<animation>`要素のユニークな識別子を含んだ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。

start はオプションの属性で、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判断するのに利用されます。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。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時間が位置する場合には、補間された値が利用されることになります。デフォルト値は0.0です。

end はオプションの属性で、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。

### 関連要素

`<animation_clip>`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>library_animation_clips</code>
子要素	<code>asset</code> 、 <code>instance_animation</code> 、 <code>extra</code>
その他	なし

### 備考

`<asset>`要素は、まったく指定しないか、もしくは1つだけ指定しなければなりません。

`<instance_animation>`要素は、1つまたは複数指定しなければなりません。

`<extra>`要素は、まったく指定しないか、もしくは任意の数だけ指定できます。

それぞれの子要素の順番は、`<asset>`、`<instance_animation>`、`<extra>`でなければなりません。

## 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

### 概要

`<asset>`要素は、その親要素に関連したアセット管理情報を定義するためのものです。

### コンセプト

コンピュータは、膨大な量の情報を保存しており、アセットというのは、識別可能で1つの単位として管理できるように組織化された情報セットのことです。そういった情報をソフトウェアツールと人間が管理して理解できるようにするために、さまざまな属性を利用してアセットを表します。

### 属性

`<asset>`要素に属性はありません。

### 関連要素

`<asset>`要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<code>camera</code> 、 <code>COLLADA</code> 、 <code>light</code> 、 <code>material</code> 、 <code>technique</code> 、 <code>texture</code>
子要素	<code>contributor</code> 、 <code>created</code> 、 <code>keywords</code> 、 <code>modified</code> 、 <code>revision</code> 、 <code>subject</code> 、 <code>title</code> 、 <code>units</code> 、 <code>up_axis</code>
その他	なし

`contributor` 要素には、その親要素に貢献した人に関連したデータが含まれます。`contributor` 要素は、まったく指定しないか、もしくは複数記述してもかまいません。

`created` 要素には、親要素が作成された日付と時刻が XML スキーマの `dateTime` プリミティブ型として ISO 8601 形式で表現されて含まれます。`created` 要素は、まったく指定しないか、もしくは1つだけ記述することができます。

`modified` 要素には、親要素が最後に修正された日付と時刻が XML スキーマの `dateTime` プリミティブ型として ISO 8601 形式で表現されて含まれます。`modified` 要素は、まったく指定しないか、もしくは1つだけ記述することができます。

`revision` 要素には、親要素のためのリビジョン情報が含まれます。`revision` 要素は、まったく指定しないか、もしくは1つだけ記述することができます。

`title` 要素には、親要素のタイトル情報が含まれます。`title` 要素は、まったく指定しないか、もしくは1つだけ記述することができます。

`subject` 要素には、親要素のトピックスに関して記述した情報が含まれます。`subject` 要素は、まったく指定しないか、もしくは1つだけ記述することができます。

`keywords` 要素には、親要素を検索する際の条件に利用される言葉のリストが含まれます。`keywords` 要素は、まったく指定しないか、もしくは複数記述してもかまいません。

`unit` 要素には、測定単位について記述した情報が含まれます。単位の名前 (`name`) とメートル法換算時の値 (`meter`) を表す属性があります。`unit` 要素は、要素は、まったく指定しないか、もしくは1つだけ記述することができます。`name` 属性のデフォルト値は「`meter`」です。また、`meter` 属性のデフォルト値は「`1.0`」です。

up\_axis 要素には、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。X\_UP、Y\_UP、Z\_UP のいずれかが指定できます。この要素は、それぞれどの軸を上、どの軸を右、どの軸を内側とみなすのかを指定します。

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

デフォルト値は Y\_UP です。up\_axis 要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

## 備考

それぞれの子要素は、<contributor>、<created>、<keywords>、<modified>、<revision>、<subject>、<title>、<units>、<up\_axis>の順番となります。

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

## bool\_array

### 概要

`<bool_array>`要素は、Boolean 値の同種の配列用の保存場所を宣言するためのものです。

### 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とまいません。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	xs:nonNegativeLong
id	xs:ID
name	xs:NCName

count は必須の属性で、配列中の値の数を表します。

id はオプションの属性で、この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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<bool_array>`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	<a href="#">source</a>
子要素	子要素はありません
その他	なし

### 備考

`<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>`要素へ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

### 概要

`<camera>`要素は、シーン階層またはシーングラフへのビューを宣言するためのものです。`<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

カメラというのは、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、カメラの光学系レンズから見たシーンとして、これがカメラのビューポイントとなります。カメラの光学系は、入射光の焦点をイメージに合わせます。イメージの焦点は、カメラの撮影装置（フィルム）の面に合わせられ、結果として生じるイメージ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。

<code>id</code>	<code>xs:ID</code>
<code>name</code>	<code>xs:NCName</code>

`id` はオプションの属性で、`<camera>`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。

`name` はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<camera>`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>library_cameras</code>
子要素	<code>asset</code> 、 <code>optics</code> 、 <code>imager</code> 、 <code>extra</code>
その他	なし

### 備考

カメラテクニックの要素には、`<optics>`要素が 1 つ、また 0 個か 1 つの`<imager>`要素が含まれていなければなりません。

単純なカメラの場合、汎用的なテクニックでは、標準パラメータを利用して視野と視野錐台を記述した`<optics>`要素だけが含まれていれば問題ありません。

`<asset>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

`<extra>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

### 例

以下は、視野角 45 度で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

### 概要

<channel>要素は、アニメーションの出力チャネルを宣言するためのものです。

### コンセプト

時間の経過とともにアニメーションがその値を変換していくのにもなって、チャネルに対してその値が出力されます。アニメーションのチャネルでは、変更された値をアニメーションエンジンからどこに保存するのか、その場所を表します。チャネルは、アニメーションされた値を受け取るデータ構造を対象とします。

### 属性

<channel>要素には、以下の属性があります。

id	xs:ID
name	xs:NCName
source	xs:anyURL
target	xs:token

id はオプションの属性で、<channel>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。

name はオプションの属性で、この要素のテキスト文字列名です。

source は必須の属性で、URL 表記を利用して、サンプラの場所を表します。

target は必須の属性で、サンプラの出力にバインドされた要素の場所を表します。このテキスト文字列は、第 2 章の「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。

### 関連要素

<channel>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	<a href="#">animation</a>
子要素	子要素はありません
その他	なし

### 備考

アドレス形式を表す target 属性については、「アドレス構文」のを参照してください。

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

## COLLADA

### 概要

COLLADA スキーマは XML をベースとしています。そのため、整形 XML ドキュメントにするには「ドキュメントルート要素」、つまりドキュメントのエントリが必要となります。

### コンセプト

<COLLADA>要素は、COLLADA スキーマに準拠しているコンテンツであることを表すドキュメントルートを宣言するためのものです。

### 属性

<COLLADA>要素には、以下の属性があります。

version	xs:string
xmlns	xs:anyURI

version は必須の属性で、インスタンス文書が準拠している COLLADA スキーマのバージョンです。現時点で指定可能な値は 1.4.0 だけです。

インスタンス文書のスキーマを特定するために、この要素に XML Schema 名前空間属性の xmlns が指定できます。

### 関連要素

<COLLADA>要素は、以下の要素と関連性があります。

出現回数	1 回
親要素	親要素はありません
子要素	asset、library_animations、library_animation_clips、library_cameras、library_controllers、library_effects、library_force_fields、library_geometries、library_images、library_lights、library_materials、library_nodes、library_physics_materials、library_physics_models、library_physics_scenes、library_visual_scenes、scene、extra
その他	なし

### 備考

<COLLADA>要素は、COLLADA インスタンス文書中のドキュメントエントリ（ルート要素）です。

ドキュメントルートには、<asset>要素が 1 つ含まれていなければなりません。

ドキュメントルートには、<library\_\*>要素をまったく含めないか、または任意の数だけ含めることができます。

ドキュメントルートには、<scene>要素をまったく含めないか、または 1 つだけ含めることができます。

それぞれの子要素を指定する場合、以下の順番で指定しなければなりません。

1. <asset>要素
2. <library\_\*>要素
3. <scene>要素

## 例

以下は、スキーマのバージョンが「1.4.0」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/2005/10/COLLADASchema" version="1.4.0">
  <asset>
    <created/>
    <modified/>
  </asset>
  <library_geometries/>
  <library_visual_scenes/>
  <scene />
</COLLADA>
```



## contributor

### 概要

<contributor>要素は、アセット管理のために作成者情報を定義するためのものです。

### コンセプト

現在の制作ラインでは、特にアートのサイズが着実に大きくなるのにもなって、複数の人間がおそらく複数のツールを利用して、1つのアセットに携わることが増えてきています。そのため、この作成者情報は、アセット管理システムにとって重要な意味を持ちます。

### 属性

<contributor>要素に属性はありません。

### 関連要素

<contributor>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">asset</a>
子要素	author、authoring_tool、comments、copyright、source_data
その他	なし

### 備考

それぞれの子要素は、<author>、<authoring\_tool>、<comments>、<copyright>、<source\_data>の順番となります。

### 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>
    <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

### 概要

<controller>要素は、汎用的な制御情報の宣言をカテゴリ化するためのものです。

### コンセプト

コントローラというのは、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、つまりメカニズムです。

### 属性

<controller>要素には、以下の属性があります。

id	xs:ID
name	xs:NCName

id はオプションの属性で、<controller>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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

<controller>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	library_controllers
子要素	asset、skin、morph、extra
その他	なし

### 備考

<controller>要素には、制御データを表す要素が含まれます。データの実際の型と複雑度の詳細を表すのは、子要素に任せられています。

<asset>要素は、まったく指定しないか、もしくは1つだけ記述することができます。

<skin>または<morph>の要素は1つだけ指定できますが、どちらか片方しか指定できません。

<extra>要素は、任意の数だけ指定できます。

それぞれの子要素は、<asset>、<skin>または<morph>、<extra>の順番となります。

### 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>
```

---

## directional

### 概要

`<directional>`要素は、方向性光源を表すためのものです。

### コンセプト

`<directional>`要素は、方向性光源を表すために必要なパラメータを宣言します。方向性光源というのは、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、Z 軸へ進みます。光源の実際の方向は、光源がインスタンス化されたノードの変換で定義されます。

### 属性

`<directional>`要素に属性はありません。

### 関連要素

`<directional>`要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<code>technique_common</code>
子要素	<code>color</code>
その他	なし

### 備考

`<directional>`要素は、`<light>`要素の下で`<technique_common>`の子としてだけ指定できます。

`<color>`要素を 1 つだけ指定しなければなりません。この`<color>`要素には、光源のカラーを指定する 3 つの浮動小数点を含めておきます。sid 属性を指定することもできます。

### 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>
```

---

## extra

### 概要

<extra>要素は、親要素に関係した追加情報を宣言するためのものです。

### 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<extra>要素を利用すると、アプリケーションへの実際の補足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<extra>要素の追加情報を、任意の XML 要素やデータを含んだテクニック（technique）として表現するようになっています。

### 属性

<extra>要素には、以下の属性があります。

id	xs:ID
name	xs:NCName
type	xs:NMTOKEN

id はオプションの属性で、<extra>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。

name はオプションの属性で、この要素のテキスト文字列名です。

type はオプションの属性で、データ値の型を表します。このテキスト文字列は、アプリケーションが理解できるものでなければなりません。

### 関連要素

<extra>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	COLLADA、scene、library_animations、library_animation_clips、library_cameras、library_controllers、library_effects、library_force_fields、library_geometries、library_images、library_lights、library_materials、library_nodes、library_physics_materials、library_physics_models、library_physics_scenes、library_visual_scenes、animation、animation_clip、instance_animation、camera、optics、imager、controller、skin、morph、joints、vertex_weights、targets、convex_mesh、mesh、spline、profile_COMMON、profile_COMMON::technique、texture、force_field、image、light、material、instance_effect、node、instance_camera、instance_controller、instance_geometry、instance_light、instance_node、physics_material、physics_model、rigid_body、rigid_constraint、instance_physics_model、physics_scene、instance_force_field、visual_scene、instance_physics_scene、instance_visual_scene
子要素	technique
その他	なし

## 備考

<technique>要素は任意の数だけ利用でき、まったく指定しなくてもかまいません。

## 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アウトライン化した<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>
```

## float\_array

### 概要

`<float_array>`要素は、浮動小数点の値を含んだ同種の配列用の保存場所を宣言するためのものです。

### 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ともしません。単純に、浮動小数点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	xs:unsignedLong
id	xs:ID
name	xs:NCName
digits	xs:short
magnitude	xs:short

count は必須の属性で、配列中の値の数を表します。

id はオプションの属性で、この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。

name はオプションの属性で、この要素のテキスト文字列名です。

digits はオプションの属性で、配列中に含めることが可能な浮動小数点の有効桁数（10 進数）を表します。デフォルト値は 6 です。

magnitude はオプションの属性で、配列中に含めることが可能な浮動小数点の最大指数を表します。デフォルト値は 38 です。

### 関連要素

`<float_array>`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	<a href="#">source</a>
子要素	子要素はありません
その他	なし

### 備考

`<float_array>`要素には、浮動小数点の値のリストが含まれます。これらの値は、`<source>`要素へのデータのリ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>
```

## geometry

### 概要

<geometry>要素は、ジオメトリ情報の宣言をカテゴリ化するためのものです。ジオメトリというのは、点、線、角度、面、立体などの計測やプロパティや関係などを扱うための一連の数学です。<geometry> 要素には、メッシュや凸状メッシュやスプラインの宣言が含まれます。

### コンセプト

ジオメトリの記述には、さまざまな形式があります。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、さまざまな属性（カラーや法線など）を持った頂点の情報を受け付けるように標準化されています。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ておきます。

- スプライン
- ベジエ
- 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だけ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
name	xs:NCName

id はオプションの属性で、<geometry>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。

name はオプションの属性で、<geometry>要素の名前を含むテキスト文字列です。

### 関連要素

<geometry>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	library_geometries
子要素	asset、mesh、convex_mesh、extra、spline
その他	なし

### 備考

<geometry>要素には、ジオメトリデータを表す要素が含まれます。データの実際の型と複雑度の詳細を表すのは、子要素に任せられています。

<asset>要素は、まったく指定しないか、もしくは1つだけ記述することができます。

<convex\_mesh>、<mesh>、または<spline>の要素は、1つだけ記述しなければなりません。いずれか1つだけが記述できます。

<extra>要素は、任意の数だけ指定してかまいません。

## 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/>  
  </geometry>  
</library_geometries>
```



---

## IDREF\_array

### 概要

<IDREF\_array>要素は、ID 参照値を含んだ同種の配列の保存場所を宣言するためのものです。

### 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	xs:unsignedLong
id	xs:ID
name	xs:NCName

count は必須の属性で、配列中の値の数を表します。

id はオプションの属性で、この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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

<IDREF\_array>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	<a href="#">source</a>
子要素	子要素はありません
その他	なし

### 備考

<IDREF\_array>要素には、XML IDREF の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、ドキュメント中の<[node](#)>要素を参照する<IDREF\_array>要素の例です。

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

## image

### 概要

<image>要素は、オブジェクトのグラフィカルな表現用の保存場所を宣言するためのものです。  
 <image>要素は、ラスター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。

### コンセプト

デジタル画像データは、大きく分けて、ラスター、ベクトル、ハイブリッドの3種類の形式があります。ラスター画像は、画素（ピクセル）と呼ばれ輝度やカラー値の並びから構成され、このピクセルが集まって1つのピクチャーを構成しています。またベクトル画像では、曲線、直線、形状など、数学的な式を利用して、ピクチャーや図を記述します。さらにハイブリッド画像は、ラスター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ピクチャーを記述します。ラスター画像データは、N次元の配列として構成されます。テクスチャの参照を行う関数を利用することで、変位、法線、高さなどのフィールド値といった、カラー以外の値にもアクセスでき、この配列構造を有効に利用することが可能です。

### 属性

<image>要素には、以下の属性があります。

id	xs:ID
name	xs:NCName
format	xs:string
height	xs:nonNegativeInteger
width	xs:nonNegativeInteger
depth	xs:nonNegativeInteger

id はオプションの属性で、<image>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。  
 name はオプションの属性で、この要素のテキスト文字列名です。  
 format はオプションの属性で、画像形式を表す文字列値です。  
 height はオプションの属性で、ピクセルを単位として画像の高さを表す整数値です。  
 width はオプションの属性で、ピクセルを単位として画像の幅を表す整数値です。  
 depth はオプションの属性で、ピクセルを単位として画像の深さを表す整数値です。2次元の画像の場合、深さは1です。これがデフォルト値となっています。

### 関連要素

<image>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	library_images、effect、profile_CG、profile_GLSL、profile_COMMON、profile_GLES、technique
子要素	asset、data、init_from、extra
その他	なし

## 備考

<image>要素では、<init\_from>を指定するか、もしくは<data>で画像データを埋め込むことで、外部の画像ファイルを指定することもできます。  
<data>子要素には、埋め込まれた画像データを表す 16 進数の符号化されたバイナリオクテットの並びが含まれます。

## 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>Textures/WoodFloor-01.png</init_from>  
  </image>  
</library_images>
```

## imager

### 概要

<imager>要素は、たとえば、フィルムや CCD といったカメラの画像センサーを表すためのものです。

### コンセプト

カメラの光学系は、画像を（通常は平面の）センサーに投影します。<imager>要素は、このセンサーが光のカラーと強度をどのように数値化するかを定義するためのものです。

実際の光の強度は、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。また、無限に多様化した波長の光子を含むこともあります。これに対して表示装置は、非常に限定されたダイナミックレンジを利用するようになっており、普通、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、画像合成の最後のステップ（レンダリング）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数として表し、実際のピクセルのカラーを float3s または float の配列（マルチスペクトル・レンダラ）として保存するようになっており、その後、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 画像を作成するために、トーンマッピングを実行します。なお、多くのレンダラは、ハイダイナミックレンジ（HDR）のオリジナル画像を後で「再露光」できるように保存することもできます。

### 関連要素

<imager>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	camera
子要素	technique、extra
その他	なし

### 備考

<imager>要素は、オプションです。共通プロファイルでは省略されており、デフォルトの解釈は、以下のようになっています。

- 輝度の線形マッピング
- 0..1 の範囲へのクランプ（コンポーネントのフレームバッファごとに 8 ビットという条件で、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 センサ付きの本物そっくりのカメラを記述する<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

### 概要

`<input>`要素は、データソースの入力セマンティクスを宣言するためのものです。

### コンセプト

`<input>`要素では、コンシューマが必要とする入力結び付きを宣言します。

### 属性

`<input>`要素には、以下の属性があります。

offset	xs:unsignedLong
semantic	xs:NMTOKEN
source	xs:anyURL
set	xs:unsignedLong

offset 属性は、インデックスのリストへのオフセットを表します。2 つの`<input>`要素が同じオフセットを共有している場合、同じ値にインデックス化されます。これは、インデックスリスト用の簡単な圧縮形式として機能すると同時に、インデックスの利用される順番を定義することになります。

`<input>` 要素が `<lines>`、`<linestrips>`、`<polygons>`、`<polylist>`、`<triangles>`、`<trifans>`、`<tristrips>`のいずれかの要素の子の場合、offset は必須の属性となります。

semantic は必須の属性で、入力の結び付きの意味をユーザ定義するものです。

source は必須の属性で、データソースの場所を表します。

set 属性は、どの入力を単一セットとしてグループ化すべきなのかを表します。これは、複数の入力と同じセマンティクスを共有する場合に役立ちます。

### 関連要素

`<input>`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>joints</code> 、 <code>lines</code> 、 <code>linestrips</code> 、 <code>polygons</code> 、 <code>polylist</code> 、 <code>sampler</code> 、 <code>targets</code> 、 <code>triangles</code> 、 <code>trifans</code> 、 <code>tristrips</code> 、 <code>vertex_weights</code> 、 <code>vertices</code>
子要素	子要素はありません
その他	なし

### 備考

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

`<input>`要素には、値が COLOR の semantic 属性を指定できます。それらのカラー入力は RGB (float3) です。

`<extra>`要素は、任意の数だけ指定できます。

## 例

以下は、頂点の位置と法線、それに<polygon>要素のテクスチャ空間接線とともに、テクスチャ座標の2つのセットを表す6つの<input>要素の例です。

```
<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>
```

## instance\_animation

### 概要

<instance\_animation>要素は、COLLADA アニメーションリソースのインスタンス化を宣言するためのものです。

### コンセプト

オブジェクトの実際のデータ表現を一度しか保存しない場合もありますが、オブジェクトはシーン中で複数表すことができます。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ます。それぞれのオブジェクトのインスタンスは、ユニークであったり、他のインスタンスと同じデータを共有する場合があります。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。ユニークではない（共有されている）インスタンスは、そのオブジェクトの他のインスタンスとデータの一部またはすべてを共有します。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。

### 属性

<instance\_animation>要素には、以下の属性があります。

url                      xs:anyURL

url は必須の属性で、インスタンス化するオブジェクトの場所を URL で表します。

### 関連要素

<instance\_animation>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	<a href="#">animation_clip</a>
子要素	<a href="#">extra</a>
その他	なし

### 備考

url 属性は、#文字で始まる相対 URL 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。断片的な識別子は、インスタンス化する要素の ID で構成された XPointer のショートハンド・ポインタです。

url 属性で他のリソースへのパスを含む外部参照を行う際には、絶対 URL もしくは相対 URL を使います。COLLADA では、インスタンス同士でデータを共有する方法を規定していません。このデータ共有の方法は、ランタイム時のアプリケーションに任されています。



## 例

以下は、「anim」という ID で識別されたローカル定義の<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>
</library_animation_clips>
```

## instance\_camera

### 概要

<instance\_camera>要素は、COLLADA カメラリソースのインスタンス化を宣言するためのものです。

### コンセプト

オブジェクトの実際のデータ表現が一度しか保存されない場合もありますが、オブジェクトを複数シーン中で表示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ます。

それぞれのオブジェクトのインスタンスは、ユニークであったり、他のインスタンスと同じデータを共有する場合があります。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。ユニークではない（共有されている）インスタンスは、そのオブジェクトの他のインスタンスとデータの一部またはすべてを共有します。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。

### 属性

<instance\_camera>要素には、以下の属性があります。

url                      xs:anyURL

url は必須の属性で、インスタンス化するオブジェクトの場所を URL で表します。

### 関連要素

<instance\_camera>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	node
子要素	extra
その他	なし

### 備考

url 属性は、#文字で始まる相対 URL 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。断片的な識別子は、インスタンス化する要素の ID で構成された XPointer のショートハンド・ポインタです。

url 属性で他のリソースへのパスを含む外部参照を行う際には、絶対 URL もしくは相対 URL を使います。COLLADA では、インスタンス同士でデータを共有する方法を規定していません。このデータ共有の方法は、ランタイム時のアプリケーションに任されています。

## 例

以下は、「cam」という ID で識別されたローカル定義の<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

### 概要

`<instance_controller>`要素は、COLLADA コントローラリソースのインスタンス化を宣言するためのものです。

### コンセプト

オブジェクトの実際のデータ表現が一度しか保存されない場合もありますが、オブジェクトを複数シーン中で表すことができます。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ます。それぞれのオブジェクトのインスタンスは、ユニークであったり、他のインスタンスと同じデータを共有する場合があります。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。ユニークではない（共有されている）インスタンスは、そのオブジェクトの他のインスタンスとデータの一部またはすべてを共有します。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。

### 属性

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

url                      xs:anyURL

url は必須の属性で、インスタンス化するオブジェクトの場所を URL で表します。

### 関連要素

`<instance_controller>`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>node</code>
子要素	<code>bind_material</code> 、 <code>skeleton</code> 、 <code>extra</code>
その他	なし

### 備考

url 属性は、#文字で始まる相対 URL 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。断片的な識別子は、インスタンス化する要素の ID で構成された XPointer のショートハンド・ポインタです。

url 属性で他のリソースへのパスを含む外部参照を行う際には、絶対 URL もしくは相対 URL を使います。`<skeleton>`要素は、必要な joint ノードの検索を開始するために、スキンコントローラのある場所を表すのに利用されます。この要素は、モーフィングコントローラでは意味がありません。

`<extra>`要素は、任意の数だけ利用できます。

## 例

以下は、「skin」と識別されたローカル定義の<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と識別された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>
<node>
  <instance_controller url="#skin"/>
    <skeleton>#skel02</skeleton>
  </instance_controller>
</node>
```

## instance\_geometry

### 概要

`<instance_geometry>`要素は、COLLADA ジオメトリリソースのインスタンス化を宣言するためのものです。

### コンセプト

オブジェクトの実際のデータ表現が一度しか保存されない場合もありますが、オブジェクトを複数シーン中で表すことができます。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ます。それぞれのオブジェクトのインスタンスは、ユニークであったり、他のインスタンスと同じデータを共有する場合があります。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。ユニークではない（共有されている）インスタンスは、そのオブジェクトの他のインスタンスとデータの一部またはすべてを共有します。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。

### 属性

`<instance_geometry>`要素には、以下の属性があります。

url                      xs:anyURL

url は必須の属性で、インスタンス化するオブジェクトの場所を URL で表します。

### 関連要素

`<instance_geometry>`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>node</code>
子要素	<code>bind_material</code> 、 <code>extra</code>
その他	なし

### 備考

url 属性は、#文字で始まる相対 URL 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。断片的な識別子は、インスタンス化する要素の ID で構成された XPointer のショートハンド・ポインタです。

url 属性で他のリソースへのパスを含む外部参照を行う際には、絶対 URL もしくは相対 URL を使います。`<bind_material>`要素は、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するのに利用します。この要素で、単一のジオメトリを、毎回、異なった表示でシーン中に複数インスタンス化することができます。

## 例

以下は、「cube」という ID で識別されたローカル定義の<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

### 概要

`<instance_light>`要素は、COLLADA 光源リソースのインスタンス化を宣言するためのものです。

### コンセプト

オブジェクトの実際のデータ表現が一度しか保存されない場合もありますが、オブジェクトを複数シーン中で表示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ます。

それぞれのオブジェクトのインスタンスは、ユニークであったり、他のインスタンスと同じデータを共有する場合があります。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。ユニークではない（共有されている）インスタンスは、そのオブジェクトの他のインスタンスとデータの一部またはすべてを共有します。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。

### 属性

`<instance_light>`要素には、以下の属性があります。

url                      xs:anyURL

url は必須の属性で、インスタンス化するオブジェクトの場所を URL で表します。

### 関連要素

`<instance_light>`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>node</code>
子要素	<code>extra</code>
その他	なし

### 備考

url 属性は、#文字で始まる相対 URL 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。断片的な識別子は、インスタンス化する要素の ID で構成された XPointer のショートハンド・ポインタです。

url 属性で他のリソースへのパスを含む外部参照を行う際には、絶対 URL もしくは相対 URL を使います。COLLADA では、インスタンス同士でデータを共有する方法を規定していません。このデータ共有の方法は、ランタイム時の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 で識別されたローカル定義の<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

### 概要

`<instance_node>`要素は、COLLADA ノードリソースのインスタンス化を宣言するためのものです。

### コンセプト

オブジェクトの実際のデータ表現が一度しか保存されない場合もありますが、オブジェクトを複数シーン中で表示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ます。

それぞれのオブジェクトのインスタンスは、ユニークであったり、他のインスタンスと同じデータを共有する場合があります。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。ユニークではない（共有されている）インスタンスは、そのオブジェクトの他のインスタンスとデータの一部またはすべてを共有します。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。

### 属性

`<instance_node>`要素には、以下の属性があります。

url                   xs:anyURL

url は必須の属性で、インスタンス化するオブジェクトの場所を URL で表します。

### 関連要素

`<instance_node>`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>node</code>
子要素	<code>extra</code>
その他	なし

### 備考

url 属性は、#文字で始まる相対 URL 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。断片的な識別子は、インスタンス化する要素の ID で構成された XPointer のショートハンド・ポインタです。

url 属性で他のリソースへのパスを含む外部参照を行う際には、絶対 URL もしくは相対 URL を使います。

`<extra>`要素は、任意の数だけ利用できます。

## 例

以下は、「myNode」という ID で識別されたローカル定義の<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

### 概要

`<instance_visual_scene>`要素は、COLLADA の `visual_scene` リソースのインスタンス化を宣言するためのものです。

### コンセプト

オブジェクトの実際のデータ表現が一度しか保存されない場合もありますが、オブジェクトを複数シーン中で表すことができます。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ます。それぞれのオブジェクトのインスタンスは、ユニークであったり、他のインスタンスと同じデータを共有する場合があります。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。ユニークではない（共有されている）インスタンスは、そのオブジェクトの他のインスタンスとデータの一部またはすべてを共有します。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。

### 属性

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

`url`                      `xs:anyURL`

`url` は必須の属性で、インスタンス化するオブジェクトの場所を URL で表します。

### 関連要素

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

出現回数	スキーマ中に定義された要素の数
親要素	<code>scene</code>
子要素	<code>extra</code>
その他	なし

### 備考

`url` 属性は、#文字で始まる相対 URL 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。断片的な識別子は、インスタンス化する要素の ID で構成された XPointer のショートハンド・ポインタです。

`url` 属性で他のリソースへのパスを含む外部参照を行う際には、絶対 URL もしくは相対 URL を使います。

## 例

以下は、「vis\_scene」という ID で識別されたローカル定義の<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

### 概要

`<int_array>`要素は、整数値を保持する同種の配列用の保存場所を宣言するためのものです。

### 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ともしません。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	xs:nonNegativeLong
id	xs:ID
name	xs:NCName
minInclusive	xs:integer
maxInclusive	xs:integer

count は必須の属性で、配列中の値の数を表します。

id はオプションの属性で、この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。

name はオプションの属性で、この要素のテキスト文字列名です。

minInclusive はオプションの属性で、配列に保持可能な最も小さい整数値を表します。デフォルト値は-2147483648 です。

maxInclusive はオプションの属性で、配列に保持可能な最も大きい整数値を表します。デフォルト値は 2147483647 です。

### 関連要素

`<int_array>`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	<a href="#">source</a>
子要素	子要素はありません
その他	なし

### 備考

`<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表した`<int_array>`要素の例です。

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

## joints

### 概要

`<joints>`要素は、ジョイントノードと属性データとの関連付けを宣言するためのものです。

### コンセプト

`<joints>`要素は、ジョイント、スケルトン、ノードを属性データと関連付けます。COLLADA では、スケルトン中の各ジョイントのバインド逆行列（影響力）で指定します。

### 属性

`<joints>`要素に属性はありません。

### 関連要素

`<joints>`要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<code>skin</code>
子要素	<code>input</code> 、 <code>extra</code>
その他	なし

### 備考

`<input>`要素は、少なくとも 2 つ指定しなければならず、その 1 つでは JOINT セマンティックスを指定していなければなりません。

JOINT セマンティックスが指定された入力で参照されている`<source>`には、ジョイントノードを識別するための `sid` が指定された`<Name_array>`が含まれているべきです。この `sid` は、個々のインスタンスが個別にアニメーション化される際に、スキンコントローラが複数回インスタンス化されるように、IDREF の代わりに利用されます。

`<input>`要素が`<joints>`要素の子である場合、`<input>`要素に `idx` 属性を指定してはなりません。

`<extra>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

子要素の順番は、`<input>`、`<extra>`でなければなりません。

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



## library\_animations

### 概要

`<library_animations>`要素は、`animation` 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもともない、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1つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別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
name	xs:NCName

id はオプションの属性で、`<library_animation>`要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_animations>`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	COLLADA
子要素	<code>asset</code> 、 <code>animation</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_animations>`要素には、1つ以上の`<animation>`要素が記述できます。

`<asset>`要素は、まったく指定しないか、もしくは1つだけ記述することができます。

`<extra>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。

例

以下は、`<library_animations>`要素の例です。

```
<library_animations>
  <animation/>
</library_animations>
```

## library\_animation\_clips

### 概要

`<library_animation_clips>`要素は、`animation_clip` 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもない、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1つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別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
name	xs:NCName

id はオプションの属性で、`<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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_animation_clips>`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	COLLADA
子要素	<code>asset</code> 、 <code>animation_clip</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_animation_clip>`要素には、1つ以上の`<animation_clip>`要素が記述できます。

`<asset>`要素は、まったく指定しないか、もしくは1つだけ記述することができます。

`<extra>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。

### 例

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

```
<library_animation_clips>
  <animation_clip/>
</library_animation_clips>
```

## library\_cameras

### 概要

`<library_cameras>`要素は、`camera` 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもない、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 1 つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別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。

<code>id</code>	<code>xs:ID</code>
<code>name</code>	<code>xs:NCName</code>

`id` はオプションの属性で、`<library_cameras>`要素のユニークな識別子を含んだ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。

`name` はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_cameras>`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>COLLADA</code>
子要素	<code>asset</code> 、 <code>camera</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_cameras>`要素には、1 つ以上の`<camera>`要素が記述できます。

`<asset>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

`<extra>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。

### 例

以下は、`<library_cameras>`要素の例です。

```
<library_cameras>
  <camera/>
</library_cameras>
```

## library\_controllers

### 概要

`<library_controllers>`要素は、`controller` 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもともない、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 1 つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別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
name	xs:NCName

id はオプションの属性で、`<library_controllers>`要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_controllers>`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	COLLADA
子要素	<code>asset</code> 、 <code>controller</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_controllers>`要素には、1 つ以上の`<controller>`要素が記述できます。

`<asset>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

`<extra>`要素は、まったく指定しないか、任意の数だけ記述できます。

### 例

以下は、`<library_controllers>`要素の例です。

```
<library_controllers>
  <controller/>
</library_controllers>
```

## library\_effects

### 概要

`<library_effects>`要素は、effect 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもともない、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 1 つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別のリソース中に保存しておくことができます。

### 属性

`<library_effects>`要素には、以下の属性があります。

id	xs:ID
name	xs:NCName

id はオプションの属性で、`<library_effects>`要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_effects>`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	COLLADA
子要素	<code>asset</code> 、 <code>effect</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_effects>`要素には、1 つ以上の`<effect>`要素が記述できます。

`<asset>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

`<extra>`要素は、まったく指定しないか、任意の数だけ記述できます。

### 例

以下は、`<library_effects>`要素の例です。

```
<library_effects>
  <effect/>
</library_effects>
```

## library\_force\_fields

### 概要

<library\_force\_fields>要素は、force\_field 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもとない、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 1 つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別のリソース中に保存しておくことができます。

### 属性

<library\_force\_fields>要素には、以下の属性があります。

id	xs:ID
name	xs:NCName

id はオプションの属性で、<library\_force\_fields>要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

<library\_force\_fields>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	COLLADA
子要素	asset、force_field、extra
その他	なし

### 備考

<library\_force\_fields>要素には、1 つ以上の<force\_field>要素が記述できます。

<asset>要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

<extra>要素は、まったく指定しないか、任意の数だけ記述できます。

### 例

以下は、<library\_force\_fields>要素の例です。

```
<library_force_fields>
  <force_field/>
</library_force_fields>
```

## library\_geometries

### 概要

`<library_geometries>`要素は、`geometry` 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもともない、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 1 つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別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
name	xs:NCName

id はオプションの属性で、`<library_geometries>`要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_geometries>`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	COLLADA
子要素	<code>asset</code> 、 <code>geometry</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_geometries>`要素には、1 つ以上の`<geometry>`要素が記述できます。

`<asset>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

`<extra>`要素は、まったく指定しないか、任意の数だけ記述できます。

### 例

以下は、`<library_geometries>`要素の例です。

```
<library_geometries>
  <geometry/>
</library_geometries>
```

## library\_images

### 概要

`<library_images>`要素は、`image` 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもない、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 1 つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別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
name	xs:NCName

id はオプションの属性で、`<library_images>`要素のユニークな識別子を含んだ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。  
name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_images>`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	COLLADA
子要素	<code>asset</code> 、 <code>image</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_images>`要素には、1 つ以上の`<image>`要素が記述できます。  
`<asset>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。  
`<extra>`要素は、まったく指定しないか、任意の数だけ記述できます。

### 例

以下は、`<library_images>`要素の例です。

```
<library_images>
  <image/>
</library_images>
```



## library\_lights

### 概要

`<library_lights>`要素は、`light` 要素のモジュールを宣言するためのものです

### コンセプト

データセットがより大きくなり複雑化するのにもない、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 1 つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別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。

<code>id</code>	<code>xs:ID</code>
<code>name</code>	<code>xs:NCName</code>

`id` はオプションの属性で、`<library_lights>`要素のユニークな識別子を含んだ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。  
`name` はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_lights>`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>COLLADA</code>
子要素	<code>asset</code> 、 <code>light</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_lights>`要素には、1 つ以上の`<light>`要素が記述できます。  
`<asset>`要素は、まったく指定しないか、もしくは1つだけ記述することができます。  
`<extra>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。

### 例

以下は、`<library_lights>`要素の例です。

```
<library_lights>
  <lights/>
</library_lights>
```

## library\_materials

### 概要

`<library_materials>`要素は、`material` 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもない、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 1 つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別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
name	xs:NCName

id はオプションの属性で、`<library_materials>`要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_materials>`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	COLLADA
子要素	<code>asset</code> 、 <code>material</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_materials>`要素には、1 つ以上の`<material>`要素が記述できます。

`<asset>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

`<extra>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。

### 例

以下は、`<library_materials>`要素の例です。

```
<library_materials>
  <material/>
</library_materials>
```

## library\_nodes

### 概要

`<library_nodes>`要素は、`node` 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもとない、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 1 つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別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。

<code>id</code>	<code>xs:ID</code>
<code>name</code>	<code>xs:NCName</code>

`id` はオプションの属性で、`<library_nodes>`要素のユニークな識別子を含んだ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。  
`name` はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_nodes>`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>COLLADA</code>
子要素	<code>asset</code> 、 <code>node</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_nodes>`要素には、1 つ以上の`<node>`要素が記述できます。  
`<asset>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。  
`<extra>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。

### 例

以下は、`<library_nodes>`要素の例です。

```
<library_nodes>
  <node/>
</library_nodes>
```

## library\_physics\_models

### 概要

`<library_physics_models>`要素は、`physics_model` 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもない、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 1 つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別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。

id	xs:ID
name	xs:NCName

id はオプションの属性で、`<library_physics_models>`要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_physics_models>`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	COLLADA
子要素	<code>asset</code> 、 <code>physics_model</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_physics_models>`要素には、1 つ以上の`<physics_model>`要素が記述できます。

`<asset>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

`<extra>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。例以下は、`<library_physics_models>`要素の例です。

```
<library_physics_models>
  <physics_model/>
</library_physics_models>
```

## library\_physics\_scenes

### 概要

`<library_physics_scenes>`要素は、`physics_scene` 要素のモジュールを宣言するためのものです。

### コンセプト

データセットがより大きくなり複雑化するのにもない、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1つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別のリソース中に保存しておくことができます。

### 属性

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

id	xs:ID
name	xs:NCName

id はオプションの属性で、`<library_physics_scenes>`要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<library_physics_scenes>`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	COLLADA
子要素	<code>asset</code> 、 <code>physics_scene</code> 、 <code>extra</code>
その他	なし

### 備考

`<library_physics_scenes>`要素には、1つ以上の`<physics_scene>`要素が記述できます。

`<asset>`要素は、まったく指定しないか、もしくは1つだけ記述することができます。

`<extra>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。例以下は、`<library_physics_scenes>`要素の例です。

```
<library_physics_scenes>
  <physics_scene/>
</library_physics_scenes>
```

## library\_visual\_scenes

### 概要

<library\_visual\_scenes>要素は、visual\_scene 要素のモジュールを宣言するためのものです。

### コンセプト

データセットが大きく複雑化するのにもない、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 1 つの方法は、何らかの条件でデータをいくつか小さなモジュールセットに分割することです。そういったモジュールセットを、ライブラリとして別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
name	xs:NCName

id はオプションの属性で、<library\_visual\_scenes>要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

<library\_visual\_scenes>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	COLLADA
子要素	asset、visual_scene、extra
その他	なし

### 備考

<library\_visual\_scenes>要素には、1 つ以上の<visual\_scene>要素が記述できます。

<asset>要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

<extra>要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。

### 例

以下は、<library\_visual\_scenes>要素の例です。

```
<library_visual_scenes>
  <visual_scene/>
</library_visual_scenes>
```

## light

### 概要

<light>要素は、シーンを照らす光源を宣言するためのものです。

### 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、無限に遠く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、各種のパターンや周波数で光を放射します。

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。

ポイント光源は、空間中の既存の位置から全方向に光を放射します。ポイント光源の輝度は、光源までの距離が大きくなるのにもなって減衰します。

平行光源は、空間中の無限に遠くにある既存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、空間中の既存の位置から特定の方向に光を放射します。スポット光源からの光は、円錐形に放射されます。放射角が光源の方向から広がるのにもなって、輝度が減衰します。また、スポット光源の輝度は、光源からの距離が大きくなるのにもなって減衰します。

### 属性

<light>要素には、以下の属性があります。

id	xs:ID
name	xs:NCName

id はオプションの属性で、<light>要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

<light>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	library_lights
子要素	asset、technique_common、technique、extra
その他	なし

### 備考

<asset>要素は、まったく指定しないか、もしくは1つだけ記述することができます。

<technique\_common>要素が 1 つだけ含まれていなければなりません。また<light>の子として、<technique\_common>要素には、<ambient>、<directional>、<point>、または<spot>の要素が1つだけ含まれていなければなりません。

<technique>要素は任意の数だけ指定できます。

<extra>要素は任意の数だけ指定できます。

## 例

以下は、夕暮れを描写するためにシーン中にインスタンス化され、さらに回転される平行光源を表す<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>
      <instance_light url="#sun"/>
      <rotate>1 0 0 -10</rotate>
    </node>
  </visual_scene>
</library_visual_scenes>
```



## lines

### 概要

`<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付きを宣言するためのものです。

### コンセプト

`<lines>`要素は、頂点の属性を結び付けて個々の直線にするために必要な情報を提供します。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参照します。メッシュで記述された各直線には、頂点が2つずつあります。最初の直線は1番目と2番目の頂点から作成され、2番目の直線は、3番目と4番目の頂点から作成されます。

### 属性

`<lines>`要素には、以下の属性があります。

name	xs:NCName
count	xs:nonNegativeInteger
material	xs:NCName

name はオプションの属性で、この要素のテキスト文字列名です。

count は必須の属性で、線のプリミティブの数を表します。

material はオプションの属性で、マテリアルのシンボルを宣言します。シンボルは、インスタンス化の時点でマテリアルとバインドされます。material 属性を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。

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	<code>input</code> 、 <code>p</code> 、 <code>extra</code>
その他	なし

### 備考

`<lines>`要素には、1つの `p` 要素 (`p` は primitive の頭文字です) が含まれます。`p` 要素は、任意の数の直線の頂点属性を表します。

`<p>`要素中の各インデックスは、それぞれの順番に応じて別の入力を参照することになります。`<p>`要素の最初のインデックスは、オフセットが0のすべての入力を参照し、2番目のインデックスはオフセットが1のすべての入力を参照します。線の各頂点は、それぞれの入力への1つのインデックスで構成されます。それぞれの入力を利用した後、その次のインデックスはオフセットが0の入力を再度参照し、新しい頂点を開始することになります。

`<input>`要素は、全く指定しないか、もしくは任意の数だけ記述することができます。

`<p>`要素は、最大でも1つだけしか記述することができません。

子要素の並びは、`<input>`、`<p>`、`<extra>`の順番でなければなりません。

`<extra>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。

## 例

以下は、<lines>要素の例です。3 つの<input>要素を 1 つにまとめてあり、最後 2 つの入力では同じオフセットを利用しています。

```
<mesh>
  <source id="position"/>
  <source id="texcoord"/>
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <lines count="1">
    <input semantic="VERTEX" source="#verts" offset="0"/>
    <input semantic="TEXCOORD" source="#texcoord" offset="1"/>
    <input semantic="TEXCOORD" source="#texcoord" offset="1"/>
    <p>0 0 1 1</p>
  </lines>
</mesh>
```

## linestrips

### 概要

`<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付きを宣言するためのものです。

### コンセプト

`<linestrips>`要素は、頂点属性を結び付けて折れ線（ラインストリップ）形式にするために必要な情報を提供します。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参照します。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。ラインストリップ中の個々の線分は、現在の頂点とその前の頂点から作成されます。

### 属性

`<linestrips>`要素には、以下の属性があります。

name	xs:NCName
count	xs:nonNegativeInteger
material	xs:anyURL

name はオプションの属性で、この要素のテキスト文字列名です。

count は必須の属性で、ラインストリップのプリミティブの数を表します。

material はオプションの属性で、マテリアルのシンボルを宣言します。シンボルは、インスタンス化の時点でマテリアルとバインドされます。material 属性を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。

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	<code>input</code> 、 <code>p</code> 、 <code>extra</code>

### 備考

`<linestrips>`要素には、`p` 要素の並び（`p` は primitive の頭文字です）が含まれます。それぞれの `p` 要素は、任意の数の折れ線の線分の頂点属性を記述します。

`<p>`要素中の各インデックスは、それぞれの順番に応じて別の入力を参照することになります。`<p>`要素の最初のインデックスは、オフセットが 0 のすべての入力を参照し、2 番目のインデックスはオフセットが 1 のすべての入力を参照します。線の各頂点は、それぞれの入力への 1 つのインデックスで構成されます。それぞれの入力を利用した後、その次のインデックスはオフセットが 0 の入力を再度参照し、新しい頂点を開始することになります。

`<input>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。

`<p>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。

子要素の並びは、`<input>`、`<p>`、`<extra>`の順番でなければなりません。

`<extra>`要素は、まったく指定しないか、もしくは任意の数だけ記述することができます。

## 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

### 概要

<lookat>要素には、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。  
<lookat>要素には、以下を記述する 3 つの数学的なベクトルが含まれます。

1. オブジェクトの位置
2. 注視点の位置
3. 上方向

### コンセプト

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。そのため、lookat 変換を利用すれば、視点や注視点や方向を直観的に指定することができます。

### 属性

<lookat>要素には、以下の属性があります。

sid                      xs:NCName

sid はオプションの属性で、この要素のサブ識別子を含んだテキスト文字列値です。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。

### 関連要素

<lookat>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	node
子要素	子要素はありません
その他	なし

### 備考

<lookat>要素には、9 個の浮動小数点の値のリストが含まれます。OpenGL®ユーティリティ (GLU) の実装と同じく、これらの値で以下の 3 つのベクトルを構成します。

1. 視点を指定する : Px、Py、Pz
2. 注視点を指定する : Ix、Iy、Iz
3. 上方向を指定する : UPx、UPy、UPz

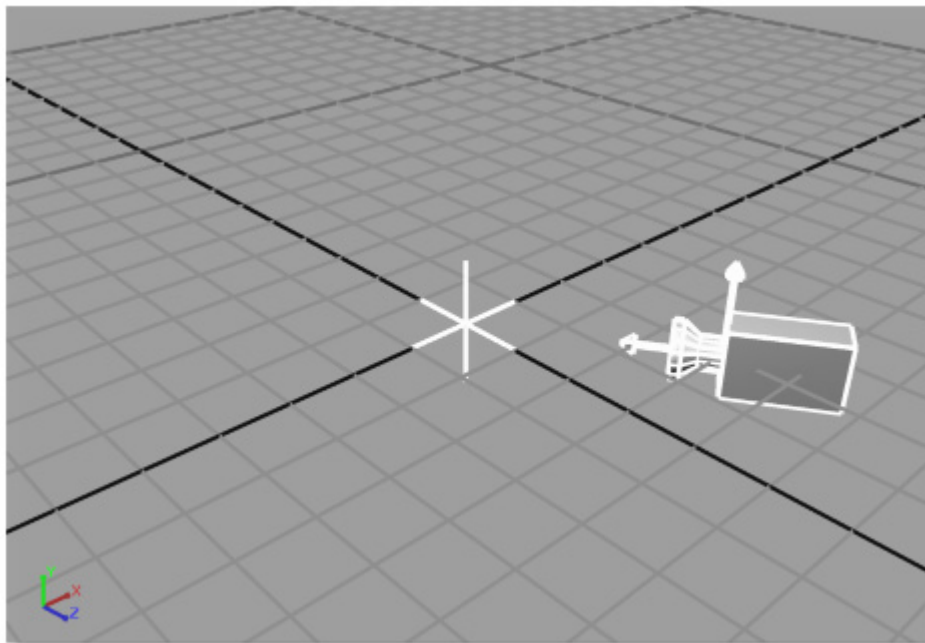
等価ビュー行列を計算する際、視点は原点、注視点は Z 軸のマイナス方向に割り当てられます。また、上方向の軸は、視平面 Y 軸のプラス方向に割り当てられます。  
値はどれもローカルオブジェクト座標で指定します。

## 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 name="Camera" id="Camera">
  <instance url="#camera"/>
  <lookat>
    2.0  0.0  3.0  <!-- eye position (X,Y,Z)      -->
    0.0  0.0  0.0  <!-- interest position (X,Y,Z)   -->
    0.0  1.0  0.0  <!-- up-vector position (X,Y,Z)  -->
  </lookat>
  ...
```

図 3-1 <lookat>要素：3 次元の十字線で注視点の位置を表しています。



## material

### 概要

`<material>`要素は、ジオメトリオブジェクトの視覚的な外観を記述するためのものです。

### コンセプト

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（材料）のプロパティを記述する各種のパラメータを指定することができます。このマテリアルのプロパティが、最終的に作成される物体の視覚的な外観を表すレンダリング時のパラメータになります。

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。固定機能の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行う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、環境光、拡散反射、正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をプログラマが定義します。これらのパラメータで、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に情報を与えることになります。

### 属性

`<material>`要素には、以下の属性があります。

id	xs:ID
name	xs:NCName

id はオプションの属性で、`<material>`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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<material>`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>library_materials</code>
子要素	<code>asset</code> 、 <code>instance_effect</code> 、 <code>extra</code>
その他	なし

### 備考

`<asset>`要素は、まったく指定しないか、もしくは1つだけ記述することができます。

`<instance_effect>`要素を1つだけ記述できます。

`<extra>`要素は任意の数だけ指定できます。

子要素の並びは、`<asset>`、`<instance_effect>`、`<extra>`でなければなりません。

## 例

以下は、<material>要素の例です。このマテリアルは、<library\_materials>要素に含まれています。

```
<library type="MATERIAL">
  <material name="Blue" 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">
        <float3>16.0</float3>
      </setparam>
    </instance_effect>
  </material>
</library>
```



## matrix

行列変換は、座標系中の各点、もしくは座標系そのものに対する数学的な変更を具体化するものです。

### 概要

`<matrix>`要素には、浮動小数点値の  $4 \times 4$  行列が含まれています。

### コンセプト

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のテクニックを利用します。3 次元座標系の一般的な形式は、 $4 \times 4$  の行列式で表します。これらの行列式は、基準座標系のつながりを表すために、シーングラフを介して階層構造化することが可能です。

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順序で表します。詳しくは、以下の例を参照してください。

### 属性

`<matrix>`要素には、以下の属性があります。

sid                      xs:NCName

sid はオプションの属性で、この要素のサブ識別子を含んだテキスト文字列値です。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。

### 関連要素

`<matrix>`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>node</code>
子要素	子要素はありません
その他	なし

### 備考

`<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。これらの値は、行列合成に適した  $4 \times 4$  の列順の行列に構成されます。

### 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

### 概要

<mesh>要素には、基本的なメッシュのジオメトリを記述するのに必要な頂点とプリミティブの情報が含まれています。

### コンセプト

メッシュは、主に頂点とプリミティブの情報を含んだジオメトリを記述する際の一般的な形式です。頂点情報というのは、メッシュ面上の特定の点に結び付けられた属性の集合です。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。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">geometry</a>
子要素	<a href="#">source</a> 、 <a href="#">vertices</a> 、 <a href="#">lines</a> 、 <a href="#">linestrips</a> 、 <a href="#">polygons</a> 、 <a href="#">polylist</a> 、 <a href="#">triangles</a> 、 <a href="#">trifans</a> 、 <a href="#">tristrips</a>
その他	なし

### 備考

<source>要素は、<mesh>の最初の要素として、1 つ以上記述されていなければなりません。<source>要素は、メッシュの頂点データの集合を提供することになります。<vertices>要素は、1 つだけ記述されていなければなりません。<vertices>要素は、メッシュの頂点属性を記述して、その固有のトポロジを確立することになります。頂点データから形成されるジオメトリプリミティブを記述するために、<mesh>要素には、以下のプリミティブ要素をまったく指定しないか、もしくは任意の数だけ記述することができます。

- 直線プリミティブを含む[lines](#)要素
- ラインストリップ・プリミティブを含む[linestrips](#)要素
- 穴を持つことが出来るポリゴン・プリミティブを含む[polygons](#)要素
- 穴を持たないポリゴン・プリミティブを含む[polylist](#)要素
- 三角形プリミティブを含む[triangles](#)要素

- 三角形ファン・プリミティブを含む<trifans>要素
- 三角形ストリップ・プリミティブを含む<tristrips>要素

<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、直接位置を記述するのではなく、頂点に対するインデックスを持ちます。メッシュの頂点は、semantic 属性の値が POSITION である<input>要素を最低でも 1 つは含んで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 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。

<extra>要素は、0 個以上記述することができます。

子要素の並びは、<source>、<vertices>、プリミティブ要素、<extra>という順番でなければなりません（プリミティブ要素の箇所は、<lines>、<linestrips>、<polygons>、<polylist>、<triangles>、<trifans>、<tristrips>の任意の組み合わせを意味します）。

## 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"/>
</mesh>
```

## morph

### 概要

<morph>要素は、固定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するためのものです。ブレンド可能な個々のメッシュ（モーフィングのターゲット）を指定する必要があります。また、それらのメッシュをどのように組み合わせるのか指定するには、method 属性を利用します。さらに、ブレンド操作の基準、あるいは参照用として、いくつかの方式で利用される「ベースメッシュ」もあります（詳しくは、以下の説明を参照してください）。

### コンセプト

一般に、メッシュをモーフィングした結果は、それぞれのメッシュ（静的メッシュやスキニングなど）を線形に組み合わせたものとなります。これらの入力メッシュは、モーフィングターゲットと呼ばれます。大きな制約として、これらのモーフィングターゲットは（モーフィングターゲットが違う位置にある場合でも）、どれも同じ頂点セットを持たなければなりません。モーフィングターゲットを組み合わせた場合、それらの<vertices>要素にあるデータだけが補間されます。したがって、すべてのモーフィングターゲットの<vertices>要素は、同じ構造でなければなりません。<vertices>要素の中にない頂点属性に対しては、ベースメッシュの頂点属性がそのまま利用され、他のモーフィングターゲット中の頂点属性は無視されます。

<morph>要素は、ベースメッシュ、他のメッシュのセット、ウェイトのセット、さらに、これらを組み合わせる方法として指定します。モーフィングターゲットを組み合わせるための方法が各種あり、どれを利用するのかを method 属性で指定します。以下の 2 つの方法が一般的です。

- NORMALIZED
 
$$(\text{Target1}, \text{Target2}, \dots) * (w1, w2, \dots) = (1-w1-w2-\dots) * \text{BaseMesh} + w1 * \text{Target1} + w2 * \text{Target2} + \dots$$
- RELATIVE
 
$$(\text{Target1}, \text{Target2}, \dots) + (w1, w2, \dots) = \text{BaseMesh} + w1 * \text{Target1} + w2 * \text{Target2} + \dots$$

### 属性

<morph>要素には、以下の属性があります。

source	xs:anyURL
method	MorphMethodType

source は必須の属性で、ベースメッシュを表します。

method 属性では、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。method 属性を指定しなかった場合のデフォルト値は NORMALIZED です。

## 関連要素

`<morph>`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>controller</code>
子要素	<code>source</code> 、 <code>targets</code> 、 <code>extra</code>
その他	なし

## 備考

`<source>`要素は、少なくとも2つ記述しなければなりません。

`<targets>`要素は、1つだけ記述しなければなりません。

`<extra>`要素は、任意の数だけ記述できます。

## 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/>
  <extra/>
</morph>
```

## Name\_array

### 概要

`<Name_array>`要素は、シンボル名の同質の配列の保存場所を宣言するためのものです。

### コンセプト

`<Name_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ともしません。単純に、XML の名前値の並びとして記述します。

### 属性

`<Name_array>`要素には、以下の属性があります。

count	xs:unsignedLong
id	xs:ID
name	xs:NCName

count は必須の属性で、配列中の値の数を表します。

id はオプションの属性で、この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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

`<Name_array>`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	<a href="#">source</a>
子要素	子要素はありません
その他	なし

### 備考

`<Name_array>`要素には、XML の名前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、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>
```

## node

ノードは、シーン中の各要素の階層構造的な関係を具体化するものです。

### 概要

`<node>`要素は、シーン中の一部を表すために利用します。1 つのノードは、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。

### コンセプト

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。ノードはグラフ中にある情報のポイントで、アークはノードを他のノードと結び付けます。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。本ドキュメントでは、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、アークはパス（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。

<code>id</code>	<code>xs:ID</code>
<code>name</code>	<code>xs:NCName</code>
<code>sid</code>	<code>xs:NCName</code>
<code>type</code>	JOINT または NODE
<code>layer</code>	ListOfNames

`id` はオプションの属性で、`<node>`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。

`name` はオプションの属性で、`<node>`要素の名前を含んだテキスト文字列です。

`sid` はオプションの属性で、この要素のサブ識別子を含んだテキスト文字列値です。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。

`type` はオプションの属性で、`<node>`要素のタイプを表します。デフォルト値は NODE です。

`layer` はオプションの属性で、そのノードが属しているレイヤの名前を表します。たとえば、「foreground glowing」という値は、ノードが「foreground」レイヤと「glowing」レイヤの両方に属していることを意味します。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。

### 関連要素

`<node>`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>library_nodes</code> 、 <code>node</code> 、 <code>visual_scene</code>
子要素	<code>asset</code> 、 <code>lookat</code> 、 <code>matrix</code> 、 <code>rotate</code> 、 <code>scale</code> 、 <code>skew</code> 、 <code>translate</code> 、 <code>instance_camera</code> 、 <code>instance_controller</code> 、 <code>instance_geometry</code> 、 <code>instance_light</code> 、 <code>instance_node</code> 、 <code>node</code> 、 <code>extra</code>
その他	なし

## 備考

<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。

要素	解説
node	階層構造を再帰的に定義できます
extra	補足情報が定義できます
asset	アセット管理情報を表現できます
instance_camera	カメラオブジェクトをインスタンス化することができます
instance_controller	コントローラオブジェクトをインスタンス化することができます
instance_geometry	ジオメトリオブジェクトをインスタンス化することができます
instance_light	光源オブジェクトをインスタンス化することができます
instance_node	他のノードの階層構造をインスタンス化することができます

<node>要素は、それぞれの変換子要素が記述されている順番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。つまり、変換要素は<node>要素の座標系を変換することになります。数学的に言うと、変換要素はそれぞれ行列に変換され、指定された順序で右から掛け合わされ、その積によって座標系が変換されることになります。それぞれの子要素の並びは、<asset>、変換要素の任意の組み合わせ、<instance\_camera>、<instance\_controller>、<instance\_geometry>、<instance\_light>、<instance\_node>、<node>、<extra>の順番でなければなりません（変換要素というのは、<lookat>、<matrix>、<rotate>、<scale>、<skew>、<translate>です）。

## 例

以下は、2 つの<node>要素を持った<scene>要素のアウトラインを示した例です。2 つのノードの名前は、それぞれ「earth」と「sky」です。

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



## optics

### 概要

<optics>要素は、画像を画像センサの上に投影するカメラに付いた装置を表すためのものです。

### 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カテゴリ化されます。

- 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。可変焦点の「ズームレンズ」には、実際には 11 枚以上のレンズと可変絞り（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。

### 関連要素

<optics>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	camera
子要素	technique_common、technique、extra
その他	なし

### 備考

共通プロファイルでは<perspective>と<orthographic>の光学タイプを定義しており、それ以外の<optics>要素のタイプは、プロファイル固有のテクニック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>45.0</yfov>
        <aspect_ratio>1.33333</aspect_ratio>
        <znear>0.1</znear>
        <zfar>32767.0</zfar>
      </perspective>
    </technique_common>
  </optics>
</camera>
```

## orthographic

### 概要

`<orthographic>`要素は、正投影カメラの視野を記述するためのものです。

### コンセプト

正投影というのは、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、オブジェクトの明確なサイズはカメラからの距離に依存しません。

### 属性

`<orthographic>`要素に属性はありません。

### 関連要素

`<orthographic>`要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	technique_common
子要素	xmag、ymag、aspect_ratio、znear、zfar
その他	なし

### 備考

`<orthographic>`要素は`<camera><optics>`の下に`<technique_common>`の中にだけ記述できます。`<orthographic>`要素には、1つの`<xmag>`要素、1つの`<ymag>`要素、`<xmag>`と`<ymag>`の両方の要素、もしくは`<xmag>`と`<ymag>`のどちらかと、もしくは`<aspect_ratio>`要素が含まれていなければなりません。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>`要素が指定されていない場合、アスペクト比は、`<xmag>`または`<ymag>`の要素と、現在のビューポートから計算されます。

`<xmag>`要素には、ビューの水平倍率を表す浮動小数点の値が含まれます。また、sidが含まれる場合もあります。

`<ymag>`要素には、ビューの垂直倍率を表す浮動小数点の値が含まれます。また、sidが含まれる場合もあります。

`<aspect_ratio>`要素には、視野のアスペクト比を表す浮動小数点の値が含まれます。また、sidが含まれる場合もあります。

`<znear>`要素は、1つだけ記述されていなければなりません。`<znear>`要素には、近方クリッピングプレーンへの距離を表す浮動小数点の値が含まれます。また、sidが含まれる場合もあります。

`<zfar>`要素は、1つだけ記述されていなければなりません。`<zfar>`要素には、遠方クリッピングプレーンへの距離を表す浮動小数点の値が含まれます。また、sidが含まれる場合もあります。

### 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）を指定している`<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

### 概要

`<param>`要素は、親要素のパラメータに関する情報を宣言するためのものです。

### 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。パラメータの基本的な宣言では、パラメータの名前、データ型、値となるデータを記述します。パラメータ名で特定の関数やプログラムを識別することになり、パラメータの型で値の符号化方法を表します。さらに、パラメータの値が実際のデータとなります。

### 属性

`<param>`要素には、以下の属性があります。

name	xs:NCName
semantic	xs:token
type	xs:NMTOKEN
sid	xs:NCName

name はオプションの属性で、この要素のテキスト文字列名です。  
 semantic はオプションの属性で、パラメータの意味をユーザ定義するためのものです。  
 type は必須の属性で、データ値の型を表します。このテキスト文字列は、アプリケーションで理解可能な形式でなければなりません。  
 sid はオプションの属性で、この要素のサブ識別子を含むテキスト文字列です。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。

### 関連要素

`<param>`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	<a href="#">accessor</a> 、 <a href="#">bind_material</a>
子要素	子要素はありません
その他	なし

### 備考

`<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>
```

## perspective

### 概要

`<perspective>`要素は、透視カメラの視野を記述するためのものです。

### コンセプト

透視図というのは、見る人から距離によって決まる物体の相対的な大きさの関係を表します。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。

### 属性

`<perspective>`要素に属性はありません。

### 関連要素

`<perspective>`要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	technique_common
子要素	xfov, yfov, aspect_ratio, znear, zfar
その他	なし

### 備考

`<perspective>`要素は`<camera>``<optics>`の下に`<technique_common>`の中にだけ記述できます。`<perspective>`要素には、1つの`<xfov>`要素、1つの`<yfov>`要素、`<xfov>`と`<yfov>`の両方の要素、もしくは`<xfov>`と`<yfov>`のどちらかと`<aspect_ratio>`要素が含まれていなければなりません。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。

`<yfov>`要素には、ビューの垂直倍率を表す浮動小数点の値が含まれます。また、sidが含まれる場合もあります。

`<aspect_ratio>`要素には、視野のアスペクト比を表す浮動小数点の値が含まれます。また、sidが含まれる場合もあります。

`<znear>`要素は、1つだけ記述されていなければなりません。`<znear>`要素には、近方クリッピングプレーンへの距離を表す浮動小数点の値が含まれます。また、sidが含まれる場合もあります。

`<zfar>`要素は、1つだけ記述されていなければなりません。`<zfar>`要素には、遠方クリッピングプレーンへの距離を表す浮動小数点の値が含まれます。また、sidが含まれる場合もあります。

### 例

以下は、水平視野角 90 度を指定する`<perspective>`要素の例です。

```
<perspective>
  <xfov sid="animated_zoom">1.0</xfov>
  <aspect_ratio>1.333</aspect_ratio>
  <znear>0.1</znear>
  <zfar>1000.0</zfar>
</perspective>
```

## point

### 概要

`<point>`要素は、ポイント光源を記述するためのものです。

### コンセプト

`<point>`要素は、ポイント光源を記述するために必要なパラメータを宣言します。ポイント光源は、空間中の既存の位置から全方向に光を放射します。ポイント光源の輝度は、光源までの距離が大きくなるのにもなって減衰します。光源の位置は、インスタンス化されるノードの変換によって定義されます。

### 属性

`<point>`要素に属性はありません。

### 関連要素

`<point>`要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	technique_common
子要素	color、constant_attenuation、linear_attenuation、quadratic_attenuation
その他	なし

### 備考

`<point>`要素は、`<light>`要素の下での`<technique_common>`の子としてだけ指定できます。  
`<color>`要素は、1 つだけ指定しなければなりません。`<color>`要素には、光源のカラーを指定する 3 つの浮動小数点が含まれます。また、sid 属性が含まれる場合もあります。  
特定の距離での光の全体の減衰は、`<constant_attenuation>`、`<linear_attenuation>`、`<quadratic_attenuation>`を利用して計算されます。その際、以下の公式が利用されます。

$$A = \text{constant\_attenuation} + \text{Dist} * \text{linear\_attenuation} + \text{Dist}^2 * \text{quadratic\_attenuation}$$

### 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

### 概要

`<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付きを宣言するためのものです。

### コンセプト

`<polygons>`要素は、頂点の属性を結び付けて個々のポリゴンを作成するために必要な情報を提供します。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、`<polygons>`要素でインデックス参照します。

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。

さまざまな操作で、表面のポイントの正確な方向が必要となります。法線ベクトルでこの方向を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。この曖昧な回転をなくす 1 つの方法は、同じポイントでの表面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）、この方法で表面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表面空間との間の変換を行う  $3 \times 3$  行列が定義できることになります。

接線と法線で、表面座標系の 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、以下のようにドキュメント中で異なる用途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 を記述した `<input>`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>`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。

```
count          xs:nonNegativeInteger
material       xs:anyURL
```

`count` は必須の属性で、ポリゴンプリミティブの数を表します。  
`material` はオプションの属性で、マテリアルのシンボルを宣言します。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。`material` 属性を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

`<polygons>`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	<code>input</code> 、 <code>p</code> 、 <code>ph</code> 、 <code>extra</code>
その他	なし

## 備考

<polygons>要素には、p 要素の並び（p は primitive の頭文字です）が含まれます。それぞれの p 要素で、個々のポリゴンの頂点属性を記述します。

1 つまたは複数の穴を持ったポリゴンは、<ph>要素として指定します。それぞれの<ph>要素には、1 つの<p>要素と、1 つまたは複数の<h>要素が含まれていなければなりません。その場合、<p>要素でポリゴンのインデックスを指定し、個々の<h>要素で穴のインデックスを表します。

<p>要素（または<h>要素）の中のインデックスは、それぞれの順番に依存して異なる入力を参照します。<p>要素中の最初のインデックスは、オフセットが 0 のすべての入力を参照します。2 番目のインデックスはオフセットが 1 のすべての入力を参照することになります。ポリゴンの個々の頂点は、それぞれの入力への 1 つのインデックスで構成されます。すべての入力を利用した後、その次のインデックスは、またオフセットが 0 の入力を再度参照し、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表面を記述することになります。

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、ライティングを有効にするために、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。

<input>要素は、任意の数だけ含めることができます。なくてもかまいません。

<p>要素は、任意の数だけ含めることができます。なくてもかまいません。

## 例

以下は、1 つの正方形を記述した<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>要素のセマンティクスに依存して位置と法線データが含まれています。<p>要素のインデックス値は、入力値を利用する順番を表しています。

```
<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、どのようにジオメトリ接線を指定するのかを示した例です（法線と接線の入力は、どちらもオフセットが 1 であるため、<p>要素中の 1 つのエントリを共有す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、どのようにテクスチャ空間の接線を指定するのかわを示した例です（テクスチャ空間の接線は、入力へのオフセットや順番ではなく、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>
  <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

### 概要

`<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バインドを宣言するためのものです。

### コンセプト

`<polylist>`要素は、複数の頂点属性を1つにまとめ、それらの頂点を個々のポリゴンに体系化するために必要な情報を指定するためのものです。

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参照します。

`<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。

さまざまな操作で、表面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、同じポイントでの表面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）、この方法で表面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表面空間との間の変換を行う $3 \times 3$ 行列が定義できることになります。

接線と法線で、表面座標系の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、以下のようにドキュメント中で異なる用途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 を記述した `<input>`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>`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。

name	xs:NCName
count	xs:nonNegativeInteger
material	xs:NCName

count は必須の属性で、ポリゴンプリミティブの数を表します。

material はオプションの属性で、マテリアルのシンボルを宣言します。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。material 属性を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。

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>
個要素	<code>input</code> 、 <code>vcount</code> 、 <code>p</code> 、 <code>extra</code>
その他	なし

## 備考

<vcount>要素には、<polylist>要素で記述されている各ポリゴンの辺の数を表す整数値のリストが含まれます。

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表面を記述することになります。

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、ライティングを有効にするために、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。

<input>要素は、まったく指定しないか、もしくは複数記述することができます。

<vcount>要素は、まったく指定しないか、もしくは1つだけ記述することができます。

<p>要素は、まったく指定しないか、もしくは1つだけ記述することができます。

<extra>要素は、まったく指定しないか、もしくは1つだけ記述することができます。

## 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、これらの<source>要素には、<input>要素の semantic 属性にしたがって位置データと法線データが含まれます。<p>要素のインデックス値は、利用される入力値の順番を示しています。

```
<mesh>
  <source id="position" />
  <source id="normal" />
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <polygons 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>
</polygons>
</mesh>
```

## rotate

### 概要

`<rotate>`要素には、回転角度と回転軸を表す数学的なベクトルが含まれています。

### 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、座標系に対して方向を変えたり、座標値を入れ替えたりするために、回転変換を利用します。つまり、回転は、ローカル座標の原点に対する座標軸の変換を意味します。

### 属性

`<rotate>`要素には、以下の属性があります。

sid                      xs:NCName

sid はオプションの属性で、この要素のサブ識別子を含んだテキスト文字列値です。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。

### 関連要素

`<rotate>`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	<a href="#">node</a>
子要素	子要素はありません
その他	なし

### 備考

`<rotate>`要素には、OpenGL®や RenderMan®の回転仕様と同様に、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 ]、および度数で表された角度を表します。

### 例

以下は、Y 軸を基軸として 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

### 概要

<sampler>要素は、N 次元の関数を宣言するためのものです。

### コンセプト

COLLADA では、アニメーションの関数曲線は 1 次元の<sampler>要素で表し、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。つまり、サンプリングポイント（キーフレーム）が、サンプラの入力データソースになります。アニメーション・チャンネルは、サンプラの出力データ値をターゲットに出力します。

### 属性

<sampler>要素には、以下の属性があります。

id                      xs:ID

id はオプションの属性で、<sampler>要素のユニークな識別子を含んだ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。

### 関連要素

<sampler>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	animation
子要素	input
その他	なし

### 備考

<input>要素を、1 つまたはそれ以上指定しなければなりません。

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 の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、補間の種類として、LINEAR、BEZIER、CARDINAL、HERMITE、BSPLINE、STEP が認識できます。

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。

## 例

以下は、「Box-Trans-X」という id のキーフレームの **source** 要素の X 軸の値を求める<**sampler**> 要素の例です。

```
<animation>
  <sampler id="Translate-X-Sampler">
    <input semantic="INPUT" source="#Box-Trans-X-Time"/>
    <input semantic="OUTPUT" source="#Box-Trans-X"/>
    <input semantic="INTERPOLATION" source="#Box-Interp"/>
  </sampler>
</animation>
```

## scale

### 概要

`<scale>`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ています。

### コンセプト

スケーリングは、回転や平行移動を行わずに、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、スケール変換を利用します。

### 属性

`<scale>`要素には、以下の属性があります。

sid	xs:NCName
-----	-----------

sid はオプションの属性で、この要素のサブ識別子を含んだテキスト文字列値です。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。

### 関連要素

`<scale>`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>node</code>
子要素	子要素はありません
その他	なし

### 備考

`<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。

### 例

以下は、物体（座標系）の大きさを一律に 2 倍に増やす変換を記述するための`<scale>`要素の例です。

```
<scale>
  2.0 2.0 2.0
</scale>
```

## scene

シーンは、特定の 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。

### 概要

<scene>要素は、シーン階層（シーングラフ）のベースを宣言するためのものです。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コンテンツの大部分は、シーンに含まれます。

### コンセプト

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、視覚情報と関連したデータをノードとして含む閉路なし有向グラフ（DAG）、つまりツリー構造のデータです。シーングラフ構造は、データの最適処理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。

出現回数	0 回、または 1 回
親要素	COLLADA
子要素	instance_physics_scene、instance_visual_scene、extra
その他	なし

### 備考

<COLLADA>文書要素（ルート要素）の中には、<scene>要素が最大 1 つ宣言されます。シーングラフは、<scene>の基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。<instance\_physics\_scene>要素は任意の数だけ指定できます。<instance\_visual\_scene>要素は任意の数だけ指定できます。<extra>要素は任意の数だけ指定できます。それぞれの要素の順番は、<instance\_physics\_scene>、<instance\_visual\_scene>、<extra>であるべきです。

### 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>
```

## skeleton

### 概要

<skeleton>要素は、スキンコントローラで必要となるジョイントノードの検索をスキンコントローラがどこから始めるのかを指定するためのものです。

### 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、さまざまな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。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">instance_controller</a>
子要素	子要素はありません
その他	なし

### 備考

### 例

以下は、<skeleton>要素がどのようにして、skin と識別された同一のローカル定義された<controller>要素を参照する 2 つのコントローラインスタンスを一つのスケルトンの異なった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"/>
    </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"/>
    <skel01/>
  </instance_controller>
</node>
<node>
  <instance_controller url="#skin"/>
    <skel02/>
  </instance_controller>
</node>

```

## skew

### 概要

<skew>要素には、角度と、回転軸と平行移動の軸を表す 2 つ数学的なベクトルが含まれています。

### コンセプト

スキュー処理（剪断変形）というのは、特定の座標軸に沿って物体を変形させることで、変形軸成分の値を変形軸に対して平行移動させます。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。

### 属性

<skew>要素には、以下の属性があります。

sid	xs:NCName
-----	-----------

sid はオプションの属性で、この要素のサブ識別子を含んだテキスト文字列値です。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。

### 関連要素

<skew>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	node
子要素	子要素はありません
その他	なし

### 備考

RenderMan®の仕様と同様に、<skew>要素には 7 つの浮動小数点値のリストが含まれます。これらの値は、度数で表された角度と、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。

### 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

### 概要

<skin>要素には、重み付きブレンド・スキニングを記述するのに必要となる頂点とプリミティブの情報が含まれています。

### コンセプト

キャラクタ・スキニングの場合、アニメーションエンジンは、スキニングされたキャラクタの関節（骨格）を動かします。スキンのトポロジーを構成している関節とメッシュの頂点との関連を記述するのが、スキン・メッシュです。関節は、制御アルゴリズムにしたがって、スキン・メッシュ頂点の変換に影響を与えることになります。

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混在させます。

古典的なスキニング・アルゴリズムでは、ノード（関節とも呼ばれます）の行列でジオメトリの点（メッシュの頂点など）を変換し、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）と、それに対応する重みの組合せは「インフルエンス（影響力）」と呼ばれます。さらに、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（スケルトン）」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スキニング・アルゴリズムの実行

前処理の結果、つまり「スキニング情報」は、以下の情報から構成されています。

- バインド形状：「デフォルトの形状」とも呼ばれます。これは、スキンが骨格にバインドされた時点でのスキンの形状です。バインド形状には、対応する <mesh>頂点の位置（必須）と、オプションでさらに頂点の属性を含むことができます。
- 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。通常、このノードごとの情報は「バインド行列」で表現されます。バインド行列というのは、バインド時のノードのローカル座標からワールド座標への変換行列です。

スキニング・アルゴリズムの変換は、すべてバインド・ポーズと関連して行われます。この相対変換は、通常、骨格中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す。

頂点の新しい（スキニングされた）位置を導き出すために、それぞれのインフルエンス・ノードのスキニング行列で、頂点のバインド位置を変換し、その結果に対してブレンド加重を利用して加重平均を取ります。

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。この操作で、各ノードのバインド・ポーズ変換が事実上打ち消され、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。

通常、バインド処理では、以下の操作が必要となります。

- スキンの現在の形状をバインド形状として保存する。
- バインド行列を計算して保存する。
- フォールオフ機能をいくつか備えたデフォルトのバインド加重値を生成する。特定の頂点から関節が遠いほど、インフルエンスが少なくなります。また、加重値が 0 のインフルエンスは無視してかまいません。

したがって、アーティストは、通常、メッシュ上に「ペイント」することで、自分で加重値を変更できるようにします。

## 属性

<skin>要素には、以下の属性があります。

source	xs:anyURL
--------	-----------

source は必須の属性で、ベースメッシュ（ベースメッシュまたはモーフィングメッシュ）を参照する URI を含みます。スキンメッシュのバインド形状も表します。

## 関連要素

<skin>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	controller
子要素	bind_shape_matrix、source、joints、vertex_weights、extra
その他	なし

## 備考

<bind\_shape\_matrix>要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。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。<bind\_shape\_matrix>要素が指定されていない場合、<bind\_shape\_matrix>として単位行列が利用されます。

<source>要素は、3 回またはそれ以上指定しなければなりません。<source>要素で、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。

<joints>要素は、1 つだけ指定しなければなりません。この要素は、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。

<vertex\_weights>要素は、1 つだけ指定しなければなりません。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。<vertex\_weights>では、関節の配列に対して-1 というインデックスはバインド形状を参照することになります。重み付けは、利用する前に正常にしておくべきです。

<extra>要素は任意の数だけ指定できます。

子要素の並びは、<bind\_source\_matrix>、<source>、<joints>、<vertex\_weights>、<extra>の順番でなければなりません。

## 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 Spine1 Spine2 Head </Name_array>
      ...
    </source>
    <source id="Weights">
      <float_array count="4"> 0.0 0.33 0.66 1.0 </float_array>
      ...
    </source>
    <source id="Inv_bind_mats">
      <float_array count="64"> ... </float_array>
      ...
    </source>
  </skin>
</controller>
```

```
<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

### 概要

<source>要素は、参照する<input>要素のセマンティ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するためのものです。

### コンセプト

データソースは、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。データソースには、情報へのアクセス方式が用意されています。このアクセス方式は、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、またはデータを生成するプログラムとして、ローカルに保存することができます。

### 属性

<source>要素には、以下の属性があります。

id	xs:ID
name	xs:NCName

id はオプションの属性で、<source>要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

<source>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	morph、animation、mesh、convex_mesh、skin
子要素	IDREF_array、Name_array、bool_array、float_array、int_array、technique_common、technique
その他	なし

### 備考

配列要素（<bool\_array>、<float\_array>、<int\_array>、<Name\_array>、<IDREF\_array>）のいずれか一つは、まったく指定しないか、もしくは 1 つだけ記述することができます。指定する場合には、いずれか 1 つしか指定できません。

<technique\_common>要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。<source>の子として、<technique\_common>要素には<accessor>要素が 1 つだけ含まれていなければなりません。

<technique>要素は、まったく指定しないか、もしくは複数記述することができます。

### 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

### 概要

<spline>要素には、制御頂点（CV）とセグメント情報を持つ複数のセグメントスプラインを表すための十分な情報が含まれています。

### コンセプト

制御頂点ごととセグメントごとのどちらの情報も、制御頂点に対して保存されます。セグメントごとのデータは、特定の制御頂点から始まるスプラインセグメントに対して適用されます。それぞれの制御頂点には任意の数の属性が指定できますが、属性の数と種類は、1 つのスプライン中で個々の制御頂点に対して一定でなければなりません。制御頂点ごととセグメントごとの情報としては、以下のものがあります。

- 制御頂点の位置（2d、3d、または 4d : NURBS）
- セグメントの曲率を制御する接線
- 2 つの隣接するセグメントの連続性を制御する接線
- セグメントの種類：1 つのスプライン中の各セグメントで別々の補間方法（リニアやベジエなど）が利用できます。
- 断片ごとの線形近似（テッセレーション）のステップ数
- カスタムの制御頂点ごとのデータ

<spline>の構成は、<mesh>と非常によく似ています。<spline>には、属性ストリームを組み立てるために、属性と<control\_vertices>要素を提供する<source>要素が含まれます。

### 属性

<spline>要素には、以下の属性があります。

closed                bool

closed はオプションの属性で、最初と最後の制御頂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、スプラインがオープンであることを意味します。

### 関連要素

<spline>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	geometry
子要素	source、control_vertices、extra
その他	なし

### 備考

<source>要素は、1 つまたはそれ以上指定されなければなりません。<source>要素では、スプラインの制御頂点とセグメントのための値を提供します。

<control\_vertices>要素は、1 つだけ指定されなければなりません。スプラインの制御頂点を表すのに利用します。



それぞれの制御頂点は、semantic 属性が POSITION の<input>要素を少なくとも 1 つの持たなければなりません。  
 共通プロファイルでは、<control\_vertices>スプライン用に以下の<input>セマンティックスが定義されています。

名前	種類	解説	デフォルト値
POSITION	float2, float3, float4	制御頂点の位置	なし
INTERPOLATION	Name	制御頂点から始まるセグメントを表す多項式の種類。共通プロファイルでの種類は、以下の通りです: LINEAR, BEZIER, HERMITE, CARDINAL, BSPLINE, NURBS	LINEAR
IN_TANGENT	float2, float3	制御頂点の前に位置するセグメント図形を制御する接線 (BEZIER または HERMITE)	なし
OUT_TANGENT	float2, float3	制御頂点の後に位置するセグメント図形を制御する接線 (BEZIER または HERMITE)	なし
CONTINUITY	Name	制御頂点での連続性制限を定義します (BEZIER や HERMITE などの分割可能な接線を持ったセグメントに適用されます)。共通プロファイルでの種類は、以下の通りです: C0, C1, G1	C1
LINEAR_STEPS	int	制御頂点に続くスプラインセグメントで利用される断片ごとの線形近似のステップ数 (UNIFORM_LINEAR_STEPS が FALSE であると想定した場合)	なし

## 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="#particles-Pos"/>
    <input semantic="INTERPOLATION" source="#particles-Interp"/>
    <input semantic="LINEAR_STEPS" source="#particles-LinSteps"/>
  </control_vertices>
</spline>
```

## spot

### 概要

`<spot>`要素は、スポット光源を表すためのものです。

### コンセプト

`<spot>`要素では、スポット光源を表すのに必要なパラメータを宣言します。スポット光源は、空間中の既存の位置から特定の方向に光を放射します。スポット光源からの光は、円錐形に放射されます。放射角が光源の方向から広がるのにもなって、輝度が減衰します。また、スポット光源の輝度は、光源からの距離が大きくなるのにもなって減衰します。

光源の位置は、インスタンス化されるノードの変形で定義されます。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、 $-Z$  軸方向に下がることになります。光源の実際の方向は、光源がインスタンス化されるノードの変形として定義されます。

### 属性

`<spot>`要素に属性はありません。

### 関連要素

`<spot>`要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<code>technique_common</code>
子要素	<code>color</code> 、 <code>constant_attenuation</code> 、 <code>linear_attenuation</code> 、 <code>quadratic_attenuation</code> 、 <code>falloff_angle</code> 、 <code>falloff_exponent</code>
その他	なし

### 備考

`<spot>`要素は、`<light>`要素の下での`<technique_common>`の子としてだけ指定できます。

`<color>`要素は、1 つだけ指定しなければなりません。`<color>`要素には、光源のカラーを指定する 3 つの浮動小数点のスポット値が含まれます。`sid` 属性を指定することも可能です。

特定の距離での光の全体の減衰は、`<constant_attenuation>`、`<linear_attenuation>`、`<quadratic_attenuation>`を利用して計算されます。その際、以下の公式が利用されます。

$$A = \text{constant\_attenuation} + \text{Dist} * \text{linear\_attenuation} + \text{Dist}^2 * \text{quadratic\_attenuation}$$

光の方向を基準にした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

### 概要

`<targets>`要素は、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するためのものです。

### 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。

出現回数	スキーマ中に定義された要素の数
親要素	<code>morph</code>
子要素	<code>input</code> 、 <code>extra</code>
その他	なし

### 備考

`<input>`要素は、少なくとも2つ指定しなければなりません。1つはMORPH\_TARGETのsemantic属性を持ったもの、もう1つはMORPH\_WEIGHTのsemantic属性を持ったものです。

`<input>`要素が`<targets>`の子の場合、offset属性を指定してはなりません。

`<extra>`要素は任意の数だけ指定できます。

### 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

### 概要

<technique>要素は、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するためのものです。各テクニクは、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。

### コンセプト

テクニクというのは、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、これはプロファイルで表されています。テクニクのコンテキストは 2 つの情報で定義します。つまり、プロファイルと、インスタンス文書中のその親要素です。通常、テクニクは「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 つ以上指定されている場合、インポート時に、両方ではなく、どちらか一方が選ばれます。インポートを行うアプリケーションがどのプロファイルをサポートしているのかを基準にして選ばれます。テクニクにはアプリケーションデータとプログラムが含まれ、1 つの単位として管理可能なアセットとすることができます。

### 属性

<technique>要素には、以下の属性があります。

profile            xs:NMTOKEN

profile は必須の属性で、プロファイルの種類を表します。テクニク用のプラットフォームや機能の対象を表したベンダ定義された文字列です。

<technique>要素には、コンテンツ検証に利用する追加スキーマを表すために、xmlns 属性が (XML Schema Language として) 指定できます。

### 関連要素

<technique>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	extra、source、light、optics、imager、force_field、physics_material、physics_scene、rigid_body、rigid_constraint、instance_rigid_body
子要素	以下を参照
その他	なし

### 備考

<technique>要素には、任意の整形式の XML データが記述できます。整形式の任意のデータは、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、実際の検証は行われません。

### 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>
```

## translate

### 概要

`<translate>`要素には、X 軸方向、Y 軸方向、Z 軸方向への距離を表す数学的なベクトルが含まれます。

### コンセプト

平行移動は、回転を行うことなく、座標系中の物体の位置を変更します。コンピュータグラフィックスでは、座標系に対する位置を決めたり、値を移動したりするために、平行移動変換を利用します。つまり、平行移動は、ローカルな座標系の原点の移動を意味します。

### 属性

`<translate>`要素には、以下の属性があります。

sid	xs:NCName
-----	-----------

sid はオプションの属性で、この要素のサブ識別子を含んだテキスト文字列値です。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。

### 関連要素

`<translate>`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	<a href="#">node</a>
子要素	子要素はありません
その他	なし

### 備考

`<translate>`要素には、3 つの浮動小数点の値が含まれます。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。

### 例

以下は、X 軸方向にそって 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

### 概要

`<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付きを宣言するためのものです。

### コンセプト

`<triangles>`要素は、頂点の属性を結び付けて、個別の三角形を構成するために必要な情報を提供します。頂点配列の情報は属性配列として別に提供され、`<triangles>`要素でインデックス参照します。メッシュによって記述される三角形には、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 番目の頂点から構成され、2 番目の三角形は、4 番目、5 番目、6 番目の頂点から構成されることになります。

### 属性

`<triangles>`要素には、以下の属性があります。

name	xs:NCName
count	xs:nonNegativeInteger
material	xs:NCName

name はオプションの属性で、この要素のテキスト文字列名です。

count は必須の属性で、三角形プリミティブの数を表します。

material はオプションの属性で、マテリアルのシンボルを宣言します。シンボルは、インスタンス化の時点でマテリアルとバインドされます。material 属性を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。

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	<code>input</code> 、 <code>p</code> 、 <code>extra</code>
その他	なし

### 備考

`<triangles>`要素には、`p` 要素 (`p` は `primitive` の頭文字です) の並びが含まれます。`p` 要素は、個々の三角形の頂点属性を表します。

`<p>`要素中の各インデックスは、それぞれの順番に応じて異なる入力を参照することになります。`<p>`要素の最初のインデックスは、オフセットが 0 のすべての入力を参照し、2 番目のインデックスはオフセットが 1 のすべての入力を参照します。線の各頂点は、それぞれの入力への 1 つのインデックスで構成されます。それぞれの入力を利用した後、その次のインデックスはオフセットが 0 の入力を再度参照し、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、それぞれの三角形の表面を記述することになります。

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。

`<input>`要素は、まったく指定しないか、もしくは複数記述することができます。

`<p>`要素は、まったく指定しないか、もしくは 1 つだけ記述することができます。

`<extra>`要素は、0 個以上記述することができます。

子要素の並びは、`<input>`、`<p>`、`<extra>`の順番でなければなりません。

## 例

以下は、2 つの三角形を記述する`<triangles>`要素の例です。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>`要素によって指定されています。`<p>`要素のインデックス値は、入力値が利用される順序を表します。

```
<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

### 概要

`<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付きを宣言するためのものです。

### コンセプト

`<trifans>`要素は、頂点の属性を結び付けて、連結した三角形を作成するために必要な情報を提供します。

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参照します。

メッシュによって記述された各三角形には、頂点が 3 つずつあります。最初の三角形は、1 番目、2 番目、3 番目の頂点から構成されます。それ以降の三角形は、現在の頂点と、最初の頂点と直前の頂点を再利用して構成されます。

### 属性

`<trifans>`要素には、以下の属性があります。

name	xs:NCName
count	xs:nonNegativeInteger
material	xs:NCName

name はオプションの属性で、この要素のテキスト文字列名です。

count は必須の属性で、三角形ファンプリミティブの数を表します。

material はオプションの属性で、マテリアルのシンボルを宣言します。シンボルは、インスタンス化の時点でマテリアルとバインドされます。material 属性を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	<code>input</code> 、 <code>p</code> 、 <code>extra</code>
その他	なし

### 備考

`<trifans>`要素には、`p` 要素 (`p` は primitive の頭文字です) の並びが含まれます。`p` 要素は、任意の数の連結した三角形の頂点属性を表します。

`<p>`要素中の各インデックスは、それぞれの順番に応じて異なる入力を参照することになります。`<p>`要素の最初のインデックスは、オフセットが 0 のすべての入力を参照し、2 番目のインデックスはオフセットが 1 のすべての入力を参照します。線の各頂点は、それぞれの入力への 1 つのインデックスで構成されます。それぞれの入力を利用した後、その次のインデックスはオフセットが 0 の入力を再度参照し、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、それぞれの三角形の表面を記述することになります。

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。

`<input>`要素は、まったく指定しないか、もしくは複数記述することができます。



<p>要素は、まったく指定しないか、もしくは複数記述することができます。  
 <extra>要素は、まったく指定しないか、もしくは1つだけ記述することができます。  
 子要素の並びは、<input>、<p>、<extra>でなければなりません。

## 例

以下は、2 つの三角形を記述する<trifans>要素の例です。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>要素によって指定されています。<p>要素のインデックス値は、入力値が利用される順序を表します。

```
<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

### 概要

<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付きを宣言するためのものです。

### コンセプト

<tristrips>要素は、頂点の属性を結び付けて、連結した三角形を作成するために必要な情報を提供します。

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参照します。

メッシュによって記述された各三角形には、頂点が 3 つずつあります。最初の三角形は、1 番目、2 番目、3 番目の頂点から構成されます。それ以降の三角形は、現在の頂点と、最初の頂点と直前の頂点を再利用して構成されます。

### 属性

<tristrips>要素には、以下の属性があります。

name	xs:NCName
count	xs:nonNegativeInteger
material	xs:NCName

name はオプションの属性で、この要素のテキスト文字列名です。

count は必須の属性で、三角形ストリッププリミティブの数を表します。

material はオプションの属性で、マテリアルのシンボルを宣言します。シンボルは、インスタンス化の時点でマテリアルとバインドされます。material 属性を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	input、p、extra
その他	なし

### 備考

<tristrips>要素には、p 要素 (p は primitive の頭文字です) の並びが含まれます。p 要素は、任意の数の連結した三角形の頂点属性を表します。

<p>要素中の各インデックスは、それぞれの順番に応じて別の入力を参照することになります。<p>要素の最初のインデックスは、オフセットが 0 のすべての入力を参照し、2 番目のインデックスはオフセットが 1 のすべての入力を参照します。線の各頂点は、それぞれの入力への 1 つのインデックスで構成されます。それぞれの入力を利用した後、その次のインデックスはオフセットが 0 の入力を再度参照し、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 (と 3 番目と 5 番目...) の三角形では反時計回りで、2 番目 (と 4 番目と 6 番目...) の三角形では時計回りとなり、それぞれの三角形の表面を記述することになります。

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。

<input>要素は、まったく指定しないか、もしくは複数記述することができます。  
 <p>要素は、まったく指定しないか、もしくは複数記述することができます。  
 <extra>要素は、0 個以上記述することができます。  
 子要素の並びは、<input>、<p>、<extra>でなければなりません。

## 例

以下は、2 つの三角形を記述する<tristrips>要素の例です。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>要素によって指定されています。<p>要素のインデックス値は、入力値が利用される順序を表します。

```
<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

### 概要

`<vertex_weights>`要素は、スキン処理で利用される関節と重み付けの組み合わせを記述するためのものです。

### 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
-------	------

count は必須の属性で、ベースメッシュ中の頂点の数を表します。

### 関連要素

`<vertex_weights>`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>skin</code>
子要素	<code>input</code> 、 <code>vcount</code> 、 <code>v</code> 、 <code>extra</code>
その他	なし

### 備考

`<input>`要素は、2 つまたはそれ以上指定しなければなりません。`<input>`要素の 1 つは、`<vertex_weights>`の子として、JOINT という semantic 属性を持っていなければなりません。

`<input>`要素は、関連付ける関節と属性を表します。

`<vcount>`は、まったく指定しないか、もしくは 1 つ指定しなければなりません。`<vcount>`要素は、個々の頂点に関連付けられた骨の数を表します。

`<v>`要素は、まったく指定しないか、もしくは 1 つ指定しなければなりません。`<v>`要素は、個々の頂点に関連付けられた骨と属性を表します。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べきです。

`<extra>`要素は、0 個以上記述することができます。

子要素の順番は、`<input>`、`<vcount>`、`<v>`、`<extra>`でなければなりません。

### 例

以下は、空の`<vertex_weights>`要素の例です。

```
<skin>
  <vertex_weights count="">
    <input semantic="JOINT"/>
    <input/>
    <vcount/>
    <v/>
    <extra/>
  </vertex_weights>
```

```
</skin>
```

以下は、もっと具体的な<vertex\_weights>要素の例です。<vcount>要素で、最初の頂点には骨が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、最初の頂点の重み付けがバインド形状への weights[0]で、weights[1]は骨0、weights[2]は骨1である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

### 概要

<vertices>要素は、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
name	xs:NCName

id はオプションの属性で、<vertices>要素のユニークな識別子を含んだ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。

name はオプションの属性で、この要素のテキスト文字列名です。

### 関連要素

<vertices>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	<a href="#">mesh</a> 、 <a href="#">convex mesh</a>
子要素	<a href="#">input</a> 、 <a href="#">extra</a>
その他	なし

### 備考

<input>要素は、1 つまたは複数指定しなければなりません。1 つの<input>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、semantic 属性が POSITION でなければなりません。

<input>要素は、<vertices>要素の子のばあい、offset 属性を指定してはなりません。

<extra>要素は、0 個以上記述することができます。

### 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

### 概要

`<visual_scene>`は、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、視覚情報と関連したデータをノードとして含む閉路なし有向グラフ (DAG)、つまりツリー構造のデータです。シーングラフ構造は、データの最適処理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
name	xs:NCName

id はオプションの属性で、`<visual_scene>`要素のユニークな識別子を含んだ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。

name はオプションの属性で、`<visual_scene>`要素の名前を含んだテキスト文字列です。

### 関連要素

`<visual_scene>`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>library visual_scenes</code>
子要素	<code>asset</code> 、 <code>node</code> 、 <code>evaluate scene</code> 、 <code>extra</code>
その他	なし

### 備考

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。

`<visual_scene>`要素には、任意の数の子要素を含めることができます。`<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。

### 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/2004/COLLADASchema.xsd" version="1.1.0">
  <library_visual_scenes>
    <visual_scene id="world">
      <node id="root"/>
    </visual_scene>
  </library_visual_scenes>
</scene>
```

```
        <instance_visual_scene url="#world"/>
    </scene>
</COLLADA>
```



---

## 4章 共通プロフィール

---

このページは空白です。

## 概要

COLLADA スキーマでは、設定プロファイルに準拠した情報表現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そのものは、現時点では COLLADA スキーマの範囲外ですが、スキーマ中で扱えるようにする方法もあります<sup>1</sup>。

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します。`<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、スキーマの拡張にともなって、COLLADA では共通プロファイルの定義も提供する必要があります。

## 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以下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。つまり、`<param>` 要素の `name` 属性の値は、すべて大文字でなければなりません。以下に例を示しておきます。

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

- パラメータの型が COLLADA スキーマ、XML Schema、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。つまり、`<param>` 要素の `type` 属性の値は、この規則にしたがわなくてはなりません。以下に例を示しておきます。

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

- 入力とパラメータのセマンティック名は大文字にします。つまり、`<input>` と `<newparam>` の要素の `semantic` 属性の値は、すべて大文字にしなければなりません。以下に例を示しておきます。

```
<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

COLLADA 共通プロファイルは、`<technique_common>` または `<profile_COMMON>` の要素で宣言します。以下に例を示しておきます。

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

<sup>1</sup> XML Schema 言語では、制約された値セットを定義するための `<xs:key>` と `<xs:keyref>` という要素が定義されています。この制約セットで、XPath 言語 1.0 のサブセットを利用して、指示されている要素と属性に特定の範囲内で値が正しく該当するかどうか検証できます。

<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、テクニックによって変わりません。

## インタフェースとしてのパラメータ

COLLADA の<param>要素または<newparam>要素は、name または semantics を指定して、特定の範囲内でバインディング可能なパラメータを形成するシンボルを宣言します。したがって、正規のパラメータは、名前とセマンティクスの両方で規定することになります。つまり、共通プロファイル中のパラメータは標準的なものであり、既知のものとなります。

パラメータの型は、C/C++言語と同じようにオーバーロードすることができます。つまり、パラメータの型がかならずしも厳密に一致しなくてもバインドすることが可能です。ただし、この場合のパラメータの型は、integer から float、float3 から float4、bool から int といった簡単（で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互換性を維持する必要があります。

## 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 属性での参照に利用されるメンバを指定するシンボル名の一覧も記載しておきます。

共通プロファイル中の<param>要素の name 属性と<newparam>要素の semantic 属性の値は、以下の通りです。

名前	型	代表的なコンテキスト	説明	デフォルト値
A	float	<material>、 <texture>	アルファカラー成分	なし
ANGLE	float	<animation>、 <light>	オイラー角	なし
B	float	<material>、 <texture>	青カラー成分	なし
DOUBLE SIDED	float	<material>	レンダリングの状態	なし
G	float	<material>、 <texture>	緑カラー成分	なし
P	float	<geometry>	3 番目のテクスチャ座標	なし
Q	float	<geometry>	4 番目のテクスチャ座標	なし
R	float	<material>、 <texture>	赤カラー成分	なし
S	float	<geometry>	1 番目のテクスチャ座標	なし
T	float	<geometry>	2 番目のテクスチャ座標	なし
TIME	float	<animation>	時間（秒）	なし
U	float	<geometry>	1 番目の汎用パラメータ	なし
V	float	<geometry>	2 番目の汎用パラメータ	なし
W	float	<animation>、 <controller>、 <geometry>	4 番目のデカルト座標	なし
X	float	<animation>、 <controller>、 <geometry>	1 番目のデカルト座標	なし
Y	float	<animation>、 <controller>	2 番目のデカルト座標	なし

		<geometry>		
Z	float	<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<input>要素の semantic 属性の値は、以下の通りです。

セマンティックス	説明
BINORMAL	従法線ベクトル
COLOR	カラー座標ベクトル
INPUT	サンプラ入力
IN TANGENT	先行する制御ポイントの接線ベクトル
INTERPOLATION	サンプラの補間種類
INV BIND MATRIX	ローカル座標からワールド座標への変換行列の逆行列
JOINT	スキンのインフルエンス識別子
MORPH TARGET	メッシュ・モーフィング用のモーフィングターゲット
MORPH WEIGHT	メッシュ・モーフィングの加重値
NORMAL	法線ベクトル
OUTPUT	サンプラ出力
OUT TANGENT	後続の制御ポイントの接線ベクトル
POSITION	ジオメトリの座標ベクトル
TANGENT	接線ベクトル

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

名前	型	説明
'(' # ')' '[' '(' # ')' ']'	float	行列またはベクトルのフィールド
A	float	アルファカラー成分
ANGLE	float	オイラー角
B	float	青カラー成分
G	float	緑カラー成分
P	float	3 番目のテクスチャ座標
Q	float	4 番目のテクスチャ座標
R	float	赤カラー成分
S	float	1 番目のテクスチャ座標
T	float	2 番目のテクスチャ座標
TIME	float	時間 (秒)
U	float	1 番目の汎用パラメータ
V	float	2 番目の汎用パラメータ
W	float	4 番目のデカルト座標
X	float	1 番目のデカルト座標
Y	float	2 番目のデカルト座標
Z	float	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表記を利用できる点に注意してください。

このページは空白です。

---

## 5章 ツールの要件とオプション

---

このページは空白です。



---

## 概要

COLLADA に完全に準拠したツールはどれも、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、当然、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、値の解釈を統一したり、重要なユーザ設定可能なオプションを提供したり、ツールに独自の機能を組み込む方法の詳しい規定といったものです。本章では、こういった問題点に優先順位を付けて解説します。

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。

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、以降で解説する要件はどれもツールに組み込んであります。これらの重要なデータ解釈とオプションは、クロスプラットフォームのゲーム開発パイプラインで必要となるインターオペラビリティやカスタマイズ性を生み出すためのものです。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を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。COLLADA 規格適合性試験というのは、Maya®、XSI、3DS Max のエクスポートとインポートのテストを自動化するツールセットです。それぞれのテストケースは、ツールの COLLADA インポート/エクスポート用プラグインで得られたコンテンツと、ネイティブのコンテンツを比較するようになっており、比較の結果は HTML ページとスプレッドシートの両方にキャプチャされます。

## 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すべて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	平行移動のエクスポートが可能でなければなりません。
拡大/縮小	拡大/縮小のエクスポートが可能でなければなりません。
回転	回転のエクスポートが可能でなければなりません。
親子関係	親子関係のエクスポートが可能でなければなりません。
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のエクスポートが可能でなければなりません。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のエクスポートが可能でなければなりません。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	傾斜のエクスポートが可能でなければなりません。
透明度/反射率	透明度や反射率の追加マテリアル・パラメータのエクスポートが可能でなければなりません。
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）のエクスポートが可能でなければなりません。
ジオメトリなしの変換	ジオメトリのないもの（ローケータや NULL など）を変換できる必要があります。

#### 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 テクスチャのエクスポートが可能でなければなりません。
RGBA テクスチャ	任意の数の RGBA テクスチャのエクスポートが可能でなければなりません。
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のエクスポートが可能でなければなりません。
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）のエクスポートが可能でなければなりません。
マルチテクスチャリング	マテリアルごとに複数のテクスチャのエクスポートが可能でなければなりません。
面単位のマテリアル	面単位のマテリアルのエクスポートが可能でなければなりません。

## 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のエクスポートが可能でなければなりません。
頂点法線	頂点法線のエクスポートが可能でなければなりません。
頂点従法線	頂点従法線のエクスポートが可能でなければなりません。
頂点接線	頂点接線のエクスポートが可能でなければなりません。
頂点 UV 座標	頂点 UV 座標（テクスチャ座標とは別）のエクスポートが可能でなければなりません。
頂点カラー	頂点カラーのエクスポートが可能でなければなりません。
カスタム頂点属性	カスタム頂点属性のエクスポートが可能でなければなりません。

## 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、大雑把なキーフレームと、複雑な制御や制約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には、ユーザがサンプリングレート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、どれも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
- シューダパラメータ
- 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

かならず一定のアニメーションデータがエクスポートできるよう保証するために、アプリケーションは、シーン中に含まれているアニメーションクリップごとに<animation>要素を 1 つエクスポートすべきです。たとえば、走っているアニメーションと歩いているアニメーションの各クリップをユーザが別々に定義した場合、エクスポート用のアプリケーションは、走っているアニメーションに関係したチャンネルをすべて 1 つのアニメーションオブジェクトにまとめ、また歩いているアニメーションに関係したチャンネルをすべてもう 1 つのアニメーションオブジェクトにまとめるべきです。1 つのシーン中で複数のアニメーションクリップをアプリケーションがサポートしていない場合、アプリケーションのデフォルト動作として、シーン中の全部のアニメーションチャンネルも含め、単一のアニメーションとしてエクスポートすべきです。これらのアニメーションのリファクタリングは、外部ツールで行えます。

データ	エクスポートの可/不可
可変サンプリングレート	可変サンプリングレートを利用したアニメーションのエクスポートが可能でなければなりません。これは、アニメーションの異なる部分に対してユーザが別のサンプリングレートを指定してエクスポートを行うことを可能にします。
バインドポーズ法線	バインドポーズ法線のエクスポートが可能でなければなりません。
ボーン	ボーンアニメーションのエクスポートが可能でなければなりません。
骨格アニメーション	骨格アニメーションのエクスポートが可能でなければなりません。
スムーズバインディングによる骨格のアニメーション	スムーズバインディングによる骨格のアニメーションがエクスポート可能でなければなりません。
光源パラメータのアニメーション	光源パラメータのアニメーションのエクスポートが可能でなければなりません。
カメラアニメーション	カメラアニメーションのエクスポートが可能でなければなりません。
キーフレーム変換アニメーション	キーフレーム変換アニメーションをエクスポートする必要があります。
アニメーション関数曲線	アニメーション関数曲線をエクスポートする必要があります。

### シーンデータ

データ	エクスポートの可/不可
空のノード	空のノードをエクスポートする必要があります。
カメラ	カメラのエクスポートが可能でなければなりません。
スポットライト	スポット光源のエクスポートが可能でなければなりません。
平行光源	平行光源のエクスポートが可能でなければなりません。
点光源	点光源のエクスポートが可能でなければなりません。
面光源	面光源のエクスポートが可能でなければなりません。
環境光	環境光源のエクスポートが可能でなければなりません。
静止物体のバウンディングボックス	静止物体のバウンディングボックスのエクスポートが可能でなければなりません。
アニメーション物体のバウンディングボックス	アニメーション物体のバウンディングボックスのエクスポートが可能でなければなりません。

### 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オプション	三角形リストのエクスポートが可能でなければなりません。
ポリゴンリストのエクスポートオプション	ポリゴンリストのエクスポートが可能でなければなりません。
焼付け行列・オプション	焼付け行列のエクスポートが可能でなければなりません。
単一の<matrix>要素オプション	各ノードに<matrix>要素を 1 つだけ含んだインスタンス文書のエクスポートが可能でなければなりません（詳しくは以下のセクションを参照してください）。

### 単一の<matrix>要素オプション

COLLADA では、指定された順序で合成される各類の変換要素のスタックによって「変換」を表現することができます。このような表現は、アプリケーション内部で複数の独立した変換ステージを利用する際に、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、単一の焼付け<matrix>だけを保存する機能を残しながら、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、その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

エクスポートの機能は、すべて、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、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、あらゆるエクスポート・オプションの逆の機能をできるだけ提供する必要があります。

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、もっと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、ツールによって認識されず、後でエクスポートされるために保持されるものが含まれていたとしても、すべてインポート可能でなければなりません。エクスポートされたデータ中に同期を必要とするものがあるかどうかを外部のツールで調べるには、<asset>要素を利用します。

### 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。

このページは空白です。

---

## 6章 COLLADA フィジックス

---

このページは空白です。



## 物理的な単位について

それぞれの値が正しく一致する限りにおいては、量子と相対論的な効果を見捨ててニュートン物理学のシミュレーションを適切に行うことができます。たとえば、距離と長さをメートル単位で指定でき、時間が秒単位の場合、その影響力はニュートン物理学にしたがうことになるはずです。こういった理由から、多くのフィジックスエンジンは単位がなく、COLLADA フィジックスも、それ自身で各コンポーネントの単位を指定するよう強制していません。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。

## 慣性について

物体を加速させるのにどれくらいの力が必要なのかを記述する量は「慣性」と呼ばれ、 $I$  という文字で表されます。これと同じ意味で、回転に関しては「慣性モーメント」と呼ばれています。たとえば、回転の中心から距離  $r$  の位置にある単一の無限に小さな質量  $m$  では（アームとも呼ばれます）、以下ようになります。

$$I = mr^2$$

こういった質量の角運動量は、慣性と角運動速度の積として求められます。

任意の形状と分散質量を持った剛体の場合、その物体は、可変の質量を持ったバラバラの粒子セットとみなすことができます。

物体の慣性モーメントは、それぞれの粒子の質量の積と、回転軸からの距離の平方を合計したものとなります。

慣性は、任意のポイントと、剛体への相対的な任意の角度で導き出すことが可能です。しかしながら、質量の中心（重心）で表現する方がもっと簡単です（むしろ、その方がより直感的です。というのも、自由に回転される物体は、常に重心で回転されるからです）。

一般に、慣性は 2 階層目のテンソル（張筋）として表現され、「3×3 行列」の形式で記述されます。これは、次のようにして計算されます。

$$I_{CM} \equiv \begin{bmatrix} \sum_i m_i [y_i^2 + z_i^2] & -\sum_i m_i x_i y_i & -\sum_i m_i x_i z_i \\ -\sum_i m_i y_i x_i & \sum_i m_i [x_i^2 + z_i^2] & -\sum_i m_i y_i z_i \\ -\sum_i m_i z_i x_i & -\sum_i m_i z_i y_i & \sum_i m_i [x_i^2 + y_i^2] \end{bmatrix}$$

- $m$  は、粒子の質量です。
- $(x_i, y_i, z_i)$  は、粒子の位置を表します。

慣性テンソルの対角線成分は慣性モーメントと呼ばれ、これに対して、非対角線成分は慣性の積を意味します。

慣性テンソル（CM の位置）は、シンメトリックである点に注意してください。

また、参照フレームが剛体の（ローカルの）主軸に揃えられている場合、非対角線成分は 0 となります。純粋な対角線成分の 3×3 テンソルのようなものは、3 つの値（float3）で表現することになります。そのため、たとえば、「アイソレーション」中の左上の要素を調べると、次のように解釈することができます。つまり、ローカル X 軸に対する回転の場合、慣性は、それぞれの粒子の質量に「Y-Z プレーン上のアーム」の平方を掛け合わせたものの合計となります。もちろん、これは慣性モーメント（ $I = mr^2$ ）の定義です。

また、（質量の）オフセンターの位置でサンプリングする場合には、さらに追加の慣性モーメントがあります。

$$I_i \equiv \begin{bmatrix} m \square y_0^2 \square z_0^2 \square & -m x_0 y_0 & -m x_0 z_0 \\ -m y_0 x_0 & m \square z_i^2 \square x_i^2 \square & -m y_0 z_0 \\ -m z_0 x_0 & -m z_0 y_0 & m \square x_0^2 \square y_0^2 \square \end{bmatrix}$$

- m は、物体の質量です。
- (x0, y0, z0) は、慣性が計算されるポイントです。

任意のポイントでの物体の慣性は、慣性テンソル（CM の位置）と、この「オフセット」との合計となります。

---

## 新しいジオメトリの種類

フィジックスエンジンは、任意の形状よりも、解剖学的な形状（プリミティブ）と凸包をコリジョン形状として利用した方がより効率的に処理できます。そのため、COLLADA 1.4.0 では、<convex\_mesh>だけでなく、<box>や<sphere>といったプリミティブなジオメトリの種類がいくつか追加されています。これらはレンダリングを意図したものではありません。理由は、メッシュやサブ分割された表面の方が、レンダリング目的にもっと適しているからです。しかしながら、先に述べた表現は、以下の理由から、他のすべてのジオメトリの種類と互換性を持ちます。

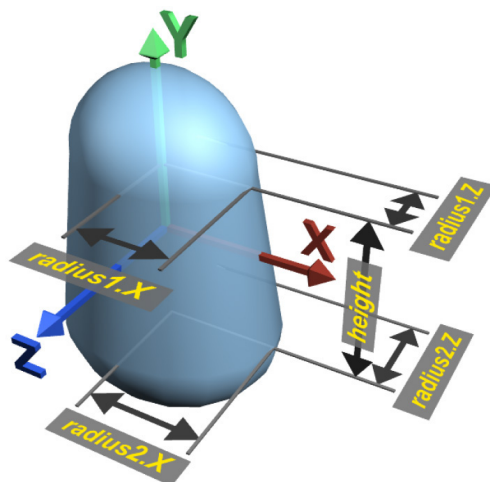
- 可能な限り一貫性が得られるように努力するのが一般であるから（コードの重複などを減らすため）。
- フィジックスで利用されるコリジョン形状を視覚化すると役立つアプリケーションもあり、この一貫性があれば、そういったものを通常の方法でインスタンス化/レンダリングできるから。
- コリジョン検出よりも、解剖学的な形状の方が役に立つから。たとえば、一部のモデラーでは、特定の領域を照らすプリミティブ（球体やディスクなど）が利用できます。これらは、放射マテリアルを持った（通常はインスタンス化された）ジオメトリプリミティブとして表現することが可能です。

このシステム（解剖学的な形状＋凸状メッシュ）で任意の形状が記述できます。したがって、<bounding\_box>要素はもはや必要ありません。

## ジオメトリプリミティブの座標系規則

`<cylinder>`、`<tapered_cylinder>`、`<capsule>`、`<tapered_capsule>`では、主軸は Y 軸（右手系で上方向の正の Y）で、以下の例に示したように、X 軸と Z 軸に沿って半径を指定します。

```
<tapered_cylinder>
  <height> 2.0 </height>
  <radius1> 1.0 2.0 </radius1>    <!-- radius1.X and radius1.Z -->
  <radius2> 1.5 1.8 </radius2>    <!-- radius2.X and radius2.Z -->
</tapered_cylinder>
```



---

## box

### 概要

軸に沿って中央揃えされたボックスプリミティブ

### コンセプト

ジオメトリプリミティブまたは解剖的な形は、フィジックスのコリジョン形状に役立ちます。詳しくは「新しいジオメトリの種類」を参照してください。

### 属性

<box>要素に属性はありません。

### 関連要素

<box>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">shape</a>
子要素	以下の解説を参照
その他	なし

### 子要素

名前/例	解説	デフォルト値	出現回数
<half extents>	ボックスの広がりを表す 3 つの浮動小数点	なし	1

### 例

```
<box>
  <half_extents> 2.5  1.0  1.0 </half_extents>
</box>
```

# capsule

## 概要

ローカルの Y 軸に沿って中央揃えされたカプセルプリミティブ

## コンセプト

ジオメトリプリミティブまたは解剖的な形は、フィジックスのコリジョン形状に役立ちます。詳しくは「新しいジオメトリの種類」を参照してください。

## 属性

<capsule>要素に属性はありません。

## 関連要素

<capsule>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">shape</a>
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<height>	半球の中央につながった線セグメントの長さを表す浮動小数点	なし	1
<radius>	カプセル（おそらく楕円形）の半径を表す 2 つの浮動小数点	なし	1

## 例

```
<capsule>
  <height> 2.0 2.0 </height>
  <radius> 1.0 1.0 </radius>
</capsule>
```

## convex\_mesh

### 概要

### コンセプト

`<convex_mesh>` の定義は `<mesh>` と同じですが、完全な記述（`<source>`、`<vertices>`、`<polygons>` など）の代わりに、形状を導き出す別の `<geometry>` を単に指し示すだけである点が違います。つまり、その `<geometry>` の凸包を計算すべきであることを意味し、オプションの `convex_hull_of` 属性で示すことになります。

フィジックスのために（レンダリングに利用される）`<mesh>` が再利用でき、ドキュメントのサイズを最小限に抑えることが可能で、オリジナルの `<mesh>` へのリンクを維持できるため、非常に役立ちます。`<convex_mesh>` を記述する最小限の方法は、それぞれの頂点を（`<vertices>` 要素と対応するソースを用いて）指定し、その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
----------------	-----------

`convex_hull_of` はオプションの属性で、その凸包を計算するためのジオメトリの URI 文字列を表します。

### 関連要素

`<convex_mesh>` 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>geometry</code>
子要素	子要素はありません
その他	なし

### 備考

詳細と子要素のリストに関しては `<mesh>` の解説を参照してください。

### 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

## 概要

シリンダプリミティブは、ローカルの Y 軸に対して中央に揃えられます。

## コンセプト

ジオメトリプリミティブまたは解剖的な形は、フィジックスのコリジョン形状に役立ちます。詳しくは「新しいジオメトリの種類」を参照してください。

## 属性

<cylinder>要素に属性はありません。

## 関連要素

<cylinder>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">shape</a>
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<height>	Y 軸に沿ったシリンダの長さを表す浮動小数点	なし	1
<radius>	シリンダ（楕円形も可能）の半径を表す 2 つの浮動小数点	なし	1

## 例

```
<cylinder>
  <height> 2.0 </height>
  <radius> 1.0 1.0 </radius>
</cylinder>
```

# force\_field

## 概要

`force_field` は汎用的なコンテナです。現時点では、`technique` と `extra` の要素だけを持つことができます。

## コンセプト

`force_field` は剛体などの物理的なオブジェクトに対して影響を及ぼし、`physics_scene` の基でインスタンス化されるか、もしくは `physics_model` のインスタンスです。

## 属性

`<force_field>`要素には、以下の属性があります。

id	xs:ID
----	-------

id はオプションの属性で、`<force_field>`要素のユニークな識別子を含んだ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。

## 関連要素

`<force_field>`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	<a href="#">library force fields</a>
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<code>&lt;technique&gt;</code>		なし	1 回、またはそれ以上
<code>&lt;extra&gt;</code>		なし	任意

## 備考

現時点では、`<force_field>`用の共通プロファイル/テクニクはありません。`<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>...
      </program>
    </technique>
  </force_field>
</library_force_fields>
```



## instance\_physics\_model

### 概要

この要素を利用すると、別のフィジックスモデルを持つフィジックスモデル、またはフィジックスシーン中のフィジックスモデルをインスタンス化することができます。

### コンセプト

この要素は、2 つの目的に利用します。つまり、あるフィジックスモデルを定義時に別のフィジックスモデルの中へ階層的に埋め込むためと、フィジックスシーン中の完全なフィジックスモデルをインスタンス化するためです。どちらの場合においても含まれている剛体のパラメータと、制限をオーバーライドすることが可能です。

フィジックスシーンの中でフィジックスモデルをインスタンス化する場合、最低でも、フィジックスモデルに含まれている個々の剛体が、影響を与える視覚的な変換ノードと結び付けられているべきです。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（その剛体に関しても同様です）を決めることになります。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対象となることも可能で、非ダイナミックな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。

### 属性

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

sid	xs:NCName
url	xs:anyURI
parent	xs:anyURI

url は必須の属性で、どの`<physics_model>`をインスタンス化するかを表します。

parent 属性は、視覚的シーン中のノードの 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基でインスタンス化することができ、最初の位置と方向を決め、剛体の運動に影響を与えるようにアニメーション化することが可能です。この属性はオプションです。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、ワールドの基で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。

sid はオプションの属性で、親要素中でのユニークな識別子を指定します。この属性を利用することで、アニメーション用に`<instance_physics_model>`インスタンスの要素を対象とすることができます。

### 関連要素

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

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">physics_scene</a> 、 <a href="#">physics_model</a>
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<code>&lt;instance_rigid_body target="#SomeNode"&gt;</code>	<code>&lt;rigid_body&gt;</code> 要素をインスタンス化して、プロパティの一部またはすべてをオーバーライドします。 target 属性では、剛体インスタンスで書きされた変換を持つ <code>&lt;node&gt;</code> 要素を定義します。	なし	任意
<code>&lt;instance_rigid_constraint&gt;</code>	プロパティの一部をオーバーライドするために、 <code>&lt;rigid_constraint&gt;</code> 要素をインスタンス化します。 この要素には target 属性はありません。理由は、 <code>&lt;rigid_constraint&gt;</code> 子要素でどの <code>&lt;node&gt;</code> 要素を対象とするのか定義するからです。	なし	任意
<code>&lt;extra&gt;</code>		なし	任意
<code>&lt;instance_force_field&gt;</code>	フィジックスモデルに影響を及ぼす <code>&lt;force_field&gt;</code> 要素をインスタンス化します。	なし	任意

## instance\_rigid\_body

### 概要

この要素は、<instance\_physics\_model>の中の<rigid\_body>をインスタンス化します。

### コンセプト

剛体というのは、突き詰めると、<scene>の中にある<node>の変換セットで、<physics\_model>の直下か、もしくは<rigid\_constraint>の下に位置しています。  
<physics\_model>をインスタンス化する際、最低でも、<physics\_model>に含まれている剛体は、関連する<node>要素と結び付けられていなければなりません。  
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
- オプションで特定のインスタンスで<rigid\_body>のパラメータをオーバーライドするため
- <rigid\_body>インスタンスの最初の状態（線速度と角運動速度）を指定するため

### 属性

<instance\_rigid\_body>要素には、以下の属性があります。

sid	xs:NCName
body	xs:NCName
target	xs:anyURI

sid はオプションの属性で、親要素中でのユニークな識別子を指定します。この属性を利用することで、アニメーション用の<rigid\_body>インスタンスの要素を対象とすることができます。  
body は必須の属性で、どの<rigid\_body>をインスタンス化するかを表します。  
target は必須の属性で、この<rigid\_body>インスタンスでどの<node>が影響を受けるのかを表します。

### 関連要素

<instance\_rigid\_body>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">instance_physics_model</a>
子要素	以下の解説を参照
その他	なし

## 子要素

`<instance_rigid_body>`要素とその `technique` は`<rigid_body>`要素の場合と同じ子を持ち、そのテクニック要素は以下を追加します。

名前/例	解説	デフォルト値	出現回数
<code>&lt;velocity&gt;</code>	<code>rigid_body</code> インスタンスの初期状態の線速度を指定します。	0, 0, 0	0 または 1
<code>&lt;angular_velocity&gt;</code>	<code>rigid_body</code> インスタンスの初期状態の角運動速度を、各軸に対して秒ごとの角度を単位として X-Y-Z オイラー回転の形式で指定します。	0, 0, 0	0 または 1

## 例

```
<physics_scene id="ColladaPhysicsScene">
  <instance_physics_model sid="firstCatapultAndRockInstance"
    url="#catapultAndRockModel" parent="#catapult1">

<!-- この physics_model 中の剛体の属性をオーバーライドして、 -->
<!-- rigid_body の初期速度を指定する。 -->
    <instance_rigid_body body="./rock/rock" target="#rockNode">
      <technique_common>
        <linear_velocity>0 -1 0</linear_velocity> <!--オプションのオーバーライド-->
        <mass>10</mass> <!--大きくする -->
      </technique_common>
    </instance_rigid_body>

<!-- このインスタンスはノードに剛体を割り当てるだけで、オーバーライドはしない。 -->
    <instance_rigid_body body="./catapult/base" target="#baseNode"/>
  </instance_physics_model>
</physics_scene>
```

# physics\_material

## 概要

この要素は、オブジェクトの物理的なプロパティを定義します。パラメータを伴ったテクニック/プロファイルが含まれます。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。

## コンセプト

フィジックスのマテリアルは、<library\_physics\_materials>要素の下に保存されてインスタンス化されることが可能です。。

## 属性

<physics\_material>要素には、以下の属性があります。

id	xs:ID
----	-------

id はオプションの属性で、<physics\_material>要素のユニークな識別子を含んだ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。

## 関連要素

<physics\_material>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	library_physics_materials
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<technique_common>	フィジックス・マテリアル用の共通テクニックを定義します。	なし	1
<technique>	フィジックス・マテリアル用のプロファイルを定義します。	なし	1 またはそれ以上

## technique\_common の子要素

名前/例	解説	デフォルト値	出現回数
<static_friction> 0.23 </static_friction>	静的摩擦係数	0	0 または 1
<dynamic_friction> 0.23 </static_friction>	動的摩擦係数	0	0 または 1
<restitution> 0.2 </restitution>	弾力または弾性とも呼ばれ、衝撃に温存される運動エネルギーの比率（通常、0.0～1.0 の範囲）	0	0 または 1

## 例

```
<physics_material id="WoodPhysMtl">
  <technique_common>
    <static_friction> 0.23 </static_friction>
    <dynamic_friction> 0.12 </dynamic_friction>
    <restitution> 0.05 </restitution>
  </technique_common>
</physics_material>
```

# physics\_model

## 概要

この要素で、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

この要素は、<physics\_scene>の基でインスタンス化される物理オブジェクトを定義してグループ化するのに利用します。  
フィジックスモデルは、単一の剛体のように単純であるか、もしくは骨と身体のパーツ（つまり、個々の剛体）を持ち、それらを筋肉（つまり、剛体制約）で結び付けている人物のキャラクタで生化学的に表現されたように複雑となります。  
また、フィジックスモデルには、事前に定義された他のフィジックスモデルを含めることも可能です。たとえば、家のフィジックスモデルには、煉瓦で造られた壁など、インスタンス化された数多くのフィジックスモデルを含めることができます。  
この要素でそういったモデルの構造を定義し、<instance\_physics\_model>要素で<physics\_model>をインスタンス化して、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。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。

## 属性

<physics\_model>要素には、以下の属性があります。

id	xs:ID
name	xs:NCName

id はオプションの属性で、<physics\_model>要素のユニークな識別子を含んだ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。  
name はオプションの属性で、<physics\_model>要素の名前を含んだテキスト文字列です。

## 関連要素

<physics\_model>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	<a href="#">library physics models</a>
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<code>&lt;rigid_body sid="..."&gt;</code>	<code>&lt;rigid_body&gt;</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。	なし	任意
<code>&lt;rigid_constraint sid="..."&gt;</code>	<code>&lt;rigid_constraint&gt;</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。	なし	任意
<code>&lt;instance_physics_model sid="..." url="#..."&gt;</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。	なし	任意
<code>&lt;asset&gt;</code>		なし	0 または 1
<code>&lt;extra&gt;</code>		なし	任意

## 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">

    <!-- カタパルトのベース。インラインを定義 -->
    <rigid_body sid="base">
      <technique_common>
        <dynamic>FALSE</dynamic>
        <shape>
          <instance_geometry url="#catapultBaseConvexMesh"/>
          <physics_material url="#catapultBasePhysicsMaterial"/>
        </shape>
        <linear_velocity>2 0 0</linear_velocity>
      <!-- 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>
    </technique_common>
  </rigid_body>

    <!-- 同じように、カタパルトの上部を定義する。 -->
    <rigid_body sid="top">

    </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 body="./base">
        <translate sid="translate">-2. 1. 0</translate>
      </ref_attachment>
      <attachment 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 0.0 </min>

```



```

        <max> 180.0 0.0 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>
</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://feelingsoftware.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>
    <instance_physics_model sid="catapult" url="#catapultModel"/>
</physics_model>
</library_physics_models>

```

## physics\_scene

### 概要

この要素は、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繰り返し処理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。

<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
name	xs:NCName

id はオプションの属性で、<physics\_scene>要素のユニークな識別子を含んだ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。  
name はオプションの属性で、<physics\_scene>要素の名前を含んだテキスト文字列です。

### 関連要素

<physics\_scene>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	<a href="#">library_physics_scenes</a>
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<code>&lt;instance_physics_model&gt;</code>	<code>&lt;physics_model&gt;</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。	なし	任意
<code>&lt;asset&gt;</code>		なし	0 または 1
<code>&lt;technique_common&gt;</code>	<code>physics_scene</code> の共通テクニックのパラメータを指定します。	なし	1
<code>&lt;technique&gt;</code>	カスタムのテクニックを指定します。	なし	任意
<code>&lt;extra&gt;</code>		なし	任意

## 例

```

<library_physics_scenes>
<!-- 通常の physics scene -->
  <physics_scene id="ColladaPhysicsScene">
    <technique_common>
      <timestep>3.e-002</timestep>
      <gravity>0 -9.8 0</gravity>
    </technique_common>
    <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>
        <linear_velocity>0 -1 0</linear_velocity> <!-- オプションのオーバーライド -->
        <mass>10</mass> <!-- 大きくする -->
      </technique_common>
    </instance_rigid_body>
    <instance_rigid_body body="./catapult/top" target="#catapultTopNode"/>
    <instance_rigid_body body="./catapult/base" target="#baseNode"/>
  </instance_physics_model>
</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ムがインスタンス化されるシーン -->
<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>

<!-- 親ノードの1つをインスタンス化して physics_model を複製する。 -->

```

```
<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>

<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

## 概要

この要素は、無限プレーンのプリミティブを表します。

## コンセプト

ジオメトリプリミティブまたは解剖的な形は、フィジックスのコリジョン形状に役立ちます。詳しくは「新しいジオメトリの種類」を参照してください。

## 属性

<plane>要素に属性はありません。

## 関連要素

<plane>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">shape</a>
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<equation>	プレーンの方程式の係数を表す 4 つの浮動小数点 $Ax + By + Cz + D = 0$	なし	1

## 例

```
<plane>
<!-- //Plane equation: Ax + By + Cz + D = 0 -->
<!-- //A, B, C, D coefficients (normal & D) -->
  <equation> 0.0 1.0 0.0 0.0 </equation>    // The X-Z plane (ground) -->
</plane>
```

# rigid\_body

## 概要

この要素を利用すると、形状を崩さないシミュレーションした物体を記述することができます。それらの物体は、各種の制約（関節やボール管継手など）で結び付けられていたり、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、剛体や制約などは<physics\_model>要素にカプセル化されます。

## コンセプト

剛体は、コリジョン検出のためのパラメータと形状の階層構造で構成されています。この階層構造中の各形状は、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、拡大・縮小、回転、変形可能となっています。これらの形状は、1 つまたは複数の<shape>要素で記述します。

## 属性

<rigid\_body>要素には、以下の属性があります。

sid	xs:NCName
name	xs:NCName

sid 属性は、<rigid\_body>要素の範囲化された識別子を含むテキスト文字列です。この値は、兄弟関係の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。  
sid は必須の属性で、<physics\_model>がインスタンス化される際に、それぞれの剛体を視覚的な<node>に関連付けるのに利用されます。  
name はオプションの属性で、<rigid\_body>要素の名前を含むテキスト文字列です。

## 関連要素

<rigid\_body>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	physics model
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<technique_common>	剛体のための共通プロファイルの表現を指定します。	なし	1
<technique>	複数の表現を可能とするために、剛体の対象プロファイルを指定します。	なし	任意
<extra>	<rigid_body>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)	なし	任意

## rigid\_body/technique\_common の子要素

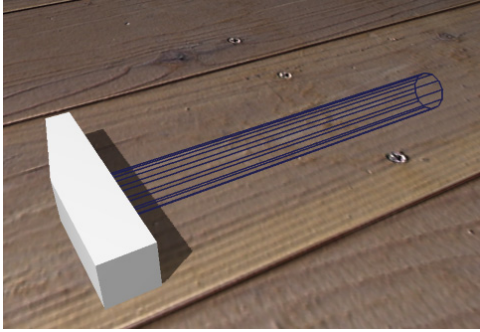
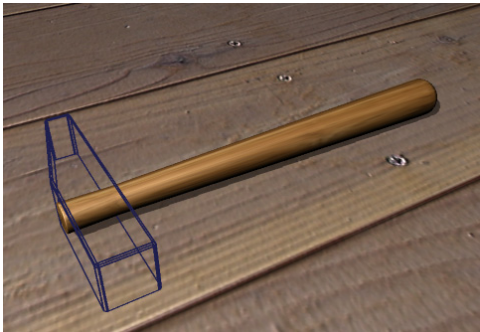
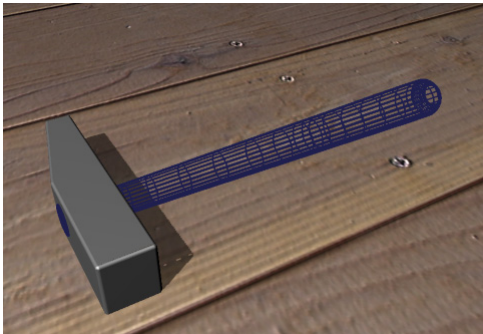
名前/例	解説	デフォルト値	出現回数
<code>&lt;dynamic&gt;FALSE&lt;/dynamic&gt;</code>	FALSE の場合、 <code>rigid_body</code> は動かすことができません。	TRUE	0 または 1
<code>&lt;mass_frame&gt;</code> <code>  &lt;translate&gt; ...</code> <code>  &lt;/translate&gt;</code> <code>  &lt;rotate&gt; ...</code> <code>&lt;/rotate&gt;</code> <code>&lt;/mass_frame&gt;</code>	「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。	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>&lt;inertia&gt;</code> <code>  1 1 1</code> <code>&lt;/inertia&gt;</code>	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）。	質量、形状の体積、重心から導き出されます。	0 または 1
<code>&lt;mass&gt;0.5&lt;/mass&gt;</code>	剛体の全体の質量	密度 X の全体の形状の体積から導き出されます。	0 または 1
<code>&lt;param&gt;</code>	非共通テクニック/プロファイルのためのユーザ定義パラメータ	なし	任意
<code>&lt;physics_material&gt;</code> または <code>&lt;instance_physics_material&gt;</code>	<code>rigid_body</code> のため <code>physics_material</code> を定義または参照します	なし	1

## 密度、質量、慣性（テンソル）の定義規則

- 剛体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。
- 質量を定義した場合、密度は無視されます。
- 体積と全体の質量は、それぞれの形状が単に交差している場合でも、形状の体積と質量の合計として計算されます。和集合や差分といった論理演算（CSG）は行われません。
- 剛体の質量や慣性などは、最終的な定義です。形状の質量の合計がその値にまで加算されなかった場合、剛体の質量まで加算されるように正規化されます。たとえば、質量全体が 6 で、2 つの形状を持つ物体（mass = 1 と mass = 2）は、全体の質量が (6) = 2+4 とみなされます。

## 例

以下は、複合化した剛体の例です。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、レンダリング対象（テクスチャ化/テーパ化された柄の部分と、傾斜された頭の部分）となる点に注意してください。

<pre> &lt;library_geometries&gt;   &lt;geometry id="hammerHeadForPhysics"&gt;     &lt;mesh&gt;       ...     &lt;/mesh&gt;   &lt;/geometry&gt; </pre>	
<pre> &lt;geometry id="hammerHandleToRender"&gt;   &lt;mesh&gt;     ...   &lt;/mesh&gt; &lt;/geometry&gt; </pre>	
<pre> &lt;geometry id="hammerHeadToRender"&gt;   &lt;mesh&gt;     ...   &lt;/mesh&gt; &lt;/geometry&gt; &lt;library_geometries&gt; </pre>	



```

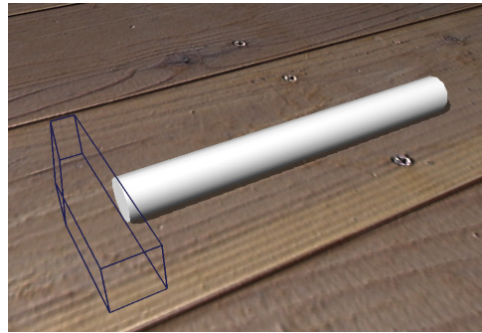
<library_physics_models>
  <physics_model
    id="HammerPhysicsModel">
    <rigid_body
      sid="HammerHandleRigidBody">
        <technique_common>

          <mass> 0.25 </mass>
          <mass_frame> ... </mass_frame>
          <inertia> ... </inertia>
          <shape>
            <physics_material
              url="#WoodPhysMtl"/>
            <!-- This geometry is small and not
              used elsewhere, so it is inlined -->
              <cylinder>
                <height> 8.0 </height>
                <radius> 0.5 </radius>
              </cylinder>

            </shape>
            <shape>
              <mass> 1.0 </mass>
              <!-- This geometry is
                referenced rather than inlined -->
                <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

### 概要

この要素で、<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ることができます。

### コンセプト

一般に、バネやボール管継手など、いろいろな種類の制約を利用していくつかの剛体を結び付けると、面白いフィジックスモデルを組み立てることができます。COLLADA では、2 つの剛体をリンクしたり、または剛体とシーン階層中（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。制約プリミティブ要素の膨大な組み合わせを定義する代わりに、COLLADA では非常に柔軟な 1 つの要素を提供しています。つまり、汎用的なレベル 6 の自由度（DOF）を持った制約です。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、ボール管継手、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。

- 剛体のローカル空間、もしくはシーン階層中の座標フレームへの相対的な変形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、変形の特定の角度または回転の角度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、単一の変形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	xs:NCName
name	xs:NCName

sid 属性は、<rigid\_constraint>要素の範囲化されたユニークな識別子を含んだ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。

name はオプションの属性で、<rigid\_body>要素の名前を含むテキスト文字列です。

### 関連要素

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

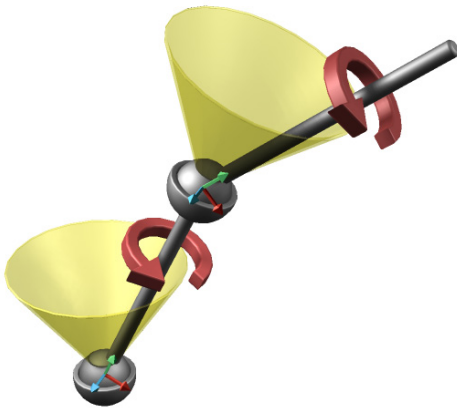
出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">physics_model</a>
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<code>&lt;ref_attachment rigid_body="./SomeRigidBody"&gt; ...</code>	参照フレームとして利用される (rigid_body または node) アタッチメントを定義します。rigid_body 属性は、同じ <a href="#">physics_model</a> 中で剛体への相対的な参照です。	なし	1
<code>&lt;attachment rigid_body="./SomeRigidBody"&gt; ...&gt;</code>	rigid_body または node へのアタッチメントを定義します。rigid_body 属性は、同じ <a href="#">physics_model</a> 中で剛体への相対的な参照です。	なし	1
<code>&lt;technique_common&gt;</code>	<a href="#">rigid_constraint</a> の対象となる共通プロファイルを指定します。	なし	1
<code>&lt;<a href="#">technique</a>&gt;</code>	複数の表現を許可するための <a href="#">rigid_constraint</a> 用の対象プロファイルを指定します。	なし	任意
<code>&lt;<a href="#">extra</a>&gt;</code>	ユーザ定義された複数の表現が可能なデータ	なし	任意

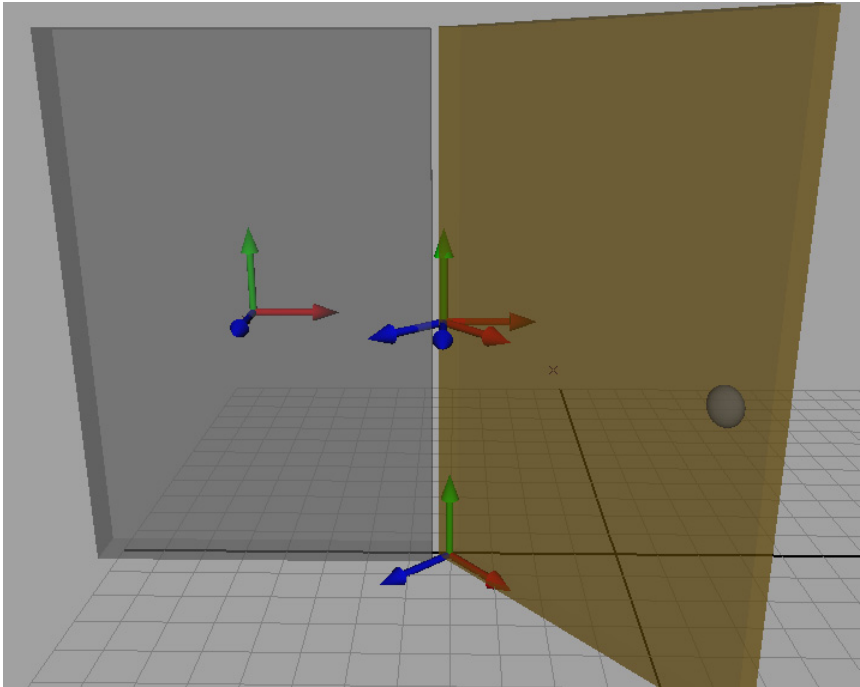
**rigid\_constraint と technique\_common のための子要素**

名前/例	解説	デフォルト値	出現回数
<code>&lt;enabled&gt;</code> TRUE <code>&lt;/enabled&gt;</code>	FALSE の場合、 <code>&lt;constraint&gt;</code> は剛体に対して何の影響も及ぼしません。	TRUE	0 または 1
<code>&lt;interpenetrate&gt;</code> TRUE <code>&lt;/interpenetrate&gt;</code>	アタッチされた剛体が互いに突き抜けているかどうかを表します。	0.0 (回転も変形も許可されません)	0 または 1
「揺れる円錐とひねり」の角度制限を持つ 2 つの制約：  <code>&lt;limits&gt;</code> <code>&lt;linear&gt;</code> <code>&lt;min&gt; 0 0 0 &lt;/min&gt;</code> <code>&lt;max&gt; 0 0 0 &lt;/max&gt;</code> <code>&lt;/linear&gt;</code> <code>&lt;swing_cone_and_twist&gt;</code> <code>&lt;min&gt;</code> -15.0 -15.0 -INF <code>&lt;/min&gt;</code> <code>&lt;max&gt;</code> 15.0 15.0 INF <code>&lt;/max&gt;</code> <code>&lt;/swing_cone_and_twist&gt;</code> <code>&lt;/limits&gt;</code>	<code>&lt;limits&gt;</code> 要素は、制約の上限（自由度と範囲）を指定するための柔軟な方法です。この要素には、以下を含めることができます。 ・ <code>&lt;swing_cone_and_twist&gt;</code> 要素と <code>&lt;linear&gt;</code> 要素  制約の上限を表現する新しい方法が標準化されれば、新しく強く型付けされた子要素が追加されることになります。具体的な制約の上限の記述に XML 要素のようなものがあるまでは、カスタムの <code>&lt;technique&gt;</code> を利用すべきです。 <code>&lt;linear&gt;</code> 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。 <code>&lt;swing_cone_and_twist&gt;</code> 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。 X と Y の上限は「揺れる円錐」を表し、Z の上限は「ひねりの角度」の範囲を表します（左側の図を参照）。 INF と -INF の値は無限の +/- に相当し、対象となる軸に沿った上限がないことを意味します。 それぞれの上限は、 <code>ref_attachment</code> の空間で表現します。	0.0 (回転も移動も許可されない)	0 または 1



<pre>&lt;spring&gt;   &lt;linear&gt;  &lt;stiffness&gt;5.4544&lt;/stiffness&gt;   &lt;damping&gt;0.4132&lt;/damping&gt;   &lt;target_value&gt;3 &lt;/target_value&gt; &lt;/linear&gt;  &lt;spring&gt;   &lt;angular&gt;  &lt;stiffness&gt;5.4544&lt;/stiffness&gt;   &lt;damping&gt;0.4132&lt;/damping&gt;   &lt;target_value&gt;90 &lt;/target_value&gt; &lt;/angular&gt; &lt;/spring&gt;</pre>	距離（LINEAR）または角度（ANGULAR）を基準にしたスプリング。剛度（スプリング係数とも呼ばれます）は、威力/距離（または威力/角度）を単位とします。ref_attachment の空間で表現されます。	無限剛度の制約（スプリングなし）  stiffness: 1 damping: 0 target_value: 0	0 または 1
--	---	---	---------

例



上記は、ヒンジ部分を持ったドアの例です。壁の剛体（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、剛体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"/>
    <rigid_body sid="wallRigidBody"/>
```

```

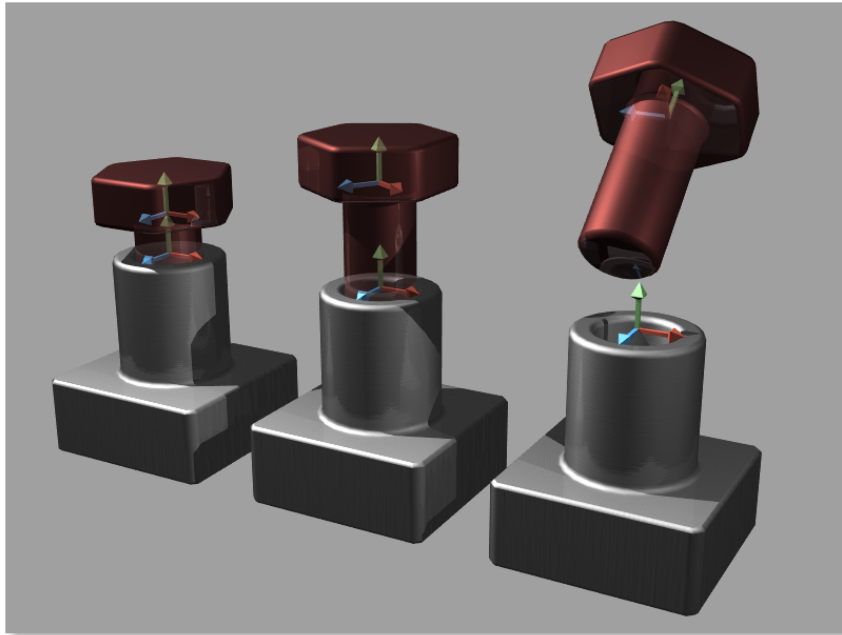
<rigid_constraint sid="rigidHingeConstraint">
  <ref_attachment body="#wallRigidBody">
    <translate sid="translate">5 0 0</translate>
  </ref_attachment>
  <attachment body="#doorRigidBody">
    <translate sid="translate">0 8 0</translate>
    <rotate sid="rotateX">0 1 0 -45.0</rotate>
  </attachment>

<!-- sid 属性を追加して、アニメーションからの制限をターゲットとする。-->
  <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>

```

## 中断可能な制約

以下の図は、3 種類の設定で示した 2 つの剛体を表しています。



赤と青と緑の矢印は、それぞれの剛体に対してアタッチされたフレーム（位置と方向）の X 軸、Y 軸、Z 軸を表しています。制約が回転の自由度だけであり（たとえば、ボール管継手）、中断不可能な場合、2 つのアタッチされたフレームの世界空間の位置は一致するはずです。これと同様に、制約に回転の自由度がなく、また中断不可能な場合には、アタッチされたフレームの世界空間の方向は同一のものとなります。

上記の図に示した制約には、2 つの自由度があります。つまり、中央に示されているように Y 軸に沿った回転と、右側に示されているように Y 軸に沿った変形です。左側の例の設定は、この制約が中断可能

であることを示しています。すなわち、Y 軸に沿った影響力が一定の閾値に達すると、制約が中断され、上側の剛体の動きを制限しなくなります。アタッチされたフレームは個々の剛体のローカル空間で表現されていますが、物理的な影響力とパラメータ（たとえば、制限）は、ワールド空間で表現されて計算されます。

# shape

## 概要

この要素で、<rigid\_body>のコンポーネントを記述することができます。

## コンセプト

剛体には、コリジョン検出用に 1 つの形状または形状の階層を含めることができます。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、拡大/縮小、回転、変形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。

- <physics\_material>の定義またはインスタンス
- 物理的なプロパティ（質量や慣性など）
- 変換（<scale>、<rotate>、<translate>）
- <geometry>のインスタンスまたはインライン化された定義

形状には、穴が空いていてもかまいません（たとえば、ウサギの形のチョコレート）。これは、体積全体で質量が分散されているのではなく、表面近くで分散されていることを意味します。それにしたがって、質量、慣性、密度、重心の属性を設定すべきです。

COLLADA では形状を拡大・縮小することができますが、一部のフィジックスエンジンでは拡大/縮小された解剖的な形状（たとえば、カプセル）をサポートしていません。

## 属性

<shape>要素に属性はありません。

## 関連要素

<shape>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	rigid body の technique common
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<hollow>TRUE</hollow>	TRUE の場合、形状の表面に沿って質量が分散されます。	TRUE	0 または 1
インライン定義またはインスタンス： <physics_material id="...">... </physics_material> または <physics_material url="...">/>	この形状に利用される<physics_material>	<shape>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



名前/例	解説	デフォルト値	出現回数
<code>&lt;mass&gt; 0.5 &lt;/mass&gt;</code>	形状の質量	密度 X の形状の体積から導き出されます。	0 または 1
<code>&lt;density&gt; 0.5 &lt;/density&gt;</code>	形状の密度	質量と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>&lt;box&gt;...&lt;/box&gt;</code> または <code>&lt;instance_geometry url=""/&gt;</code>	形状のジオメトリ。ジオメトリのインライン化と参照の両方が行えます。 <code>&lt;plane&gt;</code> 、 <code>&lt;box&gt;</code> 、 <code>&lt;sphere&gt;</code> 、 <code>&lt;cylinder&gt;</code> 、 <code>&lt;tapered_cylinder&gt;</code> 、 <code>&lt;capsule&gt;</code> 、 <code>&lt;tapered_capsule&gt;</code> はインライン化され、他の種類のジオメトリ ( <code>&lt;mesh&gt;</code> 、 <code>&lt;convex_mesh&gt;</code> 、 <code>&lt;spline&gt;</code> など) は、 <code>&lt;instance_geometry&gt;</code> 要素で参照されます。	なし	1
<code>&lt;scale&gt;</code> 、 <code>&lt;rotate&gt;</code> 、 <code>&lt;translate&gt;</code>	<code>&lt;node&gt;</code> の下と同じ	変換は行われません。	任意
<code>&lt;asset&gt;</code>		なし	0 または 1
<code>&lt;extra&gt;</code>		なし	任意

## 備考

詳しくは、`<rigid_body>`要素を参照してください。

## 例

```

<library_rigid_bodies>
  <rigid_body id="HammerHandleRigidBody">
    <technique_common>
      <shape id="HammerHandleShape">
        <mass> 0.25 </mass>
        <physics_material url="#WoodPhysMtl"/>
        <instance_geometry url="#hammerHandleForPhysics"/>
      </shape>
    </technique_common>
  </rigid_body>
</library_rigid_bodies>

```

---

## sphere

### 概要

中央揃えされた球体プリミティブ

### コンセプト

ジオメトリプリミティブまたは解剖的な形は、フィジックスのコリジョン形状に役立ちます。詳しくは「新しいジオメトリの種類」を参照してください。

### 属性

<sphere>要素に属性はありません。

### 関連要素

<sphere>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">shape</a>
子要素	以下の解説を参照
その他	なし

### 子要素

名前/例	解説	デフォルト値	出現回数
<radius>	球体の半径を表す浮動小数点	なし	1

### 例

```
<sphere>
  <radius> 1.0 </radius>
</sphere>
```

# tapered\_capsule

## 概要

ローカルの Y 軸に沿って中央揃えされたテーパ化されたカプセルプリミティブ

## コンセプト

ジオメトリプリミティブまたは解剖的な形は、フィジックスのコリジョン形状に役立ちます。詳しくは「新しいジオメトリの種類」を参照してください。

## 属性

<tapered\_capsule>要素に属性はありません。

## 関連要素

<tapered\_capsule>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	<a href="#">shape</a>
子要素	以下の解説を参照
その他	なし

## 子要素

名前/例	解説	デフォルト値	出現回数
<height>	半球の中央とつながっている線のセグメントの長さを表す浮動小数点	なし	1
<radius1>	正 (height/2) の Y 値でテーパ化されたカプセルの半径を表す 2 つの浮動小数点。テーパ化されたカプセルの両端は楕円となります。	なし	1
<radius2>	負 (height/2) の Y 値でテーパ化されたカプセルの半径を表す 2 つの浮動小数点。テーパ化されたカプセルの両端は楕円となります。	なし	1

## 例

```
<tapered_capsule>
  <height> 2.0 </height>
  <radius1> 1.0 1.0 </radius1>
  <radius2> 1.0 0.5 </radius2>
</tapered_capsule>
```

## tapered\_cylinder

### 概要

ローカルの Y 軸に沿って中央揃えされたテーパー化されたシリンダプリミティブ

### コンセプト

ジオメトリプリミティブまたは解剖的な形は、フィジックスのコリジョン形状に役立ちます。詳しくは「新しいジオメトリの種類」を参照してください。

### 属性

<tapered\_cylinder>要素に属性はありません。

### 関連要素

<tapered\_cylinder>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	<a href="#">shape</a>
子要素	以下の解説を参照
その他	なし

### 子要素

名前/例	解説	デフォルト値	出現回数
<height>	Y 軸に沿うシリンダの長さを表す浮動小数点	なし	1
<radius1>	正 (height/2) の Y 値でテーパー化されたシリンダの半径を表す 2 つの浮動小数点。テーパー化されたシリンダの両端は楕円となります。	なし	1
<radius2>	負 (height/2) の Y 値でテーパー化されたシリンダの半径を表す 2 つの浮動小数点。テーパー化されたシリンダの両端は楕円となります。	なし	1

### 例

```
<tapered_cylinder>
  <height> 2.0 </height>
  <radius1> 1.0 2.0 </radius1>
  <radius2> 1.5 1.8 </radius2>
</tapered_cylinder>
```

---

## 7章 COLLADA FX

---

このページは空白です。

## レンダリング状態

### 概要

異なる FX プロファイルは、<pass>要素の中で利用可能な別々のレンダリング状態セットを持ちます。一般に、それぞれのレンダリング状態の要素は、以下の宣言に準拠します。

```
<render_state value="some_value" param="param_reference"/>
```

この際、value 属性でレンダリング状態に固有の値が設定でき、また param 属性で、状態用の param 中に保存された値を利用することができます。

これらの状態の詳細に関しては、OpenGL の仕様書を参照してください。

以下の表に、<profile\_CG>、<profile\_GLSL>、<profile\_GLES>のレンダリング状態を示しておきます。それぞれのレンダリング状態は、GLES プロファイル用に明記されている違い以外は基本的に同じです。

状態	値	GLES での違い
alpha_func func  value	NEVER、LESS、LEQUAL、 EQUAL、GREATER、NOTEQUAL、 GEQUAL、ALWAYS  float 値の 0.0～1.0 が含まれる	
blend_func src dest	[src と dest の両方] ZERO、ONE、SRC_COLOR、 ONE_MINUS_SRC_COLOR、 DEST_COLOR、 ONE_MINUS_DEST_COLOR、 SRC_ALPHA、 ONE_MINUS_SRC_ALPHA、 DEST_ALPHA、 ONE_MINUS_DEST_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	GLES にはない
blend_equation	FUNC_ADD、FUNC_SUBTRACT、 FUNC_REVERSE_SUBTRACT、 MIN、MAX	GLES にはない
blend_equation_separate rgb alpha	blend_equation の値と同じ	GLES にはない
color_material face mode	FRONT、BACK、 FRONT_AND_BACK EMISSION、AMBIENT、 DIFFUSE、SPECULAR、 AMBIENT AND DIFFUSE	GLES にはない

cull_face	FRONT、BACK、 FRONT AND BACK	
depth_func	NEVER、LESS、LEQUAL、 EQUAL、GREATER、NOTEQUAL、 GEQUAL、ALWAYS	
fog_mode	LINEAR、EXP、EXP2	
fog_coord_src	FOG_COORDINATE、 FRAGMENT_DEPTH	GL ES にはない
front_face	CW、CCW	
light_model_color_control	SINGLE_COLOR、 SEPARATE_SPECULAR_COLOR	GL ES にはない
logic_op	CLEAR、AND、AND_REVERSE、 COPY、AND_INVERTED、NOOP、 XOR、OR、NOR、EQUIV、 INVERT、OR_REVERSE、 COPY_INVERTED、NAND、SET	
polygons_mode Face mode	FRONT、BACK、 FRONT_AND_BACK POINT、 LINE、FILL	GL ES にはない
shade_model	FLAT、SMOOTH	
stencil_func Func  Ref mask	NEVER、LESS、LEQUAL、 EQUAL、GREATER、NOTEQUAL、 GEQUAL、ALWAYS  符号なし byte 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	
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	GL ES にはない
stencil_mask_separate Face mask	FRONT、BACK、 FRONT_AND_BACK 符号なし byte	GL ES にはない
light_enable	論理値 この要素には、光源を指定する index 属性があります。	enable_light
light_ambient	float4 値 この要素には、光源を指定する index 属性があります。	
light_diffuse	float4 値 この要素には、光源を指定する index 属性があります。	
light_specular	float4 値 この要素には、光源を指定する index 属性があります。	



light_position	float4 値 この要素には、光源を指定する index 属性があります。	
light_constant_attenuation	Float 値 この要素には、光源を指定する index 属性があります。	
light_linear_attenuation	Float 値 この要素には、光源を指定する index 属性があります。	
light_quadratic_attenuation	float 値 この要素には、光源を指定する index 属性があります。	
light_spot_cutoff	float 値 この要素には、光源を指定する index 属性があります。	
light_spot_direction	float3 値 この要素には、光源を指定する index 属性があります。	
light_spot_exponent	float 値 この要素には、光源を指定する index 属性があります。	
texture1D	Sampler1D 型 この要素には、テクスチャユニットを指定する index 属性があります。	GL ES にはない
texture2D	Sampler2D 型 この要素には、テクスチャユニットを指定する index 属性があります。	GL ES にはない < <a href="#">texture_pipeline</a> >を参照。
texture3D	Sampler3D 型 この要素には、テクスチャユニットを指定する index 属性があります。	GL ES にはない
textureCUBE	SamplerCUBE 型 この要素には、テクスチャユニットを指定する index 属性があります。	GL ES にはない
textureRECT	SamplerRECT 型 この要素には、テクスチャユニットを指定する index 属性があります。	GL ES にはない
textureDEPTH	SamplerDEPTH 型 この要素には、テクスチャユニットを指定する index 属性があります。	GL ES にはない
texture1D_enable	Boolean 型 この要素には、テクスチャユニットを指定する index 属性があります。	GL ES にはない
texture2D_enable	Boolean 型 この要素には、テクスチャユニットを指定する index 属性があります。	GL ES にはない < <a href="#">texture_pipeline</a> >を参照。
texture3D enable	Boolean 値	GL ES にはない

	この要素には、テクスチャユニットを指定する index 属性があります。	
textureCUBE_enable	Boolean 値 この要素には、テクスチャユニットを指定する index 属性があります。	GLES にはない
textureRECT_enable	Boolean 値 この要素には、テクスチャユニットを指定する index 属性があります。	GLES にはない
textureDEPTH_enable	Boolean 値 この要素には、テクスチャユニットを指定する index 属性があります。	GLES にはない
texture_env_color	float4 値 この要素には、テクスチャユニットを指定する index 属性があります。	GLES にはない <texture_pipeline>を参照。
texture_env_mode	String 値 この要素には、テクスチャユニットを指定する index 属性があります。	GLES にはない <texture_pipeline>を参照。
clip_plane	float4 値 この要素にはクリッピングプレーンを指定する index 属性があります。	
clip_plane_enable	Boolean 値 この要素にはクリッピングプレーンを指定する index 属性があります。	
blend color	float4 値	
clear color	float4 値	
clear stencil	Int 値	
clear depth	float 値	
color mask	Bool4 値	
depth bounds	float2 値	
depth mask	Boolean 値	
depth range	float2 値	
fog density	float 値	
fog start	float 値	
fog end	float 値	
fog color	float4 値	
light model ambient	float4 値	
lighting enable	Boolean 値	
line stipple	Int2 値	
line width	float 値	
material ambient	float4 値	
material diffuse	float4 値	
material emission	float4 値	
material shininess	float 値	
material specular	float4 値	
model view matrix	float4x4 値	

point distance attenuation	float3 値	
point fade threshold size	float 値	
point size	float 値	
point size min	float 値	
point size max	float 値	
polygon offset	float2 値	
projection matrix	float4x4 値	
scissor	Int4 値	
stencil mask	Int 値	
alpha_test_enable	Boolean 値	enable_alpha_test
auto normal enable	Boolean 値	GLES にはない
blend enable	Boolean 値	enable_blend
color_logic_op_enable	Boolean 値	enable_color_logic_op
cull_face_enable	Boolean 値	enable_cull_face
depth bounds enable	Boolean 値	GLES にはない
depth clamp enable	Boolean 値	GLES にはない
depth_test_enable	Boolean 値	enable_depth_test
dither enable	Boolean 値	enable_dither
fog enable	Boolean 値	enable_fog
light model local viewer enable	Boolean 値	GLES にはない
light_model_two_side_enable	Boolean 値	enable_light_model_two_side
line_smooth_enable	Boolean 値	GLES にはない
line_stipple_enable	Boolean 値	GLES にはない
logic_op_enable	Boolean 値	enable_logic_op
multisample_enable	Boolean 値	enable_multisample
normalize_enable	Boolean 値	enable_normalize
point_smooth_enable	Boolean 値	GLES にはない
polygon_offset_fill_enable	Boolean 値	enable_polygon_offset_fill
polygon_offset_line_enable	Boolean 値	GLES にはない
polygon_offset_point_enable	Boolean 値	GLES にはない
polygon_smooth_enable	Boolean 値	GLES にはない
polygon_stipple_enable	Boolean 値	GLES にはない
rescale_normal_enable	Boolean 値	enable_rescale_normal
sample_alpha_to_coverage_enable	Boolean 値	enable_sample_alpha_to_coverage
sample_alpha_to_one_enable	Boolean 値	enable_sample_alpha_to_one
sample_coverage_enable	Boolean 値	enable_sample_coverage

7-8 COLLADA – Digital Asset Schema、リリース 1.4.0

scissor_test_enable	Boolean 値	enable_scissor_test
stencil_test_enable	Boolean 値	enable_stencil_test
gl_hook_abstract	GL 拡張からレンダリング状態を追加するための要素	GL ES にはない
texture_pipeline	String 値 <texture_pipeline>パラメータの名称	GL ES のみ
texture_pipeline_enable	Boolean 値	GL ES のみ

---

## alpha

### 概要

<texture\_pipeline>コマンドのアルファ要素を定義します。これは、混合モードのテクスチャ操作です。

### 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<texture\_pipeline>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。

operator	REPLACE   MODULATE   ADD   ADD_SIGNED   INTERPOLATE   SUBTRACT	glTexEnv(TEXTURE_ENV, COMBINE_ALPHA, operator)での利用を想定
scale	float	glTexEnv(TEXTURE_ENV, ALPHA_SCALE, scale)での利用を想定

### 関連要素

<alpha>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	<a href="#">texcombiner</a>
子要素	<a href="#">argument</a>
その他	なし

<argument>要素は、実行する特定の操作に必要な引数を設定します。

### 備考

### 例

<texture\_pipeline>を参照してください。

---

## annotate

### 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

注釈というのは SYMBOL=VALUE という形式のオブジェクトで、SYMBOL はユーザ定義された識別子であり、VALUE は強く型付けされた値です。注釈は、効果ランタイムからアプリケーションに対してメタ情報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:NCName	この要素のテキスト文字列名（オプション）
------	-----------	----------------------

### 関連要素

<annotate>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	<a href="#">effect</a> 、 <a href="#">technique</a> 、 <a href="#">pass</a> 、 <a href="#">newparam</a> 、 <a href="#">setparam</a> 、 <a href="#">generator</a>
子要素	bool、bool2、bool3、bool4、int、int2、int3、int4、float、float2、float3、float4、float2x2、float3x3、float4x4、string
その他	なし

### 備考

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>
```

## array

### 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の並びで、多次元配列は配列型の配列として宣言されます。

配列は、サイズを宣言しても、または未サイズのままでもかまいません。未サイズの配列は、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。

length	xs:positiveInteger	配列中の要素数を指定する
--------	--------------------	--------------

length 属性で、配列中の要素の数を指定します。

### 関連要素

`<array>`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>newparam</code> 、 <code>setparam</code>
子要素	<code>VALUE_TYPES</code> 、 <code>CG_PARAM_TYPE</code> 、 <code>GLSL_PARAM_TYPE</code> 、 <code>array</code> 、 <code>usertype</code> 、 <code>connect_param</code>
その他	なし

### 備考

作成した後、配列のインデックス化と構造体の非参照を行う通常の CG シンタックス（例：`array[3].element`）を利用して、`<setparam>`の宣言の中で配列要素を直接記述できます。

### 例

```
<newparam sid="numbers">
  <array length="4">
    <float>1.0</float>
    <float>2.0</float>
    <float>3.0</float>
    <float>4.0</float>
  </array>
</newparam>
<setparam ref="numbers[2]">
  <float>2.5</float>
</setparam>
```

## argument

### 概要

<argument>要素は、テクスチャユニットの混合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ためのものです。

### 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。

idx	int 0-2
source	TEXTURE   CONSTANT   PRIMARY   PREVIOUS
operand	親が RGB の場合： SRC_COLOR   ONE_MINUS_SRC_COLOR   SRC_ALPHA   ONE_MINUS_SRC_ALPHA 親が alpha の場合： SRC_ALPHA   ONE_MINUS_SRC_ALPHA
unit	xs:NCName

注意：以下のリストで、##は結合を意味し、idx と source は値を指定する場所を意味します。source は、引数のソースデータがどこにあるのかを表します。

- 親が RGB の場合、glTexEnv(TEXTURE\_ENV, SRC##idx##\_RGB, source) の呼び出しを暗に意味します。
- 親が alpha の場合、glTexEnv(TEXTURE\_ENV, SRC##idx##\_ALPHA, source) の呼び出しを暗に意味します。

operand は、ソースから値をどのように読み込むのか詳しい情報を表します。

- 親が RGB の場合の場合、glTexEnv(TEXTURE\_ENV, OPERAND##idx##\_RGB, source) の呼び出しを暗に意味します。
- 親が alpha の場合、glTexEnv(TEXTURE\_ENV, OPERAND##idx##\_ALPHA, source) の呼び出しを暗に意味します。

unit は、ソースが読み込まれる元のテクスチャユニットの名前を引数に対して提供します。この属性は、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。

- GLES 1.0 の場合、<texenv>要素中のすべての引数は、同じテクスチャユニットを参照しなければなりません。混合クロスバーがないからです。
- GLES 1.1 の場合、テクスチャの混合クロスバーが利用可能であるため、unit 属性でどのテクスチャユニットの名前をも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。

出現回数	スキーマ中に定義された要素の数
親要素	RGB、alpha
子要素	子要素はありません
その他	なし

## 備考

<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。

# bind

## 概要

値をシェーダへの均一の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

均一パラメータを持ったシェーダは、コンパイル時に個々の入力に対してバインドされた値を持つことができ、また実行時には均一の値に割り当てられた値を必要とします。これらの値は、リテラル値、定数パラメータ、均一パラメータのいずれでもかまいません。定数値の場合、それらの宣言はコンパイラによって利用され、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。

symbol	xs:NCName	シェーダへの均一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す。<bind>が<shader>の子の場合にだけ有効です。
semantic	xs:NCName	どの効果パラメータをバインドするのか指定します。<bind>が<instance material>の子の場合にだけ有効です。
target	xs:token	指定したセマンティックスにバインドする値の場所を指定します。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。<bind>が<instance material>の子の場合にだけ有効です。

## 関連要素

<bind>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	shader、instance material
子要素	VALUE TYPES、param
その他	なし

VALUE\_TYPES と<param>子要素は、<bind>要素が<shader>要素の子の場合にだけ利用できます。

## 備考

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均一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

```
<bind ref="diffusecol">
  <float3> 0.30 .52 0.05 </float3>
</bind>
<bind ref="lightpos">
  <param ref="OverheadLightPos_03">
</bind>
...
<instance_material symbol="RedMat" target="#RedCGEffect">
  <bind semantic="LIGHTPOS0" target="#LightNode/translate"/>
</instance_material>
```

## bind\_material

### 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均一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ようにリクエストすることができます。

```
<polygons name="leftarm" count="2445" material="bluePaint">
```

この抽象的なシンボルは特定のマテリアルインスタンスにバインドする必要があり、これは `<bind_material>` ブロックを持つ `<instance_geometry>` の最中に行われます。マテリアルのために名前でもリクエストされたジオメトリがスキャンされ、実際のマテリアルは、それらのシンボルにバインドされます。

マテリアルがバインドされていますが、やはり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、どの光源がマテリアルに利用されるのでしょうか？ また、1 つのオブジェクトに対してテクスチャ座標のセットが 1 つ必要な効果で、ジオメトリがテクスチャ座標を 2 セット定義している場合、どのセットが効果に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。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">instance_geometry</a> 、 <a href="#">instance_controller</a>
子要素	<a href="#">param</a> 、 <a href="#">technique</a> 、 <a href="#">technique common</a>
その他	なし

### 備考

`<bind_material>` の中の `<param>` オブジェクトは、アニメーションのターゲットに追加されます。これらのオブジェクトは、`<effect>` の内部レイアウトを解析するために、アニメーションのターゲット化システムを必要とせずに、通常の方法で入力パラメータにバインドすることが可能です。

## 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" url="/scene/light01.pos"/>
        <bind semantic="TEXCOORD0" url="BeechTree#texcoord2"/>
      </instance_material>
    </technique_common>
  </bind_material>
</instance_geometry>
```

## blinn

### 概要

<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表面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 Blinn BRDF 近似にしたがってシェーディング処理されます。

### コンセプト

<blinn>シェーダは、共通の Blinn シェーディング公式を利用します。

```
color = emissive + <ambient> * ambient_light + <diffuse> * max(N . L, 0) +
<specular>*max(H . I, 0)<shininess>
```

半角ベクトル  $H$  を利用している点に注意してください。たとえば、 $(I+L)/2$  というように、ユニット Eye と Light ベクトルとの間の半分として計算されます。

### 属性

<blinn>要素に属性はありません。

### 関連要素

<blinn>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">technique</a>
子要素	emission、 <a href="#">ambient</a> 、diffuse、reflective、specular、shininess、reflectivity、transparent、transparency、index of refraction
その他	なし

### 子要素

名前/例	解説	デフォルト値	出現回数
<emission>	<common_color_or_texture_type>を利用して、このオブジェクトの表面から放射される光源の量を宣言します。	なし	0 または 1
<ambient>	<common_color_or_texture_type>を利用して、このオブジェクトの表面から放射される環境光の量を宣言します。	なし	0 または 1
<diffuse>	<common_color_or_texture_type>を利用して、このオブジェクトの表面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
<specular>	<common_color_or_texture_type>を利用して、このオブジェクトの表面から反射される鏡面反射光の量を宣言します。	なし	0 または 1
<shininess>	<common_float_or_param_type>を利用して、このオブジェクトの表面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。	なし	0 または 1
<reflective>	完全鏡面反射のカラーを <common_color_or_texture_type>として	なし	0 または 1

	宣言します。		
<reflectivity>	<common_float_or_param_type>を利用して、反射された光に追加する完全鏡面反射の量（0.0～1.0 の値）を宣言します。	なし	0 または 1
<transparent>	完全鏡面反射のカラーを<common_color_or_texture_type>として宣言します。	なし	0 または 1
<transparency>	<common_float_or_param_type>を利用して、反射されたカラーに追加する完全鏡面反射の量を 0.0～1.0 のスカラー値として宣言します。	なし	0 または 1
<index_of_refraction>	<common_float_or_param_type>を利用して、完全鏡面反射の反射インデックスを単一のスカラーインデックスとして宣言します。	なし	0 または 1

## 備考

## 例

code

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	xs:NCName	オプション。この要素のサブ識別子を含んだテキスト文字列。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。
-----	-----------	--

--	--

関連要素

<code>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	<a href="#">technique</a> 、 <a href="#">generator</a> 、 <a href="#">profile CG</a> 、 <a href="#">profile GLSL</a>
子要素	子要素はありません
その他	なし

備考

インラインソースコードでは、すべての XML 識別子の文字をエスケープしておく必要があります。たとえば、"<"は"&lt;"としておきます。

例

```

<code sid="lighting_code">
atrix4x4 mat : MODELVIEWMATRIX;
float4 lighting_fn( varying float3 pos : POSITION,
    ...
</code>

```



---

## color\_clear

### 概要

レンダリングターゲット表面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表面を空のキャンバス、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。クリア文が含まれていない場合、ターゲット表面は、レンダリングが開始しても変更されません。

### 属性

<color\_clear>要素には、以下の属性があります。

index	xs:nonNegativeInteger	設定するマルチレンダリングのターゲット
-------	-----------------------	---------------------

### 関連要素

<color\_clear>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	<a href="#">pass</a>
子要素	子要素はありません
その他	なし

### 備考

現時点のプラットフォーム（Q4 2005）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、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具体的に設計されており、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

### 概要

特定のパスで、カラーの情報を出力から受け取る<surface>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）では、フラグメントシェーダに対して、パスごとに 1 つ以上の値を出力させたり、または標準の深度とステンシルの単位を任意のオフスクリーンバッファに書き込んだり、そこから読み込んだりするためにリダイレクトさせることが可能です。これらの要素は、FX ランタイムに対して、事前に定義されているどのサーフィスを利用するのかを伝えることになります。

### 属性

<color\_target>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 1 つのインデックス
slice	xs:nonNegativeInteger	ターゲット<surface>中のサブイメージのインデックス。単一 MIP マップレベル、ユニークなキューブ面、または 3 次元テクスチャのレイヤを含みます。

### 関連要素

<color\_target>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	pass
子要素	子要素はありません
その他	なし

### 備考

現在のプラットフォーム（Q4 2005）では、MRT の設定がかなり限定されています。たとえば、カラーバッファは 4 つだけで（すべて同じサイズで同じピクセルフォーマットでなければならない）、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しかアクティブにすることができません。COLLADA FX 宣言は、こういった制約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>中の特性の MRT の宣言を検証する必要があり、検証に失敗した場合にはエラーのフラグを設定しなければなりません。

<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">
  <surface type="2D"/>
</newparam>
<pass>
  <color_target>surfaceTex</color_target>
</pass>

```

## common\_color\_or\_texture\_type

### 概要

`<profile_COMMON>`効果の中の固定機能シェーダのカラー属性を宣言するためのタイプです。

### コンセプト

### 属性

`<common_color_or_texture_type>`に属性はありません。

### 関連要素

`<common_color_or_texture_type>`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>constant</code> 、 <code>lamert</code> 、 <code>phong</code> 、 <code>blinn</code>
子要素	<code>color</code> 、 <code>param</code> 、 <code>texture</code>
その他	なし

### 子要素

名前/例	解説	デフォルト値	出現回数
<code>&lt;color sid="mySID"&gt;</code>	リテラルのカラーを RGBA の順番で 4 つの浮動小数点で指定する。	なし	1
<code>&lt;param ref="myParam"&gt;</code>	<code>&lt;float4&gt;</code> に直接キャスト可能な現在の範囲で事前に定義されたパラメータを参照して値を指定する。	なし	1
<code>&lt;texture texture="myParam" texcoord="myUVs"&gt;</code>	セマンティックスによって <code>&lt;bind_material&gt;</code> の中のテクスチャ座標のストリームに結び付けられる事前に定義された <code>&lt;sampler2D&gt;</code> オブジェクトと <code>&lt;float2&gt;</code> パラメータを参照して値を指定する。 <code>&lt;texture&gt;</code> 宣言には、プラットフォーム固有の <code>&lt;extra&gt;</code> 情報を含めることが可能。	なし	1

### 備考

### 例

---

## common\_float\_or\_param\_type

### 概要

`<profile_COMMON>`効果の中の固定機能シェーダのスカラー属性を宣言するためのタイプです。

### コンセプト

### 属性

`<common_float_or_param_type>`要素に属性はありません。

### 関連要素

`<common_float_or_param_type>`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>constant</code> 、 <code>lamBERT</code> 、 <code>phong</code> 、 <code>blinn</code>
子要素	<code>float</code> 、 <code>param</code>
その他	なし

### 子要素

名前/例	解説	デフォルト値	出現回数
<code>&lt;float sid="mySID"&gt;</code>	リテラルの浮動小数点スカラーで値を指定する（例： <code>&lt;float&gt; 3.14 &lt;/float&gt;</code> ）	なし	1
<code>&lt;param&gt;</code>	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値を指定する。	なし	1

### 備考

### 例

---

## compiler\_options

### 概要

シェーダコンパイラ用のコマンドライン操作を含んだ文字列です。

### コンセプト

### 属性

`<compiler_options>`要素に属性はありません。

### 関連要素

`<compiler_options>`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	<a href="#">shader</a>
子要素	子要素はありません
その他	なし

### 備考

### 例

```
<compiler_options> -o3 -finlinelevel 6 </compiler_options>
```

---

## compiler\_target

### 概要

シェーダでコンパイラがどのプロファイルまたはプラットフォームを対象としているのかを宣言した文字列です。

### コンセプト

一部の FX ランタイムコンパイラは、複数のプラットフォームやハードウェア用のコードが生成できます。この宣言は、コンパイラが対象とするプロファイルを保持します。

### 属性

<compiler\_target>要素に属性はありません。

### 関連要素

<compiler\_target>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	<a href="#">shader</a>
子要素	子要素はありません
その他	なし

### 備考

### 例

## connect\_param

### 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くのシェーダの複数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、抽象インタフェー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。たとえば、均一の入力パラメータとして「Light」というタイプの抽象インタフェースをシェーダが持つ場合、この宣言は、そのパラメータへの具体的な<usertype>構造体のインスタンスを接続することで完全に解決できます。

### 属性

<connect\_param>要素には、以下の属性があります。

ref	xs:NCName	現在のパラメータに結び付けるターゲットパラメータへの参照
-----	-----------	------------------------------

### 関連要素

<connect\_param>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	setparam、array、usertype
子要素	子要素はありません
その他	なし

### 備考

### 例

```
<setparam ref="scene.light[2]">
  <connect_param ref="OverheadSpotlight_B"/>
</setparam>
```

## constant

### 概要

<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表面を常に生成する固定機能パイプラインを宣言します。

### コンセプト

反射カラーは、以下のように計算されます。

```
color = emission
```

### 属性

<constant>要素に属性はありません。

### 関連要素

<constant>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">technique</a>
子要素	emission、reflective、reflectivity、transparent、transparency、index of refraction
その他	なし

### 子要素

名前/例	解説	デフォルト値	出現回数
<emission>	<common_color_or_texture_type>を利用して、このオブジェクトの表面から放射される光源の量を宣言します。	なし	0 または 1
<reflective>	完全鏡面反射のカラーを<common_color_or_texture_type>として宣言します。	なし	0 または 1
<reflectivity>	<common_float_or_param_type>を利用して、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。	なし	0 または 1
<transparent>	完全鏡面反射光源のカラーを<common_color_or_texture_type>として宣言します。	なし	0 または 1
<transparency>	<common_float_or_param_type>を利用して、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。	なし	0 または 1
<index_of_refraction>	<common_float_or_param_type>を利用して、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。	なし	0 または 1
<float sid="mySID">	リテラルの浮動小数点スカラーで値を指定します (例: <float> 3.14 </float>)。	なし	1



<param>	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値を指定します。	なし	1
---------	---	----	---

**備考****例**

## depth\_clear

### 概要

レンダリングターゲットの表面をクリアするかどうか、またクリアする場合にはどの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表面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があります。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、ターゲットの表面はレンダリングが始まってでも変更されません。

### 属性

<depth\_clear>要素には、以下の属性があります。

index	xs:nonNegativeInteger	設定するマルチレンダリングターゲット
-------	-----------------------	--------------------

### 関連要素

<depth\_clear>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	<a href="#">pass</a>
子要素	子要素はありません
その他	なし

### 備考

現在のプラットフォーム（Q4 2005）では、MRT の設定がかなり限定されています。たとえば、カラーバッファは 4 つだけで（すべて同じサイズで同じピクセルフォーマットでなければならない）、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しかアクティブにすることができません。COLLADA FX 宣言は、こういった制約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>中の特性の MRT の宣言を検証する必要がある、検証に失敗した場合にはエラーのフラグを設定しなければなりません。

### 例

```
<depth_clear index="0">0.0</depth_clear>
```

## depth\_target

### 概要

特定のパスで、カラーと深度とステンシルの各情報を出力から受け取る<surface>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) では、フラグメントシェーダに対して、パスごとに 1 つ以上の値を出力させたり、または標準の深度とステンシルの単位を任意のオフスクリーンバッファに書き込んだり、そこから読み込んだりするためにリダイレクトさせることが可能です。これらの要素は、FX ランタイムに対して、事前に定義されているどのサーフェイスを利用するのかを伝えることになります。

### 属性

<depth\_target>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 1 つのインデックス
slice	xs:nonNegativeInteger	ターゲット<surface>中のサブイメージのインデックス。単一 MIP マップレベル、ユニークなキューブ面、または 3 次元テクスチャのレイヤを含みます。

### 関連要素

<depth\_target>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	pass
子要素	子要素はありません
その他	なし

### 備考

現在のプラットフォーム (Q4 2005) では、MRT の設定がかなり限定されています。たとえば、カラーバッファは 4 つだけで (すべて同じサイズで同じピクセルフォーマットでなければならない)、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しかアクティブにすることができません。COLLADA FX 宣言は、こういった制約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>中の特性の MRT の宣言を検証する必要があり、検証に失敗した場合にはエラーのフラグを設定しなければなりません。

<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深度バッファセットを利用します。

### 例

```
<newparam sid="depthSurface">
  <surface type="2D"/>
</newparam>
<pass>
  <color_target>depthSurface</color_target>
</pass>
```

---

## draw

### 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するユーザ定義文字列を指定します。

### 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らない場合があります。あるパスはモデルを送り出さなければならない、または別のパスは、オフスクリーンバッファ中の各ピクセルに対してフラグメントシェーダを実行するためにフルスクリーンの四角形が必要なのにに対して、さらに別のパスでは、正面のポリゴンしか必要ない場合があります。`<draw>` は、どんなジオメトリが特定のパスで想定されているのかを FX ランタイムに対して示すセマンティックスとして利用可能なユーザ定義文字列を宣言します。

### 属性

`<draw>` 要素に属性はありません。

### 関連要素

`<draw>` 要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">pass</a>
子要素	子要素はありません
その他	なし

### 備考

### 例

## effect

### 概要

COLLADA 効果の記述を表します。

### コンセプト

プログラマブル・パイプラインでは、高レベルな言語を利用して、3 次元パイプラインの各段階をプログラミングすることができます。これらのシェーダでは特定のデータを渡さなければならないことが多く、また正しく機能させるために、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ておく必要があります。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、サンプラー、シェーダ、入出力パラメータ、均一パラメータ、レンダリング状態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。

<effect>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
name	xs:NCName	効果の名称

### 関連要素

<effect>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	<a href="#">library effects</a>
子要素	<a href="#">asset</a> 、 <a href="#">annotate</a> 、 <a href="#">image</a> 、 <a href="#">newparam</a> 、 <a href="#">profile_CG</a> 、 <a href="#">profile_GLSL</a> 、 <a href="#">profile COMMON</a>
その他	なし

### 備考

### 例

---

## generator

### 概要

手続き型の表面ジェネレータです。

### コンセプト

### 属性

<generator>要素に属性はありません。

### 関連要素

<generator>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	<a href="#">surface</a> (CG と GLSL の範囲のみ)
子要素	<a href="#">annotate</a> 、 <a href="#">blinn</a> 、 <a href="#">include</a> 、 <a href="#">name</a> 、 <a href="#">setparam</a>
その他	なし

### 備考

### 例

---

## include

### 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	xs:NCName	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
url	xs:anyURI	リソースがある場所

### 関連要素

<include>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	<a href="#">technique</a> 、 <a href="#">generator</a> 、 <a href="#">profile CG</a> 、 <a href="#">profile GLSL</a>
子要素	子要素はありません
その他	なし

### 備考

### 例

```
<include sid="ShinyShader" url="file://assets/source/shader.glsl"/>
```

## instance\_effect

### 概要

`<instance_effect>`要素は、COLLADA マテリアルリソースのインスタンス化を宣言するためのものです。

### コンセプト

オブジェクトの実際のデータ表現は一度しか格納できません。しかしオブジェクトはシーン中で複数表示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ます。それぞれのオブジェクトのインスタンスは、ユニークであったり、他のインスタンスと同じデータを共有する場合があります。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。ユニークではない（共有されている）インスタンスは、そのオブジェクトの他のインスタンスとデータの一部またはすべてを共有します。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。

### 属性

`<instance_effect>`要素には、以下の属性があります。

url	xs:anyURI	必須。インスタンス化するオブジェクトの場所の URL。
-----	-----------	-----------------------------

### 関連要素

`<instance_effect>`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	<a href="#">material</a> 、 <a href="#">render</a>
子要素	<a href="#">technique hint</a> 、 <a href="#">setparam</a> 、 <a href="#">extra</a>
その他	なし

### 備考

### 例

```
<material id="BlueCarPaint" name="Light blue car paint">
  <instance_effect ref="CarPaint">
    <technique_hint platform="Xbox360" sid="precalc_texture">
      <setparam ref="diffuse_color">
        <float3> 0.3 0.25 0.85 </float3>
      </setparam>
    </instance_effect>
  </material>
```



## instance\_material

### 概要

`<instance_material>`要素は、COLLADA マテリアルリソースのインスタンス化を宣言するためのものです。

### コンセプト

オブジェクトの実際のデータ表現を一度しか格納できません。しかしオブジェクトはシーン中で複数表示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ます。

それぞれのオブジェクトのインスタンスは、ユニークであったり、他のインスタンスと同じデータを共有する場合があります。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。ユニークではない（共有されている）インスタンスは、そのオブジェクトの他のインスタンスとデータの一部またはすべてを共有します。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。

### 属性

`<instance_material>`要素には、以下の属性があります。

target	xs:anyURL	必須。インスタンス化するオブジェクトの場所の URL。
symbol	xs:NCName	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。

### 関連要素

`<instance_material>`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	technique common
子要素	<a href="#">bind</a> 、 <a href="#">extra</a>
その他	なし

### 備考

### 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" url="/scene/light01.pos"/>
        <bind semantic="TEXCOORD0" url="BeechTree#texcoord2"/>
      </instance_material>
    </technique_common>
  </bind_material>
</instance_geometry>
```

## lambert

### 概要

`<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表面を常に生成する固定機能パイプラインを宣言します。

### コンセプト

反射カラーは、以下のように計算します。

```
color = emission
```

### 属性

`<lambert>`要素に属性はありません。

### 関連要素

`<lambert>`要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<code>technique</code>
子要素	<code>emission</code> 、 <code>reflective</code> 、 <code>reflectivity</code> 、 <code>transparent</code> 、 <code>transparency</code> 、 <code>index of refraction</code>
その他	なし

### 子要素

名前/例	解説	デフォルト値	出現回数
<code>&lt;emission&gt;</code>	<code>&lt;common_color_or_texture_type&gt;</code> を利用して、このオブジェクトの表面から放射される光源の量を宣言します。	なし	0 または 1
<code>&lt;reflective&gt;</code>	完全鏡面反射のカラーを <code>&lt;common_color_or_texture_type&gt;</code> として宣言します。	なし	0 または 1
<code>&lt;reflectivity&gt;</code>	<code>&lt;common_float_or_param_type&gt;</code> を利用して、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。	なし	0 または 1
<code>&lt;transparent&gt;</code>	完全鏡面反射光源のカラーを <code>&lt;common_color_or_texture_type&gt;</code> として宣言します。	なし	0 または 1
<code>&lt;transparency&gt;</code>	<code>&lt;common_float_or_param_type&gt;</code> を利用して、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。	なし	0 または 1
<code>&lt;index_of_refraction&gt;</code>	<code>&lt;common_float_or_param_type&gt;</code> を利用して、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。	なし	0 または 1
<code>&lt;float sid="mySID"&gt;</code>	リテラルの浮動小数点スカラーで値を指定します (例: <code>&lt;float&gt; 3.14 &lt;/float&gt;</code> )。	なし	1
<code>&lt;param&gt;</code>	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値を指定します。	なし	1

備考

例

---

## modifier

### 概要

`<newparam>` 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。

### コンセプト

COLLADA FX パラメータ宣言で、定数、外部または均一パラメータを指定することができます。

### 属性

`<modifier>` 要素に属性はありません。

### 関連要素

`<modifier>` 要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">newparam</a>
子要素	子要素はありません
その他	なし

### 備考

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。

### 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>
```

---

## name

### 概要

シェーダ関数のエントリシンボルを指定します。

### コンセプト

シェーダコンパイラは、シェーダオブジェクトまたはバイナリコードにコンパイルする関数名を必要とします。翻訳ユニットというパラダイムを利用する FX ランタイムでは、オプションで、内部シンボルを検索するために翻訳ユニットまたはシンボルテーブルを指定することができます。

### 属性

<name>要素には、以下の属性があります。

source	xs:NCName	シンボルがある<code>または<include>ブロックのオプションの sid
--------	-----------	--

### 関連要素

<name>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	<a href="#">shader</a> 、 <a href="#">generator</a>
子要素	子要素はありません
その他	なし

### 備考

### 例

## newparam

### 概要

FX ランタイム中に新しい名前付きの<param>オブジェクトを作成して、宣言時に型と初期値と追加属性を割り当てます。

### コンセプト

パラメータは、FX 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。

### 属性

<newparam>要素には、以下の属性があります。

sid	xs:NCName	パラメータの識別子（つまり変数名）
-----	-----------	-------------------

### 関連要素

<newparam>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	<a href="#">effect</a> 、 <a href="#">technique</a> 、 <a href="#">profile CG</a> 、 <a href="#">profile COMMON</a> 、 <a href="#">profile GLSL</a>
子要素	<a href="#">annotate</a> 、 <a href="#">semantic</a> 、 <a href="#">modifier</a> 、 <a href="#">VALUE TYPES</a>
その他	なし

### 備考

### 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>
```

## param

### 概要

シェーダバインディング宣言で事前に定義されたパラメータを参照します。

### コンセプト

パラメータは、FX 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る<param>オブジェクトです。

### 属性

<param>要素には、以下の属性があります。

name	xs:NCName
sid	xs:NCName
semantic	ns:NMTOKEN
type	ns:NMTOKEN

### 関連要素

<param>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	bind、texture1D、texture2D、texture3D、textureCUBE、textureRECT、textureDEPTH
子要素	子要素はありません
その他	なし

### 備考

### 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>
<pass>
  <shader>
    <bind symbol="inColor">
      <param ref="diffuseColor"/>
    </bind>
  </shader>
</pass>
```

## pass

### 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状態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状態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状態を「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、グラフィックス状態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状態の設定と均一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	xs:NCName	オプションのラベルで、名前で <code>&lt;pass&gt;</code> を指定できるようにします。また必要であれば、テクニ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。
-----	-----------	---

### 関連要素

`<pass>`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	<a href="#">technique</a>
子要素	<a href="#">annotate</a> 、 <a href="#">color_target</a> 、 <a href="#">depth_target</a> 、 <a href="#">stencil_target</a> 、 <a href="#">color_clear</a> 、 <a href="#">depth_clear</a> 、 <a href="#">stencil_clear</a> 、 <a href="#">draw</a> 、 <a href="#">RENDER_STATES</a> 、 <a href="#">shader</a>
その他	なし

### 備考

`<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」のローパス重量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">
  <depth_test_enable value="true"/>
  <depth_func value="LEQUAL"/>
  <shader stage="VERTEX">
    <name>goochVS</name>
    <bind symbol="LightPos">
      <param ref="effectLightPos"/>
    </bind>
  </shader>
  <shader stage="FRAGMENT">
```



```
        <name>passThruFS</name>  
    </shader>  
</pass>
```

## phong

### 概要

<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表面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。

### コンセプト

<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。

$$\text{color} = \text{emissive} + \langle \text{ambient} \rangle * \text{ambient\_light} + \langle \text{diffuse} \rangle * \max(\mathbf{N} \cdot \mathbf{L}, 0) + \langle \text{specular} \rangle * \max(\mathbf{H} \cdot \mathbf{I}, 0) \langle \text{shininess} \rangle$$

Blinn シェーディングで利用している半角ベクトル H ではなく、完全反射ベクトル R を利用している点に注意してください。

### 属性

<phong>要素に属性はありません。

### 関連要素

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">technique</a>
子要素	emission、 <a href="#">ambient</a> 、diffuse、specular、shininess、reflective、reflectivity、transparent、transparency、index of refraction
その他	なし

### 子要素

名前/例	解説	デフォルト値	出現回数
<emission>	<common_color_or_texture_type> を利用して、このオブジェクトの表面から放射される光源の量を宣言します。	なし	0 または 1
<ambient>	<common_color_or_texture_type> を利用して、このオブジェクトの表面から放射される環境光の量を宣言します。	なし	0 または 1
<diffuse>	<common_color_or_texture_type> を利用して、このオブジェクトの表面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
<specular>	<common_color_or_texture_type> を利用して、このオブジェクトの表面から反射される鏡面反射光の量を宣言します。	なし	0 または 1
<shininess>	<common_float_or_param_type> を利用して、このオブジェクトの表面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。	なし	0 または 1
<reflective>	完全鏡面反射のカラーを <common_color_or_texture_type> として宣言します。	なし	0 または 1
<reflectivity>	<common_float_or_param_type> を利用して、反射された光に追加する完全鏡面反射の量 (0.0~1.0 の値) を宣言します。	なし	0 または 1

<transparent>	完全鏡面反射のカラーを <common_color_or_texture_type>として宣言します。	なし	0 または 1
<transparency>	<common_float_or_param_type>を利用して、反射されたカラーに追加する完全鏡面反射の量を 0.0～1.0 のスカラー値として宣言します。	なし	0 または 1
<index_of_refraction>	<common_float_or_param_type>を利用して、完全鏡面反射の反射インデックスを単一のスカラーインデックスとして宣言します。	なし	0 または 1

## 備考

## 例

# profile\_CG

## 概要

プラットフォーム固有のデータ型と<technique>宣言のブロックをオープンします。

## コンセプト

<profile\_CG>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\_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。

## 属性

<profile\_CG>には、以下の属性があります。

platform	xs:NMTOKEN	オプション。プラットフォームのタイプ。これはベンダ定義された文字列で、プラットフォームまたは technique 用の機能ターゲットを表します。
----------	------------	--

## 関連要素

<profile\_CG>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	<a href="#">effect</a>
子要素	<a href="#">code</a> 、 <a href="#">include</a> 、 <a href="#">image</a> 、 <a href="#">newparam</a> 、 <a href="#">technique</a>
その他	なし

## 備考

## 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>
  <technique id="default" sid="default">
    <code>
void passthroughVS (in varying float4 pos,
  in uniform float3 light_pos,
  in uniform float4x4 mat : MODELVIEWPROJ,
  out varying float4 oPosition : POSITION,
```

```

        out varying float3 oToLight : TEXCOORD0 )
    { oPosition = mul(modelViewProj, position);
      oToLight = light_pos - pos.xyz;
      return;
    }

float3 diffuseFS (in uniform float3 flat_color,
                  in varying float3 to_light : TEXCOORD0 ) : COLOR
    { return flat_color * clamp(1.0 - sqrt(dot(to_light, to_light)),
                                0.0, 1.0);
    }
</code>
<pass sid="single_pass">
    <shader stage="VERTEX">
        <name>passthroughVS</name>
        <bind symbol="light_pos">
            <param ref="lightpos"/>
        </bind>
    </shader>
    <shader stage="FRAGMENT">
        <name>diffuseFS</name>
        <bind symbol="flat_color">
            <param ref="color"/>
        </bind>
    </shader>
</pass>
</technique>
</profile_CG>

```

## profile\_COMMON

### 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、プラットフォームとは独立した固定機能シェーダの値と宣言をすべてカプセル化し、すべてのプラットフォームで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。

### 属性

<profile\_COMMON 要素に属性はありません。

### 関連要素

<profile\_COMMON>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	<a href="#">effect</a>
子要素	<a href="#">image</a> 、 <a href="#">newparam</a> 、 <a href="#">technique</a>
その他	なし

### 子要素

名前/例	解説	デフォルト値	出現回数
<image id="myID" name="./BrickTexture" format="R8G8B8A8" height="64" width="128" depth="1">	標準の COLLADA イメージリソースを宣言します。	なし	任意
<newparam sid="mySID"> <semantic> DIFFUSECOLOR </semantic> <float3> 1 2 3 </float3> </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スを伴った<float>、<float2>、<float3>、<float4>、<surface>、<sampler2D>）から新しいパラメータを作成します。	なし	任意
<technique>	この効果用の technique を 1 つだけ宣言します。このノードには、<asset>と<image>と<extra>を含み、<constant>、<lambert>、<phong>、<blinn>のいずれか 1 つを追加することができます。	なし	1

## 備考

## 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>myDiffuseColor</param></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

### 概要

プラットフォーム固有のデータ型と<technique>宣言のブロックをオープンします。

### コンセプト

<profile\_GLES>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\_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。

### 属性

<profile\_GLES>には、以下の属性があります。

platform	xs:NMTOKEN	オプション。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。
----------	------------	---

### 関連要素

<profile\_GLES>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	effect
子要素	image、newparam、technique
その他	なし

<newparam>には GLES の新しいオブジェクトが含まれます。<newparam>用の GLES 固有の VALUE\_TYPES には、<texture\_pipeline>と<sampler\_state>があります。

### 備考

### 例



---

## profile\_GLSL

### 概要

プラットフォーム固有のデータ型と<technique>宣言のブロックをオープンします。

### コンセプト

<profile\_GLSL>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\_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。

### 属性

<profile\_GLSL>要素に属性はありません。

### 関連要素

<profile\_GLSL>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	<a href="#">effect</a>
子要素	<a href="#">code</a> 、 <a href="#">include</a> 、 <a href="#">image</a> 、 <a href="#">newparam</a> 、 <a href="#">technique</a>
その他	なし

### 備考

### 例

---

## RGB

### 概要

<texture\_pipeline>コマンドの RGB 部分を定義します。これは、混合モードのテクスチャ操作です。

### 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。

### 属性

<RGB>要素には、以下の属性があります。

operator	REPLACE   MODULATE   ADD   ADD_SIGNED   INTERPOLATE   SUBTRACT   DOT3_RGB   DOT3_RGBA	glTexEnv(TEXTURE_ENV, COMBINE_RGB, operator)での利用を想定
scale	float	glTexEnv(TEXTURE_ENV, RGB_SCALE, scale)での利用を想定

### 関連要素

<RGB>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	<a href="#">texcombiner</a>
子要素	<a href="#">argument</a>
その他	なし

<argument>要素は、実行する特定の操作に必要な引数を設定します。

### 備考

### 例

<texture\_pipeline>を参照してください。

---

## sampler1D

### 概要

1次元のテクスチャサンプラです。

### コンセプト

### 属性

<sampler1D>要素に属性はありません。

### 関連要素

<sampler1D>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">newparam</a> 、 <a href="#">setparam</a> 、 <a href="#">usertype</a>
子要素	<a href="#">source</a> 、 <a href="#">wrap_s</a> 、 <a href="#">minfilter</a> 、 <a href="#">magfilter</a> 、 <a href="#">mipfilter</a> 、 <a href="#">border_color</a> 、 <a href="#">mipmap_maxlevel</a> 、 <a href="#">mipmap_bias</a>
その他	なし

<[source](#)>要素は、0 個または 1 つ指定できます。  
<[wrap\\_s](#)>要素は、0 個または 1 つ指定できます。  
<[minfilter](#)>要素は、0 個または 1 つ指定できます。  
<[magfilter](#)>要素は、0 個または 1 つ指定できます。  
<[mipfilter](#)>要素は、0 個または 1 つ指定できます。  
<[border\\_color](#)>要素は、0 個または 1 つ指定できます。  
<[mipmap\\_maxlevel](#)>要素は、0 個または 1 つ指定できます。  
<[mipmap\\_bias](#)>要素は、0 個または 1 つ指定できます。

### 備考

### 例

---

## sampler2D

### 概要

2次元のテクスチャサンプラです。

### コンセプト

### 属性

<sampler2D>要素に属性はありません。

### 関連要素

<sampler2D>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">newparam</a> 、 <a href="#">setparam</a> 、 <a href="#">usertype</a>
子要素	<a href="#">source</a> 、 <a href="#">wrap_s</a> 、 <a href="#">wrap_t</a> 、 <a href="#">minfilter</a> 、 <a href="#">magfilter</a> 、 <a href="#">mipfilter</a> 、 <a href="#">border_color</a> 、 <a href="#">mipmap_maxlevel</a> 、 <a href="#">mipmap_bias</a>
その他	なし

<[source](#)>要素は、0 個または1 指定できます。  
<[wrap\\_s](#)>要素は、0 個または1 指定できます。  
<[wrap\\_t](#)>要素は、0 個または1 指定できます。  
<[minfilter](#)>要素は、0 個または1 指定できます。  
<[magfilter](#)>要素は、0 個または1 指定できます。  
<[mipfilter](#)>要素は、0 個または1 指定できます。  
<[border\\_color](#)>要素は、0 個または1 指定できます。  
<[mipmap\\_maxlevel](#)>要素は、0 個または1 指定できます。  
<[mipmap\\_bias](#)>要素は、0 個または1 指定できます。

### 備考

### 例

---

## sampler3D

### 概要

3次元のテクスチャサンプラです。

### コンセプト

### 属性

<sampler3D>要素に属性はありません。

### 関連要素

<sampler3D>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">newparam</a> 、 <a href="#">setparam</a> 、 <a href="#">usertype</a>
子要素	<a href="#">source</a> 、 <a href="#">wrap_s</a> 、 <a href="#">wrap_t</a> 、 <a href="#">wrap_p</a> 、 <a href="#">minfilter</a> 、 <a href="#">magfilter</a> 、 <a href="#">mipfilter</a> 、 <a href="#">border_color</a> 、 <a href="#">mipmap_maxlevel</a> 、 <a href="#">mipmap_bias</a>
その他	なし

<[source](#)>要素は、0 個または1 指定できます。  
<[wrap\\_s](#)>要素は、0 個または1 指定できます。  
<[wrap\\_t](#)>要素は、0 個または1 指定できます。  
<[wrap\\_p](#)>要素は、0 個または1 指定できます。  
<[minfilter](#)>要素は、0 個または1 指定できます。  
<[magfilter](#)>要素は、0 個または1 指定できます。  
<[mipfilter](#)>要素は、0 個または1 指定できます。  
<[border\\_color](#)>要素は、0 個または1 指定できます。  
<[mipmap\\_maxlevel](#)>要素は、0 個または1 指定できます。  
<[mipmap\\_bias](#)>要素は、0 個または1 指定できます。

### 備考

### 例

---

## samplerCUBE

### 概要

キューブマップ用のテクスチャサンプラです。

### コンセプト

### 属性

<samplerCUBE>要素に属性はありません。

### 関連要素

<samplerCUBE>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">newparam</a> 、 <a href="#">setparam</a> 、 <a href="#">usertype</a>
子要素	<a href="#">source</a> 、wrap_s、wrap_t、wrap_p、minfilter、magfilter、mipfilter、border color、mipmap maxlevel、mipmap bias
その他	なし

<source>要素は、0 個または 1 つ指定できます。  
<wrap\_s>要素は、0 個または 1 つ指定できます。  
<wrap\_t>要素は、0 個または 1 つ指定できます。  
<wrap\_p>要素は、0 個または 1 つ指定できます。  
<minfilter>要素は、0 個または 1 つ指定できます。  
<magfilter>要素は、0 個または 1 つ指定できます。  
<mipfilter>要素は、0 個または 1 つ指定できます。  
<border\_color>要素は、0 個または 1 つ指定できます。  
<mipmap\_maxlevel>要素は、0 個または 1 つ指定できます。  
<mipmap\_bias>要素は、0 個または 1 つ指定できます。

### 備考

### 例

---

## samplerRECT

### 概要

3次元のテクスチャサンプラです。

### コンセプト

### 属性

<samplerRECT>要素に属性はありません。

### 関連要素

<samplerRECT>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">newparam</a> 、 <a href="#">setparam</a> 、 <a href="#">usertype</a>
子要素	<a href="#">source</a> 、 <a href="#">wrap_s</a> 、 <a href="#">wrap_t</a> 、 <a href="#">minfilter</a> 、 <a href="#">magfilter</a> 、 <a href="#">mipfilter</a> 、 <a href="#">border_color</a> 、 <a href="#">mipmap_maxlevel</a> 、 <a href="#">mipmap_bias</a>
その他	なし

<[source](#)>要素は、0個または1つ指定できます。  
<[wrap\\_s](#)>要素は、0個または1つ指定できます。  
<[wrap\\_t](#)>要素は、0個または1つ指定できます。  
<[minfilter](#)>要素は、0個または1つ指定できます。  
<[magfilter](#)>要素は、0個または1つ指定できます。  
<[mipfilter](#)>要素は、0個または1つ指定できます。  
<[border\\_color](#)>要素は、0個または1つ指定できます。  
<[mipmap\\_maxlevel](#)>要素は、0個または1つ指定できます。  
<[mipmap\\_bias](#)>要素は、0個または1つ指定できます。

### 備考

### 例

---

## sampler\_state

### 概要

<profile\_GLES>用の 2 次元のテクスチャサンプラ状態です。これは、1 つまたは複数の <texture\_pipeline>で参照されるサンプラ固有の状態のバンドルです。

### コンセプト

### 属性

<sampler\_state>要素に属性はありません。

### 関連要素

<sampler\_state>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	<a href="#">newparam</a> 、 <a href="#">setparam</a> 、 <a href="#">usertype</a>
子要素	<a href="#">wrap_s</a> 、 <a href="#">wrap_t</a> 、 <a href="#">minfilter</a> 、 <a href="#">magfilter</a> 、 <a href="#">mipfilter</a> 、 <a href="#">mipmap_maxlevel</a> 、 <a href="#">mipmap_bias</a>
その他	なし

<wrap\_s>要素は、0 個または 1 つ指定できます。  
<wrap\_t>要素は、0 個または 1 つ指定できます。  
<minfilter>要素は、0 個または 1 つ指定できます。  
<magfilter>要素は、0 個または 1 つ指定できます。  
<mipfilter>要素は、0 個または 1 つ指定できます。  
<border\_color>要素は、0 個または 1 つ指定できます。  
<mipmap\_maxlevel>要素は、0 個または 1 つ指定できます。  
<mipmap\_bias>要素は、0 個または 1 つ指定できます。

### 備考

### 例



## semantic

### 概要

パラメータ宣言の目的を記述するメタ情報を指定します。

### 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情報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）
- 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。

出現回数	スキーマ中に定義された要素の数
親要素	<a href="#">newparam</a>
子要素	子要素はありません
その他	なし

### 備考

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。

### 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>
```

## setparam

### 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。

<setparam>のインスタンスはどれも等しい点に注意してください。特定の<profile\_\*>の中の<setparam>は、プラットフォーム固有のデータ型と定義へのアクセス権を持ちます。これに対して、<instance\_material>ブロック中の<setparam>は、共通の COLLADA データ型のプールからの値だけを割り当てることができます。

<setparam>は、事前に注釈を削除されたパラメータに対して注釈を追加する 1 つの方法です。高度な言語プロファイルでは、<array length="N"/>要素を利用して、サイズが未定義の配列に具体化された配列サイズを割り当てるのに<setparam>が利用できます。また、<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るのにも利用できます。

### 属性

<setparam>要素には、以下の属性があります。

ref	xs:NCName	値セットを持つ事前に定義されたパラメータを参照します。
-----	-----------	-----------------------------

### 関連要素

<setparam>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	<a href="#">technique</a> 、 <a href="#">generator</a> 、 <a href="#">instance effect</a>
子要素	<a href="#">annotate</a> 、 <a href="#">VALUE TYPES</a> 、 <a href="#">usertype</a> 、 <a href="#">array</a> 、 <a href="#">connect param</a>
その他	なし

### 備考

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

```
<setparam ref="light_Direction">
  <annotate name="UIWidget"> <string>text</string> </annotate>
  <float3> 0.0 1.0 0.0 </float3>
</setparam>
```

## shader

### 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均一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。「翻訳ユニット」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。

均一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。

stage	プラットフォーム固有の列挙	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（例：VERTEX、FRAGMENT など）。
-------	---------------	--

### 関連要素

<shader>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	pass
子要素	annotate、compiler_target、common_color_or_texture_type、name、bind
その他	なし

### 備考

## 例

```
<shader stage="VERTEX">
  <compiler_target>ARBVP1</compiler_target>
  <name source="ThinFilm2">main</entry>
  <bind symbol="lightpos">
    <param ref="LightPos_03"/>
  </bind>
</shader>
```

## stencil\_clear

### 概要

レンダリングターゲット表面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表面を空のキャンバス、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。これらの<stencil\_clear>宣言では、どの値を利用するのか指定します。クリア文が含まれていない場合、ターゲット表面は、レンダリングが開始しても変更されません。

### 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。
-------	-----------------------	--------------------------------

### 関連要素

<stencil\_clear>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	pass
子要素	子要素はありません
その他	なし

### 備考

現時点のプラットフォーム（Q4 2005）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、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具体的に設計されており、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">0.0</stencil_clear>
```

## stencil\_target

### 概要

特定のパスで、ステンシルの各情報を出力から受け取る<surface>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）では、フラグメントシェーダに対して、パスごとに 1 つ以上の値を出力させたり、または標準の深度とステンシルの単位を任意のオフスクリーンバッファに書き込んだり、そこから読み込んだりするためにリダイレクトさせることが可能です。これらの要素は、FX ランタイムに対して、事前に定義されているどのサーフェイスを利用するのかを伝えることになります。

### 属性

<stencil\_target>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いずれか 1 つを指し示します。
slice	xs:nonNegativeInteger	単一の MIP マップレベル、ユニークな立方体の面、または 3 次元テクスチャのレイヤも含め、対象とする<surface>の中のサブイメージを指し示します。

### 関連要素

<stencil\_target>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	pass
子要素	子要素はありません
その他	なし

### 備考

現在のプラットフォーム（Q4 2005）では、MRT の設定がかなり限定されています。たとえば、カラーバッファは 4 つだけで（すべて同じサイズで同じピクセルフォーマットでなければならない）、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しかアクティブにすることができません。COLLADA FX 宣言は、こういった制約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>中の特性の MRT の宣言を検証する必要があり、検証に失敗した場合にはエラーのフラグを設定しなければなりません。  
 <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">
  <surface type="2D"/>
</newparam>
<pass>
  <stencil_target>surfaceTex</stencil_target>
</pass>

```

## surface

### 概要

テクスチャサンプルのソースとレンダリングパスのターゲットの両方として利用されるリソースを宣言します。

### コンセプト

<surface>というのは、GPU レンダリングのために<image>を抽象的に汎用化したもので、たとえば、事前にフィルタリングされた N レベルの MIP マッピングされたイメージなど、複数の<image>リソースを単一のオブジェクトにリンクすることが可能で、6 つの四角形のテクスチャを立方体マップに結び付けることができます。

<surface>オブジェクトは、個々のピクセルのフィールドのサイズとレイアウトを表すデータ形式を持ち、<size>を利用してピクセルの絶対値にサイズ化したり、<viewport\_ratio>を利用してビューポートの分数としてサイズ化することが可能です。また、<mip\_levels>を利用して決まった数の MIP マップレベルを宣言することもできます。

<surface>オブジェクトは、事前に存在する<image>オブジェクトセットから、それぞれの ID の順番リストを<init\_from>に指定して初期化することができます。また、<generator>要素を利用して、表面の各ピクセルに対してソースコードを評価することでプログラマ的に初期化することも可能です。

### 属性

<surface>要素には、以下の属性があります。

type	fx_surface_type_enum	必須。表面のタイプを指定します。1D、2D、3D、CUBE、DEPTH、RECT のいずれか1つを指定します。
------	----------------------	---

### 関連要素

<surface>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	newparam、setparam
子要素	init_from、generator、format、size、viewport_ratio、mip_levels、mipmap generate
その他	なし

### 備考

### 例

```
<surface type="CUBE">
  <init_from> sky01 sky02 sky03 sky04 sky05 ground </init_from>
  <format> R5G6B5 </format>
  <size> 64 64 </size>
  <mip_levels> 0 </mip_levels>
</surface>
```

## technique

### 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プラー、シェーダ、パラメータ、パスに関する記述を保持します。

### コンセプト

テクニクは、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クを含めることができ、それぞれのテクニ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。

- あるテクニクで高い LOD バージョンの効果を記述し、2 番目のテクニ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。
- 異なるゲームの状態、たとえば、昼間のテクニクと夜間のテクニクなどの効果を記述し、通常のテクニクと「魔法が有効」なテクニクを記述する場合。

### 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	xs:NCName	オプション。この要素のサブ識別子を含んだテキスト文字列値です。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。

### 関連要素

<technique>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	profile_CG、profile_COMMON、profile_GLSL、bind_material、source、light、optics、imager
子要素	asset、annotate、blinn、include、newparam、setparam、pass
その他	なし

### 備考

ツールを利用して効果のテクニクを自動的に生成することができ、テクニクを最善の<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

### 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です。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しなければなりません。

### 属性

<technique\_hint>要素には、以下の属性があります。

platform	xs:NCName	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。
ref	xs:NCName	プラットフォームの名前を参照します。

### 関連要素

<technique\_hint>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	<a href="#">instance effect</a>
子要素	子要素はありません
その他	なし

### 備考

### 例

```
<technique_hint platform="PS3" ref="HighLOD">  
<technique_hint platform="OpenGL|ES" ref="twopass">
```

---

## texcombiner

### 概要

<texture\_pipeline>コマンドを定義します。これは混合モードのテクスチャ操作です。

### コンセプト

この要素は、割り当てられているテクスチャユニットの混合状態を設定します。  
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。

### 属性

<texcombiner>要素に属性はありません。

### 関連要素

<texcombiner>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	texture_pipeline
子要素	constant、RGB、alpha
その他	なし

<constant>要素は、glTexEnv(TEXTURE\_ENV, TEXTURE\_ENV\_COLOR, value)のための float4 です。

<RGB>要素は、テクスチャ混合コマンドの RGB コンポーネントを設定します。

<alpha>要素は、テクスチャ混合コマンドのアルファコンポーネントを設定します。

### 備考

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混合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

### 概要

<texture\_pipeline>コマンドを定義します。これは単純な非混合モードのテクスチャ操作です。

### コンセプト

この要素は、割り当てられる先のテクスチャユニットの状態を設定します。  
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。

### 属性

<texenv>要素には、以下の属性があります。

operator	REPLACE   MODULATE   DECAL   BLEND   ADD	入力フラグメントに対して実行する操作
unit	xs:NCName	テクスチャユニット

### 関連要素

<texenv>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	<a href="#">texture_pipeline</a>
子要素	<a href="#">constant</a>
その他	なし

<constant>要素は、glTexEnv(TEXTURE\_ENV, TEXTURE\_ENV\_COLOR, value)のための float4  
です。

### 備考

割り当てられる先のテクスチャユニットに対して glTexEnv(TEXTURE\_ENV, TEXTURE\_ENV\_MODE,  
operator) の呼び出しを行うものと想定しています。

### 例

<texture\_pipeline>を参照してください。

## texture\_pipeline

### 概要

通常モードと混合モードで `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 マルチテクスチャ状態を定義する順番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、混合モードでテクスチャユニットの設定を定義します。
- `<texenv>` は、非混合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。

フラグメントシェーダを有効にするために、パスは `<texture_pipeline>` と `<texture_pipeline_enable>` の状態を利用します。

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 ( `<texture_unit>` ) 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 の場合、テクスチャは既存のユニットからのものでなければならず、そのため、 `source="texture"` をともなう 2 つの引数は、同じ `<texture_unit>` 要素を参照していない場合には不正なものとなります。

### 属性

`<texture_pipeline>` 要素には、以下の属性があります。

sid	xs:NCName
-----	-----------

### 関連要素

`<texture_pipeline>` 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	<a href="#">profile GLES</a>
子要素	<a href="#">texcombiner</a> 、 <a href="#">texenv</a>
その他	なし

### 備考

それぞれの要素はコマンドのタイプです。

## 例

```

<texture_pipeline sid="terrain-transition-shader">
  <texcombiner>
    <constant> 0.0f, 0.0f, 0.0f, 1.0f </constant>
    <RGB operator="INTERPOLATE">
      <argument idx="0" source="TEXTURE" operand="SRC_RGB" unit="gravel"/>
      <argument idx="1" source="TEXTURE" operand="SRC_RGB" unit="grass"/>
      <argument idx="2" source="TEXTURE" operand="SRC_ALPHA" unit="transition"/>
    </RGB>
    <alpha operator="INTERPOLATE">
      <argument idx="0" source="TEXTURE" operand="SRC_ALPHA" unit="gravel"/>
      <argument idx="1" source="TEXTURE" operand="SRC_ALPHA" unit="grass"/>
      <argument idx="2" source="TEXTURE" operand="SRC_ALPHA" unit="transition"/>
    </alpha>
  </texcombiner>
  <texcombiner>
    <RGB operator="MODULATE">
      <argument idx="0" source="PRIMARY" operand="SRC_RGB"/>
      <argument idx="1" source="PREVIOUS" operand="SRC_RGB"/>
    </RGB>
    <alpha operator="MODULATE">
      <argument idx="0" source="PRIMARY" operand="SRC_ALPHA"/>
      <argument idx="1" source="PREVIOUS" operand="SRC_ALPHA"/>
    </alpha>
  </texcombiner>
  <texenv unit="debug-decal-unit" operator="DECAL"/>
</texture_pipeline>

```

---

## texture\_unit

### 概要

テクスチャユニットの定義型です。これらのテクスチャユニットは、<texture\_pipeline>コマンド中での利用に応じて、ハードウェアテクスチャユニットにマッピングされます。

### コンセプト

<texture\_pipeline>がハードウェアの上限を一度に超えない限りは、ハードウェアで対応している以上のテクスチャユニットを定義することも可能で、フラグメントシェーダも有効となります（GLESのバージョンにも依存します）。

### 属性

<texture\_unit>要素には、以下の属性があります。

sid	xs:NCName
-----	-----------

### 関連要素

<texture\_unit>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	<a href="#">RGB</a> 、 <a href="#">alpha</a>
子要素	<a href="#">surface</a> 、 <a href="#">sampler state</a> 、 <a href="#">texcoord</a>
その他	なし

<surface>要素は、テクスチャに利用される表面の surface パラメータを参照します。

<sampler\_state>要素は、表面のサンプリングに利用される状態を持った<sampler\_state>パラメータを参照します。

<texcoord>要素には、テクスチャユニットが利用しなければならないテクスチャ座標配列のセマンティックス名を表す semantic 属性が含まれます。配列は<bind\_material>を利用してマッピングされます。

### 備考

### 例

<texture\_pipeline>を参照してください。

## usertype

### 概要

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、Cg 言語 1.4 の仕様の一部として導入されたもので、オブジェクトのクラスのため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ダ中でだけ宣言でき、`<usertype>` 宣言は、テクニッ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。

name	xs:NCName	現在のソースコード翻訳ユニット中にある構造体宣言のための識別子です。
------	-----------	------------------------------------

### 関連要素

`<usertype>` 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	<a href="#">newparam</a> (cg newparam)
子要素	CG VALUE TYPE、 <a href="#">array</a> 、 <a href="#">usertype</a> 、 <a href="#">connect param</a>
その他	なし

### 備考

`<usertype>` の各要素は、値が設定されている順番で各リーフノードを処理するか、もしくは一連の `<setparam>` 宣言を利用して名前で各リーフノードにアクセスして、`<newparam>` 中で作成時に初期化することができます。

### 例

```
<newparam sid="lightsource">
  <usertype name="spotlight">
    <float3> 10 12 10 </float3>
    <float3> 0.3 0.3 0.114 </float3>
  </usertype>
</newparam>
```



## VALUE\_TYPES

### 概要

異なる FX プロファイルは、強く型付けされた利用可能な別々のパラメータセットを持ちます。

### COLLADA の範囲の VALUE\_TYPES

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、surface、sampler1D、sampler2D、sampler3D、samplerCUBE、samplerRECT、samplerDEPTH、enum

### GLSL の範囲の VALUE\_TYPES

bool、bool2、bool3、bool4、int、int2、int3、int4、float、float2、float3、float4、float2x2、float3x3、float4x4、surface、sampler1D、sampler2D、sampler3D、samplerCUBE、samplerRECT、samplerDEPTH、enum

### CG の範囲の VALUE\_TYPES

bool、bool2、bool3、bool4、bool1x1、bool1x2、bool1x3、bool1x4、bool2x1、bool2x2、bool2x3、bool2x4、bool3x1、bool3x2、bool3x3、bool3x4、bool4x1、bool4x2、bool4x3、bool4x4、int、int2、int3、int4、int1x1、int1x2、int1x3、int1x4、int2x1、int2x2、int2x3、int2x4、int3x1、int3x2、int3x3、int3x4、int4x1、int4x2、int4x3、int4x4、float、float2、float3、float4、float1x1、float1x2、float1x3、float1x4、float2x1、float2x2、float2x3、float2x4、float3x1、float3x2、float3x3、float3x4、float4x1、float4x2、float4x3、float4x4、half、half2、half3、half4、half1x1、half1x2、half1x3、half1x4、half2x1、half2x2、half2x3、half2x4、half3x1、half3x2、half3x3、half3x4、half4x1、half4x2、half4x3、half4x4、fixed、fixed2、fixed3、fixed4、fixed1x1、fixed1x2、fixed1x3、fixed1x4、fixed2x1、fixed2x2、fixed2x3、fixed2x4、fixed3x1、fixed3x2、fixed3x3、fixed3x4、fixed4x1、fixed4x2、fixed4x3、fixed4x4、surface、sampler1D、sampler2D、sampler3D、samplerCUBE、samplerRECT、samplerDEPTH、enum

### GLES の範囲の VALUE\_TYPES

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、texture\_unit、surface、sampler\_state、texture\_pipeline、enum

このページは空白です。

---

## Appendix A キューブの例

---

このページは空白です。

## 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/2005/COLLADASchema" version="1.3.0">
  <library type="GEOMETRY">
    <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 profile="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>
        </source>
        <vertices id="box-Vtx">
          <input semantic="POSITION" source="#box-Pos"/>
        </vertices>
        <polygons count="6">
          <input semantic="VERTEX" source="#box-Vtx" idx="0"/>
          <p>0 2 3 1</p>
          <p>0 1 5 4</p>
          <p>6 7 3 2</p>
          <p>0 4 6 2</p>
          <p>3 7 5 1</p>
          <p>5 7 6 4</p>
        </polygons>
      </mesh>
    </geometry>
  </library>
  <scene id="DefaultScene">
    <node id="Box" name="Box">
      <instance url="#box"/>
    </node>
  </scene>
</COLLADA>
```

このページは空白です。

---

# 用語集

---

このページは空白です。



- **属性**：XML 要素は、任意の数の属性を持つことができます（なくてもかまいません）。属性は、開始タグの中のタグ名の直後に指定します。それぞれの属性は、名前と値のペアで構成します。属性の値の部分は、かならず引用符（" "）で囲みます。属性は、それが結び付けられている要素についての意味的な情報を提供します。以下に例を示しておきます。

```
<tagName attribute="value">
```

- **コメント**：XML ファイルには、コメントのテキストを含めることができます。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 -->
```

- **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。
- **名前**：一般に属性の名前は、それが属している要素との関係を表す意味を持っています。以下に例を示しておきます。

```
<Array size="5" type="xs:float">
  1.0 2.0 3.0 4.0 5.0
</Array>
```

これは、size と type という 2 つの属性を持った Array という名前の要素です。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。

- **タグ**：個々の 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。

- **検証**：XML それ自体では、いかなる文書構造やスキーマをも記述していません。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**：解析の際に、属性値は常にテキストデータとして扱われます。
- **XML**：XML（eXtensible Markup Language）は拡張可能なマークアップ言語で、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 自体が、要素、属性、コメント、テキストデータからなる構造化された言語です。
- **XML Schema**：XML Schema 言語は、構文、構造、意味において、同一の規則にしたがう XML 文書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。