COLLADA – Digital Asset Schema
リリース 1.4.1

仕様書

2006 年 6 月

編集責任者: Mark Barnes, Sony Computer Entertainment Inc.

Publication date: April 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

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て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 インタラクティブ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と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 エクスポータ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®の Cg や Pixar の 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：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 3 章：スキーマとリファレンスの概要	スキーマとその設計について、およびリファレンスの章でスキーマがどのように表されているかについての概要。COLLADA の共通プロファイルの説明も含まれています。
第 4 章：COLLADA のコア要素参照	COLLADA スキーマのコア要素についての詳しいリファレンス
第 5 章：COLLADA フィジックス リファレンス	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 6 章：COLLADA FX リファレンス	COLLADA FX の要素についての詳しいリファレンス
付録 A:COLLADA 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（XML 用語を含む）
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 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 bold	ファイル名を示します。
> Courier New bold	右角括弧（>）を先頭に付けて、コマンドを示します。
青色	ハイパーリンクを示します。

このページは空白です。

1章　設計上の考慮事項

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、さらには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、インタラクティブアプリケーションのコンテンツ作成との橋渡しに有益なものと想定しています。また、インタラクティブアプリケーションの多くにおいて、COLLADA は、最終的な配信メカニズムではなく、むしろ製作時のパイプラインとして利用されることを想定しています。たとえば、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にもかかわらず、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、ディベロッパやデジタルコンテンツ制作（DCC）、ハードウェア、ミドルウェアベンダの間の共同作業を促進すること。

以降のセクションでは、COLLADA の目標について解説し、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。

ほとんどの 3D アプリケーションにおいて、最も重要な部分はデジタルアセットです。
ディベロッパは、大量のコストをかけて制作してきた大量のデジタルアセットを、不明瞭な専用のフォーマットで蓄積しています。このような専用ツールからデータをエクスポートするには、複雑な専用のソ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、エクスポータをも更新し続ける必要があります。
また、ハードウェア・ベンダは、新たなハードウェアを利用するために、ますます複雑なアセットを必要とするようになってきています。必要なデータはツールの中に存在するかもしれませんが、そのデータをツールからエクスポートする方法がない場合もあります。このようなデータのエクスポート処理は、ディベロッパが高度な機能を利用する上でも、ハードウェア・ベンダが新製品を普及させる上でも障害となるような、複雑な処理だということです。
ミドルウェアやツールのベンダは、ミドルウェアやツールをあらゆるツールチェーンに組み込んで、ディベロッパが利用できるようにする必要がありますが、これはほとんど無理です。ミドルウェア・ベンダが成功するためには、拡張性のある独自のツールチェーンおよびフレームワークを提供して、それをディベロッパに採用させる必要があります。このため、ゲームディベロッパは、同一のプロジェクトで複数の異なる 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、何らかのバイナリ形式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て、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減らすばかりでなく、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。

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ージメントに対するソリューションを促進する。
- COLLADA のエクスポータやインポータなどのツールを 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タやインポータ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の設計と今後のCOLLADAの改良を図る。
- 拡張が簡単にできるエクスポータやインポータを作成することを、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、そのエクスポータやインポータ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、たとえばアンドゥ・スタックのようなツール固有の情報であるとか、あるいは複雑なシェーダのように、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、エクスポータおよびインポータに対して強制する。

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の地位にある Sony Computer Entertainment (SCE) こそがふさわしかったのです。SCE には、ミドルウェアやディベロッパプログラムにおいて、ツール・ベンダとゲームディベロッパの仲介を行ってきた歴史があったので、この共通フォーマットの問題を、次世代プラットフォーム用のコンテンツの品質や量を向上させたい、という要求とともに、議題に挙げることができました。
私たちは、この運動を常に SCE が推進しようとは思っておらず、時機が来れば、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。

2章　ツールの要件とオプション

概要

COLLADA に完全に準拠したツールはどれも、COLLADA スキーマによって表現されたあらゆるデータの仕様に対応する必要があります。しかしながら、単純にスキーマ仕様に準拠する以上の要件が求められる場合もあります。たとえば、値の解釈を統一する、重要なユーザ設定可能なオプションを提供する、ツールに独自の機能を組み込む方法を詳しく規定する、といったものです。本章の目的は、こういった問題点に優先順位を付けることです。
それぞれの「要件」のセクションでは、個々の対応ツールがかならず実装しなければならないオプションについて詳しく述べてあります。ただし、指定された情報をアプリケーションで利用できない場合は例外です。たとえば、レイヤをサポートしていないツールでは（そういったレイヤ情報をエクスポートするのが通常は必須だと仮定した場合）、レイヤ情報をエクスポートできなくてもかまいません。しかしながら、レイヤをサポートしたツールは、すべてレイヤ情報を適切にエクスポートできなければなりません。
また、「オプション」のセクションでは、かならずしも実装する必要のない（ただし、一部のユーザにとっては重要で高度な）オプション機能などのメカニズムについて述べてあります。
以降で解説する要件はどれも、ツールの品質を保証すると同時に、COLLADA の目的に沿っていることを保証するために、ツールに組み込まれるものです。こ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相互運用性やカスタマイズ性を備えるためのものです。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が著しく損なってしまう可能性があります。このセクションは、そういった問題が発生してしまうのを最小限に抑えるために記述されています。
このセクションで必要とされている各機能は、COLLADA 規格適合性試験に用意されている複数のテストケースでチェックしてあります。COLLADA 規格適合性試験とは、Maya®、XSI、3DS Max のエクスポータとインポータ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。

エクスポータ

範囲

COLLADA エクスポータの役割は、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（ロケータや NULL など）の変換が可能でなければなりません。

マテリアルとテクスチャ

データ	エクスポートの可/不可
RGB テクスチャ	任意の数の RGB テクスチャ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。
前処理済みの手続き型テクスチャ座標	前処理済みの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。
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区別されます）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、大まかなキーフレームと、複雑な制御や制約（コンストレイント）を利用してアニメーションを表現します。両者は、アニメーションが再生されて最終的な出力を提供する際に、アプリケーションによって組み合わせられます。アプリケーションがエクスポートするアニメーションデータを解析する際、そのアニメーションで利用されている制約やコントローラが全部は実装できず、オリジナルに対して忠実ではないアニメーションが出力されてしまう可能性があります。したがって、個々の変換データを指定された間隔で定期的にエクスポートするオプションが必要となります。サンプルがキーフレームより優先される場合には、ユーザがサンプリングレートを指定できなければなりません。
利用可能なアニメーション用のパラメータは、すべてエクスポートできなければなりません。そのようなパラメータとしては、以下のものがあります。

- マテリアルパラメータ

- テクスチャパラメータ
- UV 位置パラメータ
- 光源パラメータ
- カメラパラメータ
- シェーダパラメータ
- 全体環境パラメータ
- メッシュ構築パラメータ
- ノードパラメータ
- ユーザパラメータ

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。
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。
スケルトンアニメーション	スケルト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。
スムーズバインディングによるスケルトンのアニメーション	スムーズバインディングによるスケルトン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。
アニメーション関数曲線	アニメーション関数曲線のエクスポートが可能でなければなりません。

シーンデータ

データ	エクスポートの可/不可
空のノード	空のノードのエクスポートが可能でなければなりません。
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	スポットライト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。
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。

エクスポータ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。
前処理済み行列のオプション	前処理済み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 1 つの<**matrix**>要素だけを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、各類の変換要素をスタックによって指定した順序で合成する、という形式で「変換」を表現できます。このような表現は、アプリケーション内部で、段階的な複数の変換を利用する際に、変

換を正しく保存したり交換したりするのに便利です。ただし、この機能をアプリケーションで実装する際には、前処理済みの単一の**<matrix>要素**だけを保存する機能をユーザオプションとして提供すべきです。
この要件には、変換スタック内部の特定の要素を対象としているデータ（アニメーションなど）はどれも、代わりに、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

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

オプション

エクスポータは、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

エクスポータは、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タ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タの逆機能とは異なっていたり、より詳しい説明が必要な場合についてだけです。

要件

COLLADAに準拠したデータは、すべてインポート可能でなければなりません。一部のデータがツールによって認識されない場合であっても、後でエクスポートされるために保持されなければなりません。外部ツールにおいて、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。

3章　スキーマとリファレンスの概要

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のには、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、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。この場合だと、属性の名前はidで、値はhereとなります。
XML の語彙の詳細に関しては「用語集」を参照してください。

アドレス構文

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

- 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 の構文は短縮ポインタと呼ばれます。短縮ポインタで指定するのは、インスタンス文書中の要素の id 属性の値です。

url属性とsource属性では、URIフラグメント識別子の前にシャープ記号（#）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" />
<bind 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。この章に記載された一般用語集サブセクションには、共通プロファイル中のこのフィールドの値が記載されています。
  - 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>
```

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

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>
          <perspective>
          <yfov sid="YFOV">45.0</yfov>
          <aspect_ratio>1.33333</aspect_ratio>
          <znear>1.0</znear>
          <zfar>1000.0</zfar>
          </perspective>
      </technique_common>
      <technique profile="OTHER">
          <param sid="YFOV" type="float">45.0</param>
          <otherStuff type="MySpecialCamera">DATA</otherStuff>
      </technique>
   </optics>
</camera>
```

それぞれのテクニ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・チャネルが必要となってしまうでしょう。
異なるテクニックの対象パラメータが異なる値を必要とする場合、ディベロッパが個別のアニメーション・チャネルを使用できることは言うまでもありません。

共通プロファイル

COLLADAスキーマでは、設定プロファイルに準拠した情報表現のコンテクストを設定する**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[1]。
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、スキーマの拡張に伴って、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。

[1] 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。

- 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。以下に例を示します。

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

- パラメータの型がCOLLADAスキーマ、XML スキーマ、C/C++言語のいずれかの基本型に対応する場合には、小文字にします。それ以外の型名は、名前を構成する単語単位で頭文字を大文字にします。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。以下に例を示します。

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

- 入力とパラメータのセマンティックス名は大文字にします。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。以下に例を示します。

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

<**technique_common**>要素の範囲外にある要素は、共通プロファイルはもちろん、いかなるプロファイルにも含まれません。たとえば、<**polygons**>要素の範囲内に存在する<**input**>要素は共通プロファイルには含まれず、テクニック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	代表的なコンテクスト	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**>	レンダリングステート	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**>、<**geometry**>	2 番目のデカルト座標	なし
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	ジオメトリの従法線ベクトル
COLOR	カラー座標ベクトル
CONTINUITY	制御点での連続性の制限
IMAGE	ラスターまたは MIP レベルの入力
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
LINEAR_STEPS	制御点の次のスプラインセグメントで利用される区分線形近似のステップ数
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	ジオメトリの接線ベクトル
TEXBINORMAL	テクスチャの従法線ベクトル
TEXCOORD	テクスチャ座標ベクトル
TEXTANGENT	テクスチャ接線ベクトル
UV	一般的なパラメータベクトル
VERTEX	メッシュ頂点
WEIGHT	スキンのインフルエンスの重みの値

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

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。

スキーマ・レファレンスの構成

スキーマ・レファレンスの章では、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。
関連要素	要素の制約事項や相互関係について述べてあります。
子要素	有効な子要素の詳細
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。

子要素の規則

子要素テーブルには、指定された要素の子要素がすべてリストされます。各子要素について

- 子要素のメインエントリがリファレンス章の 1 つにある場合、子要素の用法、属性、子要素の子の詳細については、そこで参照してください。
- メインエントリがリファレンス章にない場合、またはローカルな子要素のプロパティがメインエントリによって異なる場合、子要素の情報は、子要素テーブルまたは追加の要素専用サブセクションで指定されます。

例

名前/例	解説	デフォルト値	出現回数
<**camera**>	子要素の説明。 *（<**camera**>に関するメインのリファレンスエントリがあります。詳細はそこで参照してください。）*	N/A（該当なし、または指定なし）	1 以上
<yfov sid="...">	sid はオプションです。 *（**yfov** に関するメインのリファレンスエントリはありません。詳細はここで規定されます。）*		
<technique_common>	下記サブセクションを参照してください。 *（詳細はここで規定されますが、別表になります。）*		

このページは空白です。

4章　COLLADA のコア要素参照

概要

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

accessor

概要

<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します。offsetとstrideの属性を指定することで、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	**uint**	配列へのアクセス回数。 必須。
offset	**uint**	配列から最初に読み出す値のインデックス。デフォルト値は 0 です。オプション。
source	**xs:anyURI**	URL 表現を利用してアクセスする配列の場所を表します。必須。
stride	**uint**	配列へ毎回アクセスする際に、1 つの単位としてみなす値の数。デフォルト値は 1 で、これは単一の値にアクセスすることを意味します。オプション。

関連要素

<accessor>要素は、以下の要素と関連性があります。

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	**下のサブセクションを参照してください**
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<param>	**<param>**要素のタイプ属性は、この要素が**<accessor>**要素の子であるときには次の一連の配列タイプに限定されます。**int**、**float**、**Name**、**bool**、および**IDREF**。	なし	0 以上

備考

<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>要素は出力の一部とはみなされないため、バインドされません。
<accessor>要素のsource属性では、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 つの整数値からなるストリームを表した<**accessor**>要素の例です。2 番目の<**param**>要素にはname属性がないため、配列中の 2 番目の値をすべて読み飛ばしています。

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

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

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

ambient (core)

概要

環境光源を記述します。

FX要素の**<ambient>**要素の詳細については、「**common_color_or_texture_type**」を参照してください。

コンセプト

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

属性

<**ambient**>要素には属性はありません。

関連要素

<**ambient**>要素は、以下の要素と関連性があります。

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	下のサブセクションを参照してください
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<color>	光源のカラーを表す 3 つの浮動小数点数を含みます。sid 属性を持つことも可能です。		1

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。

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

アニメーション情報の宣言をカテゴリ化します。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要素が含まれます。これらの要素は、同時に実行すべきアニメーションがグループ化されるように並べられています。

コンセプト

アニメーションは、時間の経過に伴うオブジェクトまたは値の変化を記述します。通常、アニメーションは、動いている錯覚を与えるために利用されます。一般的なアニメーションの技法としては、キーフレームアニメーションがあります。
キーフレームはデータを 2 次元（2D）サンプリングしたものです。最初の次元は入力と呼ばれ、通常は時間が使用されますが、他の任意の値でもかまいません。2 番目の次元は出力と呼ばれ、アニメートされている最中の値を表します。キーフレームと補間アルゴリズムを組み合わせることで、キーフレーム間の時間に対する中間の値が計算され、キーフレーム間の区間全体の出力値のセットが生成されます。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**	**<animation>** 要素のユニークな識別子を含んだテキスト文字列。この値はインスタンス文書中でユニークではければなりません。オプション。
name	**xs:NCName**	要素のテキスト文字列名。オプション。

関連要素

<**animation**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**library_animations**、**animation**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<animation>			0 以上（「備考」を参照）
<source>			0 以上

名前/例	解説	デフォルト値	出現回数
<sampler>			0 以上（「備考」を参照）
<channel>			0 以上（「備考」を参照）
<extra>			0 以上

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、子要素によって表されています。

子要素は以下のルールに従います。

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

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、より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

アニメーションクリップとしてまとめて利用するアニメーション曲線のセクションを定義します。

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 つのセットを複数の部分として切り分ける目的で利用します。たとえば、キャラクタ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のアニメーションを切り分けたり、同じキャラクタの別々の部分（上半身と下半身など）を切り分けるのにも利用できます。
現時点では、アニメーションクリップを COLLADA ドキュメント中でインスタンス化することはできません。エンジンや他のツールで利用するためのものです。

属性

<**animation_clip**>要素には、以下の属性があります。

id	**xs:ID**	<**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。オプション。
start	**xs:double**	クリップの開始時間を表します（単位は秒）。クリップの開始時間を表します（単位は秒）。この時間は、キーフレームデータで利用されているものと同じで、どのキーフレームセットをクリップに含めるのかを判断する目的で利用されます。クリップがいつ再生されるのかを開始時間で指定するわけではありません。指定した時間が、参照されているアニメーションの 2 つのキーフレーム間に位置する場合には、補間された値が利用されます。デフォルト値は 0.0 です。 オプション。
end	**xs:double**	クリップの終了時間を表します（単位は秒）。開始時間と同じように利用されます。終了時間を指定しなかった場合、最も長いアニメーションの終了時間が値として利用されます。 オプション。
name	**xs:NCName**	オプション

関連要素

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

出現回数	スキーマ中に定義された要素の数
親要素	**library_animation_clips**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<instance_animation>			1 以上
<extra>			0 以上

備考

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

その親要素に関連したアセット管理情報を定義します。

コンセプト

コンピュータは、膨大な量の情報を保存しています。アセットは、1 つのまとまりとして識別し管理できるように組織化された情報セットのことです。さまざまな属性を用いてアセットを表すことで、この情報を、ソフトウェアツールおよび人間の両方によって管理および理解できるようにします。アセット情報は階層的であることが多く、その場合大きなアセットの各パーツは、それぞれ別のアセットとして管理される、より小さい部分に区分けされます。

属性

<**asset**>要素には属性はありません。

関連要素

<**asset**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**camera**, **COLLADA**, **light**, **material**, **technique**, **source**, **geometry**, **image**, **animation**, **animation_clip**, **controller**, **extra**, **node**, **visual_scene**, **library_***, **effect**, **force_field**, **physics_material**, **physics_scene**, **physics_model**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<contributor>	その親要素の貢献者に関連したデータ		0 以上
<created>	親要素が作成された日付と時刻。XML スキーマの dataTime 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		1
<keywords>	親要素の検索基準として使用する単語のリスト		0 または 1
<modified>	親要素が最後に修正された日付と時刻。 XML スキーマの dataTime 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		1
<revision>	親要素のためのリビジョン情報		0 または 1
<subject>	親要素の主題に関して記述した情報		0 または 1
<title>	親要素のタイトル情報		0 または 1

名前/例	解説	デフォルト値	出現回数
<unit **meter=…** **name=…** **/>**	測定単位について記述した情報。その光学上の属性は、 **name**: 単位の名前 **meter**: メートル法換算時の値。詳細は、「COLLADA フィジックスリファレンス」の「フィジカル単位」を参照してください。	name: meter meter: 1.0	0 または 1
<up_axis>	ジオメトリデータの座標系に関する情報。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。備考を参照してください。	Y_UP	0 または 1

備考

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

上軸の値

<up_axis>要素の値には以下の意味があります。

値	右軸	上軸	内側軸
X_UP	負の *y*	正の *x*	正の *z*
Y_UP	正の *x*	正の *y*	正の *z*
Z_UP	正の *x*	正の *z*	負の *y*

例

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

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

bool_array

概要

Boolean 値の同種の配列の保存場所を宣言します。

コンセプト

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

属性

<bool_array>要素には、以下の属性があります。

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

関連要素

<bool_array>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**source**
子要素	なし
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

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

コンセプト

カメラは、シーンの視覚的なイメージを捕らえる装置です。シーン中のカメラには位置と方向があります。これが、カメラの光学系装置やレンズから見たビューポイントとなります。
カメラの光学系は、入射光をまとめて画像を作ります。入射光によって作成される画像は、カメラの撮影装置（フィルム）の面に焦点が合わせられ、そのイメージが記録されます。

属性

<**camera**>要素には、以下の属性があります。

id	**xs:ID**	<**camera**>要素のユニークな識別子を含むテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。

関連要素

<**camera**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**library_cameras**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<optics>	標準パラメータを利用して視野角（FOV）と視錐台を記述します		1
<imager>			0 または 1
<extra>			0 以上

備考

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

例

以下は、視野角 45 度のシーンの透視投影を記述した<**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

アニメーションの出力チャンネルを宣言します。

コンセプト

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

属性

<**channel**>要素には、以下の属性があります。

source	**xs:URIFragmentType**	URL 表現を使って表したサンプラの場所。必須。
target	**xs:token**	サンプラの出力にバインドされた要素の場所。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。必須。

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	**animation**
子要素	なし
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」のセクション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 source="#Box-Translate-X-Sampler" target="Box/Trans.X"/>
  <channel source="#Box-Translate-Y-Sampler" target="Box/Trans.Y"/>
  <channel source="#Box-Translate-Z-Sampler" target="Box/Trans.Z"/>
</animation>
```

COLLADA

概要

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

コンセプト

COLLADAスキーマはXMLをベースとしています。そのため、整形式XML文書にするには「ドキュ メントルート要素」、つまり文書実体が必要となります。

属性

<COLLADA>要素には、以下の属性があります。

version	**VersionType**	インスタンス文書が準拠している COLLADA スキーマのリビジョンです。有効な値は、1.4.0 および 1.4.1 です。必須。
xmlns	**xs:anyURI**	XML Schema の名前空間属性である **xmlns** をこの要素に適用して、インスタンス文書のスキーマを特定します。
base	**xs:anyURI**	XML Base 仕様は HTML BASE の場合と同様、XML ドキュメントのパーツのベース URI を定義するファシリティを記述します。1 つの属性 **xml:base** を定義し、相対的な URI 参照を処理する際にその属性を使用するための手順を詳細に記しています。http://www.w3.org/XML/1998/namespace を参照してください。

関連要素

<**COLLADA**>要素は、以下の要素と関連性があります。

出現回数	1 回
親要素	親要素はありません
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは、**<asset>**, **<library_*>**, **<scene>**, **<extra>**の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			1
<library_animations>			0 以上
<library_animation_clips>			0 以上
<library_cameras>			0 以上
<library_controllers>			0 以上
<library_effects>			0 以上
<library_force_fields>			0 以上
<library_geometries>			0 以上

名前/例	解説	デフォルト値	出現回数
<library_images>			0 以上
<library_lights>			0 以上
<library_materials>			0 以上
<library_nodes>			0 以上
<library_physics_materials>			0 以上
<library_physics_models>			0 以上
<library_physics_scenes>			0 以上
<library_visual_scenes>			0 以上
<scene>			0 または 1
<extra>			0 以上

備考

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

例

以下は、スキーマのバージョンが「1.4.1」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/2005/11/COLLADASchema" version="1.4.1">
  <asset>
    <created/>
    <modified/>
  </asset>
  <library_geometries/>
  <library_visual_scenes/>
  <scene />
</COLLADA>
```

color

概要

親光源要素のカラーを表します。

コンセプト

<light>のコンテキストで、**<color>**要素はその親光源要素のRGBカラーを表す 3 つの浮動小数点値を含みます。

<profile_COMMON>のコンテキストでは、4 つの浮動小数点値を含みます。

属性

<color>要素には、以下の属性があります。

sid	**xs:NCName**	**<profile_COMMON>**要素の場合のみ。この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。

関連要素

<color>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	**<light>**内では：**ambient (core)**、**directional**, **point**, **spot** **<profile_COMMON>**内では： **common_color_or_texture_type**タイプの要素（**ambient**、**emission**、**diffuse**、**reflective**、**specular**、**transparent**）
子要素	なし
その他	なし

備考

例

contributor

概要

アセット管理用の作成者情報を定義します。

コンセプト

現在の制作ラインでは、特にアートのサイズが着実に大きくなるのに伴って、1 つのアセットに複数の人間、複数のツールが関わる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**>要素には属性はありません。

関連要素

<**contributor**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**asset**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<author>			0 または 1
<authoring_tool>			0 または 1
<comments>			0 または 1
<copyright>			0 または 1
<source_data>			0 または 1

備考

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

汎用的な制御情報の宣言をカテゴリ化します。

コンセプト

コントローラとは、別のオブジェクトの操作を制御管理するためのデバイスや仕組みです。

属性

<**controller**>要素には、以下の属性があります。

id	**xs:ID**	<**controller**>要素のユニークな識別子を含むテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。

関連要素

<**controller**>要素は、以下の要素と関連性があります。

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

注意： 子要素**<morph>**または**<skin>**のいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<skin>			注意を参照
<morph>			注意を参照
<extra>			0 以上

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は、子要素で表されています。

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

並行光源を表します。

コンセプト

<directional>要素は、並行光源を表すために必要なパラメータを宣言します。並行光源とは、位置に関係なく、同じ方向からすべてを照らす光源のことです。
ローカル座標での光源のデフォルトの方向ベクトルは[0, 0, -1]で、負の Z 軸方向を指しています。実際の光源の方向は、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>要素には属性はありません。

関連要素

<directional>要素は、以下の要素と関連性があります。

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	下のサブセクションを参照してください
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<color>	光源のカラーを表す 3 つの浮動小数点数を含みます。 **sid** 属性を持つことも可能です。		1

備考

<directional>要素は、<light>要素の下で<technique_common>の子としてのみ指定できます。

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

親要素に関係した追加情報を宣言します。

コンセプト

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

属性

<**extra**>要素には、以下の属性があります。

id	**xs:ID**	<**extra**>要素のユニークな識別子を含むテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
type	**xs:NMTOKEN**	データ値の型。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。オプション。

関連要素

<**extra**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**animation, animation_clip, attachment, box, camera, capsule, COLLADA, controller, control_vertices, convex_mesh, cylinder, effect, force_field, geometry, image, imager, instance_*, joints, library_*, light, lines, linestrips, material, mesh, morph, node, optics, physics_material, physics_model, physics_scene, plane, polygons, polylist, profile_COMMON, profile_COMMON::technique, ref_attachment, rigid_body, rigid_constraint, scene, shape, skin, sphere, spline, tapered_capsule, tapered_cylinder, targets, texture, triangles, trifans, tristrips, vertex_weights, vertices, visual_scene**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<technique>			1 以上

備考

例

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

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

float_array

概要

浮動小数点値の値を含んだ同種の配列の保存場所を宣言します。

コンセプト

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

属性

<**float_array**>要素には、以下の属性があります。

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

関連要素

<**float_array**>要素は、以下の要素と関連性があります。

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

備考

<**float_array**>要素には、浮動小数点値の値のリストが含まれます。これらの値は、<**source**>要素に対するデータのリポジトリとして利用されます。

例

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

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

geometry

概要

シーン内のオブジェクトの視覚的形状や外観を記述したものです。<**geometry**>要素は、ジオメトリ情報の宣言をカテゴリ化します。ジオメトリ（幾何学）とは、点、線、角度、面、立体の大きさや特性、相互関係についての研究を行う数学の一分野です。<**geometry**> 要素には、メッシュ、凸メッシュ、スプラインの宣言が含まれます。

コンセプト

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

- B スプライン
- ベジェ
- メッシュ
- NURBS
- パッチ

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

属性

<**geometry**>要素には、以下の属性があります。

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

関連要素

<**geometry**>要素は、以下の要素と関連性があります。

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

注意： 子要素**<convex_mesh>**, **<mesh>**、または**<spline>**のいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<convex_mesh>			注意を参照
<mesh>			注意を参照
<spline>			注意を参照
<extra>			0 以上

備考

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

例

以下は、指定可能な属性を持った空の<**geometry**>要素の例です。

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

IDREF_array

概要

ID 参照値を含んだ同種の配列の保存場所を宣言します。

コンセプト

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

属性

<**IDREF_array**>要素には、以下の属性があります。

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

関連要素

<**IDREF_array**>要素は、以下の要素と関連性があります。

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

備考

<**IDREF_array**>要素には、XMLのIDREFの値のリストが含まれます。これらの値は、<**source**>要素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。

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

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

image

概要

オブジェクトのグラフィカルな表現の保存場所を宣言します。<**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、輝度やカラー値を表す画素（ピクセル）の並びから構成され、このピクセルが集まって 1 つの画像を構成しています。またベクトル画像では、曲線、直線、図形などを表す数学的な式を利用して、図を記述します。さらにハイブリッド画像は、ラスタ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画像を記述します。
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。テクスチャ参照を行う関数では、この配列構造をディスプレイスメント、法線、高さといったフィールド値を格納し利用することもできます。

属性

<**image**>要素には、以下の属性があります。

id	**xs:ID**	<**image**>要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
format	**xs:token**	画像形式を表す文字列値。オプション。
height	**uint**	ピクセルを単位として画像の高さを表す整数値。オプション。
width	**uint**	ピクセルを単位として画像の幅を表す整数値。オプション。
depth	**uint**	ピクセルを単位として画像の奥行きを表す整数値。2 次元の画像の場合、奥行きは 1 です。これがデフォルト値となっています。オプション。

関連要素

<**image**>要素は、以下の要素と関連性があります。

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	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

注意： 子要素**<data>**または**<init_from>**のいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1

名前/例	解説	デフォルト値	出現回数
<data>	埋め込み画像データを表す、16 進数で符号化されたバイナリオクテットの並びを含みます。		注意を参照
<init_from>	外部の画像ファイルを指定します。		注意を参照
<extra>	埋め込み画像データを含みます。		0以上

備考

<image>要素では、<init_from>によって外部の画像ファイルを指定したり、<data>で画像データを埋め込むことができます。

例

以下は、外部のPNGアセットを参照する<image>要素の例です。

```
<library_images>
  <image name="WoodFloor">
    <init_from>Textures/WoodFloor-01.png</init_from>
  </image>
</library_images>
```

imager

概要

フィルムや CCD といったカメラの画像センサーを表します。

コンセプト

カメラの光学系は、画像を（通常は平面の）センサーに投影します。<**imager**>要素は、このセンサーによる、光のカラーと強度を数値化する方法を定義します。
現実世界では、光の輝度が極めて広いダイナミックレンジを持つことがあります。たとえば、野外のシーンでは、太陽の明るさは、木陰よりも桁違いに明るくなります。光子に含まれる波長の組み合わせも無数にあります。これに対し、表示装置では、非常に限定されたダイナミックレンジを利用します。通常は、可視範囲内の 3 つの波長、つまり、赤（Red）、緑（Green）、青（Blue）の 3 原色しか考慮しません。これらは、一般に 3 つの 8 ビット値として表されます。
したがって、画像センサーは、以下の 2 つの処理を行うようになっています。

- スペクトルサンプリング
- ダイナミックレンジのリマップ

この 2 つの処理の組み合わせはトーンマッピングと呼ばれ、画像合成（レンダリング）の最後のステップとして実行されます。レイトレーサなどの高品質のレンダラは、スペクトルの強度を内部的に浮動小数点値として表し、実際のピクセルのカラーを `float3s`（マルチベクトル・レンダラの場合は `float` の配列）として保存します。その後、トーンマッピングを実行して、グラフィックスハードウェアやモニタで表示可能な 24 ビットの RGB 画像を作成します。多くのレンダラでは、ハイダイナミックレンジ（HDR）のオリジナル画像を保存して、後で「再露光」することもできます。

属性

<imager>要素には属性はありません。

関連要素

<**imager**>要素は、以下の要素と関連性があります。

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<technique>			1 以上
<extra>			0 以上

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カメラを記述した<**camera**>要素の例です。

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

      <param name="RedGamma" type="float">2.2</param>
      <param name="GreenGamma" type="float">2.1</param>
      <param name="BlueGamma" type="float">2.17</param>

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

input

概要

データソースの入力セマンティクスを宣言します。

コンセプト

<input>要素は、コンシューマが必要とする入力の結び付きを宣言します。

属性

<input>要素には、以下の属性があります。

offset	**uint**	インデックスのリストへのオフセット。2 つの**<input>**要素で同じオフセットが使用されている場合は、同じインデックスの示す値が両方の要素で使用されます。この仕組みによって、入力の使用される順番が定義されるだけでなく、インデックスリストを簡単な圧縮形式で記述することも可能になります。**<input>**がグループB親の子である場合には必須、ただしグループA親に対しては無効。
semantic	**xs:NMTOKEN**	入力の結び付きの意味をユーザ定義するものです。必須。第 3 章の「共通グロサリー」の共通**<input>** セマンティック属性値のリストを参照してください
source	**xs:URIFragmentType**	データソースの場所。必須。
set	**uint**	単一セットとしてグループ化すべである入力。これは、複数の入力で同じセマンティックスを使用する場合に役立ちます。**<input>**がグループB親の子である場合にはオプション、ただしグループA親に対しては無効。

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	グループA: **joints**, **sampler**, **targets**, **vertices**, **control_vertices** グループB: **lines**, **linestrips**, **polygons**, **polylist**, **triangles**, **trifans**, **tristrips**, **vertex_weights**
子要素	なし
その他	なし

備考

それぞれの入力は、親要素のスコープ内で **offset** 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。

例

以下は、1 つの**<polygons>**要素に対する 6 つの**<input>**要素の例です。これらの**<input>**要素は、頂点位置、法線、テクスチャ座標とテクスチャ空間接線のセット 2 組を表しています。

```
<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

COLLADA アニメーションリソースのインスタンス化を宣言します。

コンセプト

オブジェクトの実際のデータ表現が保存されるのはおそらく一度だけですが、そのオブジェクトを複数の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。このように、シーン中で表示される個々のオブジェクトは、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、ユニークであることも、他のインスタンスと同じデータを共有することもあります。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仕組みが現在のシーンやリソースにとってローカルな場合、この仕組み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。

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
url	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URL。必須。#文字で始まる、相対 URI のフラグメント識別子を利用してローカルなインスタンスを参照します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<**instance_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	**animation_clip**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<extra>			0以上

備考

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

例

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

```
<library_animations>
  <animation id="anim"/>
</library_animations>

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

instance_camera

概要

COLLADA カメラリソースのインスタンス化を宣言します。

コンセプト

オブジェクトの実際のデータ表現が保存されるのはおそらく 1 度だけですが、そのオブジェクトは複数の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。このように、シーン中で表示される個々のオブジェクトは、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、ユニークであることも、他のインスタンスと同じデータを共有することもあります。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仕組みが、現在のシーンやリソースにとってローカルな場合、この仕組み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。

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
url	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URL。必須。 #文字で始まる、相対 URI のフラグメント識別子を利用してローカルなインスタンスを参照します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

〈**instance_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	**下のサブセクションを参照してください**
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<extra>			0以上

備考

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

例

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

```
<library_cameras>
  <camera id="cam"/>
</library_cameras>

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

instance_controller

概要

COLLADA コントローラリソースのインスタンス化を宣言します。

コンセプト

オブジェクトの実際のデータ表現が保存されるのはおそらく 1 度だけですが、そのオブジェクトは複数の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。このように、シーン中で表示される個々のオブジェクトは、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、ユニークであることも、他のインスタンスと同じデータを共有することもあります。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仕組みが、現在のシーンやリソースにとってローカルな場合、この仕組み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。

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
url	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URI。必須。#文字で始まる、相対 URI のフラグメント識別子を利用してローカルなインスタンスを参照します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<**instance_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	**node**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<skeleton>	必要な関節ノードの検索をスキンコントローラが開始すべき場所、を示します。この要素は、モーフィングコントローラに対しては意味を持ちません。		0 以上
<bind_material>			0 または 1
<extra>			0 以上

備考

例

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

```
<library_controllers>
  <controller id="skin"/>
</library_controllers>

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

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

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

```
    <node sid="Spine1">
      <node sid="Spine2">
        <node sid="Head"/>
      </node>
    </node>
  </node>
</library_nodes>
<node id="skel01">
   <instance_node url="#Skeleton1"/>
</node>
<node id="skel02">
  <instance_node url="#Skeleton1"/>
</node>
<node>
  <instance_controller url="#skin"/>
    <skeleton>#skel01</skeleton>
  </instance_controller>
</node>
<node>
  <instance_controller url="#skin"/>
    <skeleton>#skel02</skeleton>
  </instance_controller>
</node>
```

instance_geometry

概要

COLLADA ジオメトリリソースのインスタンス化を宣言します。

コンセプト

オブジェクトの実際のデータ表現が保存されるのはおそらく 1 度だけですが、そのオブジェクトは複数の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。このように、シーン中で表示される個々のオブジェクトは、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、ユニークであることも、他のインスタンスと同じデータを共有することもあります。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仕組みが、現在のシーンやリソースにとってローカルな場合、この仕組み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。

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
url	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URL。必須。#文字で始まる、相対 URI のフラグメント識別子を利用してローカルなインスタンスを参照します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<**instance_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	**node**, **shape**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<bind_material>	マテリアルシンボルをマテリアルインスタンスにバインドします。これにより、1 つのジオメトリを、各回異なる外観を伴い、シーンに複数回インスタンス化することができます。		0 または 1
<extra>			0 以上

備考

例

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

```
<library_geometries>
  <geometry id="cube"/>
</library_geometries>

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

instance_light

概要

COLLADA 光源リソースのインスタンス化を宣言します。

コンセプト

オブジェクトの実際のデータ表現が保存されるのはおそらく 1 度だけですが、そのオブジェクトは複数の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。このように、シーン中で表示される個々のオブジェクトは、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、ユニークであることも、他のインスタンスと同じデータを共有することもあります。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仕組みが、現在のシーンやリソースにとってローカルな場合、この仕組み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。

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
url	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URL。必須。#文字で始まる、相対 URI のフラグメント識別子を利用してローカルなインスタンスを参照します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<**instance_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	**node**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<extra>			0以上

備考

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

例

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

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

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

instance_node

概要

COLLADA ノードリソースのインスタンス化を宣言します。

コンセプト

オブジェクトの実際のデータ表現が保存されるのはおそらく 1 度だけですが、そのオブジェクトは複数の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。このように、シーン中で表示される個々のオブジェクトは、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、ユニークであることも、他のインスタンスと同じデータを共有することもあります。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仕組みが、現在のシーンやリソースにとってローカルな場合、この仕組み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。

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
url	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URL。必須。#文字で始まる、相対 URI の断片的な識別子を利用してローカルなインスタンスへの参照を表します。 フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<**instance_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	**node**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<extra>			0以上

備考

例

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

```
<library_nodes>
  <node id="myNode"/>
</library_nodes>

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

instance_visual_scene

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します。

コンセプト

オブジェクトの実際のデータ表現が保存されるのはおそらく 1 度だけですが、そのオブジェクトは複数の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。このように、シーン中で表示される個々のオブジェクトは、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、ユニークであることも、他のインスタンスと同じデータを共有することもあります。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仕組みが、現在のシーンやリソースにとってローカルな場合、この仕組み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。

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
url	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URL。必須。#文字で始まる、相対 URI の断片的な識別子を利用してローカルなインスタンスへの参照を表します。 フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<instance_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	**scene**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<extra>			0以上

備考

例

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

```
<library_visual_scenes>
  <visual_scene id="vis_scene"/>
</library_visual_scenes>

<scene>
  <instance_visual_scene url="#vis_scene"/>
</scene>
```

int_array

概要

整数値を保持する同種の配列の保存場所を宣言します。

コンセプト

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

属性

<**int_array**>要素には、以下の属性があります。

count	**uint**	配列中の値の数。必須。
id	**xs:ID**	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
minInclusive	**xs:integer**	配列に保持可能な最も小さい整数値。デフォルト値は-2147483648 です。オプション。
maxInclusive	**xs:integer**	配列に保持可能な最も大きい整数値。デフォルト値は2147483647 です。オプション。

関連要素

<**int_array**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**source**
子要素	なし
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

ジョイントノードと属性データとの関連付けを宣言します。

コンセプト

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

属性

<**joints**>要素には属性はありません。

関連要素

<**joints**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**skin**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<input>	**<input>**要素が**<joints>**要素の子である場合 、**<input>**要素に**offset**属性を指定してはなりません。2 つの**<input>**要素のうちの 1 つには、値が**JOINT**であるsemantic属性を指定していなければなりません。 値が**JOINT**であるsemantic属性を持つinputで参照されている**<source>**には、骨格ノードを識別するための**sid**を含む**<Name_array>**が含まれているべきです。この**sid**はIDREFの代わりに使用され、スキンコントローラを複数回インスタンス化することを可能にします。インスタンス化されたスキンコントローラでは、各インスタンスを独立してアニメートできます。		2 以上
<extra>			0 以上

備考

例

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

```
<skin source="#geometry_mesh">
  <joints>
```

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

library_animation_clips

概要

<animation_clip>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 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は、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**	<**library_animation_clips**>要素のユニークな識別子を含んだテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。

関連要素

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

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 以上
<animation_clip>			1 以上
<extra>			0 以上

備考

例

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

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

library_animations

概要

animation要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 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は、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**	<library_animation>要素のユニークな識別子を含んだテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。

関連要素

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

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<animation>			1 以上
<extra>			0 以上

備考

例

以下は、<**library_animations**>要素の例です。

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

library_cameras

概要

<camera>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 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は、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。

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

関連要素

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

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<camera>			1 以上
<extra>			0 以上

備考

例

以下は、<**library_cameras**>要素の例です。

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

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

library_controllers

概要

<controller>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 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は、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**	<**library_controllers**>要素のユニークな識別子を含んだテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。

関連要素

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

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<controller>			1 以上
<extra>			0 以上

備考

例

以下は、<**library_controllers**>要素の例です。

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

```
</library_controllers>
```

library_geometries

概要

<geometry>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 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は、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**	<**library_geometries**>要素のユニークな識別子を含んだテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。

関連要素

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

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<geometry>			1 以上
<extra>			0 以上

備考

例

以下は、<**library_geometries**>要素の例です。

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

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

library_images

概要

<image>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 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は、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**	<**library_images**>要素のユニークな識別子を含んだテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。

関連要素

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

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<image>			1 以上
<extra>			0 以上

例

以下は、<**library_images**>要素の例です。

```
<library_images>
     <image name="Rose">
      <init_from>../flowers/rose01.jpg</init_from>
   </image>
</library_images>
```

library_lights

概要

<image>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 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は、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。

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

関連要素

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

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<light>			1 以上
<extra>			0 以上

備考

例

以下は、<**library_lights**>要素の例です。

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

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

library_nodes

概要

<node>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 1 つの方法は、何らかの基準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は、ライブラリとして別々のリソース中に保存しておくことができます。

属性

<library_nodes>要素には、以下の属性があります。

id	**xs:ID**	要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。
name	**xs:NCName**	要素のテキスト文字列名。オプション。

関連要素

<library_nodes>要素は、以下の要素と関連性があります。

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	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<node>			1 以上
<extra>			0 以上

備考

例

以下は、**<library_nodes>**要素の例です。

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

  <node id="node2">
```

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

library_visual_scenes

概要

<visual_scene>要素のモジュールを宣言します。

コンセプト

データセットが大きく複雑化するのに伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 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は、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**	<**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。この値は、インスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。

関連要素

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

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<visual_scene>			1 以上
<extra>			0 以上

備考

例

以下は、<**library_visual_scenes**>要素の例です。

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

  <visual_scene id="vis_sce2">
```

```
    ...
  </visual_scene>

</library_visual_scenes>
```

light

概要

シーンを照らす光源を宣言します。

コンセプト

光源は、視覚的なシーンを照らす照明の源を表します。
光源は、シーン内に配置することも、無限遠に設定することもできます。
光源にはさまざまな属性があり、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。
- 点光源は、空間中の既存の位置から全方向に光を放射します。点光源の輝度は、光源までの距離が大きくなるのにつれて減衰します。
- 平行光源は、空間に対して無限遠である位置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。
- スポットライトは、空間中の既知の位置から特定の方向に光を放射します。スポットライトからの光は、円錐形に放射されます。スポットライトの輝度は、光源の放射角に対する角度が増えるにつれ、および光源からの距離が大きくなるにつれて減衰します。

属性

<**light**>要素には、以下の属性があります。

id	**xs:ID**	<**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。オプション。
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

<**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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<technique_common>	**<light>**の子として、**<technique_common>**要素には、**<ambient> (core)**、**<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。		1
<technique>			0 以上

名前/例	解説	デフォルト値	出現回数
<extra>			0 以上

備考

例

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

```
<library_lights>
  <light id="sun" name="the-sun">
    <technique_common>
      <directional>
        <color>1.0 1.0 1.0</color>
      </directional>
    </technique_common>
  </light>
</library_lights>
<library_visual_scenes>
  <visual_scene>
    <node>
      <instance_light url="#sun"/>
      <rotate>1 0 0 -10</rotate>
    </node>
  </visual_scene>
</library_visual_scenes>
```

lines

概要

<**mesh**>要素においてジオメトリプリミティブと頂点属性の結び付きを宣言します。

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付けされます。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**	この要素のテキスト文字列名。オプション。
count	**uint**	線のプリミティブの数。必須。
material	**xs:NCName**	マテリアルのシンボルを宣言します。シンボルは、インスタンス化の時点で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

<**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	**mesh**、**convex_mesh**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<input>			0 以上

名前/例	解説	デフォルト値	出現回数
<p>	（p は primitive の頭文字です）p 要素は、任意の数の直線の頂点属性を表します。 〈p〉要素中の各値は入力に参照されますが、参照時には各値の記述された順番が使用されます。〈p〉要素の最初の値は offset 属性の値として 0 が指定された入力によって参照され、2 番目の値は offset 属性として 1 が指定された入力によって参照されます。線の頂点は、その頂点に対する入力である値によって構成されます。すべての入力が参照された後は、次の値はオフセット 0 の値として入力によって再度参照され、新しい頂点を開始することになります。		0 または 1
<extra>			0 以上

備考

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

<**mesh**>要素においてジオメトリプリミティブと頂点属性の結び付きを宣言します。

コンセプト

<**linestrips**>要素は、頂点属性を結び付けてラインストリップ（折れ線）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付けされます。メッシュで記述された各ラインストリップは、任意の数の頂点を持ちます。ラインストリップ中の個々の線分は、現在の頂点と 1 つ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**	この要素のテキスト文字列名。オプション。
count	**uint**	ラインストリップのプリミティブの数。必須。
material	**xs:NCName**	マテリアルのシンボルを宣言します。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。オプション。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	**mesh**、**convex_mesh**
子要素	**下のサブセクションを参照してください**
その他	**なし**

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<input>			0 以上

名前/例	解説	デフォルト値	出現回数
<p>	（p は primitive の頭文字です）それぞれの p 要素は、任意の数のラインストリップの線分の頂点属性を記述します。 <p>要素中の各値は入力に参照されますが、参照時には各値の記述された順番が使用されます。<p>要素の最初の値は offset 属性の値として 0 が指定された入力によって参照され、2 番目の値は offset 属性として 1 が指定された入力によって参照されます。線の頂点は、その頂点に対する入力である値によって構成されます。すべての入力が参照された後は、次の値はオフセット 0 の値として入力によって再度参照され、新しい頂点を開始することになります。		0 以上
<extra>			0 以上

備考

<**linestrips**>要素には、**<p>**要素の並びが含まれます。

例

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

```
<mesh>
  <source id="position"/>
  <source id="normals"/>
  <source id="texcoord"/>
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <linestrips count="1">
    <input semantic="VERTEX" source="#verts" offset="0"/>
    <input semantic="NORMAL" source="#normals" offset="0"/>
    <input semantic="TEXCOORD" source="#texcoord" offset="0"/>
    <p>0 1 2</p>
  </linestrips>
</mesh>
```

lookat

概要

カメラの照準を定めるのに適した位置や方向の変換情報が含まれます。

コンセプト

<**lookat**>要素には、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

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

属性

<**lookat**>要素には、以下の属性があります。

sid	**xs:NCName**	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、親要素のスコープ内でユニークでなければなりません。オプション。

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	なし
その他	なし

備考

<**lookat**>要素には、9 個の浮動小数点値で構成されるリストが含まれます。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

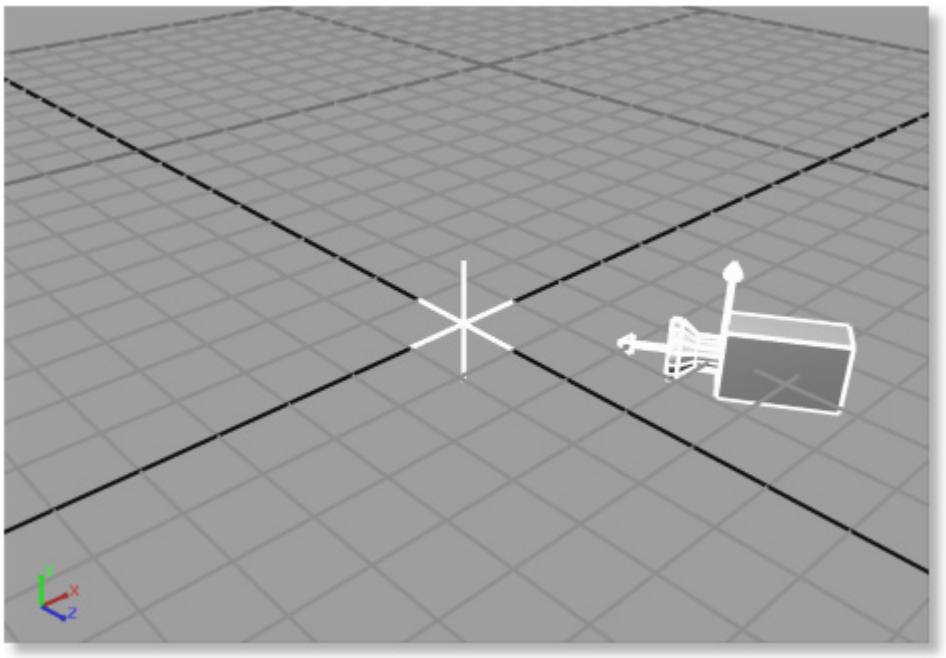
OpenGL と等価のビューイング行列を計算する際、視点は原点、注視点は *z* 軸のマイナス方向に割り当てられます。また、上方向の軸は、視平面 *y* 軸のプラス方向に割り当てられます。
値はどれもオブジェクトのローカル座標で指定します。

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 id="Camera">
  <instance_camera url="#camera1"/>
  <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 4-1　<**lookat**>要素：3Dのクロスヘアで注視点の位置を表しています。



matrix

概要

座標系中の各点や、座標系そのものに対する数学的な変更を表します。

コンセプト

<**matrix**>要素には、浮動小数点値の 4×4 行列が含まれています。
コンピュータグラフィックスでは、データの変換に線形代数を利用します。3 次元座標系は、一般的に 4×4 の行列式で表されます。これらの行列式を、シーングラフを介して階層構造化することによって、基準座標系のつながりを表すことができます。
COLLADA での行列は、数学的には「列行列」です。これらの行列は、人間が読みやすいように行優先で記述します。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**	この要素のサブ識別子を含んだテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。

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	**node**
子要素	なし
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 4×4 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

頂点とプリミティブの情報を使って基本的なジオメトリメッシュを記述します。

コンセプト

メッシュは、主に頂点とプリミティブの情報で構成される、幾何学的形状を記述するための一般的な形式です。
頂点情報とは、メッシュ面上の特定の点に結び付けられた属性セットです。それぞれの頂点には、以下のような属性データが含まれます。

- 頂点の位置
- 頂点のカラー
- 頂点の法線
- 頂点のテクスチャ座標

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

属性

<**mesh**>要素には、属性はありません。

関連要素

<**mesh**>要素は、以下の要素と関連性があります。

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。**<source>**, **<vertices>**、**プリミティブ要素**、**<extra>**（ここで、プリミティブ要素は、**<lines>**, **<linestrips>**, **<polygons>**, **<polylist>**, **<triangles>**, **<trifans>**, **<tristrips>**の任意の組み合わせを意味します）。

名前/例	解説	デフォルト値	出現回数
<source>	メッシュの頂点データの集まりを提供します。		1 以上
<vertices>	メッシュの頂点属性を記述して、トポロジー的な位置を確立します。		1
<lines>	直線プリミティブを含みます。		0 以上
<linestrips>	ラインストリッププリミティブを含みます。		0 以上
<polygons>	穴を持つことができるポリゴンプリミティブを含みます。		0 以上

名前/例	解説	デフォルト値	出現回数
<polylist>	穴を持てないポリゴンプリミティブを含みます。		0 以上
<triangles>	三角形プリミティブを含みます。		0 以上
<trifans>	三角形ファンプリミティブを含みます。		0 以上
<tristrips>	三角形ストリッププリミティブを含みます。		0 以上
<extra>			0 以上

備考

〈**mesh**〉要素は、以下のプリミティブ要素を 0 個以上持つことができます。**<lines>**, **<linestrips>**, **<polygons>**, **<polylist>**, **<triangles>**, **<trifans>**、そして**<tristrips>**.

<**mesh**>要素中の<**vertices**>要素は、メッシュの頂点を記述するために利用されます。ポリゴンや三角形などは、位置を直接記述するのではなく、メッシュの頂点を示すインデックスを指定します。メッシュの頂点には、semantic属性の値がPOSITIONである<**input**>要素が最低でも 1 つは含まれていなければなりません。
テクスチャ座標に対しては、COLLADA の右手座標系が適用されます。したがって、[0,0]の ST 座標は、テキスチャ画像が 2 次元テクスチャ・ビュアー/エディタにロードされた際に、テキスチャ画像の左下のテクセルにマッピングされます。

例

以下は、指定可能な属性を持った空の<**mesh**>要素の例です。

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

morph

概要

静的メッシュの複数セットをブレンドするのに必要なデータを記述します。ブレンド可能な個々のメッシュ（モーフィングターゲット）を指定する必要があります。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、特定のブレンド方法においては、ブレンド処理の基準、あるいは参照用として利用される「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）の線形結合となります。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。これらのモーフィングターゲットは（それぞれ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らないという制約があります。モーフィングターゲットを組み合わせると、それらの<**vertices**>要素にあるデータが補間されるだけです。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**>要素に含まれ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**>要素では、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を指定します。モーフィングターゲットを組み合わせる方法は複数あり、そのうちの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

```
(Target1, Target2, ...) * (w1, w2, ...) = (1-w1-w2-...) * BaseMesh + w1*Target1
+ w2*Target2 + ...
```

- RELATIVE

```
 (Target1, Target2, ...) + (w1, w2, ...) = BaseMesh + w1 * Target1 + w2 *
Target2 +  ...
```

属性

<**morph**>要素には、以下の属性があります。

source	**xs:anyURI**	ベースメッシュ。必須。
method	**MorphMethodType**	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。オプション。

関連要素

<**morph**>要素は、以下の要素と関連性があります。

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<source>			2 以上
<targets>			1
<extra>			0 以上

備考

例

以下は、空の<**morph**>要素の例です。

```
<morph source="#the-base-mesh" method="RELATIVE">
  <source id="morph-targets"/>
  <source id="morph-weights"/>
  <targets>
    <input semantic="MORPH_TARGET" source="#morph-targets"/>
    <input semantic="MORPH_WEIGHT" source="#morph-weights"/>
  <targets/>
  <extra/>
</morph>
```

Name_array

概要

シンボル名値を保持する同種の配列の保存場所を宣言します。

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伴いません。単純に、XMLの名前値の並びを記述します。

属性

<**Name_array**>要素には、以下の属性があります。

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

関連要素

<**Name_array**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**source**
子要素	なし
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

シーン中の各要素の階層構造的な関係を表します。

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。ノードはシーングラフ中における特定の位置の情報で表し、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。COLLADA では、内部ノードを表す場合に「ノード」という用語を利用しています。また、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。

id	**xs:ID**	この要素のユニークな識別子を含むテキスト文字列。この値はインスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素の名前を含んだテキスト文字列。オプション。
sid	**xs:NCName**	この要素のサブ識別子を含んだテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
type	**NodeType**	**<node>** 要素のタイプ。有効な値は、JOINTまたはNODEです。デフォルト値はNODEです。オプション。
layer	**ListOfNames**	そのノードが属しているレイヤの名前。たとえば、「foreground glowing」という値は、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。オプション。

関連要素

<node>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**library_nodes**、**node**、**visual_scene**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。**<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>**。ここで言う変換要素とは、**<lookat>**, **<matrix>**, **<rotate>**, **<scale>**, **<skew>**, **<translate>**のことです。

名前/例	解説	デフォルト	出現回数
<asset>	アセット管理情報を表現する		0 または 1
<lookat>			0 以上
<matrix>			0 以上
<rotate>			0 以上
<scale>			0 以上
<skew>			0 以上
<translate>			0 以上
<instance_camera>	カメラオブジェクトをインスタンス化する		0 以上
<instance_controller>	コントローラオブジェクトをインスタンス化する		0 以上
<instance_geometry>	ジオメトリオブジェクトをインスタンス化する		0 以上
<instance_light>	光源オブジェクトをインスタンス化する		0 以上
<instance_node>	他のノードの階層構造をインスタンス化する		0 以上
<node>	階層構造を再帰的に定義する		0 以上
<extra>	補足情報を定義する		0 以上

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。

<**node**>要素は、それぞれの子変換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**>要素のスコープ内で累積された変換が等しく適用されます。変換要素は<**node**>要素の座標系を変換することになります。数学的に言うと、変換要素はそれぞれ行列に変換され、記述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。

例

以下は、2 つの<**node**>要素を持った<**visual_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。

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

optics

概要

カメラに搭載されている、画像を画像センサの上に投影する装置を表します。

コンセプト

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

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

特定のカメラ光学系では、これらを複雑に組合せたものが内蔵されることがあります。たとえば、シュミット式望遠鏡には凹面レンズと凹面主鏡の両方と、さらに接眼部のレンズも含まれています。
現実の可変焦点式の「ズームレンズ」には、実際には 10 枚を超えるレンズと可変絞り（アイリス絞り）が含まれることがあります。
コンピュータグラフィックスで一般に利用されている「透視投影」カメラモデルは、「ズームレンズ」の単純な近似で、無限に小さな絞りと（レンズ関連の値である焦点距離の代わりに）直接指定された視野を持ちます。

属性

<optics>要素には属性はありません。

関連要素

<**optics**>要素は、以下の要素と関連性があります。

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<technique_common>	すべての COLLADA 実装がサポートしなければならない共通プロファイルの光学の（optics）情報を、指定します。下記サブセクションを参照してください。		1
<**technique> (core)**	共通プロファイルでないテクニック。		0 以上
<extra>			0 以上

optics / technique_common の子要素

注意：子要素**<orthographic>**または**<perspective>**のいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<orthographic>			注意を参照
<perspective>			注意を参照

備考

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

例

以下は、視野角 45 度での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

正投影カメラの視野を記述します。

コンセプト

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

属性

<orthographic>要素には属性はありません。

関連要素

<orthographic>要素は、以下の要素と関連性があります。

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	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト	出現回数
<xmag sid="...">	水平視野を度数で表す浮動小数点数。sid はオプションです。		備考を参照
<ymag sid="...">	垂直視野を度数で表す浮動小数点値。sid はオプションです。		備考を参照
<aspect_ratio sid="...">	視野のアスペクト比を表す浮動小数点値。sid が含まれる場合もあります。		備考を参照
<znear sid="...">	前方クリップ面への距離を表す浮動小数点値。sid が含まれる場合もあります。		1
<zfar sid="...">	後方クリップ面への距離を表す浮動小数点値。sid が含まれる場合もあります。		1

備考

<orthographic>要素は<camera><optics>の下の<technique_common>の中にだけ記述できます。<orthographic>要素には、以下のいずれかの要素が含まれていなければなりません。

- 1 つの<xmag>要素
- 1 つの<ymag>要素
- <xmag>と<ymag>の両方の要素

- 〈xmag〉と〈ymag〉のどちらかと〈aspect_ratio〉要素

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。

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

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

例

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

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

param

概要

親要素のパラメータに関する情報を宣言します。

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、ステート情報の一部としてパラメータを必要とします。
基本的なパラメータ宣言では、パラメータの名前、データ型、値データを記述します。パラメータ名で特定の関数やプログラムが識別され、パラメータの型で値の符号化方法が表されます。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**	この要素のテキスト文字列名。オプション。
sid	**xs:NCName**	この要素のサブ識別子を含むテキスト文字列。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
type	**xs:NMTOKEN**	データ値の型。このテキスト文字列は、アプリケーションの理解可能な形式でなければなりません。必須。
semantic	**xs:NMTOKEN**	パラメータの意味をユーザ定義するためのものです。オプション。

関連要素

<**param**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**accessor**、**bind_material**
子要素	なし
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

透視カメラの視野を記述します。

コンセプト

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

属性

<**perspective**>要素には属性はありません。

関連要素

<**perspective**>要素は、以下の要素と関連性があります。

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<xfov sid="...">	水平視野を度数で表す浮動小数点数。sid はオプションです。		「備考」を参照
<yfov sid="...">	垂直視野を度数で表す浮動小数点値。sid はオプションです。		「備考」を参照
<aspect_ratio sid="...">	視野のアスペクト比を表す浮動小数点値。sid はオプションです。		「備考」を参照
<znear sid="...">	前方クリップ面への距離を表す浮動小数点値。sid はオプションです。		1
<zfar sid="...">	後方クリップ面への距離を表す浮動小数点値。sid はオプションです。		1

備考

<**perspective**>要素は<**camera**><**optics**>の下の<**technique_common**>の中にだけ記述できます。<**perspective**>要素には、以下のいずれかの要素が含まれていなければなりません。

- 1 つの<xfov>要素
- 1 つの<yfov>要素
- <xfov>と<yfov>の両方の要素
- <xfov>と<yfov>のどちらかと<aspect_ratio>要素

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。アスペクト比は、ピクセルの幅に対するピクセルの高さの比率として定義されます。したがって、アスペクト比は、下に示すように、視野パラメータから導き出されることも、視野パラメータを導き出すために使用されることもあります。**aspect_ratio** = **yfov** / **xfov**。
中央のスクリーンピクセルは、スクリーン座標で（0, 0）であると仮定されます。

例

以下は、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

点光源を記述します。

コンセプト

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

属性

<**point**>要素には属性はありません。

関連要素

<**point**>要素は、以下の要素と関連性があります。

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<color sid="...">	光源のカラーを表す 3 つの浮動小数点値。sid はオプションです。		1
<constant_attenuation sid="...">	sid はオプションです。	1.0	0 または 1
<linear_attenuation sid="...">	sid はオプションです。	0.0	0 または 1
<quadratic_attenuation sid="...">	sid はオプションです。	0.0	0 または 1

備考

<**point**>要素は、<**light**>要素の下の<**technique_common**>の子としてだけ指定できます。
特定の距離における光の全体の減衰量は、<constant_attenuation>、<linear_attenuation>、<quadratic_attenuation>を利用して計算されます。その際、以下の計算式が利用されます。

```
  A = constant_attenuation + ( Dist * linear_attenuation ) + (( Dist^2 ) *
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

<**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します。

コンセプト

<**polygons**>要素は、頂点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付けされます。

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

さまざまな操作において、サーフェス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、同じポイントでのサーフェスの接線を指定することです。座標系の種類がすでにわかっていると仮定すると（たとえば、右手系）、この方法でサーフェスの方向を完全に指定できます。つまり、オブジェクト空間とサーフェス空間との間の変換を行う 3×3 行列が定義できることになります。
接線と法線で、サーフェス座標系の 2 つの座標軸（行列の 2 つの列）を指定します。従法線と呼ばれる 3 番目の座標軸は、接線と法線の外積として計算できます。接線には、以下に示す 2 種類があります。

- テクスチャ空間の接線：semantic属性としてTEXTANGENTとTEXBINORMALを記述し、set属性を記述した<**input**>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。

COLLADA ドキュメントでは、これら 2 つの用途および論理的な配置が異なるため、これらの 2 つの接線がサポートされています。

属性

<**polygons**>要素には、以下の属性があります。

count	**uint**	ポリゴンプリミティブの数。必須。
material	**xs:NCName**	マテリアルのシンボルを宣言します。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。オプション。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。
name	**xs:NCName**	オプション。

関連要素

<**polygons**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**mesh**、**convex_mesh**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<input>			0 以上
<p>			0 以上
<ph>			0 以上
<extra>			0 以上

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>要素中（または<h>要素）の各値は入力に参照されますが、参照時には各値の記述された順番が使用されます。<p>要素の最初の値は offset 属性の値として 0 が指定された入力によって参照され、2 番目の値は offset 属性として 1 が指定された入力によって参照されます。線の頂点は、その頂点に対する入力である値によって構成されます。すべての入力が参照された後は、次の値はオフセット 0 の値として入力によって再度参照され、新しい頂点を開始することになります。。
生成される頂点の順番は反時計回りで、これによってそれぞれのポリゴンのサーフェスを記述します。プリミティブが頂点法線を持たずに構成される場合には、ライティングを有効にするために、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。

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**>要素のセマンティッ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、ジオメトリ接線を指定する方法を示した例です（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>要素中の同じエントリが参照される点に注意してください）。

```
<mesh>
  <source id="position" />
  <source id="normal" />
  <source id="tangent" />
  <vertices id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <polygons count="1" material="#Bricks">
    <input semantic="VERTEX" source="#verts" offset="0"/>
    <input semantic="NORMAL" source="#normal" offset="1"/>
    <input semantic="TANGENT" source="#tangent" offset="1"/>
    <p>0 0  2 1  3 2  1 3</p>
  </polygons>
</mesh>
```

以下は、テクスチャ空間の接線を指定する方法を示した例です（テクスチャ空間の接線は、入力への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 id="verts">
    <input semantic="POSITION" source="#position"/>
  </vertices>
  <polygons count="1" material="#Bricks">
    <input semantic="VERTEX" source="#verts" offset="0"/>
    <input semantic="NORMAL" source="#normal" offset="1"/>
    <input semantic="TEXCOORD" source="#tex-coord" offset="2" set="0"/>
    <input semantic="TEXTANGENT" source="#tex-tangent" offset="3" set="0"/>
    <p>0 0 0 1  2 1 2 0  3 2 1 2  1 3 3 3</p>
  </polygons>
</mesh>
```

polylist

概要

<**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します。

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します。
頂点配列の情報は<**mesh**>要素の属性配列として別に提供され、<**polylist**>要素でインデックス付けされます。
<**polylist**>に記述されているポリゴンには、任意の数の頂点を含めることができます。4 つ以上の頂点を含んでいるpolylistプリミティブは、非平面である可能性もあります。

さまざまな操作で、サーフェスのポイントの正確な方向が必要となります。この方向を法線ベクトルで部分的に定義できますが、法線自体の周りの「回転」については不明瞭のままです。この曖昧な余分の回転をなくす 1 つの方法は、同じポイントでのサーフェスの接線を指定することです。座標系の種類がすでにわかっていると仮定すると（たとえば、右手系）、この方法でサーフェスの方向を完全に指定できます。つまり、オブジェクト空間とサーフェス空間との間の変換を行う 3×3 行列が定義できます。
接線と法線で、サーフェス座標系の 2 つの座標軸（行列の 2 つの列）を指定します。従法線と呼ばれる 3 番目の座標軸は、接線と法線の外積として計算できます。2 つの異なる種類の接線は、以下のように文書中で異なる目的と論理的な配置を持つので、COLLADA では、それら 2 つの接線をサポートしています。

- テクスチャ空間の接線：semantic属性としてTEXTANGENTとTEXBINORMALを記述した<**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**	この要素の文字列の名。オプション。
count	**uint**	ポリゴンプリミティブの数。必須。
material	**xs:NCName**	マテリアルのシンボルを宣言します。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	**mesh**
個要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<input>			0 以上
<vcount>			0 または 1
<p>			0 または 1
<extra>			0 以上

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サーフェスを記述します。
プリミティブが頂点法線を持たずに構成される場合には、ライティングを有効にするために、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。

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、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>
  <polylist count="3" material="#Bricks">
    <input semantic="VERTEX" source="#verts" offset="0" />
    <input semantic="NORMAL" source="#normal" offset="1" />
    <vcount>4 4 3</vcount>
    <p>0 0  2 1  3 2  1 3  4 4  6 5  7 6  5 7  8 8  10 9  9 10</p>
  </polylist>
</mesh>
```

rotate

概要

回転角度と回転軸を表す数学的なベクトルを含みます。

コンセプト

回転は、平行移動を行わずに、座標系の中の物体の向きだけを変更します。コンピュータグラフィックスでは、向きを変える際に、座標系を基準として値を変更せずに済むように回転移動を利用します。言い換えると、<**rotate**>要素は、ローカル座標の原点を中心に座標軸を変換することを意味します。

属性

<**rotate**>要素には、以下の属性があります。

sid	**xs:NCName**	この要素のサブ識別子を含んだテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。

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	**node**、**mass_frame**、**shape**、**ref_attachment**、**attachment**
子要素	なし
その他	なし

備考

<**rotate**>要素には、OpenGL®やRenderMan®の回転の仕様と同様に、4 つの浮動小数点値のリストが含まれています。これらの値は、回転軸を指定する列ベクトル [X, Y, Z]と、度数で表された角度を表します。

例

以下は、Y軸を基軸とした 90 度の回転を表す<**rotate**>要素の例です。

```
<rotate>
  0.0 1.0 0.0 90.0
</rotate>
```

sampler

概要

N 次関数を宣言します。

コンセプト

COLLADAでは、アニメーションの関数曲線を 1 次元の<**sampler**>要素で表します。<**sampler**>要素では、サンプリングポイントと、それらの補間方法を定義します。アニメーション・チャンネルの値の計算に利用する際には、アニメーションのキーフレームがサンプリングポイントとなります。

サンプリングポイント（キーフレーム）は、補間の種類のシンボル名でですが、サンプラーへの入力データソースです。アニメーション・チャンネルは、サン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**	要素のユニークな識別子を含んだテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。オプション。

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	**下のサブセクションを参照してください**
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<input>			1 以上

備考

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、ほしくは他のものを指定する場合もあります。
COLLADAでは、補間の種類として、LINEAR、BEZIER、CARDINAL、HERMITE、BSPLINE、STEPが認識されます。これらのシンボル名は、シンボル名が格納されている**<Name_array>**を含む**<source>**要素に保持されます。これらの値は、INTERPOLATION **<input>**要素によってサンプラーに供給されます。完全な<**sampler**>要素には、semantic属性がINTERPOLATIONの<**input**>要素を 1 つ含まれていなければなりません。COLLADAでは、デフォルトの補間方式が指定されていません。つまり、補間の種類を指定しなかった場合、<**sampler**>の動作はアプリケーションの定義したものとなります。

例

以下は、「group1_translate-anim-outputY」というidを持つキーフレームのsource要素のy軸方向の値を評価する**<sampler>**要素の例です。INTERPOLATION入力は、**<source>**要素に、より明確にするため以下のように示されます。

```
<animation id="group1_translate-anim">
  <source id="group1_translate-anim-inputY">
   ...
  </source>
  <source id="group1_translate-anim-outputY">
   ...
  </source>
  <source id="group1_translate-anim-interpY">
    <Name_array count="3" id="group1_translate-anim-interpY-array">
      BEZIER BEZIER BEZIER
    </Name_array>
    <technique_common>
      <accessor count="3" source="#group1_translate-anim-interpY-array">
        <param name="Y" type="Name"/>
      </accessor>
    </technique_common>
  </source>
  <sampler id="group1_translate-anim-samplerY">
    <input semantic="INPUT" source="#group1_translate-anim-inputY"/>
    <input semantic="OUTPUT" source="#group1_translate-anim-outputY"/>
    <input semantic="IN_TANGENT" source="#group1_translate-anim-intanY"/>
    <input semantic="OUT_TANGENT" source="#group1_translate-anim-outtanY"/>
    <input semantic="INTERPOLATION" source="#group1_translate-anim-interpY"/>
  </sampler>
  <channel source="#group1_translate-anim-samplerY"
target="group1/translate.Y"/>
</animation>
```

scale

概要

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を含みます。

コンセプト

拡大縮小は、回転や平行移動を伴わずに、座標系内の物体の大きさを変更します。コンピュータグラフィックスでは、座標系軸に対する値の大きさや比率を変更するために拡大縮小変換を利用します。

属性

<**scale**>要素には、以下の属性があります。

sid	**xs:NCName**	この要素のサブ識別子を含んだテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。

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	**node**
子要素	なし
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した<**scale**>要素の例です。

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

scene

概要

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）のベースを宣言します。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といった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、視覚情報および関連データをノードとして含む有向非巡回グラフ（DAG）、つまりツリー構造のデータです。シーングラフ構造は、データ処理の最適やレンダリングに役立つため、コンピュータグラフィックスでは幅広く利用されています。

属性

<**scene**>要素には属性はありません。

関連要素

<**scene**>要素は、以下の要素と関連性があります。

出現回数	0 回または 1 回
親要素	**COLLADA**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<instance_physics_scene>			0 以上
<instance_visual_scene>			0 または 1
<extra>			0 以上

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**〉下でインスタンス化された<**visual_scene**>要素から構成されます。インスタンス化された<**physics_scene**>要素は、シーンに適用される任意のフィジックス（物理）を表します。

例

以下は、idが「world」の視覚的シーンをインスタンス化する単純な<**scene**>要素の例です。

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

```
</COLLADA>
```

skeleton

概要

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します。

コンセプト

シーングラフが複雑になるにともなって、シーン中に同じオブジェクトを何度も表示しなければならない場合があります。そのため、スペースを節約するために、オブジェクトの実際のデータ表現を一度保存しておき、複数の場所で参照することが可能です。しかしながら、シーンに毎回表示する際に、さまざまにオブジェクトを変形したい状況も発生します。スキンコントローラの場合、オブジェクトの変形は外部のノードセットから派生されます。
また、同じスキンコントローラの複数のインスタンスに、ノードセットの別々のインスタンスを参照させたい場合も発生するでしょう。その場合、個々のコントローラを個別にアニメーション化する必要があります。スキンコントローラをアニメーション化するためには、それに影響を及ぼすノードをアニメーション化しなければならないからです。
さらに、別々のスキンコントローラのインスタンスに、同じノードセットを参照させる必要が出てくる場合もあります。たとえば、キャラクタ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	**instance_controller**
子要素	なし
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 1 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">
    <skeleton>#skel01</skeleton>
  </instance_controller>
</node>
<node>
  <instance_controller url="#skin">
    <skeleton>#skel02</skeleton>
  </instance_controller>
    </node>
```

skew

概要

角度と、回転軸と平行移動軸を表す 2 つ数学的なベクトルを含んでいます。

コンセプト

スキュー処理（剪断変形）とは、特定の座標軸に沿って物体を変形させることです。変形の影響を受ける値は、その値の使用する座標軸に対して平行移動されます。コンピュータグラフィックスでは、物体を変形させたり、画像の歪みを補正したりするために、この剪断変形変換を利用します。

属性

<**skew**>要素には、以下の属性があります。

sid	**xs:NCName**	この要素のサブ識別子を含んだテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。

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	なし
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、それに続く回転軸と平行移動軸を指定する 2 つの列ベクトルを表します。

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

重み付けブレンドを用いたスキニングの記述に必要な頂点とプリミティブの情報を含みます。

コンセプト

キャラクタのスキニングを行う場合には、アニメーションエンジンによって、スキニングされたキャラクタの関節（スケルトン）が動かされます。関節と、スキンのトポロジーを構成するメッシュの頂点との関連付けは、スキンメッシュによって記述します。制御アルゴリズムによってスキンメッシュの頂点を変換するうえで、関節が変換に影響を与えます。
一般的なスキニングアルゴリズムでは、隣接する関節の影響を重みの値に応じてブレンドします。
古典的なスキニングアルゴリズムでは、ジオメトリの点（メッシュの頂点など）を、ノード（関節とも呼ばれます）の行列で変換し、スカラー加重を利用して、その結果の加重平均を計算します。
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」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「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。

- 「スケルトンをスキンにバインドする」前処理
- スキニングアルゴリズムを実行し、スケルトンのポーズの変化に応じてスキン形状を変更

前処理によって作成される「スキニング情報」は、以下の情報から構成されます。

- バインド形状：「デフォルトの形状」とも呼ばれます。これは、スキンをスケルトンにバインドした時点でのスキンの形状です。バインド形状には、<**mesh**>の各頂点に対応する位置が必須として含まれるほか、オプションとして頂点の属性を含めることができます。
- インフルエンス：<**mesh**>の各頂点に対応する、ノードと重みの対で構成される可変長リストです。
- バインドポーズ：バインド時点でのすべてのインフルエンスの座標変換を表します。これは、ノードごとの情報として保存される「バインド行列」で表現されます。バインド行列とは、バインド時にノードをローカル座標からワールド座標へ変換するための行列です。

スキニングアルゴリズムにおいては、変換はすべてバインドポーズに対して相対的に行われます。この相対変換は、通常、スケルトンの各ノードについて事前に計算され、スキニング行列として保存されます。
頂点の新しい（スキングされた）位置を得るには、影響を与える各ノードのスキニング行列で、バインド形状の頂点位置を変換します。その結果に対して重み付けブレンドを利用して求められた平均が、スキニングされた位置となります。
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

- 重み付けブレンドの規定値を生成する。通常は、何らかのフォールオフ（減衰）を考慮します。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、重みが 0 の場合、インフルエンスは無視してかまいません。

このような処理によって、アーチストが重みを変更できるようになります。通常、重みの編集は、メッシュ上への「ペイント」によって行います。

属性

<**skin**>要素には、以下の属性があります。

source	**xs:anyURI**	ベースメッシュ（静的メッシュまたはモーフィングメッシュ）を参照する URI。スキンメッシュのバインド形状も表します。 必須。

関連要素

<**skin**>要素は、以下の要素と関連性があります。

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<bind_shape_matrix>	バインド前のベースメッシュの位置と方向に関して追加情報を指示します。<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。		0 または 1
<source>	特定のベースメッシュのスキニングに必要なほとんどのデータを指定します。		3 以上
<joints>	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。		1
<vertex_weights>	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、関節の配列に対するインデックスとして-1 を指定した場合、バインド形状を参照することになります。重みの値は、利用する前に正規化しておくべきです。		1
<extra>			0 以上

備考

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>
      <joints>
        <input semantic="JOINT" source="#Joints"/>
        <input semantic="INV_BIND_MATRIX" source="#Inv_bind_mats"/>
      </joints>
      <vertex_weights count="4">
        <input semantic="JOINT" source="#Joints"/>
        <input semantic="WEIGHT" source="#Weights"/>
        <vcount>3 2 2 3</vcount>
        <v>
        -1 0  0 1  1 2
        -1 3  1 4
        -1 3  2 4
        -1 0  3 1  2 2
        </v>
      </vertex_weights>
    </skin>
  </controller>
```

source

概要

<source>要素は、その<source>要素を参照する<input>要素のセマンティクスにしたがって値を提供するデータのリポジトリを宣言します。

コンセプト

データソースとは、確立された通信チャネルを介してアクセス可能な既知の情報源のことです。
データソースは、情報へのアクセス方式を提供します。このアクセス方式としては、情報の表現に応じて各種のものが実装されています。情報は、データの配列としてローカルに保存することも、データを生成するプログラムとして使用することもできます。

属性

<source>要素には、以下の属性があります。

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

関連要素

<source>要素は、以下の要素と関連性があります。

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**、**spline**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

注意：配列要素（**<bool_array>**, **<float_array>**, **<int_array>**, **<Name_array>**、または **<IDREF_array>**）を複数記述することはできません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<IDREF_array>			注意を参照
<Name_array>			注意を参照
<bool_array>			注意を参照
<float_array>			注意を参照
<int_array>			注意を参照
<technique_common>	**<source>**の子として、この要素は、**<accessor>**要素を 1 つだけ含まなければなりません。		0 または 1
<technique>			0 以上

備考

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

制御点（CV）とセグメント情報を持つ複数セグメントで構成されるスプラインを表す情報を含んでいます。

コンセプト

制御点ごとの情報と、セグメントごとの情報のいずれも、制御点に対して保存されます。セグメントごとの情報は、指定された制御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。
それぞれの制御点には任意の数の属性を指定できますが、属性の数と種類は、1 つのスプライン中で個々の制御点に対して一定でなければなりません。
制御点ごとの情報と、セグメントごとの情報としては、以下のものがあります。

- 制御点の位置（2 次、3 次、または 4 次：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異なる補間方法（LINEAR、BEZIER など）を利用可能。
- 区分線形近似のステップ数（テッセレーション）
- 制御点ごとのカスタムデータ

<**spline**>の構成は<**mesh**>と非常によく似ています。<**spline**>には、属性を提供する<**source**>要素と<control_vertices>要素が含まれており、これによって属性ストリームが組み立てられます。

属性

<**spline**>要素には、以下の属性があります。

closed	**bool**	最初と最後の制御点を結び付けているセグメントがあるかどうかを表します。デフォルト値は「false」で、開いたスプラインであることを意味します。オプション。

関連要素

<**spline**>要素は、以下の要素と関連性があります。

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	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<source>	スプラインの制御点とセグメントの値を提供します。		1 以上

名前/例	解説	デフォルト値	出現回数
<control_vertices>	スプラインの制御点を表します。		1
<extra>			0 以上

備考

制御点は、semantic属性がPOSITIONである**<input>**要素を少なくとも 1 つ持たなければなりません。COLLADAでは、多項式補間の種類として、LINEAR、BEZIER、CARDINAL、HERMITE、BSPLINE、STEP、NURBSが認識されます。これらのシンボル名は、シンボル名が格納されている**<Name_array>**を含む**<source>**要素に保持されます。これらの値は、INTERPOLATION **<input>**要素によってサンプラーに供給されます。
共通プロファイルでは、**<spline>**/<control_vertices>用として、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
POSITION	float2、float3、float4	制御点の位置	なし
INTERPOLATION	Name	制御点から始まるセグメントを表す多項式の**補間**の種類。共通プロファイルでの種類は次の通り：LINEAR、BEZIER, HERMITE、CARDINAL、BSPLINE、STEP、NURBS	LINEAR
IN_TANGENT	float2、float3	制御点の前に位置するセグメントの形状を制御する接線（BEZIER または HERMITE）	なし
OUT_TANGENT	float2、float3	制御点の後に位置するセグメントの形状を制御する接線（BEZIER または HERMITE）	なし
CONTINUITY	Name	制御点での連続性の制限を定義する。共通プロファイルでの種類は次の通り：C0、C1、G1	C1
LINEAR_STEPS	int	制御点の次のスプラインセグメントで利用される区分線形近似のステップ数	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="#CVs-Pos"/>
    <input semantic="INTERPOLATION" source="#CVs-Interp"/>
    <input semantic="LINEAR_STEPS" source="#CVs-LinSteps"/>
  </control_vertices>
</spline>
```

spot

概要

スポットライトを表します。

コンセプト

<spot>要素では、スポットライトを表すのに必要なパラメータを宣言します。スポットライトは、空間中の既知の位置から特定の方向に光を放射します。スポットライトからの光は、円錐形に放射されます。スポットライトの輝度は、光源の放射角に対する角度が増えるにつれ、および光源からの距離が大きくなるにつれて減衰します。
光源の位置は、インスタンス化されるノードの座標変換で定義されます。ローカル座標での光源のデフォルトの方向ベクトルは[0, 0, -1]で、負の Z 軸方向に向いてます。実際の放射方向は、光源がインスタンス化されるノードの座標変換として定義されます。

属性

<spot>要素には属性はありません。

関連要素

<spot>要素は、以下の要素と関連性があります。

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	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<color sid="...">	光源のカラーを表す 3 つの浮動小数点数を含みます。sid 属性を持つことも可能です。		1
<constant_attenuation sid="...">	sid はオプション。	1.0	0 または 1
<linear_attenuation sid="...">	sid はオプション。	0.0	0 または 1
<quadratic_attenuation sid="...">	sid はオプション。	0.0	0 または 1
<falloff_angle sid="...">	sid はオプション。	180.0	0 または 1
<falloff_exponent sid="...">	sid はオプション。	0.0	0 または 1

備考

<spot>要素は、<light>要素の下の<technique_common>の子としてだけ指定できます。
特定の距離における総合的な減衰は、<constant_attenuation>、<linear_attenuation>、<quadratic_attenuation>を利用して計算されます。その際、以下の計算式が利用されます。

```
  A = constant_attenuation + ( Dist * linear_attenuation ) + (( Dist^2 ) *
quadratic_attenuation )
```

光の方向を基準として減衰量を指定するには、<falloff_angle>と<falloff_exponent>を利用します。

例

以下は、<**spot**>要素の例です。

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

targets

概要

モーフィングターゲットと、その重み付け、さらに関連したユーザ定義属性を宣言します。

コンセプト

<**targets**>要素では、モーフィングターゲットとモーフィングの重み付けを宣言します。<**input**>要素で、ブレンドするメッシュセットを定義し、さらにブレンド時に利用される重み付けの配列も定義します。モーフィングターゲットに関連付ける詳細な情報を指定するのにも利用できます。

属性

<**targets**>要素には属性はありません。

関連要素

<**targets**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**morph**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<input>	1 つは semantic="MORPH_TARGET" とともに、もう 1 つは semantic="MORPH_WEIGHT" とともに記述しなければなりません。		2 以上
<extra>			0 以上

備考

例

以下は、<**targets**>要素の完全な例です。

```
<targets>
  <input source="#morph-targets" semantic="MORPH_TARGET" />
  <input source="#morph-weights" semantic="MORPH_WEIGHT" />
</targets>
```

technique (core)

概要

コンテンツの一部の処理するために使われる情報を宣言します。各テクニックは、対応するプロファイルに適合させておく必要があります。

FXの**<technique>**については、「**technique (FX)**」を参照してください。

コンセプト

テクニックとは、特定のプラットフォームやプログラムで必要となる情報を記述するためのもので、プロファイルとして表現されます。テクニックのコンテキストは、プロファイル、インスタンス文書中のその親要素という 2 つの情報で定義します。
通常、テクニックは「スイッチ」として機能します。コンテンツ中の特定の部分に対して 1 つ以上のテクニックが指定されている場合、インポート時に、両方ではなく、どちらか一方が選ばれます。使用されるプロファイルの選択は、インポートを行うアプリケーションのサポートしているプロファイルを基準にして選ばれます。テクニックにはアプリケーションデータとプログラムが含まれ、1 つの単位として管理可能なアセットを構成しています。

属性

<**technique**>要素には、以下の属性があります。

profile	**xs:NMTOKEN**	プロファイルの種類。テクニックの対象とするプラットフォームや機能表したベンダ定義の文字列。必須。

<**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言語として）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**、**bind_material**
子要素	「備考」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。整形式のデータは、すべてCOLLADAスキーマに対して検証されます。データ検証に別のスキーマを利用するように指定することも可能です。それ以外も正しいものとみなされますが、実際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>
```

technique_common

概要

すべての COLLADA 実装がサポートしなければならない共通プロファイルの特定の要素の情報を指定します。

コンセプト

各要素の**<technique_common>**引数と子はユニークです。詳しくは、各親要素を参照してください。

属性

各親要素に関するリファレンスの章のエントリを参照してください。

関連要素

<technique_common>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**bind_material**, **instance_rigid_body**, **light**, **optics**, **physics_material**, **physics_scene**, **rigid_body**, **rigid_constraint**, **source**
子要素	各親要素に関するリファレンスの章のエントリを参照してください。
その他	なし

備考

例

translate

概要

x 軸、*y* 軸、*z* 軸の各方向への距離を表す数学的なベクトルが含まれます。

コンセプト

平行移動は、座標系中の物体を回転させることなく、位置だけを変更します。コンピュータグラフィックスでは、座標系を基準として位置を決めたり、値を移動したりするために、平行移動の座標変換を利用します。逆に、平行移動は、ローカル座標系の原点を移動することとも考えられます。

属性

<**translate**>要素には、以下の属性があります。

sid	**xs:NCName**	この要素のサブ識別子を含んだテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。

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	**node**, **shape**, **mass_frame**, **ref_attachment**, **attachment**
子要素	なし
その他	なし

備考

<**translate**>要素には、3 つの浮動小数点値が含まれます。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 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

<**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します。

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付けされます。
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**	この要素のテキスト文字列名。オプション。
count	**uint**	三角形プリミティブの数。必須。
material	**xs:NCName**	マテリアルのシンボルを宣言します。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。オプション。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	**mesh**、**convex_mesh**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<input>			0 以上
<p>	（p は primitive の頭文字です）個々の三角形の頂点属性を表します。		0 または 1
<extra>			0 以上

備考

<p>要素中の各値は入力に参照されますが、参照時には各値の記述された順番が使用されます。<p>要素の最初の値は `offset` 属性の値として 0 が指定された入力によって参照され、2 番目の値は `offset` 属性として 1 が指定された入力によって参照されます。三角形の頂点は、その頂点に対する入力の値に

よって構成されます。すべての入力が参照された後は、次の値はオフセット 0 の値として入力によって再度参照され、新しい頂点を開始することになります。
生成される三角形の頂点の順番は反時計回りで、これによってそれぞれの三角形のサーフェスを記述します。
プリミティブが頂点法線を持たずに構成される場合には、ライティングを可能にするために、プリミティブごとの法線をアプリケーションが生成する場合もあります。

例

以下は、2 つの三角形を記述する<**triangles**>要素の例です。<**input**>要素によって指定されているセマンティックスによると、<**source**>要素で位置および法線のデータが記述されています。<p>要素の値の記述された順序によって、入力値の利用される順序が決まります。

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

<**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します。

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付けされます。
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、現在の頂点に加えて、1 番目の頂点と、現在の頂点の直前の頂点の再利用によって構成されます。

属性

<**trifans**>要素には、以下の属性があります。

name	**xs:NCName**	この要素のテキスト文字列名。オプション。
count	**uint**	三角形ファンプリミティブの数。必須。
material	**xs:NCName**	マテリアルのシンボルを宣言します。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。オプション。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	**mesh**、**convex_mesh**
子要素	**下のサブセクションを参照してください**
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<input>			0 以上
<p>	（p は primitive の頭文字です）任意の数の連結した三角形の頂点属性を表します。		0 以上
<extra>			0 以上

備考

<**trifans**>要素には、**<p>**要素の並びが含まれます。

<p>要素中の各値は入力に参照されますが、参照時には各値の記述された順番が使用されます。<p>要素の最初の値は offset 属性の値として 0 が指定された入力によって参照され、2 番目の値は offset 属性として 1 が指定された入力によって参照されます。三角形の頂点は、その頂点に対する入力の値によって構成されます。すべての入力が参照された後は、次の値はオフセット 0 の値として入力によって再度参照され、新しい頂点を開始することになります。
生成される頂点の順番は反時計回りで、これによってそれぞれの三角形のサーフェスを記述します。
プリミティブが頂点法線を持たずに構成される場合には、ライティングを可能にするために、プリミティブごとの法線をアプリケーションが生成する場合もあります。

例

以下は、2 つの三角形を記述する<**trifans**>要素の例です。<**input**>要素によって指定されているセマンティクスによると、2 つの<**source**>要素で位置および法線のデータが記述されています。<p>要素の値の記述された順序によって、入力値の利用される順序が決まります。

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

<**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します。

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付けされます。
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、現在の頂点に加えて、現在の頂点の直前の 2 つの頂点の再利用によって構成されます。

属性

<**tristrips**>要素には、以下の属性があります。

name	**xs:NCName**	この要素のテキスト文字列名。オプション。
count	**uint**	三角形ストリッププリミティブの数。必須。
material	**xs:NCName**	マテリアルのシンボルを宣言します。 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。オプション。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	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<input>			0 以上
<p>	（p は primitive の頭文字です。）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。		0 以上
<extra>			0 以上

備考

<**tristrips**>要素には、**<p>**要素の並びが含まれます。

<p>要素中の各値は入力に参照されますが、参照時には各値の記述された順番が使用されます。<p>要素の最初の値は offset 属性の値として 0 が指定された入力によって参照され、2 番目の値は offset 属性として 1 が指定された入力によって参照されます。三角形の頂点は、その頂点に対する入力の値によって構成されます。すべての入力が参照された後は、次の値はオフセット 0 の値として入力によって再度参照され、新しい頂点を開始することになります。
生成される頂点の順番は、最初（および 3 番目、5 番目...）の三角形では反時計回りで、2 番目（および 4 番目、6 番目...）の三角形では時計回りとなり、これによって三角形のサーフェスを記述します。プリミティブが頂点法線を持たずに構成される場合には、ライティングを可能にするために、プリミティブごとの法線をアプリケーションが生成する場合もあります。

例

以下は、2 つの三角形を記述する<**tristrips**>要素の例です。<**input**>要素によって指定されているセマンティクスによると、2 つの<**source**>要素で位置および法線のデータが記述されています。<p>要素の値の記述された順序によって、入力値の利用される順序が決まります。

```
<mesh>
  <source id="position"/>
  <source id="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

スキニング処理で利用される関節と重み付けの組み合わせを記述します。

コンセプト

<**vertex_weights**>要素は、ベースメッシュ中の各頂点に対して、それぞれの関節と重み付けを関連付けます。

属性

<**vertex_weights**>要素には、以下の属性があります。

count	**uint**	ベースメッシュ中の頂点の数。必須。

関連要素

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

出現回数	スキーマ中に定義された要素の数
親要素	**skin**
子要素	下のサブセクションを参照してください。
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<input>	**<input>**要素の一つは、**<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。		2 以上
<vcount>	個々の頂点に関連付けられたボーンの数を表します。		0 または 1
<v>	個々の頂点に関連付けられたボーンと属性を表します。 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るべきです。		0 または 1
<extra>			0 以上

備考

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、より具体的な<**vertex_weights**>要素の例です。<vcount>要素で、最初の頂点にはボーンが 3 つあり、2 番目の頂点には 2 つあるといったことを指定している点に注意してください。また、<v>要素では、最初の頂点は、バインド形状に対してweights[0]で重み付けを行い、同様にweights[1]はボーン 0 に対して、weights[2]はボーン 1 に対して重み付けを行うことも指定しています。

```
<skin>
  <source id="joints"/>
  <source id="weights"/>
  <vertex_weights count="4">
      <input semantic="JOINT" source="#joints"/>
      <input semantic="WEIGHT" source="#weights"/>
    <vcount>3 2 2 3</vcount>
    <v>
                  -1 0  0 1  1 2
                  -1 3  1 4
                  -1 3  2 4
                  -1 0  3 1  2 2
    </v>
  </vertex_weights>
</skin>
```

vertices

概要

メッシュ頂点の属性や識別情報を宣言します

コンセプト

<**vertices**>要素は、メッシュ中のメッシュ頂点を記述します。メッシュ頂点は、メッシュを構成している頂点の識別情報、およびテッセレーションに対して不変な他の頂点属性を表します。

属性

<**vertices**>要素には、以下の属性があります。

id	**xs:ID**	この要素のユニークな識別子を含んだテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。必須。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。

関連要素

<**vertices**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**mesh**、**convex_mesh**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<input>	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ー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。		1 以上
<extra>			0 以上

備考

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

COLLADA リソースのコンテンツで視覚化可能な情報セット全体を表します。

コンセプト

visual_sceneの階層構造は、シーングラフにまとめられます。シーングラフは、視覚情報および関連データをノードとして含む有向非巡回グラフ（DAG）、つまりツリー構造のデータです。シーングラフ構造は、データ処理の最適化やレンダリングに役立つため、コンピュータグラフィックスでは幅広く利用されています。

属性

<**visual_scene**>要素には、以下の属性があります。

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

関連要素

<**visual_scene**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**library_visual_scenes**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<node>			1 以上
<evaluate_scene>			0 以上
<extra>			0 以上

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/2005/11/COLLADASchema.xsd"
version="1.4.1">
  <library_visual_scenes>
    <visual_scene id="world">
      <node id="root"/>
      </visual_scene>
  </library_visual_scenes>
  <scene>
    <instance_visual_scene url="#world"/>
  </scene>
</COLLADA>
```

5章　COLLADA フィジックス リファレンス

概要

COLLADA フィジックスを用いると、コンテンツ制作者は、ビジュアルシーンのオブジェクトに物理的なプロパティを加えることが可能になります。
ビジュアルシーンのノードは、物理シミュレーションで制御できます。これは、フィジックスシーンで剛体および制約をインスタンス化することで行われます。その後、剛体は、ビジュアルシーンの適切なノードをターゲットにします。剛体は、シミュレーションに影響を及ぼす、質量や慣性などの物理的なプロパティを持ちます。剛体は、ジオメトリメッシュやカプセル、ボックスなどのコリジョン形状を 1 つまたは複数指定することで、互いにコリジョンできます。
剛体は、力学的であっても運動学的であってもかまいません。力学剛体は、単一の視覚的なノードを完全に物理シミュレーションによって操作しながら、そのノードの変換を制御します。運動学剛体（非力学剛体）は、外部のアニメーションにより制御されます。例えば、サッカープレーヤーの足に相当する運動学剛体をキーフレームのアニメーションを通して制御することは可能ですが、サッカーボールについては力学剛体の特性を持つことになるでしょう。
剛体のペアは、剛体制約を通して接続できます。剛体制約には、角および線形で自由度を指定できる多くのパラメータがあります。例えば、車輪をシャーシ（筐体）に束縛することができます。そうすると、車輪は *x* 軸にのみ沿って回転し、他の軸に沿って移動することも回転することもありません。
剛体および制約は、フィジックスモデル内でグループ化されて、車やラグドールなどの複雑な物理オブジェクトを定義します。後者は、通常、落下しているキャラクタや死にかけているキャラクタをシミュレートするのに使用されます。
バージョン 1.4 では、COLLADA フィジックスは、剛体の力学システムを表すことができるだけで、布や流動体などの特徴はサポートしていません。

物理的な単位について

使用する値を適切に一致させれば、（量子効果や相対論的効果は差し引いて）ニュートン物理学を適切にシミュレートできます。たとえば、距離と長さをメートル単位で指定し、時間を秒単位とした場合、導かれる力量の単位はニュートンとなります。
こういった理由から、多くの物理エンジンでは単位を定めておらず、COLLADA フィジックスも規格として、各構成要素の単位の指定を必須とはしていません。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。
COLLADA では、以下の測定単位を使用します。

測定	単位
時間	秒（標準単位）
角度	度（標準単位）
質量	キログラム（標準単位）
距離	メートル（デフォルトの単位）。**<asset>**要素は**<unit>**要素を含んでおり、この要素によって、距離の大きさを対応するアセットに対して再定義することができます。

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、アルファベットの I 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（アーム）が r で、無限小の質量 m を持つ単一の物体の場合、慣性モーメントは次の式で表されます。

$$I=mr^2$$

このような物体の角運動量は、慣性と角速度の積となります。
任意の形状で、質量も場所によって異なる剛体の場合、その物体はそれぞれ異なる質量を持つ粒子の集まりとみなすことができます。この物体の慣性モーメントは、それぞれの粒子の(質量)×(回転軸からの距離)^{2}の総和となります。
粒子の慣性は、剛体に対して相対的なあらゆる場所、あらゆる角度において求めることができます。しかしながら、その粒子の質量の中心（重心）で求めると簡単に表現できます（むしろ、自由回転している物体の場合、質量の重心を中心に回転するため、このほうが直感的です）。
一般に、慣性は 2 階層テンソルとして、「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。

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

- mi は、粒子の質量です。
- (xi, yi, zi)は、粒子の位置を表します。

慣性テンソルの対角成分は慣性モーメントと呼ばれ、非対角成分は慣性相乗モーメントと呼ばれます。
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。
（CM における）慣性テンソルの対角成分が同じである点に注意してください。また、基準座標系と、剛体の（ローカル座標の）主軸が並列である場合、慣性相乗モーメントは 0 となります。このような、純粋な 3×3 の対称テンソルは、3 つの値（float3）で表すことができます。
ここで、左上の要素は「孤立している」と考え、次のように解釈することができます。
ローカル座標の*x*軸を中心とした回転の場合、慣性は各粒子の（質量）×（Y-Z平面上のアーム）^{2}の総和です。これはもちろん慣性モーメントの定義（I = mr2）そのものです。
重心から外れた場所をサンプルする場合には、さらに追加の慣性モーメントがあります。

$$I_t \equiv \begin{bmatrix} m(y_0^2 + z_0^2) & -m x_0 y_0 & -m x_0 z_0 \\ -m y_0 x_0 & m(z_i^2 + x_i^2) & -m y_0 z_0 \\ -m z_0 x_0 & -m z_0 y_0 & m(x_0^2 + y_0^2) \end{bmatrix}$$

- m は物体の質量です。
- (x0, y0, z0)は慣性を計算する位置です。

任意の場所における物体の慣性モーメントは、（CM における）慣性テンソルとこの「オフセット」の和となります。

新しいジオメトリタイプ

フィジックスエンジンで使用するコリジョン形状は、任意の形状よりも、分析的な形状（プリミティブ）や凸包を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**>に加えて、<**box**>や<**sphere**>といったプリミティブなジオメトリタイプがいくつか追加されました。

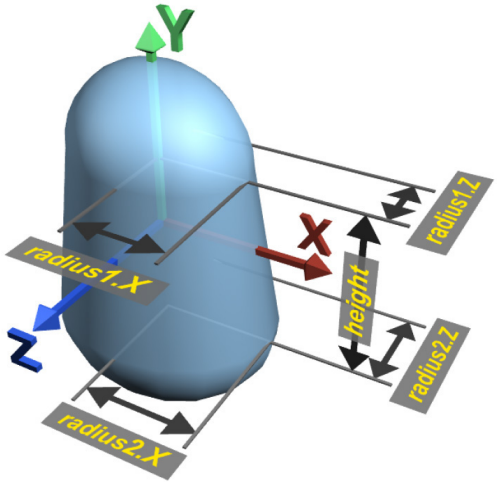
<convex_mesh>を除いてこれらの形状は、レンダリング用のものではありません。理由は、メッシュや再分割サーフェスなどの方が、レンダリング目的にはるかに適しているからです。

このシステム（分析的な形状＋凸包メッシュ）で任意の形状を記述できるため、<bounding_box>要素は廃止予定となりました。

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 および radius1.Z -->
  <radius2> 1.5 1.8 </radius2>   <!- radius2.X および radius2.Z -->
</tapered_cylinder>
```



box

概要

軸並行の中央揃えされたボックスプリミティブを宣言します。

コンセプト

ジオメトリプリミティブ（分析的な形状）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「新しいジオメトリタイプ」セクションを参照してください。

属性

<**box**>要素には属性はありません。

関連要素

<**box**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**shape**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<half_extents>	ボックスの範囲を表す 3 つの浮動小数点値	なし	1
<extra>		なし	0 以上

例

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

capsule

概要

ローカルの *y* 軸並行の中央揃えされたカプセルプリミティブを宣言します。

コンセプト

ジオメトリプリミティブ（分析的な形状）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「新しいジオメトリタイプ」を参照してください。

属性

<**capsule**>要素には属性はありません。

関連要素

<**capsule**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**shape**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<height>	半球同士をつなぐ、中央の線分の長さを表す浮動小数点値	なし	1
<radius>	カプセル（おそらく楕円形）の半径を表す 2 つの浮動小数点値	なし	1
<extra>		なし	0 以上

例

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

convex_mesh

概要

基本的なジオメトリックメッシュを記述するのに充分な情報を含むか、または参照します。

コンセプト

<convex_mesh>の定義は<mesh>と同じですが、<source>、<vertices>、<polygons>などのような完全な記述ではなく、別の<geometry>要素を単に指し示すことで形状を構成している点で異なります。別の<geometry>要素を使用する場合には、それらの凸包を計算しなければならず、このような凸包はオプションのconvex_hull_of属性で示されます。
<convex_mesh>を使用すると、レンダリングに利用される<mesh>をフィジックス用途で再利用できるため、ドキュメントのサイズを最小限に抑えたり、オリジナルの<mesh>へのリンクを維持できる、といった利点があります。
<convex_mesh>要素の記述を最小限に抑える方法は、頂点を（<vertices>要素と対応するソースを用いて）指定し、その頂点群（ポイントクラウド）の凸包をインポータに計算させることです。

属性

<convex_mesh>要素には、以下の属性があります。

convex_hull_of	**xs:anyURI**	その凸包を計算するためのジオメトリの URI 文字列。オプション。

関連要素

<convex_mesh>要素は、以下の要素と関連性があります。

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	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。**<source>**, **<vertices>**、**プリミティブ要素**、**<extra>**（ここで、プリミティブ要素は、**<lines>**, **<linestrips>**, **<polygons>**, **<polylist>**, **<triangles>**, **<trifans>**または **<tristrips>**の任意の組み合わせを意味します）。

名前/例	解説	デフォルト値	出現回数
<source>	メッシュの頂点データの集まりを提供します。		1 以上
<vertices>	メッシュの頂点属性を記述して、トポロジー的な位置を確立します。		1
<lines>	直線プリミティブを含みます。		0 以上
<linestrips>	ラインストリッププリミティブを含みます。		0 以上
<polygons>	穴を持つことができるポリゴンプリミティブを含みます。		0 以上
<polylist>	穴を持てないポリゴンプリミティブを含みます。		0 以上
<triangles>	三角形プリミティブを含みます。		0 以上
<trifans>	三角形ファンプリミティブを含みます。		0 以上

名前/例	解説	デフォルト値	出現回数
<tristrips>	三角形ストリッププリミティブを含みます。		0 以上
<extra>			0 以上

備考

例

```
    <geometry id="myConvexMesh">
      <convex_mesh>
        <source>...</source>
        <vertices>...</vertices>
        <polygons>...</polygons>
      </convex_mesh>
    </geometry>
```

or:

```
    <geometry id="myArbitraryMesh">
      <mesh>
 ...
      </mesh>
    </geometry>
    <geometry id="myConvexMesh">
      <convex_mesh convex_hull_of="#myArbitraryMesh"/>
    </geometry>
```

cylinder

概要

ローカルの *y* 軸に対して中央揃えされたシリンダプリミティブを宣言します。

コンセプト

ジオメトリプリミティブ（分析的な形状）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「新しいジオメトリタイプ」を参照してください。

属性

<**cylinder**>要素には属性はありません。

関連要素

<**cylinder**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**shape**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<height>	Y 軸に沿ったシリンダの長さを表す浮動小数点値	なし	1
<radius>	シリンダ（楕円形も可能）の半径を表す 2 つの浮動小数点値	なし	1
<extra>		なし	0 以上

例

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

force_field

概要

force_fieldは力場に使用する汎用的なコンテナを提供します。現時点では、**technique**と**extra**の要素だけを含んでいます。

コンセプト

force_fieldは剛体などの物理オブジェクトに対して影響を及ぼします。**physics_scene**もしくは**physics_model**の下でインスタンス化されます。

属性

<**force_field**>要素には、以下の属性があります。

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

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	**library_force_fields**
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>		なし	0 または 1
<**technique**>		なし	1 以上
<**extra**>		なし	0 以上

備考

現時点では、<**force_field**>用の共通プロファイル/テクニックはありません。<**technique**>要素には、整形式の任意のXMLデータを含めることができます。

例

```
<library_force_fields>
  <force_field>
    <technique profile="SomePhysicsProfile">
```

```
        <program url="#SomeWayToDescribeAForceField">
          <param> ... </param>
          <param> ... </param>
        </program>
      </technique>
    </force_field>
  </library_force_fields>
```

instance_force_field

概要

<force_field>のインスタンス化を宣言します。

コンセプト

属性

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

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
url	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URL。必須。#文字で始まる、相対 URI のフラグメント識別子を利用してローカルなインスタンスを参照します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<instance_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	**instance_physics_model, physics_scene**
子要素	下記サブセクションを参照
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<extra>			0 以上

備考

例

instance_physics_material

概要

<physics_material>のインスタンス化を宣言します。

コンセプト

属性

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

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
url	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URL。必須。#文字で始まる、相対 URI の断片的な識別子を利用してローカルなインスタンスへの参照を表します。 フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<instance_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	**rigid_body**内の**technique_common**、**instance_rigid_body**, **shape** 内の **technique_common**
子要素	下記サブセクションを参照
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<extra>			0 以上

備考

例

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や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

この要素の目的は、あるフィジックスモデルの定義中に別のフィジックスモデルを階層的に埋め込むこと、および完全なフィジックスモデルをフィジックスシーン中でインスタンス化することです。どちらの場合も、含まれている剛体のパラメータと制約をオーバーライドできます。

フィジックスシーン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。**<instance_physics_model>**内のインスタンス化された剛体と制約のみが、シミュレーションを考慮されます。インスタンス化されていない剛体に接続された制約は、シミュレーションによって破棄されるか無視されます。

さらに、インスタンス化されたフィジックスモデルの親属性を指定することもできます。この親は、フィジックスモデルの最初の位置と方向を決めることになります（その剛体に関しても同様です）。親（または祖父など）は一部のアニメーションコントローラのターゲットとなることも可能で、非力学剛体のキーフレーム運動を物理シミュレーションと組み合わせることができます。

属性

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

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。この属性を利用することで、アニメーションターゲットに**<instance_physics_model>**インスタンスの要素を指定できます。オプション。
name	**xs:NCName**	オプション。
url	**xs:anyURI**	どの**<physics_model>**をインスタンス化するかを表します。必須。#文字で始まる、相対URIのフラグメント識別子を利用してローカルなインスタンスを参照します。フラグメント識別子は、インスタンス化すべき要素のURIで構成されたXPointerの短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。
parent	**xs:anyURI**	ビジュアルシーン内のノードの **id** を指し示します。この属性を利用することで、フィジックスモデルを特定の変換ノードの下にインスタンス化することができます。この変換ノードは、フィジックスモデルの最初の位置と方向を指定することになり、アニメートして剛体の運動に影響を与えるようにすることが可能です。オプション。 デフォルトで、フィジックスモデルは、特定の変換ノードではなく、ワールドの下にインスタンス化されます。このパラメータは、現在の**<physics_model>**の親要素が**<physics_scene>**の場合にだけ意味を持ちます。

関連要素

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

出現回数	スキーマ中に定義された要素の数
親要素	**physics_scene, physics_model**
子要素	下記サブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<instance_force_field>	**<force_field>**要素をインスタンス化して、このフィジックスモデルに影響を及ぼします。	なし	0 以上
<instance_rigid_body target="#SomeNode">	**<rigid_body>**要素をインスタンス化して、そのプロパティの一部またはすべてのオーバーライドを可能にします。 target属性は、この剛体インスタンスで上書きされた変換を持つ**<node>**要素を定義します。	なし	0 以上
<instance_rigid_constraint>	**<rigid_constraint>**要素をインスタンス化して、プロパティの一部をオーバーライドします。 **<rigid_constraint>**子要素が、対象としている**<node>**要素を定義するので、この要素にはtarget属性がありません。	なし	0 以上
<extra>		なし	0 以上

instance_physics_scene

概要

<physics_scene>のインスタンス化を宣言します。

コンセプト

属性

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

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
url	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URL。必須。#文字で始まる、相対 URI のフラグメント識別子を利用してローカルなインスタンスを参照します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<instance_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	**scene**
子要素	下のサブセクションを参照
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<extra>			0 以上

備考

例

instance_rigid_body

概要

<instance_physics_model>内の<rigid_body>をインスタンス化します。

コンセプト

剛体とは、<scene>内の<node>の変換を設定するものであり、<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
- 特定のインスタンス内の<rigid_body>のパラメータをオーバーライドするため（オプション）
- <rigid_body>インスタンスの初期状態（線速度と角速度）を指定するため

属性

<instance_rigid_body>要素には、以下の属性があります。

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。この属性を利用することで、アニメーションターゲットに<rigid_body>インスタンスの要素を指定できます。オプション。
name	xs:NCName	オプション。
body	xs:NCName	どの<rigid_body>をインスタンス化するかを表します。必須。
target	**xs:anyURI**	どの<node>がこの<rigid_body>インスタンスの影響を受けるのかを表します。必須。#文字で始まる、相対URIのフラグメント識別子を利用してローカルなインスタンスを参照します。フラグメント識別子は、インスタンス化すべき要素のURIで構成されたXPointerの短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<instance_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	instance_physics_model
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<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。下記サブセクションを参照してください。	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	0 以上
<extra>	**<instance_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	0 以上

instance_rigid_body / technique_common の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<angular_velocity>	**rigid_body**インスタンスの各軸に対する初期角速度を、*x*-*y*-*z* オイラー回転の形式で指定します。大きさは、1 秒あたりの角度を単位とします。「物理的な単位について」を参照してください。	0, 0, 0	0 または 1
<velocity>	**rigid_body**インスタンスの初期線速度を指定します。	0, 0, 0	0 または 1
<dynamic>false</dynamic>	falseの場合、**rigid_body**は動かすことができません。	なし	0 または 1
<mass>0.5</mass>	剛体全体の質量。	密度 X 形状全体の体積から導き出されます。	0 または 1
<mass_frame> **<translate> ...** **</translate>** **<rotate> ...** **</rotate>** **</mass_frame>**	「ルート」図形のローカルの原点に対して相対的な剛体の重心と方向を定義します。 これは、慣性テンソルの非対角成分（*慣性の積*）をすべて 0 にし、対角成分（*慣性モーメント*）だけを保存することができるようにします。	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
<inertia> 1 1 1 </inertia>	**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

名前/例	解説	デフォルト値	出現回数
<physics_material> または **<instance_physics_material>**	**rigid_body**のため **physics_material** を定義または参照します。	なし	0 または 1
<shape>			0 以上

<**instance_rigid_body**>要素とそのテクニックは<**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
<velocity>	**rigid_body**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
<angular_velocity>	**rigid_body**インスタンスの初期状態の角速度を、各軸に対して*x*-*y*-*z*オイラー回転の形式で指定します。大きさは、1 秒あたりの角度を単位とします。「物理的な単位について」を参照してください。	0, 0, 0	0 または 1

例

```
    <physics_scene id="ColladaPhysicsScene">
      <instance_physics_model sid="firstCatapultAndRockInstance"
                              url="#catapultAndRockModel" parent="#catapult1">
  <!—Override attributes of a rigid_body within this physics_model      -->
  <!—and specify the initial velocity of the rigid_body                 -->
        <instance_rigid_body body="./rock/rock" target="#rockNode">
          <technique_common>
            <velocity>0 -1 0</velocity>  <!—optional overrides -->
            <mass>10</mass>                              <!—heavier            -->
          </technique_common>
        </instance_rigid_body>
  <!—This instance only assigns the rigid_body to its node. It does no overriding -->
        <instance_rigid_body body="./catapult/base" target="#baseNode"/>
      </instance_physics_model>
    </physics_scene>
```

library_force_fields

概要

<force_field>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 1 つの方法は、何らかの基準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は、ライブラリとして別々のリソース中に保存しておくことができます。

属性

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

id	**xs:ID**	**<library_force_fields>**要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。
name	**xs:NCName**	要素のテキスト文字列名。オプション。

関連要素

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

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	下記サブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<force_field>			1 以上
<extra>			0 以上

備考

例

以下は、**<library_force_fields>**要素の例です。

```
<library_force_fields>
    <force_field>
      <technique profile="AGEIA"/>
  </force_field>
</library_force_fields>
```

library_physics_materials

概要

<physics_material>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 1 つの方法は、何らかの基準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は、ライブラリとして別々のリソース中に保存しておくことができます。

属性

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

id	**xs:ID**	**<library_physics_materials>**要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。
name	**xs:NCName**	要素のテキスト文字列名。オプション。

関連要素

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

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	下記サブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<physics_material>			1 以上
<extra>			0 以上

備考

例

以下は、**<library_physics_materials>**要素の例です。

```
<library_physics_materials>
  <physics_material id="phymat1">
    ...
  </physics_material>
```

```
  <physics_material id="phymat2">
    ...
  </physics_material>
</library_physics_materials>
```

library_physics_models

概要

<physics_model>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 1 つの方法は、何らかの基準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は、ライブラリとして別々のリソース中に保存しておくことができます。

属性

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

id	**xs:ID**	要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。
name	**xs:NCName**	要素のテキスト文字列名。オプション。

関連要素

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

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	下記サブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<physics_model>			1 以上
<extra>			0 以上

備考

例

以下は、**<library_physics_models>**要素の例です。

```
<library_physics_models>
  <physics_model id="phymod1">
    ...
  </physics_model>

  <physics_model id="phymod2">
```

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

library_physics_scenes

概要

<physics_scene>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 1 つの方法は、何らかの基準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は、ライブラリとして別々のリソース中に保存しておくことができます。

属性

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

id	**xs:ID**	要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。
name	**xs:NCName**	要素のテキスト文字列名。オプション。

関連要素

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

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	下記サブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<physics_scene>			1 以上
<extra>			0 以上

備考

例

以下は、**<library_physics_scenes>**要素の例です。

```
<library_physics_scenes>
  <physics_scene id="physce1">
    ...
  </physics_scene>

  <physics_scene id="physce2">
```

```
    ...
  </physics_scene>

</library_physics_scenes>
```

physics_material

概要

オブジェクトの物理特性を定義します。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**>要素に格納してインスタンス化することができます。

属性

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

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

関連要素

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

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, shape, technique_common**
子要素	下のサブセクションを参照してください
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<asset>		なし	0 または 1
<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 以上
<extra>		なし	0 以上

physics_material / technique_common の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<dynamic_friction> 0.23 </static_friction>	運動摩擦係数	0	0 または 1
<restitution> 0.2 </restitution>	弾力または弾性とも呼ばれる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

<static_friction> 0.23 </static_friction>	静止摩擦係数	0	0 または 1

例

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

physics_model

概要

何度もインスタンス化される剛体と制約の複雑な組み合わせを構築することができます。

コンセプト

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**	この要素のユニークな識別子を含んだテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素の名前を含んだテキスト文字列です。オプション。

関連要素

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

出現回数	スキーマ中に定義された要素の数
親要素	**library_physics_models**
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>		なし	0 または 1
<**rigid_body** sid="...">	<**rigid_body**> 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	0 以上

名前/例	解説	デフォルト値	出現回数
<**rigid_constraint** sid="...">	<**rigid_constraint**> 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	0 以上
<**instance_physics_model** sid="..." url="#...">	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	0 以上
<**extra**>		なし	0 以上

例

```
<library_physics_models>
<!-- 他の Physics_model または physics_scene で再利用したり修正できるように、     -->
<!-  カタパルトの physics_model を定義する。                                      -->
      <physics_model id="catapultModel">
        <!-- 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>
            <instance_physics_material url="#catapultBasePhysicsMaterial"/>
            <shape>
              <instance_geometry url="#catapultBaseConvexMesh"/>

              <!-- カタパルトのモデルへの相対的なベースのローカル位置-->
              <translate> 0 -1 0 </translate>
            </shape>

          </technique_common>
        </rigid_body>

        <!--同じように、カタパルトの上部を定義する。 -->
        <rigid_body sid="top">
          <technique_common>
            <dynamic>true</dynamic>
            <shape>
              <instance_geometry url="#catapultTopConvexMesh"/>
              <translate> 0 3 0 </translate>
            </shape>
          </technique_common>
        </rigid_body>

        <!-- カタパルトの動きを制御する角スプリングを定義する。
             オプションで、剛体の制約を他の physics_model からコピーするために URL を指定する。 --
>
        <rigid_constraint sid="spring_constraint">
        <ref_attachment rigid_body="./base">
          <translate sid="translate">-2. 1. 0</translate>
        </ref_attachment>
        <attachment rigid_body="./top">
          <translate sid="translate">1.23205 -1.86603 0</translate>
          <rotate sid="rotateZ">0 0 1 -30.</rotate>
        </attachment>
        <technique_common>
          <limits>
            <swing_cone_and_twist>
              <min> -180.0 0.0 0.0 </min>
              <max> 180.0 0.0 0.0 </max>
```

```
              </swing_cone_and_twist>
            </limits>
            <spring>
              <angular>
                <stiffness>500</stiffness>
                <damping>0.3</damping>
                <target_value>90</target_value>
              </angular>
            </spring>
          </technique_common>
        </rigid_constraint>
      </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

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**	この要素のユニークな識別子を含んだテキスト文字列。この値は、インスタンス文書中でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素の名前を含んだテキスト文字列です。オプション。

関連要素

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

出現回数	スキーマ中に定義された要素の数
親要素	**library_physics_scenes**
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>		なし	0 または 1
<instance_force_field>	**<force_field>**要素をインスタンス化してフィジックスシーンに影響を与えます。	なし	0 以上

<instance_physics_model>	<physics_model>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	0 以上
<technique_common>	physics_sceneの共通テクニックのパラメータを指定します。下のサブセクションを参照	なし	1
<technique>	カスタムのテクニックを指定します。	なし	0 以上
<extra>		なし	0 以上

physics_scene / technique_common の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<gravity>	シーンの重力場のベクトル表現。この重力場によりもたらされた加速度の大きさと方向を表す 3 つの浮動小数点値の非正規化方向ベクトルとして、与えられます。	なし	0 または 1
<time_step>	フィジックスシーンの、秒単位で測定された積分時間ステップ。この値はエンジン固有のものです。省かれている場合は、フィジックスエンジンのデフォルト値が使用されます。浮動小数点数。	なし	0 または 1

例

```
<library_physics_scenes>
<!--通常の physics scene -->
  <physics_scene id="ColladaPhysicsScene">
     <instance_physics_model sid="firstCatapultAndRockInstance">
                    url="#catapultAndRockModel" parent"#catapult1">
<!-- オーバーライドした physics_model のインスタンス
     現在の変換行列で、ワールド空間中の physics_model の初期位置と方向を決める。 -->
      <instance_rigid_body body="./rock/rock" target="#rockNode">
        <technique_common>
          <velocity>0 -1 0</velocity> <!-- オプションのオーバーライド -->
          <mass>10</mass>                          <!--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>
    <technique_common>
      <gravity>0 -9.8 0</gravity>
      <time_step>3.e-002</time_step>
    </technique_common>
  </physics_scene>
</library_physics_scenes>
<!-- 2 つの物理的にシミュレーションされたカタパルトのアームがインスタンス化されるシーン -->
<library_visual_scenes>
  <visual_scene id="battlefield">
    <node id="catapult1">
      <translate sid="translate">0 -0.9 0</translate>
      <node id="rockNode">
```

```
        <instance_geometry url="#someRockVisualGeometry"/>
      </node>
      <node id="catapultTopNode">
        <instance_geometry url="#someVisualCatapultTopGeometry"/>
      </node>
      <node id="catapultBaseNode">
        <instance_geometry url="#someVisualCatapultBaseGeometry"/>
      </node>
    </node>
<!--  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>
</library_visual_scenes>
<scene>
<!-- visual_scene に physics_scene が適用可能であることを示す。    -->
  <instance_physics_scene url="#ColladaPhysicsScene"/>
  <instance_visual_scene url="#battlefield"/>
</scene>
```

plane

概要

無限プレーンのプリミティブを定義します。

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「新しいジオメトリタイプ」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	**shape**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<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
<extra>		なし	0 以上

例

```
    <plane>
<!--   平面の方程式: Ax + By + Cz + D = 0 -->
<!-- A, B, C, D 係数  (normal & D) -->
      <equation> 0.0   1.0   0.0   0.0 </equation>     :!-- X-Z 平面（グラウンド）  -->
    </plane>
```

rigid_body

概要

形状を崩さないシミュレーションした物体を記述します。それらの物体は、各種の制約（関節やボールジョイントなど）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コレクションで構成されています。各形状は、複雑なコリジョン形状（バウンディング形状）を構築できるように、回転、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**	**<rigid_body>**要素の範囲化された識別子を含むテキスト文字列。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。 〈**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。必須。
name	**xs:NCName**	この要素の名前を含むテキスト文字列。オプション。

関連要素

<**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	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
〈**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。下のサブセクション。	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。下のサブセクションを参照してください。	なし	0 以上
〈**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	0 以上

rigid_body/technique_common の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<dynamic>false</dynamic>	false の場合、**rigid_body**は動かすことができません。	なし	0 または 1
<mass>0.5</mass>	剛体全体の質量。	密度 X 形状全体の体積から導き出されます。	0 または 1
<mass_frame> <translate> ... </translate> <rotate> ... </rotate> </mass_frame>	「ルート」図形のローカルの原点に対して相対的な剛体の重心と方向を定義します。これは、慣性テンソルの非対角成分（慣性の積）をすべて 0 にし、対角成分（慣性モーメント）だけを保存することができます。	identity（重心はローカルの原点に位置し、主軸はローカル軸となります）	0 または 1
<inertia> 1 1 1 </inertia>	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
<**physics_material**>または **<instance_physics_material>**	**rigid_body**のため**physics_material**を定義または参照します	なし	0 または 1
<shape>			1 以上

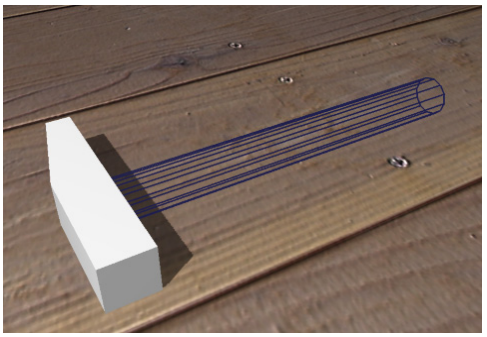
密度、質量、慣性（テンソル）の定義規則

- 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。

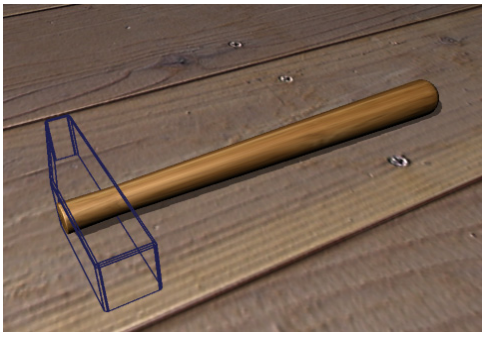
例

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、レンダリングの対象（テクスチャ化され、すぼまった柄の部分と、傾斜された頭の部分）となる点に注意してください。

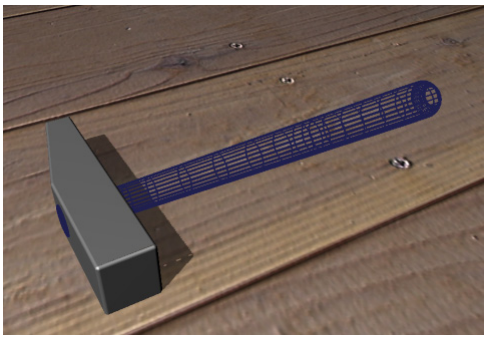
```
<library_geometries>
  <geometry id="hammerHeadForPhysics">

      <mesh>
...
      </mesh>
  <geometry>
```

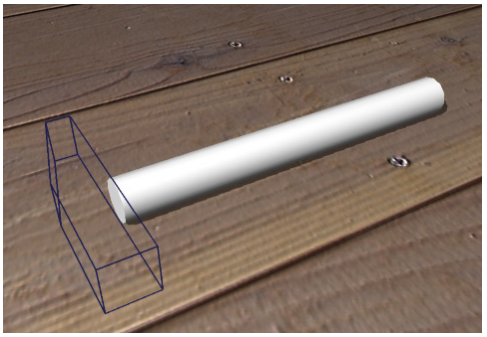


```
<geometry id="hammerHandleToRender">

    <mesh>
...
    </mesh>
  </geometry>
```



```
  <geometry id="hammerHeadToRender">

    <mesh>
...
    </mesh>
  </geometry>
<library_geometries>
```





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

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

        </shape>
        <shape>
          <mass> 1.0 </mass>
          <!- This geometry is referenced
rather than inlined -->
          <instance_physics_material
                 url="#SteelPhysMtl"/>
          <instance_geometry

url="#hammerHeadForPhysics"/>
          <translate> 0.0  4.0  0.0
</translate>
        </shape>
      </technique_common>
    </rigid_body>
  </physics model>
</library_rigid_bodies>
```

rigid_constraint

概要

<rigid_body>のようなコンポーネントを、可動パーツを持った複雑なフィジックスモデルに結び付けます。

コンセプト

一般に、バネやボールジョイントなど、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、ボールジョイント、ヒンジなど）を表現することができます。
制約は、以下のように指定します。

- 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**	<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。この値は、インスタンス文書中でユニークでなければなりません。必須。
name	**xs:NCName**	この要素の名前を含むテキスト文字列。オプション。

関連要素

<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	**physics_model**
子要素	下のサブセクションを参照
その他	なし

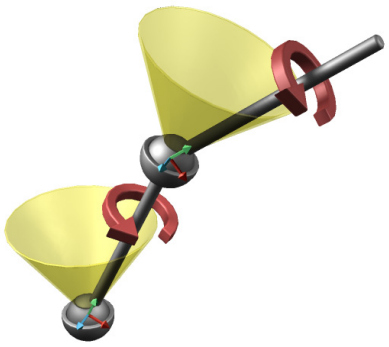
子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<ref_attachment rigid_body="./SomeRigidBody"> **<translate/>** **<rotate/>** **<extra/>** **</ref_attachment>**	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<**physics_model**>中で剛体への相対的な参照です。子要素は、どんな順番でも、何回でも記述できます。あるいは、まったく記述しなくてもかまいません。	なし	1
<attachment rigid_body="./SomeRigidBody"> **<translate/>** **<rotate/>** **<extra/>** **</attachment>**	**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<**physics_model**>中で剛体への相対的な参照です。子要素は、どんな順番でも、何回でも記述できます。あるいは、まったく記述しなくてもかまいません。	なし	1
<**technique_common**>	**rigid_constraint** の対象となる共通プロファイルを指定します。下のサブセクションを参照。	なし	1
<**technique**>	複数の表現を許可するための **rigid_constraint** 用の対象プロファイルを指定します。	なし	0 以上
<**extra**>	ユーザ定義された複数の表現が可能なデータ	なし	0 以上

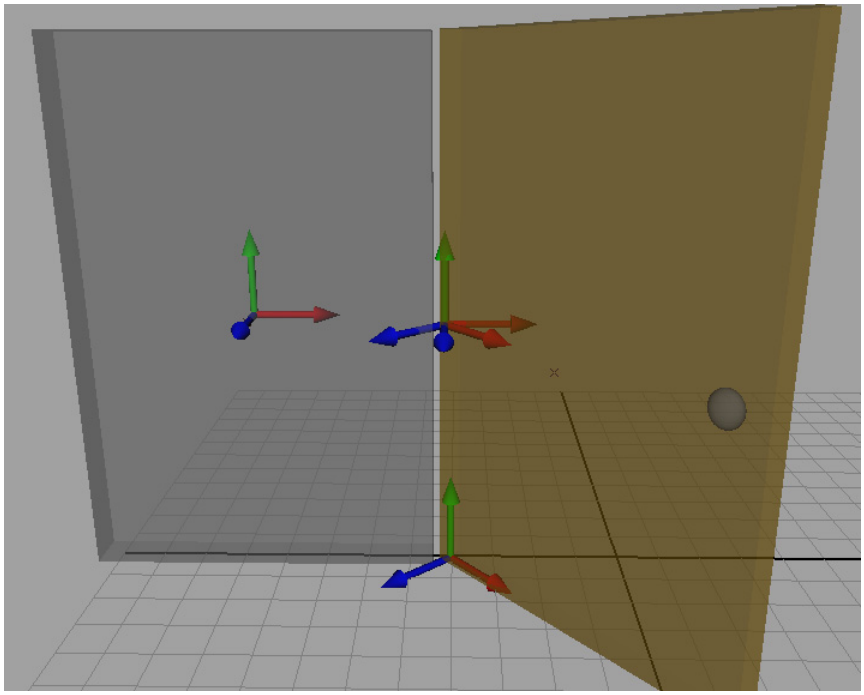
rigid_constraint と technique_common のための子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<enabled>true</enabled>	false の場合、<constraint>は剛体に対して何の影響力も及ぼしません。	true	0 または 1
<interpenetrate>true</interpenetrate>	true の場合、アタッチされた剛体が互いに突き抜ける場合があることを表します。	false	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： **<limits> <linear> <min sid="..."> 0 0 0 </min> <max sid="..."> 0 0 0 </max> </linear> <swing_cone_and_twist> <min sid="..."> -15.0 -15.0 -INF </min> <max sid="..."> 15.0 15.0 INF </max> </swing_cone_and_twist> </limits>** 	<limits>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。 ・<swing_cone_and_twist>子要素と<linear>子要素 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<**technique**>を利用すべきです。 <linear>要素は、それぞれの軸に沿った線形の（平行移動的な）上限を表します。 <swing_cone_and_twist>要素は、それぞれの回転軸に沿った角度を度数で表します。**sid** 属性はオプションです。 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、ref_attachment の空間で表現します。	**linear**: **min**: 0.0 0.0 0 .0 **max**: 0.0 0.0 0.0 **swing_cone_and_twist**: 0.0 0.0 0 .0 **min**: 0.0 0.0 0.0 **max**: 0.0 0.0 0.0 これは、完全に固定された剛体制約に対応します。すなわち、2 つの剛体は互いに相対的に動くことはありません。（回転も変形も許されていません。）	0 または 1

<spring> <linear> <stiffness>5.4544 </stiffness> <damping>0.4132 </damping> <target_value>3 </target_value> </linear> </spring> <spring> <angular> <stiffness>5.4544 </stiffness> <damping>0.4132 </damping> <target_value>90 </target_value> </angular> </spring>	距離（LINEAR）または角度（ANGULAR）を基準にしたスプリング。スプリングは、距離（**LINEAR**）または角度（**ANGULAR**）に基づきますが、3つのオプションの子要素を持つことができます。これらの子要素を使用する場合、子要素は、示した順番に記述しなければなりません。 stiffness（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。	**stiffness**: 1.0 **damping**: 0.0 **target_value**: 0.0（無限剛度の制約、つまりスプリングなし）	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">
          <technique_common>...</technique_common>
        </rigid_body>
        <rigid_body sid="wallRigidBody">
          <technique_common>...</technique_common>
        </rigid_body>

        <rigid_constraint sid="rigidHingeConstraint">
          <ref_attachment rigid_body="#wallRigidBody">
              <translate sid="translate">5 0 0</translate>
          </ref_attachment>
          <attachment rigid_body="#doorRigidBody">
              <translate sid="translate">0 8 0</translate>
              <rotate sid="rotateX">0 1 0 -45.0</rotate>
          </attachment>
<!- sid 属性を追加することで、アニメーションから制限をターゲットにすることが可能となる-->
          <technique_common>
            <limits>
              <swing_cone_and_twist>
                <min sid="swing_min">0  90 0</min>
                <max sid="swing_max">0 -90 0</max>
              </swing_cone_and_twist>
            </limits>
          </technique_common>
        </rigid_constraint>
      </physics_model>
    </library_physics_models>
```

shape

概要

<rigid_body>のコンポーネントを記述します。

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コレクション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、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（<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、サーフェス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。

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、instance_rigid_bodyのtechnique_common
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<hollow sid="...">true</hollow>	true の場合、質量は形状のサーフェスに沿って分布します。 sid はオプションです。	なし	0 または 1
<mass sid="..."> 0.5 </mass>	形状の質量を指定する浮動小数点数。 sid はオプションです。	密度 X 形状の体積から導き出されます	0 または 1

名前/例	解説	デフォルト値	出現回数
<density sid="..."> 0.5 **</density>** sid はオプションです。	形状の密度を指定する浮動小数点数	質量/形状の体積から導き出されます	0 または 1
インライン定義またはインスタンス： **<physics_material>** または **<instance_physics_material>**	この形状に使用される **<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
ジオメトリのインライン定義またはインスタンス： **<box>...** **</box>** または **<instance_geometry url=""/>**	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。 **<plane>**, **<box>**, **<sphere>**, **<cylinder>**, **<tapered_cylinder>**, **<capsule>**、および **<tapered_capsule>**はインライン化され、他のジオメトリタイプ（**<mesh>**, **<convex_mesh>**, **<spline>**など）は、**<instance_geometry>**要素を介して参照されます。	なし	1
<rotate>、<translate>	<node>の下と同じ	変換は行われません。	0 以上
<extra>		なし	0 以上

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 sid="HammerHandleRigidBody">
    <technique_common>
      <shape>
        <mass> 0.25 </mass>
        <instance_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を記述します。

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「新しいジオメトリタイプ」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	**shape**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<radius>	球体の半径を表す浮動小数点	なし	1
<extra>		なし	0 以上

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、すぼめられたカプセルプリミティブを記述します。

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「新しいジオメトリタイプ」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	**shape**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<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 値ですぼめられたカプセルの半径を表す 2 つの浮動小数点。すぼめら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 値ですぼめられたカプセルの半径を表す 2 つの浮動小数点。すぼめられたカプセルの両端は楕円となります。	なし	1
<extra>		なし	0 以上

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、すぼめられたシリンダプリミティブを記述します。

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「新しいジオメトリタイプ」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	**shape**
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<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 値ですぼめられたシリンダの半径を表す 2 つの浮動小数点。すぼめら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 値ですぼめられたシリンダの半径を表す 2 つの浮動小数点。すぼめられたシリンダの両端は楕円となります。	なし	1
<extra>		なし	0 以上

例

```
<tapered_cylinder>
  <height> 2.0 </height>
  <radius1> 1.0 2.0 </radius1>
  <radius2> 1.5 1.8 </radius2>
</tapered_cylinder>
```

このページは空白です。

6章　COLLADA FX リファレンス

概要

COLLADA FX は、カラーをビジュアルシーンに適用する方法を作成者が記述することを可能にします。これは、多くのプラットフォームとアプリケーションプログラミングインタフェース（API）すべてにわたって、マテリアルプロパティを記述するための柔軟な概念です。

COLLADA スキーマの FX 要素で記述できるものには、以下のものがあります。

- 単一パスおよび複数パスの効果。抽象マテリアルの定義です（例：プラスチック）
- 効果パラメタリゼーション（**<newparam>**を使用）
- 効果メタデータ
- シーングラフとのバインディング
- 複数のテクニック
- インラインおよび外部のソースコードまたはバイナリ

複数のアプリケーションプログラミングインタフェース（API）は、各効果を複数のプラットフォーム用に記述することを可能にする、**<profile_*>**要素を通してサポートされています。各プラットフォームを、プラットフォーム固有のデータ型、レンダリングステート、および機能を用いて完全に記述できます。

各プラットフォームでは、各効果を多くのテクニックを用いて記述できます。*テクニック*とは、レンダリングのスタイル（例：daytime、nighttime、magic、superhero_mode）、別の詳細レベル、または計算方法（例：「approximate」、「accurate」、「high_LOD」、「low_LOD」）の、ユーザーがラベル付けした（例で示したような）表記のことです。

より高いレベルで、マテリアルシステムは、効果定義のデフォルト値以外の値をパラメータに提供することで、あらかじめ定義された効果を特定のインスタンスに特化できるようにします。このことにより、単一の効果を多くのさまざまなマテリアルの基礎として使用することが可能になります。

最後に、マテリアルは、シーングラフの 1 つ以上のポイントにバインドされるとき、使用するために挿入されます。シーングラフのジオメトリ内の**<bind_material>**要素は、1 つ以上のマテリアルをインスタンス化し、それらをジオメトリのセグメントに接続することができます。マテリアルインスタンス内で、シーングラフのリソース（光源やカメラなど）とバインドしたり、テクスチャ座標をペアにしたりすることで、効果をさらに特化させることができます。

FX 要素の構成

各 FX 要素の特定の属性および子要素は、各 FX 要素が使用されるスコープに応じて変化することがあります。プロファイルスコープは、以下のようにグループ化できます。

- 共有されている（要素はすべてのスコープで有効）
- FX の外部
- 共通プロファイル
- Cg プロファイル
- Open GL ES プロファイル（GLES に短縮）
- GLSL プロファイル

以下のリストは、どの要素がどのグループに入るかを示したものです。アスタリスク（*）は、複数のグループに入り、コンテキストに応じて異なる属性およびコンテンツを持つ可能性のある要素を、示しています。

共有されている

- **<annotate>**
- **<bind> ***
- **<bind>/<param> ***
- **<code>**
- **<color_clear>**
- **<color_target>**
- **<compiler_options>**
- **<compiler_target>**
- **<depth_clear>**
- **<depth_target>**
- **<draw>**
- **<effect>**
- **<generator>**
- **<generator>/<name> ***
- **<include>**
- **<modifier>**
- **<newparam>**
- **<newparam>/<semantic>**
- **<pass>** *
- レンダリングステート
- **<sampler1D>**
- **<sampler2D>**
- **<sampler3D>**
- **<samplerRECT>**
- **<samplerCUBE>**
- **<setparam>**
- **<shader>**
- **<shader>/<name> ***
- **<stencil_clear>**
- **<stencil_target>**
- **<surface>**
- **<technique> ***
- 値のタイプ

FX の外部

- **<bind>** *
- **<bind_material>**

- **<instance_effect>**
- **<instance_material>**
- **<param>** *
- **<technique_hint>**

共通プロファイル

- **<blinn>**
- **<color>**（「**common_color_or_texture_type**」を参照）
- **<param>**（「**common_float_or_param_type**」を参照）
- **<lambert>**
- **<phong>**
- **<profile_COMMON>**
- **<texture>**　（「**common_color_or_texture_type**」を参照）
- **<technique>** *

プロファイル Cg

- **<array>**
- **<connect_param>**
- **<pass>** *
- **<profile_CG>**
- **<surface>** *… **<generator>**もサポートすることに注意
- **<technique>** *… **<code>**および**<include>**もサポートすることに注意
- **<usertype>**
- **<technique>** *

プロファイル GLSL

- **<pass>** *
- **<profile_GLSL>**
- **<surface>** *… **<generator>**もサポートすることに注意
- **<technique>** *

プロファイル OpenGL ES

- **<alpha>**
- **<argument>**
- **<constant>**
- **<newparam>** *…　**texture_pipeline**、**texture_unit**、および**sampler_state**もサポートすることに注意
- **<pass>** *
- **<profile_GLES>**
- **<RGB>**
- **<sampler_state>**

- **<setparam>*…** **texture_pipeline**, **texture_unit**、および**sampler_state**もサポートすることに注意
- **<technique> ***
- **<texcombiner>**
- **<texenv>**
- **<texture_pipeline>**
- **<texture_unit>**

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、合成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。オプション。
scale	float	glTexEnv(TEXTURE_ENV, ALPHA_SCALE, *scale*)での利用を想定。オプション。

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	**texcombiner**
子要素	下のサブセクションを参照してください
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<argument>	実行する特定の操作に必要な引数を設定します		1 から 3

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を表し、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	*SYMBOL = VALUE* 形式のオブジェクト内の *SYMBOL* を表すこの要素の、テキスト文字列名。必須。

関連要素

<**annotate**>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**effect**、**technique**、**pass**、**newparam**、**setparam**、**generator**、**shader**
子要素	下のサブセクションを参照してください
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
VALUE_TYPES	*SYMBOL = VALUE* 形式のオブジェクト内の *VALUE* を表す、強く型付けされた値。有効な種類は、次のとおりです。 **bool、bool2、bool3、bool4、int、int2、int3、int4、float、float2、float3、float4、float2x2、float3x3、float4x4、string**	なし	1

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>
```

argument

概要

<argument>要素は、テクスチャユニットの合成スタイルのテクスチャコマンドが持つRGBまたはアルファコンポーネントの引数を定義するためのものです。

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。

注意：以下の表で、##は*結合*を意味し、*idx*は、引数が親コマンド（<texenv>または<texcombiner>）内に記述されたインデックスを表します。*source*は、値を指定する場所を意味します。

source	列挙	オプション。引数のソースデータがどこ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。 **TEXTURE	CONSTANT	PRIMARY	PREVIOUS** を指定できます。デフォルト値はありません。
operand	列挙	オプション。ソースから値を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、 **glTexEnv(TEXTURE_ENV, OPERAND##*idx*##_RGB, *source*)**の呼び出しを暗に意味し、 次の値を指定できます。 **SRC_ALPHA	ONE_MINUS_SRC_ALPHA**；デフォルト値は、**SRC_ALPHA** です。 親が<alpha>の場合、 **glTexEnv(TEXTURE_ENV, OPERAND##*idx*##_ALPHA, *source*)** の呼び出しを暗に意味し、指定できる値は **SRC_ALPHA	ONE_MINUS_SRC_ALPHA**；**SRC_ALPHA**で、デフォルトは、 **SRC_ALPHA**です。
unit	**xs:NCName**	オプション。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。***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。合成クロスバーがないからです。 GLES 1.1 の場合、テクスチャの合成クロスバーが利用可能であるため、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	なし
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。

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**	配列中の要素の数。**<newparam>**では必須、**<setparam>**ではオプション。

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	**newparam**、**setparam**
子要素	下のサブセクションを参照してください
その他	なし

CG スコープ内の子要素。

子要素を任意の順で記述できます。

名前/例	解説	デフォルト値	出現回数
VALUE_TYPE	本章の最後にある「値のタイプ」の「CG スコープ値のタイプ」を参照してください。		0 以上
<connect_param>	**<newparam>**でのみ有効。		0 以上
<usertype>			0 以上
<array>			0 以上

GLSL スコープ内の子要素

子要素を任意の順で記述できます。

名前/例	解説	デフォルト値	出現回数
VALUE_TYPE	本章の最後にある「値のタイプ」の「GLSL スコープ値のタイプ」を参照してください。		0 以上
<array>			0 以上

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>
```

bind

概要

値をシェーダへのユニフォーム入力にバインドします。もしくは、インスタンス化時に効果パラメータに値をバインドします。

コンセプト

ユニフォームパラメータを持ったシェーダは、コンパイル時に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、ユニフォームパラメータのいずれでもかまいません。定数値の場合、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	シェーダへのユニフォーム入力パラメータのための識別子（正式な関数パラメータまたは特定の範囲でのグローバルな識別子）で、外部リソースにバインドされます。<**bind**>が<**shader**>の子の場合にだけ有効で必須です。
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必須です。

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	下のサブセクションを参照してください。**CG_PARAM_TYPE, GLSL_PARAM_TYPE, param**
その他	なし

子要素

注意：子要素**<param>**または値タイプのいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<param>	**<bind>**が**<shader>**の子であるときにのみ、有効。		注意を参照
***CG_PARAM_TYPE* または**	**<bind>**が**<shader>**の子であるときに		注意を参照

GLSL_PARAM_TYPE	のみ、有効。この章の最後の「値のタイプ」を参照してください。		

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ユニフォーム入力をバインドしておく必要があります。それ以外の FX ランタイムでは、バインドされていない入力をチェックできるように、シェーダを非実行可能形式のオブジェクトコードに部分コンパイルすることができます。

例

```
<shader stage="VERTEXPROGRAM">
  <name source="fooShader-code-1">main</name>
  <bind symbol="diffusecol">
    <float3> 0.30 .52 0.05 </float3>
  </bind>
  <bind symbol="lightpos">
    <param ref="OverheadLightPos_03">
  </bind>
</shader>
  ...
<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ユニフォームパラメータをバインドします。

コンセプト

ジオメトリの一部を宣言する際に、以下のように特定のマテリアルを持つ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？　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。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	**instance_geometry**、**instance_controller**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<param>	**<bind_material>**で、これらは追加されてアニメーションのターゲットになります。その後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。		0 以上
<technique_common>	下記サブセクションを参照してください。		1
<technique> (FX)			0 以上
<extra>		なし	0 以上

bind_material / technique_common の子要素

名前/例	解説	デフォルト値	出現回数
<instance_material>			1 以上

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=" RedMat "target="`">
        <bind semantic="LIGHTPOS0" target="LightNode/translate"/>
        <bind semantic="TEXCOORD0" target="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サーフェスを生成する固定機能パイプラインを宣言します。その際、鏡面反射は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、H=(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	**technique**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<emission>	このオブジェクトのサーフェスから放射される光源の量を宣言します。**common_color_or_texture_type**を参照してください	なし	0 または 1
<ambient>	このオブジェクトのサーフェスから放射される環境光の量を宣言します。**common_color_or_texture_type**を参照してください	なし	0 または 1
<diffuse>	このオブジェクトのサーフェスから反射される拡散光の量を宣言します。**common_color_or_texture_type**を参照してください	なし	0 または 1
<specular>	このオブジェクトのサーフェスから反射	なし	0 または 1

名前/例	解説	デフォルト値	出現回数
	される鏡面反射光の量を宣言します。**common_color_or_texture_type**を参照してください		
<shininess>	このオブジェクトのサーフェスから鏡面反射突起部分の鏡面度または荒さを宣言します。**common_float_or_param_type**を参照してください	なし	0 または 1
<reflective>	完全鏡面反射のカラーを宣言します。**common_color_or_texture_type**を参照してください	なし	0 または 1
<reflectivity>	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。**common_float_or_param_type**を参照してください	なし	0 または 1
<transparent>	完全鏡面反射のカラーを宣言します。**common_color_or_texture_type**を参照してください	なし	0 または 1
<transparency>	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。**common_float_or_param_type**を参照してください	なし	0 または 1
<index_of_refraction>	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。**common_float_or_param_type** を参照してください	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	**technique**、**generator**、**profile_CG**、**profile_GLSL**
子要素	なし
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する必要があります。たとえば、"<"は"<"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サーフェスをクリアするかどうか、クリアする場合にはどんな値を利用するのかを指定します。

コンセプト

描画を行う前に、レンダリングターゲットのサーフェスを空のキャンバス、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、ターゲットサーフェスは、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**	複数のレンダリング対象のうちどれが設定されているか。デフォルト値は 0 です。 GLSL と CG スコープではオプション; GLES スコープでは有効ではありません。

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	**pass**
子要素	なし
その他	なし

備考

現時点のプラットフォームでは、MRT（マルチレンダリングターゲット）を設定するルールがかなり制限されています。たとえば、MRTは、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 1 つのステンシルバッファだけがアクティブとなります。COLLADA FXの宣言は、この制限を緩めるように設計されており、FX ランタイムは、実際に適用する前に<**pass**>の中に指定されている特定のMRT宣言が可能であることを検証しなければならず、検証に失敗した場合にはエラーを返す必要があります。

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。デフォルト値は 0 です。 GLES スコープでは有効でありません。それ以外ではオプションです。
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。デフォルト値は 0 です。 GLESスコープでは有効ではありません。それ以外ではオプション。
mip	**xs:nonNegativeInteger**	デフォルト値は 0 です。 オプション。
face	**enumeration**	有効な値は、POSITIVE_X、NEGATIVE_X、POSITIVE_Y、NEGATIVE_Y、POSITIVE_Z、および NEGATIVE_Z です。デフォルト値は、POSITIVE_X です。オプション。

関連要素

<**color_target**>要素は、以下の要素と関連性があります。

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	なし
その他	なし

備考

現在のプラットフォームでは、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>効果内の固定機能シェーダ要素のカラー属性を記述するタイプ。

コンセプト

属性

common_color_or_texture_typeの要素は属性を持ちません。

関連要素

common_color_or_texture_typeの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	**constant, lambert, phong, blinn**
子要素	下記サブセクションを参照
その他	**common_color_or_texture_type**タイプの要素：ambient、diffuse、emission、reflective、specular、transparent（各親要素中にある）

子要素

注意：子要素**<color>**、**<param>**、または**<texture>**のいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<color sid="mySID">	値はリテラルのカラーであり、RGBA の順番に並んだ 4 つの浮動小数点数で指定します。	なし	注意を参照
<param ref="myParam">	値は、**<float4>**に直接キャスト可能な、現在のスコープ内で事前に定義されたパラメータを参照して指定します。	なし	注意を参照
<texture texture="myParam" texcoord="myUVs"> <extra…/> </texture>	値は、事前に定義された**<sampler2D>**オブジェクトを参照して指定します。texcoord属性は、セマンティックトークンを提供します。このトークンは、**<geometry>**インスタンスからのtexcoordの配列を**<texture_unit>**にバインドするために、**<bind_material>**内で参照されることになります。	なし	注意を参照

備考

例

common_float_or_param_type

概要

<profile_COMMON>効果内の固定機能シェーダ要素のスカラー属性を記述するタイプ。

コンセプト

属性

common_float_or_param_typeのタイプの要素は属性を持ちません。

関連要素

common_float_or_param_typeのタイプの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	**constant, lambert, phong, blinn**
子要素	下記サブセクションを参照
その他	**common_float_or_param_type**タイプの要素：**index_of_refraction**、**reflectivity**、**shininess**、**transparency**（各親要素中にある）

子要素

注意：子要素 **<float>**または**<param>**のいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<float sid="mySID">	値は、リテラルの浮動小数点スカラーで指定します。例：**<float> 3.14 </float>**	なし	注意を参照
<param ref="myParam">	値は、浮動小数点スカラーに直接キャスト可能な事前に定義されたパラメータを参照して指定します。	なし	注意を参照

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みます。

コンセプト

シェーダコンパイラは、シェーダプログラムソースコードと入力を受け入れ、それらをコンパイルして、マシンによる実行が可能なオブジェクトコードにする外部ツールです。シェーダコンパイラは、特定の操作を実行するようにそのオブジェクトコードを設定するコマンドラインオプションを、受け入れます。

属性

<**compiler_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	**shader**
子要素	なし
その他	なし

備考

<compiler_options>値テキストは、テキスト文字列としてツールに与えられたコマンドラインオプションです。

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ます。

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文字列として指定します。

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	**shader**
子要素	なし
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、ユニフォーム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:token**	現在のパラメータに結びつけるターゲットパラメータへの参照。必須。

関連要素

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

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	なし
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サーフェスを常に生成する固定機能パイプラインを宣言します。

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	**technique**
子要素	下のサブセクションを参照してください
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<emission>	このオブジェクトのサーフェスから放射される光源の量を宣言します。**common_color_or_texture_type**を参照してください。	なし	0 または 1
<reflective>	完全鏡面反射のカラーを宣言します。**common_color_or_texture_type**を参照してください。	なし	0 または 1
<reflectivity>	反射光源に追加する完全鏡面反射の量（0.0～1.0）を宣言します。**common_float_or_param_type**を参照してください。	なし	0 または 1
<transparent>	完全鏡面反射光源のカラーを宣言します。**common_color_or_texture_type**を参照してください。	なし	0 または 1
<transparency>	反射カラーに追加する完全鏡面反射光源の量（0.0～1.0）をスカラー値として宣言します。**common_float_or_param_type**を参照してください。	なし	0 または 1

名前/例	解説	デフォルト値	出現回数
<index_of_refraction>	単一スカラーインデックスとして完全鏡面反射光源の反射インデックスを宣言します。 **common_float_or_param_type**を参照してください。	なし	0 または 1

備考

例

depth_clear

概要

レンダリングターゲットのサーフェス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サーフェスをブランクのキャンバスまたはデフォルト値にリセットしなければならない場合もあります。これらの<**depth_clear**>宣言は、リセットする際にどんな値を利用するのかを指定します。クリアする指示が含まれていない場合には、ターゲットのサーフェスはレンダリングが始まっても変更されません。

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。デフォルト値は 0 です。GLSL および CG スコープではオプションであり、GLES スコープでは有効ではありません。

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	**pass**
子要素	なし
その他	なし

備考

現在のプラットフォームでは、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。

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。デフォルト値は 0 です。GLSL および CG スコープではオプションであり、GLES スコープでは有効ではありません。
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。デフォルト値は 0 です。GLSLおよびCGスコープではオプションであり、GLESスコープでは有効ではありません。
mip	xs:nonNegativeInteger	デフォルト値は 0 です。オプション。
face	enumeration	有効な値は、POSITIVE_X、NEGATIVE_X、POSITIVE_Y、NEGATIVE_Y、POSITIVE_Z、および NEGATIVE_Z です。デフォルト値は、POSITIVE_X です。オプション。

関連要素

<**depth_target**>要素は、以下の要素と関連性があります。

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	なし
その他	なし

備考

現在のプラットフォームでは、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>
  <depth_target>depthSurface</depth_target>
  <depth_clear>0.0</depth_clear>
</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ず、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	**pass**
子要素	なし
その他	なし

備考

以下のリストには、**<draw>**で使用する共通の文字列が含まれています。ただし、使用する文字列は、これらの文字列のみに限定されているわけではありません。

- GEOMETRY：この**<instance_geometry>**または**<instance_material>**に関連付けられているジオメトリ。
- SCENE_GEOMETRY:この効果を使って、このジオメトリに既に関連付けられている効果やマテリアルにではなく、シーンのジオメトリ全体を描画します。これは、シャドウバッファ生成などのテクニックのためのものです。これにより、Z 値を光源から抽出することにのみ集中できます。これは、ZBuffer が順番を処理するという前提のもとでの順序付けとは関係なく行われます。
- SCENE_IMAGE: シーン全体を自分のターゲットに描画します。各オブジェクトにとって適切な効果やマテリアルを使用します。これは、後処理のぼかしなどの効果のために、作業するシーンの正確な画像を必要とする効果のためのものです。これは、ZBuffer が順番を処理するという前提のもとでの順序付けとは関係なく行われます。
- FULL_SCREEN_QUAD：位置は 0,0～1,1 であり、UV 同士は一致しています。
- FULL_SCREEN_QUAD_PLUS_HALF_PIXEL: 位置は 0,0～1,1 であり、UV はピクセルの UV サイズの 1/2 だけ正にずれています。

例

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、ユニフォームパラメータ、レンダリングステートの設定を記述する必要があります。
さらに、一部のアルゴリズムは、効果をレンダリングするために複数のパスを必要とします。これは、パイプラインの記述を順番付けされた**<pass>**オブジェクトのコレクションに分割することでサポートされています。これらは、効果を生成するための複数の方法の 1 つを記述した**<technique>**にグループ化されます。
<effect>宣言の中の要素は、シェーダの作成/利用/管理、ソースコード、パラメータなどを処理する基盤となるライブラリコードを利用するものと想定しています。この基盤となるライブラリは「FXランタイム」と呼ばれています。
<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。必須。
name	xs:NCName	効果の名称。オプション。

関連要素

<effect>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**library_effects**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

注意：少なくとも 1 つのプロファイルを記述しなければなりませんが、どんなタイプのプロファイルも、いくつでも含めることができます。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<annotate>			0 以上

名前/例	解説	デフォルト値	出現回数
<image>			0 以上
<newparam>			0 以上
<profile_CG>			注意を参照
<profile_GLSL>			注意を参照
<profile_COMMON>			注意を参照
<extra>		なし	0 以上

備考

例

generator

概要

手続き型のサーフェスジェネレータを記述します。

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	**surface**（CGとGLSLの範囲のみ）
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

注意：少なくとも、**<code>**または**<include>**のいずれか 1 つは記述されなければなりません。

名前/例	解説	デフォルト値	出現回数
<annotate>			0 以上
<code>			注意を参照
<include>			注意を参照
<name>			1
<setparam>			0 以上

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。必須。
url	xs:anyURI	リソースがある場所。必須。

関連要素

<include>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**technique**、**generator**、**profile_CG**、**profile_GLSL**
子要素	なし
その他	なし

備考

例

```
<include sid="ShinyShader" url="file://assets/source/shader.glsl"/>
```

instance_effect

概要

COLLADA マテリアルリソースのインスタンス化を宣言します。

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_effect**>要素には、以下の属性があります。

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
url	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URI。必須。#文字で始まる、相対 URI のフラグメント識別子を利用してローカルなインスタンスを参照します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。

関連要素

<**instance_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	**material**、render
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<technique_hint>			0 以上

<setparam>			0 以上
<extra>			0 以上

備考

例

```
<material id="BlueCarPaint" name="Light blue car paint">
  <instance_effect url="CarPaint">
    <technique_hint profile="CG" platform="PS3"
   ref="precalc_texture"/>
    <setparam ref="diffuse_color">
      <float3> 0.3 0.25 0.85 </float3>
      </setparam>
  </instance_effect>
</material>
```

instance_material

概要

COLLADA マテリアルリソースのインスタンス化を宣言します。

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_material**>要素には、以下の属性があります。

sid	**xs:NCName**	この要素のサブ識別子を含んでいるテキスト文字列値。この値は、親要素のスコープ内でユニークでなければなりません。オプション。
name	**xs:NCName**	この要素のテキスト文字列名。オプション。
target	**xs:anyURI**	インスタンス化すべきオブジェクトの場所の URI。必須。#文字で始まる、相対 URI のフラグメント識別子を利用してローカルなインスタンスを参照します。フラグメント識別子は、インスタンス化すべき要素の URI で構成された XPointer の短縮ポインタです。 絶対 URL もしくは相対 URL に他のリソースへのパスが含まれる場合は、その絶対 URL もしくは相対 URL を使用して外部参照を参照します。
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。必須。

関連要素

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

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	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<bind>			0 以上
<bind_vertex_input>	**<bind_vertex_input>**要素は、インスタンス化時に頂点入力を効果パラメータにバインドします。 これは、たとえば頂点プログラムパラメータを**<source>**にバインドする際に役に立ちます。頂点プログラムは、既にソースから収集されたデータを必要とします。このデータは、**<polygons>**や**<triangles>**などの照合要素の下の **<input>**からもたらされます。入力は、**<source>**中のデータにアクセスし、そのデータがポリゴン頂点"fetch"に対応していることを保証します。バインドするために**<input>**を参照するには、**<bind_vertex_attribute>**を使用してください。 詳しくは、以下のサブセクションを参照してください。		0 以上
<extra>			0 以上

instance_material/bind_vertex_input の属性

<bind_vertex_input>には以下の属性があります。

semantic	**xs:NCName**	どの効果パラメータをバインドするかを指定します。必須。
input_semantic	**xs:NCName**	どの入力セマンティックをバインドするかを指定します。必須。
input_set	**uint**	どの入力セットをバインドするかを指定します。オプション。

セマンティックで場所を特定するときには、以下の順に検索します。

- COLLADA FX パラメータ
- シェーダがサポートしている場合は、セマンティックを用いて直接シェーダ内で
- セマンティックがサポートされていない場合は、パラメータ名を用いて直接シェーダ内で

備考

<bind_vertex_input>要素は、（**<geometry>**要素内の**<input>**要素として識別された）ジオメトリ頂点ストリームをマテリアル効果頂点ストリームセマンティックにバインドします。アプリケーションは通常、同一のセマンティック識別子を用いて頂点ストリームの自動バインディングを実行しますが、セマンティック識別子の意味において食い違いが頻繁に起こります。**<bind_vertex_input>**を使用して、通常以下のようなことによってもたらされるこれらの曖昧さを取り除きます。

- 一般化。例えば、TEXCOORD0 対 DIFFUSE-TEXCOORD
- スペリングの違い。例えば、COLOR 対 COLOUR
- 省略形
- 冗長
- 同義語

例

```
<instance_geometry url="#BeechTree">
  <bind_material>
    <param sid="windAmount" semantic="WINDSPEED" type="float3"/>
    <technique_common>
      <instance_material symbol="leaf" target="#MidsummerLeaf01"/>
      <instance_material symbol="bark" target="#MidsummerBark03">
        <bind semantic="LIGHTPOS1" target="/scene/light01/pos"/>
        <bind semantic="TEXCOORD0" target="BeechTree/texcoord2"/>
      </instance_material>
    </technique_common>
  </bind_material>
</instance_geometry>
```

下の例では、濡れた羽毛マテリアルをカモモデルに適用します。カモモデルは、通常のマップテクスチャ座標およびベースカラーテクスチャ座標を持つことができます。通常のマップテクスチャ座標は TEXCOORD0（semantic=TEXCOORD および set=0）と呼ばれ、、ベースカラーテクスチャ座標は TEXCOORD1 と呼ばれます。

テクスチャ座標（または他のジオメトリストリーム）のセマンティック名がマッチしない環境があります。例えば、濡れた羽毛マテリアルは、TEXCOORD1 と呼ばれる通常のマップテクスチャ座標および TEXCOORD0 と呼ばれるベースカラーテクスチャ座標を持つかもしれません。この場合、これら同一名称の意味がそのようにスワップされているので、これらの一致しないオブジェクトをバインドするには、**<bind_vertex_input>**を用います。

セマンティック属性はマテリアル効果内のセメンティックを参照し、一方接頭辞**input_**を持つ属性はジオメトリ頂点**<input>**ストリームを参照し、そのストリームはセマンティック名とセット番号を組み合わせたもので識別されることに、注意してください。

```
<instance_geometry url="#duck">
  <bind_material>
    <technique_common>
      <instance_material symbol="region1" target="#wet-feathers">
          <bind_vertex_input semantic="TEXCOORD1"
                 input_semantic="TEXCOORD" input_set="0"/>
          <bind_vertex_input semantic="TEXCOORD0"
                 input_semantic="TEXCOORD" input_set="1"/>
      </instance_material>
    </technique_common>
  </bind_material>
</instance_geometry>
```

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サーフェスを常に生成する固定機能パイプラインを宣言します。

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	**technique**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<emission>	このオブジェクトのサーフェスから放射される光源の量を宣言します。**common_color_or_texture_type**を参照。	なし	0 または 1
<ambient>	このオブジェクトのサーフェスから放射される環境光の量を宣言します。**common_color_or_texture_type**を参照 。		0 または 1
<diffuse>	このオブジェクトのサーフェスから反射される拡散光の量を宣言します。**common_color_or_texture_type**を参照。		0 または 1
<reflective>	完全鏡面反射のカラーを宣言します。**common_color_or_texture_type**を参照	なし	0 または 1
<reflectivity>	反射光源に追加する完全鏡面反射の量（0.0～1.0）を宣言します。**common_float_or_param_type**を参照。	なし	0 または 1
<transparent>	完全鏡面反射光源のカラーを宣言します。**common_color_or_texture_type**を参照。	なし	0 または 1

<transparency>	反射カラーに追加する完全鏡面反射光源の量（0.0～1.0）をスカラー値として宣言します。**common_float_or_param_type**を参照。	なし	0 または 1
<index_of_refraction>	単一スカラーインデックスとして完全鏡面反射光源の反射インデックスを宣言します。**common_float_or_param_type**を参照。	なし	0 または 1

備考

例

library_effects

概要

<effect>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 1 つの方法は、何らかの基準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は、ライブラリとして別々のリソース中に保存しておくことができます。

属性

<library_effects>要素には、以下の属性があります。

id	**xs:ID**	**<library_effects>**要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。
name	**xs:NCName**	要素のテキスト文字列名。オプション。

関連要素

<library_effects>要素は、以下の要素と関連性があります。

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	下記サブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<effect>			1 以上
<extra>			0 以上

備考

例

以下は、**<library_effects>**要素の例です。

```
<library_effects>
  <effect id="fullscreen_effect1">
     ...
```

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

library_materials

概要

<material>要素のモジュールを宣言します。

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伴い、単一のコンテナ中で操作するのが難しくなります。そういった複雑さに対処するための 1 つの方法は、何らかの基準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は、ライブラリとして別々のリソース中に保存しておくことができます。

属性

<library_materials>要素には、以下の属性があります。

id	**xs:ID**	要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。
name	**xs:NCName**	要素のテキスト文字列名。オプション。

関連要素

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

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	下記サブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<material>			1 以上
<extra>			0 以上

備考

例

以下は、**<library_materials>**要素の例です。

```
<library_materials>
  <material id="mat1">
    ...
  </material >
```

```
  <material id="mat2">
    ...
  </material>

</library_materials>
```

material

概要

ジオメトリオブジェクトの視覚的な外観を記述します。

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（素材）の特性を記述する各種のパラメータを持つことができます。このマテリアルの特性がレンダリング計算時のパラメータとして使用され、最終出力においてオブジェクトの外観を決定します。

実際のマテリアルのパラメータは、利用するグラフィックスレンダリングシステムに依存します。固定機能グラフィックスパイプラインでは、Phong シェーディングのような定義済みの照明モデルを解くためのパラメータが必要となります。これらのパラメータには、環境、拡散、鏡面の各反射率などが含まれます。

これに対して、プログラマブルグラフィックスパイプラインでは、マテリアルのパラメータセットをプログラマが定義します。これらの定義されたパラメータが、頂点プログラムやピクセルプログラムで定義されているレンダリングアルゴリズムで使用されます。

属性

<material>要素には、以下の属性があります。

id	**xs:ID**	要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。
name	**xs:NCName**	要素のテキスト文字列名。オプション。

関連要素

<material> 要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**library_materials**
子要素	下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<instance_effect>			1
<extra>			0 以上

備考

例

以下は、簡単な**<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_materials>
  <material id=" Blue">
    <instance_effect url="#phongEffect">
      <setparam ref="AMBIENT">
        <float3>0.0 0.0 0.1</float3>
      </setparam>
      <setparam ref="DIFFUSE">
        <float3>0.15 0.15 0.1</float3>
      </setparam>
      <setparam ref="SPECULAR">
        <float3>0.5 0.5 0.5</float3>
      </setparam>
      <setparam ref="SHININESS">
        <float>16.0</float>
      </setparam>
      </instance_effect>
  </material>
</library_materials>
```

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、外部またはユニフォームパラメータを指定することができます。

属性

〈**modifier**〉要素に属性はありません。

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	**newparam**
子要素	なし
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。有効な修飾子は、以下のとおりです。

- CONST
- UNIFORM
- VARYING
- STATIC
- VOLATILE
- EXTERN
- SHARED

例

```
<newparam sid="diffuseColor">
  <annotate name="UIWidget"><string>none</string></annotate>
  <semantic>DIFFUSE</semantic>
  <modifier>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。変換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変換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sid。オプション。

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	**shader**、**generator**
子要素	なし
その他	なし

備考

<name>要素には、エントリポイント関数の名前が含まれています。**source**はエントリポイントがどこにあるかを指定します。

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**（**GLE** と **COMMON** で） **xs:token**（その他）	パラメータの識別子（つまり変数名）。必須。

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	**effect**、**technique**、**profile_CG**、**profile_COMMON**、**profile_GLSL**、**profile_GLES**
子要素	下のサブセクションを参照
その他	なし

profile_CG および profile_CG / technique 内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

注意：子要素**<array>**，**<usertype>**、**VALUE_TYPE**のいずれか 1 つだけを記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<annotate>			0 以上
<semantic>			0 または 1
<modifier>			0 または 1
<array>			注意を参照
VALUE_TYPE	CG スコープ中の有効な値のタイプについては、「値のタイプ」セクションを参照してください。		注意を参照
<usertype>			注意を参照

profile_GLSL および profile_GLSL / technique 内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

注意：子要素**<array>**または**VALUE_TYPE**のいずれか 1 つだけを、記述しなければなりません。これらの要素は相互に排他的です。

名前/例	解説	デフォルト値	出現回数
<annotate>			0 以上
<semantic>			0 または 1
<modifier>			0 または 1
<array>			注意を参照
VALUE_TYPE	GLSL スコープ内の有効な値のタイプについては、「値のタイプ」セクションを参照してください。		注意を参照

profile_GLES、profile_GLES/technique、および effect 内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<annotate>			0 以上
<semantic>			0 または 1
<modifier>			0 または 1
VALUE_TYPE	GLES内の有効な値のタイプについては、「値のタイプ」セクションを参照してください。COLLADAスコープの値は、**<effect>**に適用されます。		1

profile_COMMON および profile_COMMON/technique 内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<semantic>			0 または 1
VALUE_TYPE	**float**、**float2**、**float3**、**surface**、および**sampler2D**のみが有効です。		1

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>要素は、**<bind_material>**内で以下の属性を持ちます。

name	**xs:NCName**	
sid	**xs:NCName**	
semantic	**ns:NMTOKEN**	
type	**ns:NMTOKEN**	

<param>要素は、他のすべての親内で以下の属性を持ちます。

ref	**xs:NCName**	必須。

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**、**bind_material**, **texture1D**, **texture2D**, **texture3D**, **textureCUBE**, **textureRECT**, **textureDEPTH**
子要素	なし
その他	なし

備考

例

```
<shader stage="VERTEX">
  <compiler_target>ARBVP1</compiler_target>
  <name source="ThinFilm2">main</name>
  <bind symbol="lightpos">
    <param ref="LightPos_03"/>
  </bind>
</shader>
```

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ステートとシェーダと設定を静的に宣言します。

コンセプト

<**pass**>は、レンダリングパイプラインのすべてのレンダリングステートとシェーダについて記述するためのもので、プログラムでジオメトリを送り出す前に、FXランタイムに対して、現在のグラフィックスのステートを「適用」するように指示するための要素です。
「静的な」宣言というのは、グラフィックスステートに適用するためにスクリプトエンジンやランタイムシステムで評価を行う必要がない宣言を意味します。<**pass**>が適用される時点では、すべてのレンダリングステートの設定とユニフォームパラメータは事前に計算されて値がわかっています。

属性

<**pass**>要素には、以下の属性があります。

sid	xs:NCName	オプションのラベルで、名前で<**pass**>を指定できるようにします。また必要であれば、テクニックを評価しながらアプリケーションで並べ替えられるようにします。オプション。

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	**technique**
子要素	下のサブセクションを参照してください
その他	なし

GLES スコープ内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<annotate>			0 以上
<color_target>			0 または 1
<depth_target>			0 または 1
<stencil_target>			0 または 1
<color_clear>			0 または 1
<depth_clear>			0 または 1
<stencil_clear>			0 または 1
<draw>			0 または 1
RENDER_STATES	「レンダリングステート」サブセクションを参照してください。		0 または 1
<extra>			0 以上

CG、GL、または GLSL スコープ内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<annotate>			0 以上
<color_target>			0 以上
<depth_target>			0 以上
<stencil_target>			0 以上
<color_clear>			0 以上
<depth_clear>			0 以上
<stencil_clear>			0 以上
<draw>			0 以上
RENDER_STATES	「レンダリングステート」サブセクションを参照してください。GLスコープでは、この子要素は、**<shader>**が使用されている場合、記述してはいけません。		0 以上
<shader>	GL スコープでは、この子要素は、***render_state*** が使用されている場合、記述してはいけません。		0 以上
<extra>		なし	0 以上

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」のローパス重畳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>
```

レンダリングステート

異なるFXプロファイルは、**<pass>**要素の中で利用できる別々のレンダリングステートセットを持ちます。

一般に、それぞれのレンダリングステートの要素は、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ステートに固有の値が設定でき、また param 属性で、ステート用の param 中に保存された値を利用することができます。いくつかの要素は、下表に明記されているように追加の index 属性を持ちます。その属性は、自身が属する要素を指定します。

```
<render_state value="some_value" param="param_reference" index="name"/>
```

これらのステートの詳細に関しては、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>**のレンダリングステートを示しておきます。それぞれのレンダリングステートは、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。

レンダリングステート	有効な値、値のタイプ、インデックス属性	GLES での違い
alpha_func **func** **value**	 **NEVER、LESS、LEQUAL、EQUAL、GREATER、NOTEQUAL、GEQUAL、ALWAYS** float 値の 0.0～1.0（0.0 と 1.0 を含む）	
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 にはありません
blend_equation	**FUNC_ADD、FUNC_SUBTACT、FUNC_REVERSE_SUBTRACT、MIN、MAX**	GLES にはありません
blend_equation_separate **rgb** **alpha**	blend_equation の値と同じ	GLES にはありません

レンダリングステート	有効な値、値のタイプ、インデックス属性	GLES での違い
color_material **face** **mode**	 **FRONT、BACK、FRONT_AND_BACK** **EMISSION、AMBIENT、DIFFUSE、SPECULAR、AMBIENT_AND_DIFFUSE**	GLES にはありません
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**	GLES にはありません
front_face	**CW、CCW**	
light_model_color_control	**SINGLE_COLOR、SEPARATE_SPECULAR_COLOR**	GLES にはありません
logic_op	**CLEAR、AND、AND_REVERSE、COPY、AND_INVERTED、NOOP、XOR、OR、NOR、EQUIV、INVERT、OR_REVERSE、COPY_INVERTED、NAND、SET**	
polygon_mode **face** **mode**	 **FRONT、BACK、FRONT_AND_BACK** **POINT、LINE、FILL**	GLES にはありません
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、DECT_WRAP**	
stencil_func_separate **front** **back** **ref** **mask**	（front と back） **NEVER、LESS、LEQUAL、EQUAL、GREATER、NOTEQUAL、GEQUAL、ALWAYS** 符号なし byte	GLES にはありません

レンダリングステート	有効な値、値のタイプ、インデックス属性	GLES での違い
	符号なし byte	
stencil_mask_separate **face** **mask**	 **FRONT**、**BACK**、**FRONT_AND_BACK** 符号なし byte	GLES にはありません
light_enable	論理値 index 属性は、光源を指定します。	
light_ambient	**float4** index 属性は、光源を指定します	
light_diffuse	**float4** index 属性は、光源を指定します	
light_specular	**float4** index 属性は、光源を指定します	
light_position	**float4** index 属性は、光源を指定します	
light_constant_attenuation	**float** index 属性は、光源を指定します	
light_linear_attenuation	**float** index 属性は、光源を指定します	
light_quadratic_attenuation	**float** index 属性は、光源を指定します	
light_spot_cutoff	**float** index 属性は、光源を指定します	
light_spot_direction	**float3** index 属性は、光源を指定します	
light_spot_exponent	**float** index 属性は、光源を指定します	
texture1D	**sampler1D**型 index属性は、テクスチャユニットを指定します。	GLES にはありません
texture2D	**sampler2D**型 index属性は、テクスチャユニットを指定します。	GLESにはありません（**<texture_pipeline>**を参照）
texture3D	**sampler3D**型 index属性は、テクスチャユニットを指定します。	GLES にはありません
textureCUBE	**samplerCUBE**型 index属性は、テクスチャユニットを指定します。	GLES にはありません
textureRECT	**samplerRECT**型 index属性は、テクスチャユニットを指定します。	GLES にはありません
textureDEPTH	**samplerDEPTH**型	GLES にはありません

レンダリングステート	有効な値、値のタイプ、インデックス属性	GLES での違い
	index属性は、テクスチャユニットを指定します。	
texture1D_enable	論理値 index 属性は、テクスチャユニットを指定します。	GLES にはありません
texture2D_enable	論理値 index 属性は、テクスチャユニットを指定します。	GLESにはありません（**<texture_pipeline>**を参照）
texture3D_enable	論理値 index 属性は、テクスチャユニットを指定します。	GLES にはありません
textureCUBE_enable	論理値 index 属性は、テクスチャユニットを指定します。	GLES にはありません
textureRECT_enable	論理値 index 属性は、テクスチャユニットを指定します。	GLES にはありません
textureDEPTH_enable	論理値 index 属性は、テクスチャユニットを指定します。	GLES にはありません
texture_env_color	**float4** index 属性は、テクスチャユニットを指定します。	GLESにはありません（**<texture_pipeline>**を参照）
texture_env_mode	**string** index 属性は、テクスチャユニットを指定します。	GLESにはありません（**<texture_pipeline>**を参照）
clip_plane	**float4** index 属性は、クリッピングプレーンを指定します。	
clip_plane_enable	論理値 index 属性は、クリッピングプレーンを指定します。	
blend_color	**float4**	
clear_color	**float4**	
clear_stencil	**int**	
clear_depth	**float**	
color_mask	**bool4**	
depth_bounds	**float2**	
depth_mask	論理値	
depth_range	**float2**	
fog_density	**float**	
fog_start	**float**	
fog_end	**float**	

レンダリングステート	有効な値、値のタイプ、インデックス属性	GLES での違い
fog_color	**float4**	
light_model_ambient	**float4**	
lighting_enable	論理値	
line_stipple	**int2**	
line_width	**float**	
material_ambient	**float4**	
material_diffuse	**float4**	
material_emission	**float4**	
material_shininess	**float**	
material_specular	**float4**	
model_view_matrix	**float4x4**	
point_distance_attenuation	**float3**	
point_fade_threshold_size	**float**	
point_size	**float**	
point_size_min	**float**	
point_size_max	**float**	
polygon_offset	**float2**	
projection_matrix	**float4x4**	
scissor	**int4**	
stencil_mask	**int**	
alpha_test_enable	論理値	
auto_normal_enable	論理値	GLES にはありません
blend_enable	論理値	
color_logic_op_enable	論理値	
cull_face_enable	論理値	
depth_bounds_enable	論理値	GLES にはありません
depth_clamp_enable	論理値	GLES にはありません
depth_test_enable	論理値	
dither_enable	論理値	
fog_enable	論理値	
light_model_local_viewer_enable	論理値	GLES にはありません
light_model_two_side_enable	論理値	
line_smooth_enable	論理値	GLES にはありません
line_stipple_enable	論理値	GLES にはありません
logic_op_enable	論理値	
multisample_enable	論理値	
normalize_enable	論理値	
point_smooth_enable	論理値	GLES にはありません

レンダリングステート	有効な値、値のタイプ、インデックス属性	GLES での違い
polygon_offset_fill_enable	論理値	
polygon_offset_line_enable	論理値	GLES にはありません
polygon_offset_point_enable	論理値	GLES にはありません
polygon_smooth_enable	論理値	GLES にはありません
polygon_stipple_enable	論理値	GLES にはありません
rescale_normal_enable	論理値	
sample_alpha_to_coverage_enable	論理値	
sample_alpha_to_one_enable	論理値	
sample_coverage_enable	論理値	
scissor_test_enable	論理値	
stencil_test_enable	論理値	
gl_hook_abstract	GL 拡張からレンダリングステートを追加するための要素。	GLES にはありません
texture_pipeline	文字列 – **<texture_pipeline>**パラメータの名前	GLES のみ
texture_pipeline_enable	論理値	GLES のみ

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サーフェスを生成する固定機能パイプラインを宣言します。その際、鏡面反射はPhong BRDF近似にしたがってシェーディング処理されます。

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。

```
color = emissive + <ambient> * ambient_light + <diffuse> * max(N.L, 0) +
<specular>*max(H.I, 0)<shininess>
```

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	**technique**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<emission>	このオブジェクトのサーフェスから放射される光源の量を宣言します。**common_color_or_texture_type**を参照	なし	0 または 1
<ambient>	このオブジェクトのサーフェスから放射される環境光の量を宣言します。**common_color_or_texture_type**を参照	なし	0 または 1
<diffuse>	このオブジェクトのサーフェスから反射される拡散光の量を宣言します。**common_color_or_texture_type**を参照	なし	0 または 1
<specular>	このオブジェクトのサーフェスから反射される鏡面反射光の量を宣言します。**common_color_or_texture_type**を参照	なし	0 または 1
<shininess>	このオブジェクトのサーフェスから鏡面反射突起部分の鏡面度または荒さを宣言します。**common_float_or_param_type**を参照	なし	0 または 1
<reflective>	完全鏡面反射のカラーを宣言します。**common_color_or_texture_type**を参照	なし	0 または 1

名前/例	解説	デフォルト値	出現回数
<reflectivity>	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。**common_float_or_param_type**を参照	なし	0 または 1
<transparent>	完全鏡面反射のカラーを宣言します。**common_color_or_texture_type**を参照	なし	0 または 1
<transparency>	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。**common_float_or_param_type**を参照	なし	0 または 1
<index_of_refraction>	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。**common_float_or_param_type**を参照	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。

id	**xs:ID**	要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。
platform	**xs:NCName**	プラットフォームのタイプ。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。デフォルトは“PC”。オプション。

関連要素

<**profile_CG**>要素は、以下の要素と関連性があります。

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	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>			0 または 1
<code>			0 以上
<include>			0 以上
<image>			0 以上
<newparam>			0 以上
<technique> (FX)			1 以上
<extra>		なし	0 以上

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>
     <newparam sid="world">
       <semantic>WORLD</semantic>
       <float4x4> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 </float4x4>
     </newparam>

     <newparam sid="worldIT">
       <semantic>WORLD_INVERSE_TRANSPOSE</semantic>
       <float4x4> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 </float4x4>
    </newparam>

     <newparam sid="worldViewProj">
       <semantic>WORLD_VIEW_PROJECTION</semantic>
       <float4x4> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 </float4x4>
     </newparam>

     <technique id="default" sid="default">
         <code>
    void VS (
         in varying float4 pos,
         in varying float3 norm,
         in uniform float3 light_pos,
         in uniform float4x4 w: WORLD,
         in uniform float4x4 wit: WORLD_INVERSE_TRANSPOSE,
         in uniform float4x4 wvp: WORLD_VIEW_PROJECTION,
         out varying float4 oPosition : POSITION,
         out varying float3 oNormal : TEXCOORD0,
         out varying float3 oToLight : TEXCOORD1 )
    {  oPosition = mul(wvp, pos);
       oNormal =  mul(wit, float4(norm, 1)).xyz;
       oToLight = light_pos - mul(w, pos).xyz;
       return;
    }

    float3 diffuseFS (
           in uniform float3 flat_color,
           in varying float3 norm : TEXCOORD0,
           in varying float3 to_light : TEXCOORD1 ) : COLOR
         {  return flat_color * saturate(NdotL),
              0.0, 1.0);
         }
         </code>
         <pass sid="single_pass">
           <shader stage="VERTEX">
             <name source="diffuse-code-1">VS</name>
             <bind symbol="light_pos">
               <param ref="lightpos"/>
             </bind>
```

```
        <bind symbol="w">
          <param ref="world"/>
        </bind>
        <bind symbol="wit">
          <param ref="worldIT"/>
        </bind>
         <bind symbol="wvp">
           <param ref="worldViewProj"/>
         </bind>
      </shader>
      <shader stage="FRAGMENT">
        <name source="diffuse-code-1">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**要素には、以下の属性があります。

id	**xs:ID**	要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。

関連要素

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

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	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>		なし	0 または 1
<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	0 以上
<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	0 以上

名前/例	解説	デフォルト値	出現回数
<technique> (FX)	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
<extra>		なし	0 以上

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。

id	**xs:ID**	要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。
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 techinique 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	下のサブセクションを参照してください
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。

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>		なし	0 または 1
<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	0 以上

名前/例	解説	デフォルト値	出現回数
<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	0 以上
<technique> (FX)	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>**, **<image>**、および**<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 以上
<extra>		なし	0 以上

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**>要素には、以下の属性があります。

id	**xs:ID**	要素のユニークな識別子を含んでいるテキスト文字列。この値は、インスタンス文書内でユニークでなければなりません。オプション。

関連要素

<**profile_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	**effect**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<asset>		なし	0 または 1
<code>			0 以上
<include>			0 以上
<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	0 以上

名前/例	解説	デフォルト値	出現回数
<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	0 以上
<technique> (FX)	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>**, **<image>**および**<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 以上
<extra>		なし	0 以上

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、合成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	**texcombiner**
子要素	下のサブセクションを参照してください
その他	なし

子要素

名前/例	解説	デフォルト値	出現回数
<argument>	実行する特定の操作に必要な引数を設定します。		1～3

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を宣言します。

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	**newparam**、**setparam**、**usertype**、**array**, **bind**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<source>			1
<wrap_s>		WRAP	0 または 1
<minfilter>		なし	0 または 1
<magfilter>		なし	0 または 1
<mipfilter>		なし	0 または 1
<border_color>		0	0 または 1
<mipmap_maxlevel>		0	0 または 1
<mipmap_bias>		0	0 または 1
<extra>		なし	0 以上

備考

例

sampler2D

概要

2 次元のテクスチャサンプラを宣言します。

コンセプト

属性

<**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	**newparam**、**setparam**、**usertype**、**array**, **bind**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<source>			1
<wrap_s>		WRAP	0 または 1
<wrap_t>		WRAP	0 または 1
<minfilter>		なし	0 または 1
<magfilter>		なし	0 または 1
<mipfilter>		なし	0 または 1
<border_color>		0	0 または 1
<mipmap_maxlevel>		255	0 または 1
<mipmap_bias>		0	0 または 1
<extra>		なし	0 以上

備考

例

sampler3D

概要

3 次元のテクスチャサンプラを宣言します。

コンセプト

属性

<**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	**newparam**、**setparam**、**usertype**、**array**, **bind**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<source>			1
<wrap_s>		WRAP	0 または 1
<wrap_t>		WRAP	0 または 1
<wrap_p>		WRAP	0 または 1
<minfilter>		なし	0 または 1
<magfilter>		なし	0 または 1
<mipfilter>		なし	0 または 1
<border_color>		0	0 または 1
<mipmap_maxlevel>		255	0 または 1
<mipmap_bias>		0	0 または 1
<extra>		なし	0 以上

備考

例

samplerCUBE

概要

キューブマップ用のテクスチャサンプラを宣言します。

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	**newparam**、**setparam**、**usertype**、**array**, **bind**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<source>			1
<wrap_s>		WRAP	0 または 1
<wrap_t>		WRAP	0 または 1
<wrap_p>		WRAP	0 または 1
<minfilter>		なし	0 または 1
<magfilter>		なし	0 または 1
<mipfilter>		なし	0 または 1
<border_color>		0	0 または 1
<mipmap_maxlevel>		255	0 または 1
<mipmap_bias>		0	0 または 1
<extra>		なし	0 以上

備考

例

samplerDEPTH

概要

奥行きマップのテクスチャサンプラを宣言します。

コンセプト

属性

<samplerDEPTH>要素には属性はありません。

関連要素

<samplerDEPTH>要素は、以下の要素と関連性があります。

出現回数	スキーマ中に定義された要素の数
親要素	**newparam**, **setparam**, **usertype**, **array**, **bind**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<source>			1
<wrap_s>		WRAP	0 または 1
<wrap_t>		WRAP	0 または 1
<minfilter>		なし	0 または 1
<magfilter>		なし	0 または 1
<extra>		なし	0 以上

備考

例

samplerRECT

概要

3 次元のテクスチャサンプラを宣言します。

コンセプト

属性

<**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	**newparam**、**setparam**、**usertype**、**array**, **bind**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<source>			1
<wrap_s>		WRAP	0 または 1
<wrap_t>		WRAP	0 または 1
<wrap_p>		WRAP	0 または 1
<minfilter>		なし	0 または 1
<magfilter>		なし	0 または 1
<mipfilter>		なし	0 または 1
<border_color>		0	0 または 1
<mipmap_maxlevel>		255	0 または 1
<mipmap_bias>		0	0 または 1
<extra>		なし	0 以上

備考

例

sampler_state

概要

<**profile_GLES**>用の 2 次元のテクスチャサンプラステートを指定します。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ステートのバンドルです。

コンセプト

属性

<**sampler_state**>要素には、以下の属性があります。

sid	**xs:NCName**	このステートの識別子。**<newparam>**および**<setparam>**ではオプションであり、**<texture_unit>**では有効でありません。

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	**newparam**、**setparam**、**texture_unit**
子要素	**<texture_unit>**内にはありません。**<newparam>**および**<setparam>**については、以下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<wrap_s>		REPEAT	0 または 1
<wrap_t>		REPEAT	0 または 1
<minfilter>		なし	0 または 1
<magfilter>		なし	0 または 1
<mipfilter>		なし	0 または 1
<mipmap_maxlevel>		255	0 または 1
<mipmap_bias>		0	0 または 1
<extra>			0 以上

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	**newparam**
子要素	なし
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:token**	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。必須。
program	**xs:NCName**	GLSLおよびCGプロファイルの**<technique>**ではオプションであり、GLESプロファイル、**<generator>**、または**<instance_effect>**では有効でありません。

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	**technique**、**generator**、**instance_effect**
子要素	下のサブセクションを参照してください
その他	なし

profile_GLSL/technique 内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<annotate>			0 以上
VALUE_TYPES	「値のタイプ」セクション内の GLSL 値のタイプを参照してください。		1
<usertype>			

名前/例	解説	デフォルト値	出現回数
<array>			

profile_CG/technique 内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
VALUE_TYPES	「値のタイプ」セクション内の CG 値のタイプを参照してください 。		1
<usertype>			
<array>			
<connect_param>			

profile_GLES/technique 内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<annotate>			0 以上
VALUE_TYPES	「値のタイプ」セクション中の GLES 値のタイプを参照してください。		1

<instance_effect>内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
VALUE_TYPES	「値のタイプ」セクション中のコア値のタイプを参照してください。		1

<generator>中の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<annotate>			0 以上
VALUE_TYPES	「値のタイプ」セクション中の GLSL 値または CG 値のタイプを、コンテキストに応じて参照してください。		1

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ユニフォーム入力にバインドするために必要なすべての情報を保持します。
COLLADA FX では、FX ランタイムのサポートに応じて、ソースコードシェーダと事前にコンパイルされたバイナリシェーダの両方が宣言できます。事前にコンパイルされたバイナリシェーダの場合、すでにコンパイル時にターゲットプロファイルが指定されていますが、バイナリヘッダをロードして解析しなくても、事前にコンパイルされたシェーダに関係した宣言を COLLADA リーダで検証できるようにするために、やはりプロファイル宣言が必要となります。
事前に定義されたパラメータ、ソースシェーダ、バイナリシェーダは、同じ名前空間/シンボルテーブル/ソースコード文字列にまとめられるとみなされるため、すべてのシンボルと関数がシェーダの宣言で利用でき、共通の関数、たとえば、共通のライティングコードを<technique>中の複数のシェーダで利用することが可能です。「変換ユニット」の考えを利用した FX ランタイムでは、それぞれのソースコードブロックに名前を付けて、名前空間をそういったユニットに分割することができます。
ユニフォーム入力パラメータを持つシェーダは、シェーダの宣言時に、事前に定義されたパラメータもしくはリテラル値をこれらの値にバインドすることができ、必要であれば、コンパイラに対してリテラルや定数の値をインライン化させることが可能です。

属性

<**shader**>要素には、以下の属性があります。

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。GLSL スコープ内の定義済み初期値は、**VERTEXPROGRAM** と **FRAGMENTPROGRAM** であり、CG スコープ内では、**VERTEX** と **FRAGMENT** です。オプション。

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	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<annotate>			0 以上
<compiler_target>			0 または 1
<name>			1

名前/例	解説	デフォルト値	出現回数
<compiler_options>			0 または 1
<bind>			0 以上

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サーフェス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サーフェスを空のキャンバス、つまりデフォルトにリセットしなければならない場合もあります。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、ターゲットサーフェスは、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。デフォルト値は 0 です。GLSL および CG のプロファイルではオプションであり、GLES プロファイルでは有効でありません。

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	なし
その他	なし

備考

現時点のプラットフォームでは、MRTを設定するルールがかなり制限されています。たとえば、どれも同じサイズとピクセルフォーマットでなければならない 4 つのカラーバッファと、すべてのカラーバッファに対して 1 つの深度バッファとステンシルバッファだけがアクティブとなります。COLLADA FXの宣言は、この制限を緩めるように設計されており、FXランタイムは、実際に適用する前に<**pass**>の中に指定されている特定のMRT宣言が可能であることを検証しなければならず、検証に失敗した場合にはエラーを返す必要があります。

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。GLSL と CG のプロファイルではオプションであり、GLES プロファイルでは有効でありません。
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。GLSLとCGのプロファイルではオプションであり、GLESプロファイルでは有効でありません。
mip	**xs:nonNegativeInteger**	デフォルト値は 0 です。オプション。
face	**enumeration**	有効な値は、POSITIVE_X、NEGATIVE_X、POSITIVE_Y、NEGATIVE_Y、POSITIVE_Z、および NEGATIVE_Z です。デフォルト値は、POSITIVE_X です。オプション。

関連要素

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

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	なし
その他	なし

備考

現在のプラットフォームでは、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、サーフェス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ム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	サーフェスのタイプを指定します。UNTYPED、1D、2D、3D、CUBE、DEPTH、RECTのいずれか 1 つでなければなりません。COMMON、GLES、およびGLSLスコープでは必須であり、CGスコープでは（または、GLES中の**<texture_unit>**では）有効ではありません。

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	**COMMON: newparam, setparam** **CG: newparam, setparam, array, bind, usertype** **GLES: newparam, setparam, texture_unit** **GLSL: newparam, setparam, array, bind**
子要素	**GLES <texture_unit>**ではありません。 それ以外は下のサブセクションを参照
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
initialization option	このサーフェスの初期化オプション。自分のサーフェスにどれが適切かを、「備考」で説明されているタイプ属性や他の特性に基づいて選択します。		0 または 1
<format>	このサーフェスに使用したいプロファイル固有およびプラットフォーム固有のテクセル形式を表している文字列を、含んでいます。この要素が指定されていない場合、アプリケーションは、sRGB ではなく、色の線形階調付きの共通形式 R8G8B8A8 を使用することになります。		0 または 1
<format_hint>	**<format>** を使って正確なフォーマットを見つけることができない場合は、アプリケーションが互換性のあるフォーマットかまたは同様のフォーマットを選択できるように、**<format_hint>**がフォーマットの重要な特徴を記述します。 以下のサブセクション「**<surface>**/**<format_hint>**内の子要素」を参照してください。		0 または 1
<size>	指定された場合、サーフェスはこの正確な寸法になるようにサイズ調整されます。**<viewport_ratio>**が使用されている場合は、記述できません。	0 0 0	0 または 1
<viewport_ratio>	指定された場合、サーフェスは、ビューポートの寸法（ピクセル）のこの比率に基づいて、ある寸法にサイズ調整されます。**<size>**が使用されている場合は、記述できません。	1	0 または 1
<mip_levels>	指定された場合、サーフェスは、指定された数の MIP レベルを含むべきです。この要素が存在しない場合は、寸法が 1 テクセルになるまですべてのミップレベルが存在することが仮定されます。ミップマップのレベルをただ 1 つ（mip=0）持つサーフェスを作成するには、これを 1 に設定します。値が 0 の場合、結果は、mip_levels が指定されなかった場合と同じことになります。すなわち、可能性のあるすべての **mip_levels** が存在することになるというわけです。	0	0 または 1
<mipmap_generate>	論理値。デフォルト値は false。値が false であり、ミップマップレベルを提供しなかったがために初期化されなかったサブサーフェスがある場合は、生成されたサーフェスはプロファイル固有でプラットフォーム固有でもある動作をすることになります。true の場合、**mipmap_generate** は、サブサーフェスの残りの初期化を実行します。	false	0 または 1

名前/例	解説	デフォルト値	出現回数
<extra>		なし	0 以上
<generator>	CG、GLES および GLSL のスコープ内で有効です。手続き型のサーフェスジェネレータを指定します。		0 または 1

<surface>/<format_hint>内の子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<channels>	フォーマットのテクセルごとのレイアウト。文字列の長さは存在するチャンネルの数を示し、各文字はチャンネルの名前を表します。通常は、0～4 つのチャンネルがあります。有効な列挙値は次のとおりです。 RGB：RGB カラーマップ。 RGBA：カラー+透明度に頻繁に使用される RGB カラー+アルファのマップ。または鏡面反射成分のようにチャンネル A にパッキングされる他のもの。 L：光マッピングに頻繁に使用される輝度マップ LA: 光マッピングに頻繁に使用される輝度+アルファのマップ D：ディスプレースメント、視差、レリーフ、またはシャドウのマッピングに頻繁に使用される奥行きマップ XYZ：通常は、標準マップ、または 3 要素の変位マップに使用されます。 XYZW: 通常は、W がレリーフマッピングまたは視差マッピングの奥行きである、標準マップに使用されます	なし	1
<range>	各チャンネルは、値の範囲を表します。いくつかの範囲例には、符号付き整数または符号なし整数のものや、0.0f～1.0f などの固定範囲内のもの、浮動小数点を利用した高位の動的範囲などがあります。有効な列挙値は次のとおりです。 SNORM：フォーマットは、-1～1 の範囲内にある 10 進値を表しています。実装は、整数、固定小数点数、浮動小数点数のいずれであってもかまいません。 UNORM：フォーマットは、0～1 の範囲内にある 10 進値を表しています。実装は、整数、固定小数点数、浮動小数点数のいずれであってもかまいません。 SINT: ファーマットは、符号付きの整数を表しています。（例：8 ビット = -128～127） UINT：形式は、符号なしの整数を表しています。例えば、8 ビット = 0～255 です。 FLOAT：ファーマットは、完全な浮動小数点数の範囲をサポートすべきです。HIGH（高）精度は、32 ビットであることが期待されます。MID（中）精度は、16～32 ビットでかまいません。LOW（低）精度は、16 ビットがであることが期待されます。	なし	1
<precision>	テクセルの各チャンネルはそれぞれ精度を持っています。通常、これらのチャンネルはひとまとめにリンクされています。正確なフォーマットが個々のチャンネ	なし	0 または 1

名前/例	解説	デフォルト値	出現回数
	ルの精度を下げることがありますが、チャンネルをひとまとめに結合することでより高い精度を適用すると引き続き同じ情報を伝達できます。有効な列挙値は次のとおりです。 LOW：整数の場合、これは通常 8 ビットを表します。浮動小数の場合は、通常 16 ビットです。 MID：整数の場合、これは通常 8～24 ビットを表します。浮動小数の場合は、通常 16～32 ビットです。 HIGH：整数の場合、これは通常 16～32 ビットを表します。浮動小数の場合は、通常 24～32 ビットです。		
<option>	アプリケーションが最適なフォーマットを選べるように支援するためのデータの関係性および他の事柄に関する追加のヒント。有効な列挙値は次のとおりです。 SRGB_GAMMA：カラーは、sRGB 2.2 ガンマ曲線（直線ではありません）に関して保存されます。 NORMALIZED3：テクセルの XYZ/RGB は、標準マップなどの場合と同様に正規化されるはずです。 NORMALIZED4：テクセルの XYZW/RGBA は、標準マップなどの場合と同様に正規化されるはずです。 COMPRESSABLE：サーフェスには、ランタイム圧縮を使うことができます。目的の**<channels>**、**<range>**、**<precision>**、および**<option>**に基づいて最適な圧縮を検討します。	なし	0 以上
<extra>		なし	0 以上

備考

<size>要素も**<viewport_ratio>**要素もない場合、サーフェスの寸法は、その初期化イメージ（**<init_*>**）から取り込まれます。寸法は、ミップレベル 0 にロードされたイメージから取り込まれます。

サーフェスのタイプ属性がUNTYPEDに設定されると、そのタイプは最初は未知であり、そのタイプを参照するテクスチャサンプラによってなど、そのタイプが使用されるコンテキストによって後で確定されます。他の任意のタイプのサーフェスは、**<setparam>**を用いて、後ほどのランタイム時に、**<newparam>**によって作成されたかのようなUNTYPEDのサーフェスに変更できます。**<setparam>**操作と、ランタイムが決定したタイプとの間にタイプの不一致がある場合、結果はプロファイル固有でプラットフォーム固有でもある動作となります。

初期化オプション

オプションの*初期化オプション*は、以下のいずれか 1 つになります。

名前/例	解説
<init_as_null>	このサーフェスは、**<setparam>**要素によって後で外部的に初期化されることになります。初期化前に使用されると、プロファイル固有でプラットフォーム固有でもある動作になります。これを含んでいる**<surface>**要素に関する大部分の要素は、**<mip_levels>**、**<mipmap_generate>**、**<size>**、**<viewport_ratio>**、および**<format>**を含み、無視されることになります。
<init_as_target>	このサーフェスを、奥行き、ステンシル、またはカラーのターゲットとして初期化します。イメージデータは必要ありません。これが使用されるとき、サーフェスが**<mipmap_generate>**を持つことがあってはなりません。
<init_cube> <all ref=… /> <primary ref=…> <order /> </primary> <face ref=… /> </init_cube	DDS などの複合イメージの CUBE を使い、サーフェス全体を初期化します。 以下のいずれか 1 つを選択します。 **<all>**：DDS などの複合イメージを 1 つ用いて、サーフェスを初期化します。 **<primary>**：ミップレベル 0 の一次サブサーフェスをすべて、DDS などの複合イメージを 1 つ用いて初期化します。この要素を使用することにより、サーフェスが要素**<mip_levels>=0** または**<mipmap_generate>**を持つようになることが予想されます。そのサブ要素**<order>**は記述されないか、または 6 個記述されます。イメージが面の順序づけを本来記述しない場合は、順序付けられたこれら一連の要素が、インデックスがどの面に属するかを記述します。 **<face>**：6 回記述されます。DDS などの複合イメージを 1 つ用いて、面の各ミップチェーンを初期化します。
<init_volume> <all ref=… /> <primary ref=… /> </init_volume>	DDS などの複合イメージの 3D を用いて、このサーフェスを初期化します。 以下のいずれか 1 つを選択します。 **<all>**：DDS などの複合イメージを 1 つ用いて、サーフェスを初期化します。 **<primary>**：DDS などの複合イメージを 1 つ用いて、サーフェスのミップレベル 0 を初期化します。この要素を使用することにより、サーフェスが要素**<mip_levels>=0** または**<mipmap_generate>**を持つようになることが予想されます。
<init_planar> <all ref=… /> </init_planar>	DDS などの複合イメージの 1D、2D、RECT、または DEPTH を用いて、このサーフェスを初期化します。 以下のものが含まれていなければなりません。 **<all>**：DDS などの複合イメージを 1 つ用いて、サーフェスを初期化します。
<init_from mip=… slice=… face=… />	特定のサーフェスタイプを意味づけするミップ、面、およびスライスの組み合わせを指定して、サーフェスのサブサーフェスを一度に 1 つ初期化します。各サブサーフェスは、DDS などの複雑な複合イメージではなく、通常の 2D イメージにより初期化されます。初期化されないサブサーフェスがある場合は、その残りのサブサーフェスの初期化が**<mipmap_generate>**によってなされる場合を除いて、このオプションは無効であり、プロファイル固有でプラットフォーム固有でもある動作をもたらします。

<format>および<format_hint>の使用

DCC ツールは、おそらく、なにも書かないか、**<format_hint>**のみを書くかのどちらかでしょう。

ゲームエンジンツールは、DirectX 4CC テクスチャコードなどのデザインケースのために、おそらく**<format>**を追加することになるでしょう。すなわち、このオプションは、正確であること、メモリスペースを最小化すること、またはパフォーマンスやクォリティーを最大化することに非常に特化していると言えます。

サーフェスを作成するアプリケーションは、以下のものを使用すべきです。

- **<format>**が存在し、その文字列が理解されている場合は、フォーマット文字列を使用します。
- **<format_hint>**が存在する場合は、そこに記述されている機能や特性を使用して、適切なフォーマットを選択します。
- **<images>**から初期化された場合は、**<image>**と互換性のあるものを使用します。
- そうでなければ、アプリケーションを支援する情報がないので、R8G8B8A8_UNORM や R16G16B16A16_UNORM などのよく知られたフォーマットを使用します。

例

```
<surface type="CUBE">
   <init_cube><all ref="sky_dds"></init_cube>
</surface>

<surface type="2D">
   <init_as_target/>
   <format>R5G6B5</format>
   <format_hint>
       <channel>RGB</channel>
       <range>UNORM</range>
       <precision>LOW</precision>
   </format_hint>
</surface>
```

technique (FX)

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。

非FX要素内の**<technique>**については、「**technique** (core)」を参照してください。

コンセプト

テクニックは、効果のレンダリングに必要なすべての必須要素を保持します。それぞれの効果には複数のテクニックを含めることができ、それぞれのテクニックで効果をレンダリングする別々の方法を表します。一般にテクニックが利用される状況としては、以下の 3 つがあります。

- あるテクニックで高い LOD バージョンの効果を記述し、2 番目のテクニックで同じ効果の低い LOD バージョンを記述するような場合。
- 同じ効果を別々の方法で記述しておき、標準の API を利用した未知のデバイスに対して最も効率的なバージョンの効果を見つけるために FX ランタイムの検証ツールを利用する場合。
- 異なるゲームのステート、たとえば、昼間のテクニックと夜間のテクニックなどの効果を記述し、通常のテクニックと「魔法が有効」なテクニックを記述する場合。

属性

<**technique**>要素には、以下の属性があります。

id	xs:ID	オプション。要素のユニークな識別子を含むテキスト文字列です。この値はインスタンス文書中でユニークでなければなりません。
sid	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, profile_GLES**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<annotate>			
<asset>			
<blinn>			
<include>			
<newparam>			
<pass>			

名前/例	解説	デフォルト値	出現回数
<setparam>			

備考

ツールを利用して効果のテクニックを自動的に生成することができ、テクニックを最善の<**asset**>として管理する（作成時間、利用可能期間、親子関係、生成に利用するツールを管理する）ことが可能です。

例

```
<effect id="BumpyDragonSkin">
  <profile_GLSL>
    <technique sid="HighLOD">
      ...
    </technique>
    <technique sid="LowLOD">
      ...
    </technique>
  </profile_GLSL>
</effect>
```

technique_hint

概要

効果で利用するテクニックのプラットフォームのヒントを追加します。

コンセプト

シェーダエディタは、効果がインスタンス化された際にデフォルトで利用するテクニックの情報が必要とします。検証の前提として、FX ランタイムがプラットフォームの文字列を認識できた場合には、推奨されているテクニックを利用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。オプション。
ref	xs:NCName	プラットフォームの名前を参照します。必須。
profile	**xs:NCName**	このヒントがどのAPIプロファイルを対象としているかを指定する文字列。テクニックを含んでいる効果内のプロファイルの名前です。プロファイルは、この属性値を「**profile_**」に付加することで作成されます。例えば、**profile_CG**を選択するには、**profile="CG"**を指定します。オプション。

関連要素

<technique_hint>要素は、以下の要素と関連性があります。

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

備考

例

```
<technique_hint platform="PS3" ref="HighLOD"/>
<technique_hint platform="OpenGL|ES" ref="twopass"/>
<technique_hint profile="CG" platform="GL" ref="HighLOD"/>
<technique_hint profile="GLES" platform="NOKIA_SW" ref="OneLight"/>
```

texcombiner

概要

<texture_pipeline>コマンドを定義します。これは合成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合成ステートを設定します。
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、value
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<constant>	glTexEnv(TEXTURE_ENV, TEXTURE_ENV_COLOR, 値)のfloat4。		0 または 1
<RGB>	テクスチャコンバイナコマンドの RGB コンポーネントを設定します。		0 または 1
<alpha>	テクスチャコンバイナコマンドのアルファコンポーネントを設定します。		0 または 1

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合成モードに変更しなければなりません。

```
glTexEnv(TEXTURE_ENV, TEXTURE_ENV_MODE, COMBINE)
```

OpenGL ESハードウェアのテクスチャユニット代入の詳細に関しては<texture_pipeline>の解説を参照してください。

例

<texture_pipeline>を参照してください。

texenv

概要

<texture_pipeline>コマンドを定義します。これは単純な非合成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ステートを設定します。
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。オプション。
unit	xs:NCName	テクスチャユニット。オプション。

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	texture_pipeline、value
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<constant>	glTexEnv(TEXTURE_ENV, TEXTURE_ENV_COLOR, 値)のfloat4。		0 または 1

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

通常モードと合成モードで glTexEnv を利用してマルチテクスチャ操作に変換されるテクスチャコマンドセットを定義します。

コンセプト

この要素には、すべての GLES マルチテクスチャステートを定義する順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、合成モードでテクスチャユニットの設定を定義します。
- <**texenv**>は、非合成モードでテクスチャユニットの設定を定義します。

それぞれのコマンドは、テクスチャユニットの名前とコマンドの利用特性を基にして、後のバインド処理の際にテクスチャユニットに割り当てられます。
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>のステートを利用します。
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**	**newparam**および**setparam**に対してオプション。 pass に対して有効でありません。
param		pass に対してオプション。 **newparam**および**setparam**に対して有効でありません。

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	GLESスコープ内では**newparam, setparam, pass**（レンダリングステート）
子要素	下のサブセクションを参照してください
その他	なし

newparam および setparam 内の子要素

子要素を任意の順で記述できます。

名前/例	解説	デフォルト値	出現回数
<texcombiner>			0 以上
<texenv>			0 以上
<extra>		なし	0 以上

pass 内の子要素

子要素を任意の順で記述できます。

名前/例	解説	デフォルト値	出現回数
<value sid=" "> **<texcombiner/>** **<texenv/>** **<extra/>**	子要素はオプションです。任意の順番で、希望に応じて何回でも記述が可能です。 sid 属性はオプションです。		

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 source="TEXTURE" operand="SRC_RGB" unit="gravel"/>
    <argument source="TEXTURE" operand="SRC_RGB" unit="grass"/>
    <argument source="TEXTURE" operand="SRC_ALPHA" unit="transition"/>
   </RGB>
   <alpha operator="INTERPOLATE">
    <argument source="TEXTURE" operand="SRC_ALPHA" unit="gravel"/>
    <argument source="TEXTURE" operand="SRC_ALPHA" unit="grass"/>
    <argument source="TEXTURE" operand="SRC_ALPHA" unit="transition"/>
   </alpha>
 </texcombiner>
 <texcombiner>
   <RGB operator="MODULATE">
    <argument source="PRIMARY" operand="SRC_RGB"/>
    <argument source="PREVIOUS" operand="SRC_RGB"/>
   </RGB>
   <alpha operator="MODULATE">
    <argument source="PRIMARY" operand="SRC_ALPHA"/>
    <argument 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**	オプション

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	**setparam, newparam**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素が存在する場合、その並びは以下の順番でなければなりません。

名前/例	解説	デフォルト値	出現回数
<surface>	テクスチャに使用されるサーフェスのサーフェスパラメータを、参照します。		0 または 1
<sampler_state>	テクスチャユニットが使用しなければならないテクスチャ座標配列にセマンティック名を提供するセマンティック属性を、含みます。配列は、**<bind_material>**を使用して、ここにマッピングされます。		0 または 1
<texcoord>	テクスチャユニットが使用しなければならないテクスチャ座標配列用のセマンティック名を提供するセマンティック属性を、含みます。配列は、**<bind_material>**を使用して、ここにマッピングされます。		0 または 1
<extra>		なし	0 以上

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: **token**	現在のソースコード変換ユニット中にある構造体宣言のための識別子です。必須。
source	**xs:NCName**	コードを参照するか、またはユーザタイプを定義する要素を含みます。必須。

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	**newparam** (cg newparam)、**array, usertype, setparam**
子要素	下のサブセクションを参照してください
その他	なし

子要素

子要素を任意の順で記述できます。

名前/例	解説	デフォルト値	出現回数
<array>			0 以上
VALUE_TYPE	「値のタイプ」セクション中の CG 値のタイプを参照してください。		0 以上
<connect_param>			0 以上
<usertype>			0 以上

名前/例	解説	デフォルト値	出現回数
<setparam>	一連のこの要素を用いて、名前によってメンバを設定します。ref 属性は、現在自分自身であるユーザタイプに関連するものです。		1 以上

備考

〈**usertype**〉の各要素は、すべてのリーフノードを順番に走査しその値を設定するか、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。
<array>、値のタイプ、**<connect_param>**、および**<usertype>**の組み合わせを使用して、順番に依存する方法でユーザタイプを初期化します。
　いくつかのユーザタイプにはデータがありません。それらは、インタフェース関数を実装するためにのみ使用できます。

例

```
<include sid="simple_cg_source" url="simple.cgfx"/>

<newparam sid="lightsource0" source="simple_cg_source">
  <usertype name="spotlight">
    <float3> 10 12 10 </float3>
    <float3> 0.3 0.3 0.114 </float3>
  </usertype>
</newparam>

<newparam sid="lightsource1" source="simple_cg_source">
  <usertype name="spotlight">
    <setparam ref="position"><float3> 10 12 10 </float3></setparam>
    <setparam ref="direction"><float3> 0.3 0.3 0.114 </float3></setparam>
  </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 の範囲のコア 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。

付録 A　COLLADA の例

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/2005/11/COLLADASchema" version="1.4.1">
<asset>
    <created>2005-11-14T02:16:38Z</created>
    <modified>2005-11-15T11:36:38Z</modified>
    <revision>1.0</revision>
  </asset>
  <library_effects>
    <effect id="whitePhong">
      <profile_COMMON>
        <technique sid="phong1">
          <phong>
            <emission>
              <color>1.0 1.0 1.0 1.0</color>
            </emission>
            <ambient>
              <color>1.0 1.0 1.0 1.0</color>
            </ambient>
            <diffuse>
              <color>1.0 1.0 1.0 1.0</color>
            </diffuse>
            <specular>
              <color>1.0 1.0 1.0 1.0</color>
            </specular>
            <shininess>
              <float>20.0</float>
            </shininess>
            <reflective>
              <color>1.0 1.0 1.0 1.0</color>
            </reflective>
            <reflectivity>
              <float>0.5</float>
            </reflectivity>
            <transparent>
              <color>1.0 1.0 1.0 1.0</color>
            </transparent>
            <transparency>
              <float>1.0</float>
            </transparency>
          </phong>
        </technique>
      </profile_COMMON>
    </effect>
  </library_effects>
  <library_materials>
    <material id="whiteMaterial">
      <instance_effect url="#whitePhong"/>
    </material>
  </library_materials>
  <library_geometries>
```

```
    <geometry id="box" name="box">
      <mesh>
        <source id="box-Pos">
          <float_array id="box-Pos-array" count="24">
           -0.5  0.5  0.5
            0.5  0.5  0.5
           -0.5 -0.5  0.5
            0.5 -0.5  0.5
           -0.5  0.5 -0.5
            0.5  0.5 -0.5
           -0.5 -0.5 -0.5
            0.5 -0.5 -0.5
          </float_array>
          <technique_common>
            <accessor source="#box-Pos-array" count="8" stride="3">
              <param name="X" type="float" />
              <param name="Y" type="float" />
              <param name="Z" type="float" />
            </accessor>
          </technique_common>
        </source>
        <source id="box-0-Normal">
          <float_array id="box-0-Normal-array" count="18">
           1.0  0.0  0.0
          -1.0  0.0  0.0
           0.0  1.0  0.0
           0.0 -1.0  0.0
           0.0  0.0  1.0
           0.0  0.0 -1.0
          </float_array>
          <technique_common>
            <accessor source="#box-0-Normal-array" count="6" stride="3">
              <param name="X" type="float"/>
              <param name="Y" type="float"/>
              <param name="Z" type="float"/>
            </accessor>
          </technique_common>
        </source>
        <vertices id="box-Vtx">
          <input semantic="POSITION" source="#box-Pos"/>
        </vertices>
        <polygons count="6" material="WHITE">
          <input semantic="VERTEX" source="#box-Vtx" offset="0"/>
          <input semantic="NORMAL" source="#box-0-Normal" offset="1"/>
          <p>0 4  2 4  3 4  1 4</p>
          <p>0 2  1 2  5 2  4 2</p>
          <p>6 3  7 3  3 3  2 3</p>
          <p>0 1  4 1  6 1  2 1</p>
          <p>3 0  7 0  5 0  1 0</p>
          <p>5 5  7 5  6 5  4 5</p>
        </polygons>
      </mesh>
    </geometry>
  </library_geometries>
  <library_visual_scenes>
    <visual_scene id="DefaultScene">
      <node id="Box" name="Box">
        <translate> 0  0  0</translate>
        <rotate> 0  0  1  0</rotate>
        <rotate> 0  1  0  0</rotate>
```

```
        <rotate> 1   0   0   0</rotate>
        <scale> 1   1   1</scale>
        <instance_geometry url="#box">
          <bind_material>
            <technique_common>
              <instance_material symbol="WHITE" target="#whiteMaterial"/>
            </technique_common>
          </bind_material>
        </instance_geometry>
      </node>
    </visual_scene>
  </library_visual_scenes>
  <scene>
    <instance_visual_scene url="#DefaultScene"/>
  </scene>
</COLLADA>
</COLLADA>
```

このページは空白です。

用語集

- **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">
```

- **COLLADA**：業界での協業による設計アクティビティ

- **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 -->
```

- **DCC**：ディジタルコンテンツ作成

- **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。