COLLADA — Digital Asset Schema
リリース 1.4.1

パッチリリースノート:リビジョン B

2007 年 8 月

編集責任者: Mark Barnes および Ellen Levy Finch, Sony Computer Entertainment Inc.

Publication date: 2007 年 8 月

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

COLLADA 1.4.1 パッチリリースノート　リビジョンB

本ドキュメントについて

本ドキュメントでは、COLLADA Digital Asset スキーマのリリース 1.4.1 での変更点について、その概要を説明します。スキーマのバージョン 1.4.1 および「*COLLADA — Digital Asset Schema リリース 1.4.1 – 仕様書*」は、以下のサイトからダウンロードできます。

http://www.khronos.org/collada/

この更新（リビジョン B）は、2006 年 6 月のリリースノート 1.4.1 の修正・拡張版で、2006 年 12 月の *パッチリリースノート：リビジョン A* に代わるものです。

- リビジョン A 以降の追加部分は陰影付きのテキストで示します。
- リビジョン A 以降の削除部分は~~取り消し線~~で示します。

追加のウェブリソース

COLLADA に関する追加情報は、以下のサイトにあります。

- http://collada.org: COLLADAに関する追加の技術情報、COLLADA拡張、プラグイン、コンディショナーの公開ディレクトリ、COLLADAに関する議論を行うための公開フォーラム。
- http://www.khronos.org/bugzilla/: COLLADAの公式バグ報告システム。

バージョン 1.4.0 以降のスキーマの変更

このリリースでのスキーマの変更点は、特に明記しない限り、既存の COLLADA 1.4.0 ドキュメントと互換性があります。

このリリースで変更されたスキーマの機能を含むスキーマ機能の詳細については、更新済みの「*COLLADA — Digital Asset Schema リリース 1.4.1 – 仕様書*」を参照してください。

<color_target>、<depth_target>、<stencil_target>は、現在、オプションのface属性およびmip属性を持ちます。

バグ K-125 を解決します。

すべての<*_target>要素は、現在、以下のような属性を持ちます。

```
<xs:attribute name="index" type="xs:nonNegativeInteger"
    use="optional" default="0"/>
<xs:attribute name="mip" type="xs:nonNegativeInteger"
    use="optional" default="0"/>
<xs:attribute name="slice" type="xs:nonNegativeInteger"
    use="optional" default="0"/>
<xs:attribute name="face" type="fx_surface_face_enum"
    use="optional" default="POSITIVE_X"/>
```

<COLLADA>のversion属性は、現在、1.4.1 をサポートします。

バグ K-171 を解決します。

version 属性の有効な値は、現在 1.4.0 および 1.4.1 です。

<COLLADA>要素は、現在、オプションのxml:base属性を持ちます。

バグ K-207 を解決します。

```
<xs:element name="COLLADA">
...
    <xs:attribute ref="xml:base"/>
```

<connect_params>のrefの属性タイプが変わりました。

バグ K-203 を解決します。

<connect_param>の ref 属性は、現在、xs:token になっています。

<convex_mesh>の子要素は、すべてオプションになりました。

バグ K-380 を解決します。

以前必要だった子要素**<source>**および**<vertices>**は、現在、オプションになっています。これにより、属性 convex_hull_of が使用されるとき、適切で明白な構文が可能になります。この属性は、指定されたメッシュの凸包がアプリケーションにより計算されるべきであることを示します。この場合、子要素の存在が誤解を招く可能性があります。1.4.0 では、convex_hull_of を曖昧でなく使用する唯一の方法は、以下に示すように、空のソースおよび頂点を定義することによってでした。

```
<convex_mesh id="cm" convex_hull_of="#someMesh">
    <source id="empty"/>
    <vertices id="verts">
        <input semantic="POSITION" source="#empty"/>
    </vertices>
</convex_mesh>
```

これは、もはや必要ありません。

注意：convex_hull_of が使用されない場合、子要素**<source>**および**<vertices>**を今までどおり指定して有効な**<convex_mesh>**を定義すべきです。

<extra>子要素がいくつかの要素に追加されました。

バグ K-174 と K-175 を解決します。

オプションで、制限のない**<extra>**子要素が以下の要素に追加されました。

- **<bind_material>**
- **<sampler1D>**
- **<sampler2D>**
- **<sampler3D>**
- **<samplerCUBE>**
- **<samplerDEPTH>**
- **<samplerRECT>**

- **<surface>**
- **<profile_CG>**
- **<profile_CG> / <technique>**
- **<profile_CG> / <technique> / <pass>**
- **<profile_GLES>**
- **<profile_GLES> / <technique>**
- **<profile_GLES> / <technique> / <pass>**
- **<profile_GLSL>**
- **<profile_GLSL> / <technique>**
- **<profile_GLSL> / <technique> / <pass>**
- **<texture_unit>**

使用されなかったfx_setparam_commonタイプが削除されました。

バグ K-140 を解決します。

fx_setparam_common は定義されても使用されなかったので、削除されました。

<input>要素のsemantic属性値リストが変更されました。

バグ K-138、K-305、K-316、K-344 を解決します（semantic 属性の値である TEXTURE を使用している COLLADA ドキュメントは、変更する必要があります）。

<input>の semantic 属性については、以下のようになっています。

- TEXTURE 値はもはや有効ではありません。これは 1.4.0 で削除されてしかるべきでした。
- 現在、以下のものが有効な値です。
  - CONTINUITY
  - LINEAR_STEPS
  - MORPH_TARGET
  - MORPH_WEIGHT
  - TEXBINORMAL
  - TEXTANGENT

<instance_material>は新しい<bind_vertex_input>子要素を持ちます。

バグ K-348 とバグ K-360 を解決します。

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

- 一般化（例：TEXCOORD0 対 DIFFUSE-TEXCOORD）
- スペリングの違い（例：COLOR 対 COLOUR）
- 省略形
- 冗長

- 同義語

存在する場合、**<instance_material>**の子要素は以下の順番で出てくる必要があります。

名前/例	解説	デフォルト値	出現回数
<bind>			0 以上
<bind_vertex_input>	**以下のテーブルを参照**		**0 以上**
<extra>			0 以上

<bind_vertex_input>は、以下の属性を持ちます。

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。

<instance_rigid_body>の<*velocity>要素は、現在、デフォルト値を持ちます。

バグ K-317 を解決します。

<instance_rigid_body>に、**<velocity>** (0, 0, 0) 用および**<angular_velocity>** (0, 0, 0) 用のデフォルト値を追加しました。

<instance_*>要素は、現在、オプションのname属性およびsid属性を持ちます。

バグ K-189 を解決します。

オプションの name および sid 属性を、今までそれらの属性を持たなかったすべての**<instance_*>**要素に追加しました。それらの要素を以下に示します。

- **<instance_animation>**
- **<instance_camera>**
- **<instance_controller>**
- **<instance_effect>**
- **<instance_force_field>**
- **<instance_geometry>**
- **<instance_light>**
- **<instance_material>**
- **<instance_node>**
- **<instance_physics_material>**
- **<instance_physics_scene>**
- **<instance_visual_scene>**

オプションの name 属性を以下の要素に追加しました。

- **<instance_physics_model>**
- **<instance_rigid_material>**

<library_physics_material>は、現在、id属性およびname属性を持ちます。

バグ K-371 を解決します。

<newparam>のsidの属性タイプが変更されました。

バグ K-203 を解決します。

<newparam>の sid 属性は、現在、<newparam>の親要素が以下の要素の場合に xs:token となっています。

- <profile_CG>
- <profile_CG> / <technique>
- <profile_GLSL>
- <profile_GLSL> / <technique>

<pass>には、もはや余分の<sequence>レイヤはありません。

バグ K-177 を解決します。

冗長な<xs:sequence>を<pass>要素から削除しました。

<pass>レンダリング状態である<color_material_enable>は、現在すべてのプロファイルに適用されます。

バグ K-390 を解決します。

gl_pipeline_setting。COLLADA FX 1.4.0 にはなかったものですが、新たに追加しました。その結果、ランタイムが glEnable(GL_COLOR_MATERIAL)および glDisable(GL_COLOR_MATERIAL)をいつ実行すべきか、または GLES、GLSL、Cg などのプロファイル内での等価な機能をいつ実行すべきかということを、作成者が指示できるようになりました。

```
<xs:element name="color_material_enable">
  <xs:complexType>
    <xs:attribute name="value" type="bool" use="optional" default="true"/>
    <xs:attribute name="param" type="xs:NCName" use="optional"/>
  </xs:complexType>
</xs:element>
```

複数の<profile_*>要素は、現在、オプションの<asset>子要素を持ちます。

バグ K-176 を解決します。

オプションの<asset>子要素は、現在、以下の要素内で有効です。

- <profile_COMMON>
- <profile_GLES>
- <profile_GLSL>
- <profile_CG>

複数の<profile_*>要素は、現在、オプションのid属性を持ちます。

バグ K-322 を解決します。

オプションの id 属性を、<profile_COMMON>、<profile_CG>、<profile_GLES>、<profile_GLSL>に追加しました。

<profile_GLES>は、現在、オプションのplatform属性を持ちます。

バグ K-323 を解決します。

オプションの platform 属性を<profile_GLES>に追加して、他の<profile_*>と一致するようにしました。

<profile_GLES> / <technique> / <annotate>子要素は、現在、無制限です。

バグ K-206 を解決します。

<profile_GLSL> / <technique>は、現在、オプションの<annotate>子要素を持ちます。

バグ K-206 を解決します。

<annotate>要素は、現在、<profile_GLSL>の<technique>のオプションの最初の子要素です。

```
<xs:element name="annotate" type="fx_annotate_common"
    minOccurs="0" maxOccurs="unbounded"/>
```

<rigid_constraint> / <technique_common>子要素は、現在、デフォルト値を持ちます。

バグ K-169 と K-210 を解決します。

<rigid_contraint> / <technique_common>内で、以下に示すように、デフォルト値をいくつかの子要素に追加しました。

```
<xs:element name="technique_common">
 <xs:complexType>
    <xs:sequence>
       <xs:element name="enabled" default="true" minOccurs="0">
          . . .
       <xs:element name="interpenetrate" default="false" minOccurs="0">
          . . .
       <xs:element name="limits" minOccurs="0">
          <xs:complexType>
             <xs:sequence>
                <xs:element name="swing_cone_and_twist" minOccurs="0">
                   <xs:complexType>
                      <xs:sequence>
                         <xs:element name="min" type="TargetableFloat3"
                                 default="0.0 0.0 0.0" minOccurs="0">
                         </xs:element>
                         <xs:element name="max" type="TargetableFloat3"
                                 default="0.0 0.0 0.0" minOccurs="0">
                         </xs:element>
                      </xs:sequence>
                   </xs:complexType>
                </xs:element>
                <xs:element name="linear" minOccurs="0">
                   <xs:complexType>
                      <xs:sequence>
                         <xs:element name="min" type="TargetableFloat3"
                                 default="0.0 0.0 0.0" minOccurs="0">
                         </xs:element>
                         <xs:element name="max" type="TargetableFloat3"
                                 default="0.0 0.0 0.0" minOccurs="0">
                         . . .
       <xs:element name="spring" minOccurs="0">
          <xs:complexType>
```

```
<xs:sequence>
   <xs:element name="angular" minOccurs="0">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="stiffness" type="TargetableFloat"
                    default="1.0" minOccurs="0">
            </xs:element>
            <xs:element name="damping" type="TargetableFloat"
                    default="0.0" minOccurs="0">
            </xs:element>
            <xs:element name="target_value" type="TargetableFloat"
                    default="0.0" minOccurs="0">
            . . .
   <xs:element name="linear" minOccurs="0">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="stiffness" type="TargetableFloat"
                    default="1.0" minOccurs="0">
            </xs:element>
            <xs:element name="damping" type="TargetableFloat"
                    default="0.0" minOccurs="0">
            </xs:element>
            <xs:element name="target_value" type="TargetableFloat"
                    default="0.0" minOccurs="0">
```

<setparam>のrefの属性タイプが変更されました。

バグ K-203 を解決します。

<setparam>の ref 属性は、現在、**<setparam>**の親要素が以下の要素である場合に xs:token です。

- **<surface>** / **<generator>**（GLSL スコープでは xs:string であり、CG スコープでは、xs:NCName でした）
- **<profile_CG>** / **<technique>**（xs:string でした）
- **<instance_effect>**（xs:string でした）
- **<profile_GLSL>** / **<technique>**（xs:string でした）
- **<usertype>**（xs:string でした）

注意：親要素が**<profile_GLES>** / **<technique>**である場合は、xs:NCName のままです。

<surface>は、現在、オプションの<format_hint>子要素を持ちます。

バグ K-111 を解決します。

<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類似したフォーマットを選択できるように、オプションの**<format_hint>**がフォーマットの重要な特徴を記述します。**<format_hint>**の有効な子要素は、**<channels>**、**<range>**、**<precision>**、**<option>**、および**<extra>**です。

<surface>は、現在、オプションの初期化子要素を持ちます。

バグ K-108、K-126 および K-178 を解決します。

サーフェスを初期化するための方法を指定します。有効な初期化子要素は、**<init_as_null>**、**<init_as_target>**、**<init_cube>**、**<init_volume>**、**<init_planar>**、**<init_from>**です。

GLSLスコープ内の<surface>は、現在、オプションの<generator>子要素を持ちます。

バグ K-343 を解決します。

これは、<glsl_param_type>が、現在、以下を含むというスキーマ変更からもたらされました。

```
<xs:element name="surface" type="glsl_surface_type"/>
```

これは以下を置き換えたものです。

```
<xs:element name="surface" type="fx_surface_common"/>
```

<surface>の子要素<init_from>は、もはやIDの配列ではありません。

バグ K-43 を解決します（<init_from>内で ID の配列を使っている COLLADA ドキュメントは変更する必要があります）。

<xs:extension base="xs:IDREFS">から<xs:extension base="xs:IDREF">に変わりました。

<surface>の子要素<format>は、もはや必要ありません。

バグ K-106 を解決します。

まったく記述しないか、または 1 つ記述することができます。

<technique_hint>は、現在、オプションのprofile属性を持ちます。

バグ K-136 を解決します。

どの API プロファイルを対象としているのかヒントを指定します。プロファイルは、「profile_」にこの属性値を付加して作成されます。たとえば、profile_CG を選択するには、profile="CG"と指定します。

<technique_hint>のplatform属性は、オプションになりました。

バグ K-136 を解決します。

<transparent>は、現在、オプションのopaque属性を持ちます。

バグ K-397、バグ K-400、および P-39 を解決します。

<blinn>、<constant>、<lambert>、<phong>内で、子要素<transparent>は、現在、以下の有効な値を持つオプションの opaque 属性を持ちます。

- A_ONE（デフォルト）：カラーのアルファチャネルから透明度情報を取り出します。ここで値 1.0 は不透明を表します。
- RGB_ZERO：各チャネルが独立して調整される、カラーの R（赤）、G（緑）、B（青）チャネルから透明度情報を取り出します。ここで値 0.0 は不透明を表します。

<transparent>もしくは<transparency>が存在する場合、透明レンダリングが有効になるので、レンダラをアルファブレンディングモードに切り替える必要があります。2 つの値を組み合わせる方法は、以下の式で定義されます。 これらの式を使って、<transparent>の不透明設定に基づき正しい結果を求めます。ここで、*fb* はフレームバッファ（すなわち、レンダリング中のものの背後にあるイメージ）であり、*mat* は透明度計算前のマテリアルカラーです。

- A_ONE 不透明モードでは、以下のとおりです。

```
result.r = fb.r * (1.0f - transparent.a * transparency) + mat.r *
    (transparent.a * transparency)
```

```
result.g = fb.g * (1.0f - transparent.a * transparency) + mat.g *
   (transparent.a * transparency)
result.b = fb.b * (1.0f - transparent.a * transparency) + mat.b *
   (transparent.a * transparency)
result.a = fb.a * (1.0f - transparent.a * transparency) + mat.a *
   (transparent.a * transparency)
```

- RGB_ZERO 不透明モードでは、以下のとおりです。

```
result.r = fb.r * (transparent.r * transparency) + mat.r *
   (1.0f -transparent.r * transparency)
result.g = fb.g * (transparent.g * transparency) + mat.g *
   (1.0f -transparent.g * transparency)
result.b = fb.b * (transparent.b * transparency) + mat.b *
   (1.0f -transparent.b * transparency)
result.a = fb.a * (luminance(transparent.rgb) * transparency) + mat.a *
   (1.0f - luminance(transparent.rgb) * transparency)
```

ここで、luminance（輝度）は、ISO/CIE カラー標準（『ITU-R Recommendation BT.709-4』を参照）に基づき、以下のようにカラーチャネルを平均して 1 つの値にする関数です。

```
luminance = (color.r * 0.212671) +
            (color.g * 0.715160) +
            (color.b * 0.072169)
```

<transparent>と**<transparency>**間の、それぞれ相互に与える影響は、以下のとおりです。

- ~~**<transparent>**は存在するが**<transparency>**は存在しない場合は、**<transparent>**で定義された不透明度にも透明度にもその割合には影響を及ぼさないことを意味します。~~
- ~~**<transparency>**は存在するが**<transparent>**が存在しない場合、**<transparency>**は、0 が完全な透明で、1 が完全な不透明であるとする A_ONE 不透明モードを前提とする計算にスケールされます。~~
- **<transparent>**が存在しない場合には、**<transparent>**は式の結果に影響を与えません。不透明モードは、デフォルトの不透明モードのままです。つまり、次の式と同じことになります。

```
transparent = <color> 1.0 1.0 1.0 1.0 </color>
```

- **<transparency>**が存在しない場合、**<transparency>**は式の結果に影響を与えません。つまり、係数が 1.0 であるのと同じことになります。

```
transparency = <float> 1.0 <float>
```

- **<transparent>**と**<transparency>**の両方が存在する場合は、両方ともその機能を発揮します。

~~**<transparent>**または**<transparency>**が指定されていない場合は、以下のように仮定することで、アプリケーションは同等の結果を得ることができます。~~

```
~~transparent = <color> 0.0 0.0 0.0 1.0 </color>~~
~~transparency = <float> 1.0 <float>~~
```

A_ONE は透明度情報が RGB チャネルでなくアルファチャネルからもたらされるということを指定するので、次の例では、カラーは指定され使用されていますが、RGB の値は透明度の計算に対して無視されます。

```
<transparent opaque=A_ONE><color>1 0 0.5 0</color></transparent>
```

<usertype>には、source属性が必須になりました。

バグ K-190 を解決します（<usertype>を含む COLLADA ドキュメントは、source 属性を追加する必要があります）。

以下の属性を追加しました。

```
<xs:attribute name="source" type="xs:NCName" use="required">
```

<usertype>は、現在、オプションの<setparam>子要素を持ちます。

バグ K-191 とバグ K-208 を解決します。

<usertype>のnameの属性タイプが変更されました。

バグ K-203 を解決します。

<usertype>の name 属性は、現在、xs:token です。

バージョン 1.4.1 の既知のスキーマ問題

<surface>の<init_*>子要素は、より柔軟性の少ない識別子を使っています。

バグ K-339。

XML データベース間で FX サーフェス（fx_surface_common または派生タイプ）を移動するときは、関連するイメージもそれと共に移動しなければなりません。これは、<surface>の<init_*>要素が外部的であってもよい URI ではなく、ローカルな IDREF を使用しているからです。

バージョン 1.4.0 以降の仕様変更

- 1.4.1 スキーマ変更を追加しました。
- スキーマに対して仕様内容を検証し、さまざまな誤りや抜けを訂正しました。
- すべての例を更新し訂正しました。
- すべての要素の「属性」および「子要素」部分を再フォーマットし、情報をより鮮明に伝達できるようにしました。
- 資料のいくつかを再編成しました。

仕様書 1.4.1 の正誤表と補遺

記載されているDOUBLE_SIDED の型が正しくありません。

バグ P-16。

ページ 3–5 の「共通用語集」の表で、DOUBLE_SIDED の型を Boolean に変更してください。

<animation>:複数の<animation>のターゲットが同じときにはどうすればよいか？

バグ K-804。

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

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

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

<unit>子要素の<asset>要素の説明が不明瞭です。

バグ K-358。

<unit>は、COLLADA 要素/オブジェクトの距離単位を定義します。この距離単位は、**<asset>**の親要素のスコープ内のすべての空間測定値に対して適用されます。ただし、よりローカルな**<asset>**によって上書きされた場合は除きます。この単位の値は自己記述的であり、実世界の測定値と整合性がとれている必要はありません。そのオプション属性は次のとおりです。

name：距離単位の名前。例：「meter」、「centimeter」、「inches」、「parsec」。この名前は、実際の測定単位名でも、仮想の単位名でもかまいません。

meter：1 距離単位が実世界の何メートルに相当するか。たとえば、名前が「meter」の場合は 1.0、「kilometer」の場合は 1000、「foot」の場合は 0.3048 になります。詳細については、仕様書の「COLLADA フィジックスリファレンス」章の「物理的な単位について」を参照してください。

<bind_material>のシンボル名バインディングはエラーを招きやすいので注意が必要です。

バグ K-436。

ジオメトリの一部が宣言されるときに、そのジオメトリが特定のマテリアルを持つということが必要になる場合があります。たとえば、次のような場合です。

```
<polygons name="leftarm" count="2445" material="bluePaint">
```

この抽象シンボルは、特定のマテリアルインスタンスにバインドされる必要があります。このインスタンス化は、**<bind_material>**要素内の**<instance_geometry>**要素を処理するときにアプリケーションによって行われます。このジオメトリが material 属性に関してスキャンされ、実際のマテリアルオブジェクトが、**<instance_material>**の symbol 属性によって指示されたとおりに、matarial 属性にバインドされます。

```
...
<material id="MyMaterial">
  ...
</material>
...
<geometry>
   ...
   <polygons name="leftarm" count="2445" material="bluePaint">
   ...
```

```
</geometry>
...
<scene>
  ...
  <instance_geometry ...>
    <bind_material>
      <technique_common>
        <instance_material symbol="bluePaint" target="MyMaterial">
          ...
        </instance_material>
      </technique_common>
    </bind_material>
  </instance_geometry>
  ...
</scene>
```

<phong>式が不正確もしくは不明確

バグ K-874。

シェーダの**<phong>**の正しい式は、以下のとおりです。

$$color = < emission > + < ambient > * al + < diffuse > * \max(N \bullet L, 0) + < specular > * \max(R \bullet I, 0)^{<shininess>}$$

それぞれ、以下のような意味となっています。

al－シーンからの環境光の寄与分を表す定数。共通プロファイルでは、**<visual_scene>**の中の**<light>**、**<technique_common>**、**<ambient>**、**<color>**の値の合計になります。

N－法線ベクトル

L－光ベクトル

I－視線ベクトル

H－半角ベクトル。単位視線ベクトルと単位光ベクトルの中間点で、H=(I+L)/2 という式を使って計算される。

R－完全反射ベクトル

<blinn>、<constant>、<lambert>、<phong>のデフォルト色が指定されない

バグ P-27。

スキーマでは、**<blinn>**、**<constant>**、**<lambert>**、**<phong>**シェーダの**<diffuse>**などの子要素について、（**common_color_or_texture_type** の中で）デフォルト色を指定していません。指定されていない子要素（**<diffuse>**など）がある場合には、その部分を除いて、指定されたシェーダ式を適用します。これは、その子要素に対して、明示的に黒を指定したのと同じ結果をもたらします。たとえば、**<diffuse>**のない**<phong>**の式は、次のようになります。

$$color = < emission > + < ambient > * al + < specular > * \max(R \bullet I, 0)^{<shininess>}$$

Axisの<camera>の説明が混乱を招く恐れがあります。

バグ K-259、P-8 および P-13。

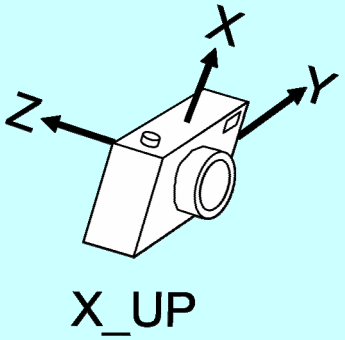
<camera>の説明内での単語「into」の使用方法が、混乱を招く恐れがあります。テキストの真意は次のとおりです（抜粋）。

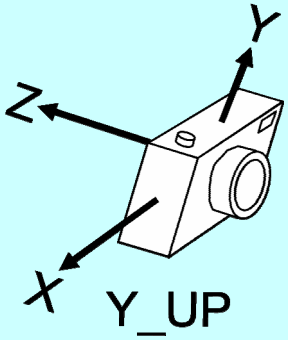
ビジュアルシーン階層またはシーングラフ「の」ビューを宣言します（Declares a view *of* the visual scene hierarchy or scene graph）。

カメラは、ビジュアルシーン「を」見ているビューアの視点を体現します（A camera embodies the eye point of the viewer looking *at* the visual scene）。

カメラの光学装置が、入ってきた光の焦点をイメージ平面「上に」合わせます（The camera optics focuses the incoming light *onto* an image plane）。

以下の図は、**<asset>**要素の**<up_axis>**子要素のさまざまな値に対する軸の向きを示したものです。最初の図は、X_UP **<node>**における Y_UP カメラを示しています。2 番目の図は、Y_UP **<node>**における Y_UP カメラのデフォルトの向きを示しています。





<control_vertices>要素が記載されていない

バグ K-535。

<control_vertices>は、スプラインの制御頂点（CV）を記述します。

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

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

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

<control_vertices>要素には、属性がありません。

<control_vertices>要素は以下の要素と関連があります。

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

<control_vertices>要素には、以下のような子要素があります。子要素がある場合には、下記の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<input>（インデックスなし）	値が **POSITION** である semantic 属性を持つ **<input>**（インデックスなし）要素が、最低でも 1 つは存在する必要があります。主見出し項目を参照してください。	*なし*	1 以上

<extra>	主見出し項目を参照してください。	該当なし	0 以上

COLLADA では、多項式補間として、LINEAR、BEZIER、CARDINAL、HERMITE、BSPLINE、STEP を認識します。これらの識別名は、**<source>**要素の中の**<Name_array>**で使われます。これらの値は、semantic 属性が **INTERPOLATION** になっている**<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、**<control_vertices>**用の**<input>**の semantic として、以下の値を定義しています。

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

注意:

- 混合補間スプラインにおいて、補間の種類が BEZIER もしくは HERMITE であるセグメントが最低 1 つある場合、各制御頂点について、IN_TANGENT 値と OUT_TANGENT 値を 1 つずつ指定する必要があります。
- 子要素のデータ型は、すべて同一である必要があります。たとえば、補間の種類が BEZIER や HERMITE の場合、
  - 有効：POSITION、IN_TANGENT、OUT_TANGENT のすべてを float2 として定義。
  - 無効：POSITION が float2 で、IN_TANGENT または OUT_TANGENT が float3。
- 子要素の間にも制約があります。たとえば、POSITION 子要素の数は、INTERPOLATION 要素の数に等しい必要があります。詳しくは、**<spline>**を参照してください。

<convex_mesh>の子要素は、すべてオプションになりました。

バグ K-382。

前に必要だった子要素**<source>**および**<vertices>**は、現在は、オプションになりました。用法の詳細については、「バージョン 1.4.0 以降のスキーマの変更」セクションを参照してください。

<instance_material>内の<bind>要素には、セマンティックによる場所の特定に関する記述がありません。

バグ K-417。

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 パラメータを検索
- 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検索
- パラメータ名を用いて COLLADA FX を検索
- 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検索

<instance_rigid_constraint>はドキュメント化されていません。

バグ K-332。

<instance_rigid_constraint>要素は、以下の属性を持ちます。

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。これにより、アニメーションに対して**<rigid_constraint>**インスタンスの要素を対象にすることが可能になります。オプション。
name	**xs:NCName**	オプション。
constraint	**xs:NCName**	どの**<rigid_constraint>**をインスタンス化すべきかを示します。必須。

<instance_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	**instance_physics_model**
子要素	下記サブセクションを参照してください。

<instance_rigid_constraint>要素は以下の子要素を持ちます。

名前/例	解説	デフォルト値	出現回数
<extra>	**<instance_rigid_constraint>**に情報を追加するユーザ定義のマルチ表現可能なデータ。	なし	0 以上

<lambert>式が不正確

バグ K-792。

<lambert>の計算が、間違って `color = emission` のように記載されています。正しい式は、以下のとおりです

$$color = < emission > + < ambient > * al + < diffuse > * \max(N \bullet L, 0)$$

<profile_COMMON>の<newparam>の有効な型が記載漏れ

バグ K-1670。

<profile_COMMON>の下の**<newparam>**の有効なパラメータのリストに**<float4>**を追加する必要があります。

<pass>レンダリング状態テーブルにいくつかの情報が存在せず、スペルの誤りがあります。

バグ K-391、K-392、K-401。

- 論理値である **color_material_enable** は、テーブルには存在しません。これは、**color_material** の使用を有効および無効にします。用法の詳細については、「バージョン 1.4.0 以降のスキーマの変更」セクションを参照してください。
- **blend_color**、**depth_bounds**、**line_stipple**、**logic_op_enable** が、GLES 内にありません。
- **clip_plane** は、GLES では bool4 であり、他のプロファイルでは float4 です。
- **stencil_op_separate**（GLES 内にない）はテーブルに存在せずに、子要素 face、fail、zfail、および zpass を持ちます。値とタイプに関する情報については、**stencil_op** および **stencil_mask_separate** を参照してください。
- blend_func：正しい値は DST_ALPHA、ONE_MINUS_DST_ALPHA です。
- blend_equation：正しい値は FUNC_SUBTRACT です。
- stencil_op：正しい値は DECR_WRAP です。

<perspective>/<aspect_ratio>の記述が正しくありません。

バグ P-25。

次の文

> The aspect ratio is defined as the ratio of a pixel's height over the pixel's width; therefore, the aspect ratio can be derived from, or be used to derive, the field of view parameters: **aspect_ratio** = **yfov** / **xfov**.

を、次のように変更してください。

> The aspect ratio is defined as the ratio of the field of view's width over its height; therefore, the aspect ratio can be derived from, or be used to derive, the field of view parameters: **aspect_ratio** = **xfov** / **yfov**.
>
> スキーマでは、以下のような指定を許可しています。
>
> - **<xfov>**を指定
> - **<yfov>**を指定
> - **<xfov>**と**<yfov>**を指定
> - **<xfov>**と**<aspect_ratio>**を指定
> - **<yfov>**と**<aspect_ratio>**を指定
>
> 最初の 2 つの指定では、アプリケーションは、カメラのアスペクト比をビューポートのアスペクト比に基づいて計算することができます。

<polygons>、<polylist>、<trifans>、<triangles>、<tristrips>の属性の例が不正確

バグ K-848。

次の文

> **<polygons>**、**<polylist>**、**<trifans>**、**<triangles>**、**<tristrips>**の例では、material 属性は URL として与えられます。たとえば、
>
> ```
> <polygons count="1" material="#Bricks">
> ```

#の文字を削除してください。

<profile_*>要素の<technique>の子が不完全であるか、もしくは子の不正な<technique> (FX)リストが訂正されていません。

バグ K-97 とバグ K-237。

<profile_*>/<technique>の正しい子要素は以下のとおりで、存在する場合、以下の順に記述されます。

名前	profile_CG	profile_COMMON	profile_GLES	profile_GLSL	出現回数
<asset>	はい	はい	はい	-	0 または 1
<annotate>	はい	-	はい	はい	0 以上
<include>	はい	-	-	はい	0 以上
<code>	はい	-	-	はい	0 以上
<newparam>	はい	はい	はい	はい	0 以上
<setparam>	はい	-	はい	はい	0 以上
<image>	はい	はい	はい	はい	0 以上
<blinn>	-	はい	-	-	1
<constant>	-	はい	-	-	
<lambert>	-	はい	-	-	
<phong>	-	はい	-	-	
<pass>	はい	-	はい	はい	1 以上
<extra>	はい	はい	はい	はい	0 以上

<sampler*>子要素の説明がない

バグ K-449、P-7、P-21。

このセクションでは、**<sampler1D>**、**<sampler2D>**、**<sampler3D>**、**<samplerCUB>**、**<samplerDEPTH>**、**<samplerRECT>**の子要素について説明します。

名前/例	解説	デフォルト値	出現回数
<source> (FX)	xs:NCName 型。**<surface>**の **sid**。**<sampler*>**は、シェーダが**<surface>**の色を解決する方法を定義しています。**<source>**は、読みとる**<surface>**を特定します。	*なし*	1
<wrap_s>	**fx_sampler_wrap_common** 型を参照。	WRAP	0 または 1
<wrap_t>	**fx_sampler_wrap_common** 型を参照。	WRAP	0 または 1
<wrap_p>	**fx_sampler_wrap_common** 型を参照。	WRAP	0 または 1
<minfilter>	テクスチャ最小化。**fx_sampler_filter_common** 列挙型。この値は、ガンマが最小化を示している場合に、テクスチャ値を取得する方法を決めます。	NONE	0 または 1
<magfilter>	テクスチャ拡大。**fx_sampler_filter_common** 列挙型。この値は、ガンマが拡大を示している場合に、テクスチャ値を取得する方法を決めます。	NONE	0 または 1
<mipfilter>	ミップマップフィルタ。**fx_sampler_filter_common** 列挙型。	NONE	0 または 1

名前/例	解説	デフォルト値	出現回数
<border_color>	クランプのあるラップモードで、テクスチャのアドレス空間の端を越えて読み取りを行うと、この色が読み取られます。**fx_color_common** 型（RGBA 順の 4 つの浮動小数点数）。	*なし*	0 または 1
<mipmap_maxlevel>	**xs:unsignedByte** 型。サンプラーが評価するプログレッシブレベルの最大数。	255	0 または 1
<mipmap_bias>	**xs:float** 型。サンプラーがミップマップレベルを評価するために使うガンマ（詳細レベルパラメータ）を補正する。	0	0 または 1
<extra>	主見出し項目を参照してください。	該当なし	0 以上

fx_sampler_filter_common型

列挙型で、有効な値は **NONE**、**NEAREST**、**LINEAR**、**NEAREST_MIPMAP_NEAREST**、**LINEAR_MIPMAP_NEAREST**、**NEAREST_MIPMAP_LINEAR**、**LINEAR_MIPMAP_LINEAR**。

プリミティブにテクスチャを適用するということは、テクスチャ画像空間からフレームバッファ画像空間への写像を意味します。一般に、サンプリングされたテクスチャ画像の再構成が伴います。フレームバッファ空間への写像に伴う一様なワーピングが行われ、さらにフィルタリングされ、最後に、フィルタリング/ワーピング/再構成された画像がリサンプリングされてから、フラグメントに適用されます。

fx_sampler_wrap_common型

ラップモードは、s、t、p テクスチャ座標の解釈に影響します。

ラップモード列挙型は、以下のような OpenGL の識別名に対応します。

ラップモード	OpenGL 識別名	解説
WRAP	**GL_REPEAT**	テクスチャ座標の整数部を無視し、小数部だけ使います。
MIRROR	**GL_MIRRORED_REPEAT**	まず、テクスチャ座標を鏡像反転させます。反転された座標は、**CLAMP_TO_EDGE** の説明にしたがって、強制的に補正されます。
CLAMP	**GL_CLAMP_TO_EDGE**	全ミップマップレベルのテクスチャ座標を強制的に補正して、テクスチャフィルタが境界テクセルをサンプリングしないようにします。 *注：***GL_CLAMP** は、サンプリング境界を越えるあらゆるテクセルを境界色に置き換えます。そのため、**CLAMP_TO_EDGE** の方が適切です。OpenGL ES 仕様からは **GL_CLAMP** 識別名が削除されたので、OpenGL ES ではさらに改善されます。
BORDER	**GL_CLAMP_TO_BORDER**	全ミップマップにおけるテクスチャ座標を強制的に補正して、対応するテクスチャ座標が[0、1]の範囲を充分に外れるフラグメントについて、テクスチャフィルタが常に境界テクセルをサンプリングするようにします。
NONE	**GL_CLAMP_TO BORDER**	NONE に対して定義された動作は、境界が黒であるデカールテクスチャリングと一致します。この計算を **GL_CLAMP_TO_BORDER** にマッピングすることは、最良近似です。

<sampler*>子要素すべてについての詳細は、OpenGL 仕様を参照してください。

<setparam>子要素の一部が不正確

バグ K-700。

<setparam>子要素の正しい順序および説明は、以下のとおりです。

profile_GLSL/technique/setparamの子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<annotate>	主見出し項目を参照してください。	該当なし	0 以上
param_value	以下のいずれかを含む必要があります。 ***glsl_value_type*:** GLSL スコープで有効な値の型については、この章の最後にある「値の型」を参照してください。 **<array>:** 主見出し項目を参照してください。	該当なし	1

profile_CG/technique/setparam、およびusertype/setparamの子要素

注： 以下のいずれかを含む必要があります。

名前/例	解説	デフォルト値	出現回数
cg_value_type	CG スコープで有効な値の型については、この章の最後にある「値の型」を参照してください。	該当なし	注を参照。
<usertype>	主見出し項目を参照してください。	該当なし	注を参照。
<array>	主見出し項目を参照してください。	該当なし	注を参照。
<connect_param>	主見出し項目を参照してください。	*なし*	注を参照。

profile_GLES/technique/setparamの子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<annotate>	主見出し項目を参照してください。	該当なし	0 以上
gles_value_type	GLES スコープで有効な値の型については、この章の最後にある「値の型」を参照してください。	該当なし	1

<instance_effect>/setparamの子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
core_value_type	有効なコア値の型については、この章の最後にある「値の型」を参照してください。	該当なし	1

<generator>/setparamの子要素

子要素が存在する場合には、以下の順序で出現する必要があります。

名前/例	解説	デフォルト値	出現回数
<annotate>	主見出し項目を参照してください。	該当なし	0 以上
value_type	有効な値の型については、この章の最後にある「値の型」を参照してください。コンテキストによって GLSL 値型もしくは CG 値型が有効です。	該当なし	1

<shader>子要素の一部が不正確

バグ K-637。

<shader>子要素の正しい順序および説明は、以下のとおりです。

名前/例	解説	デフォルト値	出現回数
<annotate>	主見出し項目を参照してください。	該当なし	0 以上
<compiler_target>	主見出し項目を参照してください。 <compiler_options>を指定した場合には、この要素も指定する必要があります。	*なし*	0 または 1
<compiler_options>	主見出し項目を参照してください。	*なし*	0 または 1
<name>	主見出し項目を参照してください。	*なし*	1
<bind>（シェーダ）	主見出し項目を参照してください。		0 以上

<skin> / <bind_shape_matrix>の説明が不明確

バグ K-569。

<bind_shape_matrix>には、4×4 の列優先行列を構成する 16 個の浮動小数点数が含まれています。この行列が COLLADA 文書に書き込まれるときには、人間にとって読みやすいように行優先順序で書き込まれます。

<translate>の説明が不明瞭です

バグ K-619。

<translate>内の次の文、

> Translations change the position of objects in a coordinate system without any rotation.

を、次のように変更してください。

> Translations change the position of objects in a local coordinate system.

<transparent>は、現在、オプションのopaque属性を持ちます。

バグ K-406。

用法の詳細については、「バージョン 1.4.0 以降のスキーマの変更」セクションを参照してください。

「COLLADA内の骨格のスキニング」には、さらに説明が必要

バグ K-419。

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

COLLADA内の骨格のスキニング

スキニングは、ジオメトリの頂点を、スケルトンのジョイント（ボーンとも呼ばれる）に対し、線形に重み付けすることにより、ジオメトリを変形させるテクニックです。

概要

スキニング<controller>は、ジオメトリをスケルトンに関連づけます。スケルトンは、休止位置（*バインドポーズ*）にあるとみなされます。バインドポーズというのは、スケルトンがジオメトリにバ

インドされた際の各ジョイントのワールド空間内での位置および方向のことです。このワールド空間は、他のワールド空間座標系と区別するためにバインドポーズ空間とも呼ばれます。

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

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

スキニングの定義

COLLADA 内のスキニングに関係する定義

- バインド形状~~（またはベースメッシュ）~~： **<skin>**要素の **source** 属性によって参照されるメッシュの頂点。
- バインド形状行列： ~~スキニングの前に~~、メッシュがスケルトンにバインドされたときのバインド形状の変換を表す単一の行列。この行列は、バインド形状をオブジェクト空間からバインド空間に変換します。
- ジョイント：スケルトンのボーンは、ジョイントによって定義されます。各ボーンのベースは、次のジョイントにまで伸びています。バインド空間では、ジョイントはバインドポーズの中にあります。スケルトンのジョイントがバインド形状にバインドされた時の位置および方向。**<visual_scene>**の中では、ジョイントの向きは、アクターのポーズとアニメーションによって定められます。ワールド空間内でのジョイントの位置は、メッシュの位置と一致しないことがあります。一致するかどうかは、メッシュをオブジェクト空間に変換するために使われるルート行列に依存します。~~semantic="JOINT"をもつ**<input>**要素によって参照された**<source>**の中で、sid によって指定されたノード。複数の sid は、通常、1 要素の名前が 1 つの sid（ノード）を表す**<Name_array>**内に格納されます。スキンコントローラをインスタンス化したときに、**<skeleton>**要素は、sid ルックアップを開始すべき場所を定義します。関節行列は、ランタイム時にこれらのノードから取得できます。~~
- 重み：特定のジョイントが頂点の最終的な位置に影響を与える度合い。頂点は、任意の数のジョイントに対して、重みの合計が 1 に等しくなるように重み付けされます。頂点の変換は、各ジョイントについて個別に行われます。複数の頂点変換結果は、重みによって線形結合され、スキニング後の頂点が生成されます。~~semantic="WEIGHT"をもつ**<input>**要素によって参照された**<source>**の中の値。通常**<float_array>**に格納され、一度に 1 つの浮動小数点値が取られます。**<vertex_weights>**要素は、スキンが用いる関節と重みの組み合わせを記述します。~~
- 逆バインドポーズ行列：このジョイントにバインド形状がバインドされた際のジョイントのバインド空間変換行列の逆行列。~~semantic="INV_BIND_MATRIX"をもつ**<input>**要素によって参照される**<source>**要素の中の値。通常、<float_array>に格納され、一度に 16 個の浮動小数点値が取り入れられます。**<joints>**要素は、関節をその逆バインド行列に関連付けます。~~

スキニングの式

バインド形状内の各頂点 v のスキニング計算は次のようになります。

$$outv = \sum_{i=0}^{n} \{((v * \mathrm{BSM}) * \mathrm{IBM}i * \mathrm{JM}i) * \mathrm{JW}\}$$

それぞれのパラメータは、以下のような意味となっています。

- *n*：頂点 *v* に影響するジョイントの数
- BSM：バインド形状行列
- IBM*i*：ジョイント *i* の逆バインドポーズ行列
- JM*i*：ジョイント *i* の~~ジョイント~~変換行列
- JW：頂点 *v* に対するジョイント *i* の影響の~~ジョイント重み/~~重み

~~通常の最適化では~~

- ~~(*v* * BSM)は、ロード時に計算され、格納されます。~~

注： *v*、BSM、IBM*i*、JW は、骨格アニメーションに関する定数の一部です。アプリケーションによっては、あらかじめ BSM に IBM*i* を掛けたり、*v* に BSM を掛けておくと便利なことがあります。

式についての注意

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

- ワールド空間スキニングでは、JM*i* は、オブジェクト空間からワールド空間へのジョイントの変換行列です。
- オブジェクト空間スキニングでは、JM*i* は、オブジェクト空間から別のオブジェクト空間へのジョイントの変換行列です。最初のオブジェクト空間変換は、バインド形状が最初に定義されたオブジェクト空間におけるジオメトリの変換です。2 番目のオブジェクト空間変換は、**<instance_controller><skeleton>**によって選択された、対象オブジェクト空間への変換です。

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

ここで参照しているスケルトンの行列がバインド形状行列ではないことに注意することが大切です。この行列は、**<instance_controller><skeleton>**によって参照されている**<visual_scene>**の中の**<node>**であり、同じ値を持たない可能性があります。**<instance_controller><skeleton>**によって参照される**<node>**を使うと、シーン中のスケルトンの位置を決め、アニメーションを行う際に、最大限の柔軟性が得られます。この柔軟性により、シーン中のスキンのバインド空間や、スケルトンのオブジェクト空間に対するあらゆる制限が取り除かれます。これは、アニメーションの位置が、選択したルートノードとの相対的位置で常に決まるためです。

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

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

スケルトンとその**<instance_controller>**に対して同時にアニメーションを行う簡単な方法が 1 つあります。**<skeleton>**のルートの中に**<instance_controller>**を配置すれば、最後の 2 つの行列

は互いに打ち消し合うので、ワールド空間スキニングと同じような解が得られるというわけです。メッシュは、常にスケルトンのあとに続きます。

<profile_COMMON>におけるテクスチャマッピング はさらに説明が必要

バグ K-360、K-449。

このセクションでは、テクスチャ、サンプラ、サーフェス、およびイメージについて概説します。

イメージをテクスチャとして使用するには、次のような要素関係を使用します。

```
texture->sampler->surface->image
```

以下では、最小の要素から順に説明します。

- **<image>**は、埋め込み対象または参照対象となるファイルデータです。これは、従来の 2D 平面の形式（BMP など）でもかまいませんし、複数のイメージ平面から成る複雑な 3D 形式（DDS や OpenEXR など）でもかまいません。イメージ自体はサーフェスではありませんが、サーフェスの形成に必要な情報のすべてがイメージ内に含まれている場合もあります。
- **<surface>**は、1 つ以上のイメージを結合することで、MIP マッピング、キューブ、ボリュームといった 3D ハードウェア概念向けに設計された、単一のまとまりのある構造を形成します。<surface>の詳細については、COLLADA 仕様書を参照してください。
- **<sampler*>**には、**<surface>**から特定の 1D、2D、または 3D 座標のデータを読み込む方法に関する指示が含まれます。これは、**<surface>**を参照し、指定された座標のデータをサンプリングするために実行すべき操作を指定します。サンプラの指示には、座標からイメージへのマッピング方法（ラップやミラーなど）に関する情報が含まれます。さらに、この指示には、座標付近の 1 つ以上のテクセルをまとめて最終的な出力カラーを生成する方法を指示するフィルタリングモードも含まれます。
- profile_COMMON の役割は、サンプラが正しいカラーを取得できるよう、ジオメトリのテクスチャ座標セット（配列）を**<sampler*>**にバインドすることです。**<texture>**上の texcoord 属性は、実際にはセマンティック名です。**<instance_geometry>**の**<instance_material>**<**bind_vertex_input>**によって**<texture>**の texcoord 座標とメッシュのテクスチャ座標配列との間に接続が確立されることが、期待されます。**<bind_vertex_input>**の詳細については、**<instance_material>**を参照してください。

一部の DCC アプリケーションでは、TEXCOORD がサンプラにプラグインされる前にその TEXCOORD を変更するための**<extra>**情報（offsetU、offsetV、rotateUV、ノイズなど）も指定されます。

次に示すのは、**<instance_material>**と、その関連要素を使ってテクスチャ処理を行うことで、**<sampler2D>**パラメータ経由で指定された**<image>**を持つマテリアルをインスタンス化する例です。

```
...
<image id="image_id">
  <init_from>image_file.dds</init_from>
</image>
...
<effect id="effect_id">
 ...
  <profile_COMMON>
    <technique sid="technique_sid">
      <newparam sid="surface_param_id">
        <surface>
          <init_from>image_id</init_from>
          ...
        </surface>
      </newparam>
      <newparam sid="sampler2D_param_id">
```

```
          <sampler2D>
            <source>surface_param_id</source>
            ...
          </sampler2D>
        </newparam>
        <lambert>
          <diffuse>
            <texture texture="sampler2D_param_id" texcoord="myUVs"/>
          </diffuse>
        </lambert>
        ...
</effect>
...
<material id="material_id">
  <instance_effect url="#effect_id" />
</material>
...
<geometry id="geometry_id">
  ...
  <input semantic="TEXCOORD" source="#..." offset=".." />
  <triangles material="material_symbol" count="...">
  ...
</geometry>
...
<scene>
  ...
  <instance_geometry url="#geometry_id">
    <bind_material>
        <technique_common>
          <instance_material symbol="material_symbol" target="#material_id">
              <bind_vertex_input semantic="myUVs" input_semantic="TEXCOORD" />
          </instance_material>
        </technique_common>
    </bind_material>
  </instance_geometry>
  ...
</scene>
```

曲線補間はさらに説明が必要

バグ K-374、K-535、K-830、P-23、P-24。

このセクションでは追加情報として、**<~~mesh~~geometry>/<spline>**〉曲線と**<animation>/<sampler>**曲線の実装について詳しく説明し、1.4.0/1.4.1 の *COLLADA* 仕様に含まれる情報の明確化を図ります。

概要

<~~mesh~~geometry>/<spline>と**<animation>/<sampler>**はどちらも、曲線を定義します。前者が画面に表示可能な曲線を表すのに対し、後者はアニメーションの作成に用いる曲線を表します。

COLLADA では、**<input>**要素に対して、曲線の補間に必要なデータを識別するための semantic 属性が定義されています。この属性の値としては、POSITION、INTERPOLATION、LINEAR_STEPS、INPUT、OUTPUT、IN_TANGENT、OUT_TANGENT、CONTINUITY などがあります。さらに、ソースの中で**<Name_array>**を使用すれば、処理対象となる曲線の種類をアプリケーションから指定できます。共通プロファイルには、値として BEZIER、LINEAR、BSPLINE、HERMITE が定義されています。このセクションでは、これらのセマンティックや曲線名が COLLADA でどのように使用されるかについて説明します。

スプライン曲線（<~~mesh~~geometry >/<spline>）

COLLADA のアニメーション曲線の仕様(**<animation>/<sampler>**)は、3 次多項式曲線(**<~~mesh~~geometry>/<spline>**)の描画プリミティブのデータフロー定義から派生しています。

曲線は複数のセグメントを使って定義されます。各セグメントは 2 つの端点によって定義されます。あるセグメントの終了点は、その次のセグメントの開始点でもあります。segment[i]の端点は、POSITION[i]と POSITION[i+1]で与えられます。したがって、n 個のセグメントを持つ曲線には、n+1 個の位置が含まれます。各点は、2 次元（2-D）と 3 次元（3-D）のどちらで定義してもかまいません。

端点間における曲線の振る舞いは、特定の補間方式と追加の係数によって決まります。各セグメントの補間時には、セグメントごとに異なる方式を使用できます。通常、あるセグメントの補間方式はその開始点に関連付けられます。したがって、segment[i]の補間方式は INTERPOLATION[i]に格納されています。ある n 個のセグメントから成る曲線があるとします。この曲線が開いている場合、INTERPOLATION[n+1]は使用されません。一方、この曲線が閉じている場合（最後のセグメントの終了点が最初のセグメントの開始点に接続されている場合）、INTERPOLATION[n+1]がその余分なセグメントの補間方式となります。**<spline>**要素の closed 属性は、曲線が閉じているのか（true）、開いているのか（false：これがデフォルト）のどちらなのかを示します。

LINEAR_STEPS はオプションの**<input>**セマンティックであり、曲線をどのくらい正確に補間する必要があるかを示します。一般に、1 つのセグメント内で複雑な曲線を実現するには、ラインセグメントの再分割による近似を行います。その再分割の回数は LINEAR_STEPS によって与えられます。

次に、COLLADA におけるスプライン定義の例を示します。

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

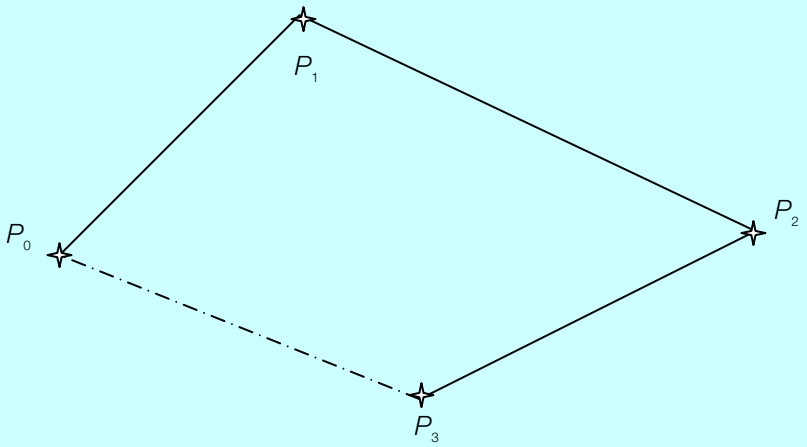
CONTINUITY はオプションの**<input>**セマンティックであり、曲線の設計時に正接がどのように拘束されたかを示します。CONTINUITY の有効な値は次のとおりです。

- *C0*（C-zero）：点に関して連続：2 つの曲線が結合箇所で同じ点を共有します。
- *C1*（C-one）：連続導関数：2 つの曲線が結合箇所で同じパラメトリック導関数を共有します。
- *G1*（Geometric continuity：ジオメトリ連続）：C0 と同じですが、さらに正接が同じ方向に向く必要があります。

1 次スプライン

線形補間がもっとも単純です。なぜなら、あるセグメントの曲線が、その開始点と終了点の間の直線になるからです。セグメント内に制御点を追加する必要はまったくありません。

以下の図は、3 つのセグメントで構成された閉じた**<spline>**を示したものです。ここでは、4 つの位置(P_0, P_1, P_2, P_3)の各ペア間で LINEAR 補間が使用されています。これは閉じたスプラインであるため、P_3 と P_0 の間に最後の（4 番目の）セグメントが存在しています。



1 次スプラインの式は、次のようになります。

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

この式の別の表現方法としては、次のように行列形式を用いる方法があります。

$$L(s) = SMC$$
$$S = \begin{bmatrix} s & 1 \end{bmatrix}$$
$$M = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$$
$$C = \begin{bmatrix} P_0 & P_1 \end{bmatrix}$$

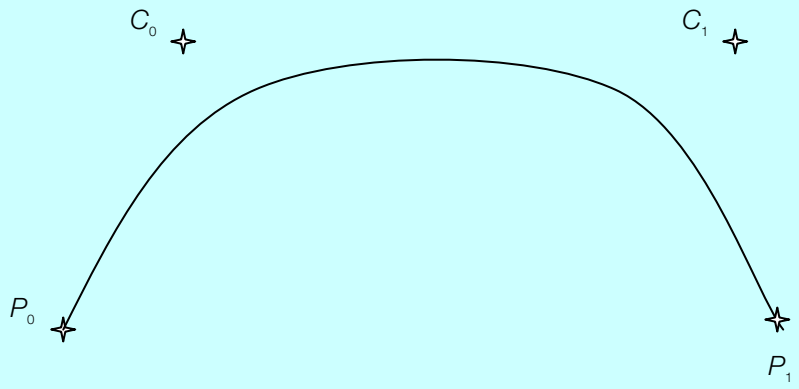
COLLADA では、LINEAR の segment[i]に対するジオメトリベクトルは、次のように定義されます。

- P_0 = POSITION[i]
- P_1 = POSITION[i+1]

ベジェ/エルミートスプライン

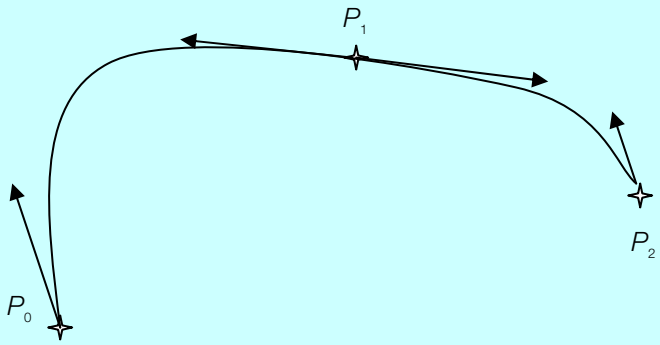
セグメントを 3 次ベジェスプラインにすることも可能です。ただしその場合、曲線の振る舞いを制御するための点が、追加で 2 つ必要となります。以下の例は、BEZIER を使って補間した 1 つのセグメントを示したものです。ここでも先ほどと同じ開始点と終了点(P_0, P_1)が存在していますが、それに加え、曲線計算に関する追加情報を提供する 2 つの制御点(C_0 と C_1)も存在しています。

注: COLLADA 1.4.1 がサポートするのは 3 次のベジェ曲線だけです。したがって、セグメントごとに 2 つの制御点が必ず存在することになります。



HERMITE（エルミート）は BEZIER（ベジェ）と等価ですが、エルミートでは制御点 C_0 と C_1 の代わりに正接 T_0 と T_1 が提供されます。

次の図は、3 次エルミート補間を用いた 2 つのセグメントで構成された曲線を示したものです。



点 P_1 には 2 つの正接が関連付けられています。2 番目のセグメントの開始を定義する正接は OUT_TANGENT と呼ばれます。なぜなら、その正接は、P_1 から「出てくる」ことで始まるセグメントに対するものだからです。最初のセグメントの終了を定義する正接は IN_TANGENT と呼ばれます。なぜなら、その正接は、P_1 に「入っていく」ことで終わるセグメントに対するものだからです。

つまり、IN_TANGENT[1]は segment[0]の終了正接、OUT_TANGENT[1]は segment[1]の開始正接です。

HERMITE と BEZIER は同一の多項式補間であり、次の変数変換が成り立ちます。

$T_0 = 3(C_0 - P_0)$

$T_1 = 3(P_1 - C_1)$

式はパラメトリック式として与えられます。0～1 の区間を動くパラメータ s に基づいて、曲線上のすべての点が計算されます。式の値は、s が 0 の場合は P_0、s が 1 の場合は P_1 となります。これらの値の外側では、式は定義されません。

COLLADA では、**<input>**のセマンティック IN_TANGENT と OUT_TANGENT に正接または制御点が格納されます。どちらが格納されるかは補間方式によって決まります。

3 次ベジェスプラインの式は、次のようになります。

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

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

$$B(s) = SMC$$

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

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

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

COLLADA では、BEZIER の segment[i]に対するジオメトリベクトルは、次のように定義されます。

- P_0 = POSITION[i]
- C_0 = OUT_TANGENT[i]
- C_1 = IN_TANGENT[i+1]
- P_1 = POSITION[i+1]

以下に、2 つのセグメントを含む 2-D（X、Y 座標による）ベジェ曲線の COLLADA サンプルを示します。

```
<spline>
  <source id="positions" >
       <float_array count="6" …>
       <technique_common><accessor>
        … <param name="X" offset="0" type="float" …
        … <param name="Y"  offset="1" type="float" …
        </accessor></technique_common>
  </source>
  <source id="interpolations" >
     <Name array count="3"> BEZIER BEZIER BEZIER</Name array>     <!-- 開いた曲線の
場合は最後のエントリを無視 -->
       <technique_common><accessor>
       … <param name="INTERPOLATION" type="Name" …
        </accessor> </technique_common> </source>
  <source id="in_tangents" >
       <float_array count="6" …>  (開いた曲線の場合は最初のエントリを無視)
       <technique_common><accessor>
        … <param name="X" offset="0" type="float" …
        … <param name="Y"  offset="1" type="float" …
          </accessor> </technique_common> </source>
  <source id="out_tangents" >
        <float_array count="6" …> <!-- 開いた曲線の場合は最後のエントリを無視 -->
        <technique_common><accessor>
         … <param name="X" offset="0" type="float" …
         … <param name="Y" offset="1" type="float" …
          </accessor></technique_common> </source>
  <control_vertices>
       <input semantic="POSITION" source = "#positions" />
       <input semantic="INTERPOLATION" source="#interpolations" />
       <input semantic="IN_TANGENT" source="#in_tangents" />
       <input semantic="OUT_TANGENT" source="#out_tangents" />
```

3 次エルミートスプラインの式は、次のようになります。

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

これは、行列形式では次のようになります。

$$H(s) = SMC$$

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

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

$$C = \begin{bmatrix} P_0 & P_1 & T_0 & T_1 \end{bmatrix}$$

$$where:$$

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

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

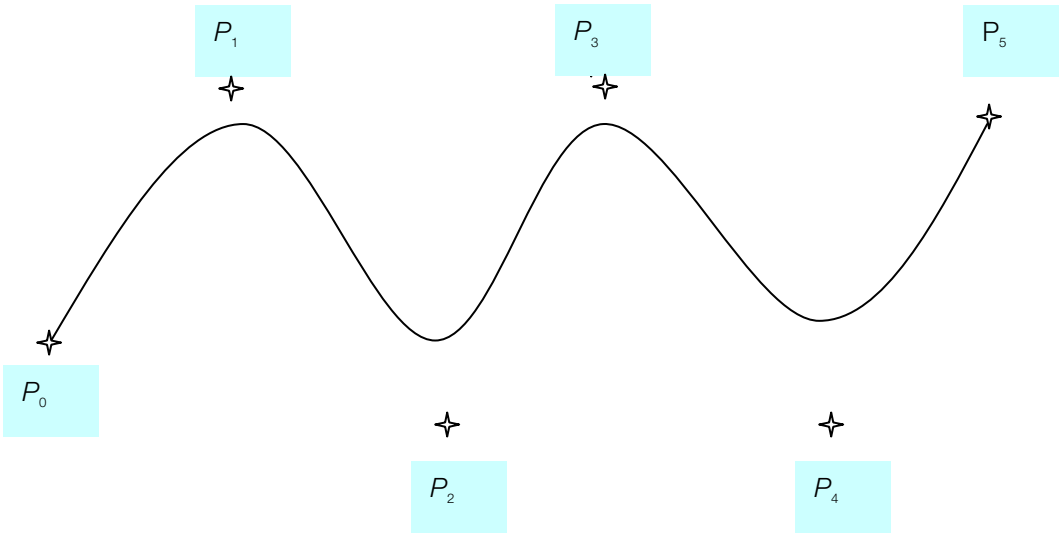
COLLADA では、HERMITE の segment[i]に対するジオメトリベクトルは、次のように定義されます。

- P_0 = POSITION[i]
- P_1 = POSITION[i+1]

- T_0 = OUT_TANGENT[i]
- T_1 = IN_TANGENT[i+1]

B スプライン

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



COLLADA 1.4.1 では一様な 3 次 B スプライン補間が定義されています。端点 P_1 と P_2 の間における曲線の振る舞いは、前の点(P_0)と次の点(P_2)を用いて以下の式で与えられます。開いた曲線の場合、P_0 には前の点というものはないので、P_0 に関する P_1 の対称点が使用されます。最後の点についても同様です。つまり、P_5 に関する P_4 の対称点が、式で使用されます。

P_i と P_{i+1} の間のセグメントに対する B スプラインを記述する式（行列形式）は、次のようになります。

$$B_i(s) = SMC$$

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

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

$$u = 1/6$$

$$C = \begin{bmatrix} P_{i-1} & P_i & P_{i+1} & P_{i+2} \end{bmatrix}$$

COLLADA でこの B スプラインのジオメトリベクトルを定義する際に必要となるのは、次の 2 つの **<input>**要素、POSITION と INTERPOLATION だけです。

- P_i = POSITION[i]
- INTERPOLATION[i]=BSPLINE.

カーディナルスプライン

カーディナルスプラインとは、その正接が端点とテンションパラメータで定義される 3 次エルミートスプラインのことです。正接の計算は次のように、セグメントの前の点と次の点に基づいて行われます。

$T_i = ½ (1-c) (P_{i+1} - P_{i-1})$

ここで、c はテンションパラメータであり、正接の長さを変更するための定数です。このパラメータは、COLLADA 1.4.1 では規定されておらず、アプリケーションによって提供されます。通常、c の値は 0 です。これは Catmull-Rom スプラインを意味します。

カーディナルスプラインは次のような行列形式で表せます（BSPLINE の場合と同じジオメトリベクトル C を使用）。

$$D_i(s) = SMC$$

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

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

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

$$C = \begin{bmatrix} P_{i-1} & P_i & P_{i+1} & P_{i+2} \end{bmatrix}$$

アニメーション曲線（<animation>/<sampler>）

アニメーションでは、対象となるパラメータが時間の経過に従ってどのように変化するかを定義するために曲線が使用されます。曲線の定義は**<~~mesh~~geometry>/<spline>**の場合に似ていますが、アニメーション用のキーが格納された 1 次元の特殊な軸が存在している点が異なります。このキーによって、ある 1 つまたは一連のパラメータが、アニメーション中に時間の経過に従ってどのように変化するかが決定されます。

キーは、通常は TIME 値ですが、ほかの任意の変数にすることもできます。たとえば、列車の車輪の回転をその列車のレール上の位置に関連付けてもかまいません。そうすれば、列車を前後に動かすことで車輪やその他のメカニズムを自動的に動かすことができます。

キー軸内のアニメーションは単調曲線に制限されます。言い換えれば、~~1 つの補間キーに対して存在可能な有効なパラメータ値は、1 つだけです。~~アニメーションキーは INPUT の昇順にソートされる必要があり、重複することはできません。これは、アニメーション曲線が閉じてはいけないことを意味します。

キーは**<source>**配列内に格納され、**<~~mesh~~geometry>/<spline>**のすべての POSITION 入力の最初の軸を置き換えます。同一キー値を持つ複数の曲線を使用すれば、複数のパラメータをアニメートできますそれらのパラメータは OUTPUT 配列によって与えられます。

要約すると、次のようになります。

POSITION[*i*].X = INPUT[*i*]

POSITION[*i*].Y = OUTPUT[*i*]

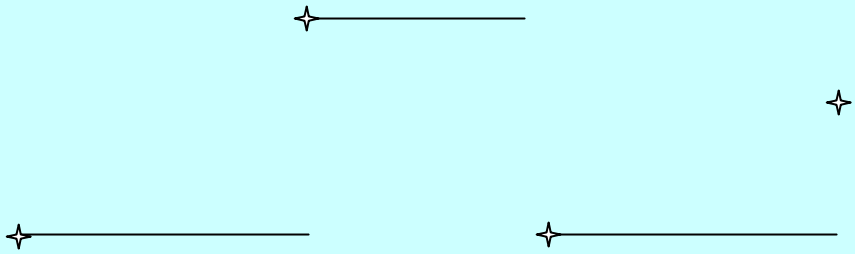
なお、曲線が n 個の場合、曲線 j の点 i は次のようになります。

POSITION[*j*][*i*] = INPUT[*j*][*i*]

POSITION[*j*][*i*+1] = OUTPUT[*j*][*i*]

STEP 補間

アニメーション曲線では、「STEP」という別の補間タイプも使えます。この場合、次のセグメントが始まるまで値がセグメントの最初の点と同じ値に維持されます。具体的には、次の曲線のようになります。



これに対する COLLADA コードは、次のとおりです。

```
<sampler>
<source id="time_axis" >
      <float_array count="4"… >
       … <technique_common><accessor>
            <param name="TIME">
         …</accessor></technique_common>
   …</source>
  <source id="positions" >
       <float_array count="4" …>
        <technique_common>… <accessor>
            <param name="name_of_parameter_animated" type="float" …
         …</accessor></technique_common>
   …</source>
  <source id="interpolations" >
     <Name array count="4"> STEP STEP STEP STEP </Name array>    <!-- 最後のエントリは無視 -->
      <technique_common>… … <accessor>
           <param name="INTERPOLATION" type="Name" …
       …</accessor></technique_common>
   …</source>
  <control_vertices>
      <input semantic="INPUT" source = "#time_axis" />
      <input semantic="OUTPUT" source="#positions" />
      <input semantic="INTERPOLATION" source="#interpolations" />
```

線形アニメーション曲線

LINEAR 補間は STEP に似ていますが、この補間では、各キー値間でパラメータの値が線形補間されます。

ベジェ/エルミートアニメーション曲線

BEZIER/HERMITE 補間は**<spline>**で説明した内容に似ていますが、POSITION 入力セマンティックが存在せず、代わりに INPUT セマンティックと OUTPUT セマンティックが存在している点が異なります。INPUT/OUTPUT セマンティックは常に 1-D パラメータです。前述したとおり、OUTPUT が 2 次元以上の場合には、複数のパラメータが同じキー値に基づいて、それぞれ独立して補間されます。IN_TANGENT セマンティックと OUT_TANGENT セマンティックはそれぞれ、1 つのキー値を持ち、その後にパラメータごとに値を 1 つずつ持ちます。

3 次ベジェ/エルミート補間については、**<spline>**ですでに定義済みのものと同じ式を使用します。ただし、パラメータ j、segment[i]に対して、次のジオメトリベクトルを使用します。

ベジェの場合：

- P_0 = (INPUT[i] , OUTPUT[j][i])
- C_0（または T_0） = (OUT_TANGENT[0][i] , OUT_TANGENT[j][i])
- C_1（または T_1） = (IN_TANGENT[0][i+1], IN_TANGENT[j][i+1])
- P_1 = (INPUT[i+1], OUTPUT[j][i+1])

特殊なケース：1-D の正接値

エクスポータによっては、縮退形式の正接を使って曲線をエクスポートする場合があります。これは COLLADA 仕様ではサポートされていません。このような縮退ケースは、関連するエクスポータのアップデートを重ねるごとに消えていくべきです。以下の情報は、あくまでも参考のために提供するにすぎません。

1-D の正接データというこの特殊なケースでは、OUT_TANGENT と IN_TANGENT はキー値を含まず、したがって OUTPUT 配列と同じ次元を持ちます。

不足しているキー値は、INPUT セグメントが提供するキーの線形補間として提供されます。ジオメトリベクトル値は次のように、通常のアニメーション曲線の場合と同様に提供されます。

- P_0 = (INPUT[i] , OUTPUT[j][i]
- C_0 = (INPUT[i]/3 + INPUT[i+1] *2/3 , OUT_TANGENT[j][i])
- C_1 = (INPUT[i]*2/3 + INPUT[i+1]/3, IN_TANGENT[j][i+1])
- P_1 = (INPUT[i+1], OUTPUT[j][i+1])

B スプライン/カーディナルアニメーション曲線

BSPLINE/CARDINAL 曲線については、先に説明したのと同じ原則が当てはまります。POSITION は、INPUT と OUTPUT を結合することで得られます。これらのアニメーション曲線には、先に定義したのと同じ式が適用されます。

第 3 章の「アドレス構文」にはsidについての説明が必要。

バグ K-1886

このセクションでは、*COLLADA 1.4.1 仕様*の第 3 章「スキーマおよびリファレンス概要」の中の「アドレス構文」のセクションに含まれる情報を解説します。

注：このセクションは、1.4.1 リビジョン B リリースノートから新たに追加されたセクションですが、*仕様*からの変更点だけを示します。

target 属性の構文は、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特定の要素の id 属性、もしくは、相対アドレスであることを示すドットセグメント（「.」）です。
- その後に、~~1 つ以上~~任意の数のスコープ付き~~副~~識別子（sid）が続きます。 各副識別子の先頭には、パスの区切り文字として、スラッシュ文字定数（/）が付加されます。 この部分が空の場合には、スラッシュ文字定数は付加されません。~~副~~識別子は、最初の部分によって特定される要素の子要素を指定します。 入れ子になった要素の場合、対象となる要素へのパスを指定するために、複数のスコープ付き~~副~~識別子が使われることもあります。

  スコープ付き識別子（sid）は、**xs:NCName** 型で、その値は、幅優先走査を使って検索されるので、親要素のスコープ内の、同じパスレベルの sid の集合の中で、一意でなくてはならないという制約があります。sid は、**<technique>**要素全体の中では一意でないことがあります。
- *（仕様中のこのセクションの残りの部分は変更なし。）*

<accessor>、<source>、<param>には、さらに説明が必要

バグ K-865

このセクションでは、**<accessor>**、**<source>**、**<param>**の使用例を提供します。

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

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

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

```
struct
{
    float x pos, y pos, z pos, x norm, y norm, z norm, text1 U,
    tex1_V, tex2_U tex2_V;
} my_array[1000];
```

このようなソースが与えられたとします。

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

その入力は以下のとおりだとします。

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

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

X_pos	Y_pos	Z_pos	X_norm	Y_norm	Z_norm	Tex1_U	Tex1_V	Tex2_U	Tex2_v
1.0	2.0	3.0							
4.0	5.0	6.0							
7.0	8.0	9.0							

アクセッサを以下のように変更します。

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

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

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

X_pos	Y_pos	Z_pos	X_norm	Y_norm	Z_norm	Tex1_U	Tex1_V	Tex2_U	Tex2_v
1.0	3.0								
4.0	6.0								
7.0	9.0								

ここで、頂点配列全体を単一の float 配列にパックしたい場合。

```
<source id=positions>
   <float_array name="values" count="30">
      1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30
   </ float _array>
   <technique_common>
      <accessor source="#values" count="3" stride=”10”>
         <param name="A" type="float"/>
         <param name="F" type="float"/>
         <param name="X" type="float"/>
      </accessor>
   </technique_common>
</source>
<source id=normals>
   <technique_common>
      <accessor source="#values" count="3" stride=”10”>
         <param type="float"/>
         <param type="float"/>
         <param type="float"/>
         <param name="A" type="float"/>
         <param name="F" type="float"/>
         <param name="X" type="float"/>
      </accessor>
   </technique_common>
</source>
<source id=texture1>
   <technique_common>
      <accessor source="#values" count="3" stride=”10”>
         <param type="float"/>
         <param type="float"/>
         <param type="float"/>
         <param type="float"/>
         <param type="float"/>
         <param type="float"/>
         <param name="A" type="float"/>
         <param name="F" type="float"/>
      </accessor>
   </technique_common>
</source>
<source id=texture2>
   <technique_common>
     <accessor source="#values" count="3" stride=”10”>
         <param type="float"/>
         <param type="float"/>
         <param type="float"/>
         <param type="float"/>
```

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

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

各**<accessor>**中の**<param>**の数を基にすれば、読み取ったのは 3 次元の位置、3 次元の法線、および、2 次元のテクスチャ座標であると推定できるはずです。

X_pos	Y_pos	Z_pos	X_norm	Y_norm	Z_norm	Tex1_U	Tex1_V	Tex2_U	Tex2_v
1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
11.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	20.0
21.0	22.0	23.0	24.0	25.0	26.0	27.0	28.0	29.0	30.0

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

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