Nodes（ノード）

概要

Nodes（ノード）（`ufbx_node`）は、FBX ファイルの **シーングラフ（Scene Graph）** を構成する要素です。
ノード自体は、変換情報（`ufbx_node.local_transform`）と階層構造（`ufbx_node.parent` / `ufbx_node.children[]`）を保持します。

ノードは **属性（attribute）** によって機能が拡張されます。たとえば、`ufbx_mesh` や `ufbx_light` などです。
1つの属性が複数のノードに参照されることもあり、同じメッシュを異なるトランスフォームでインスタンス化することができます。

ノードは共通属性を直接保持しており、たとえば：

- `ufbx_node.mesh`
- `ufbx_node.light`

などを通じてアクセスできます。
より珍しい属性は `ufbx_node.attrib` に格納されており、`ufbx_as_bone()` のようなヘルパ関数で具体的な型へキャスト（または `NULL`）できます。

逆に、「ノードがどの属性を持つか」を列挙する代わりに、「ある属性がどのノードで使用されているか」を調べることもできます。
各属性（例：`ufbx_mesh`）には `ufbx_element.instances[]` フィールドがあり、それを参照しているすべてのノードを取得できます。

Transforms（変換）

ノードのローカル変換は、**平行移動（translation）・回転（rotation）・スケール（scale）** の組み合わせで表されます。
これらは `ufbx_node.local_transform` に格納され、**親ノードに対する相対的な変換** を表します。

ノードはさらに以下のような便利フィールドも持っています：

- `ufbx_node.node_to_parent`（親ノード座標系への行列）
- `ufbx_node.node_to_world`（ワールド座標系への行列）

FBX の内部変換は非常に複雑ですが、ufbx ではこれを隠蔽するための機能を数多く備えています。
`ufbx_node` のフィールドや `ufbx_evaluate_transform()`、`ufbx_bake_anim()` などを利用すれば、複雑さを意識せずに扱えます。

（FBX の内部的な変換構造については「Node Transforms」セクションで詳細に説明されています。）

Coordinate Spaces（座標系）

FBX ファイルは任意の座標系（軸向きや単位スケール）で保存されている場合があります。
たとえば、前方／右／上方向の軸や単位長（1.0 の意味）が異なるケースです。

これに対応するため、`ufbx_scene.settings` から **軸（axes）** と **単位メートル値（unit_meters）** を取得できます。

また、`ufbx_load_opts.target_axes` および `ufbx_load_opts.target_unit_meters` を設定することで、
読み込んだシーンを希望の座標系に変換することも可能です。

変換方法は `ufbx_load_opts.space_conversion` によって指定します：

定数名	内容
`UFBX_SPACE_CONVERSION_TRANSFORM_ROOT`	ルートノードで空間変換を行う
`UFBX_SPACE_CONVERSION_ADJUST_TRANSFORMS`	各ノードのトランスフォームを補正する
`UFBX_SPACE_CONVERSION_MODIFY_GEOMETRY`	ジオメトリにスケールを焼き込み、軸変換は ADJUST_TRANSFORMS と同様に行う

ufbx では、`ufbx_load_opts.handedness_conversion_axis` を使用して **左右座標系（右手／左手）** の変換も可能です。
通常、FBX は右手座標系が主流なので、右手系を使用する場合は不要です。
左手座標系でシーンをロードする場合は、ミラー変換が必要になります。

また、カメラ（ローカル +X 向き）やライト（デフォルトではローカル -Y 向き）の軸を修正する機能もあります。
これには `ufbx_load_opts.target_camera_axes` および `ufbx_load_opts.target_light_axes` を使用します。

Coordinate Spaces in Files（ファイル内の座標系）

FBX の座標系やエクスポータの違いは、多くのユーザーを混乱させてきました。
（例：「FBX scale 100」「FBX scale 0.01」で検索するとよく出てきます。）

これは ufbx にも影響しますが、いくつかの方法で軽減可能です。

- **Blender**：通常 Z-up メートル単位で動作しますが、FBX 書き出し時に **Y-up / cm単位（100倍）** に変換します。
  `UFBX_SPACE_CONVERSION_ADJUST_TRANSFORMS` を使うと、この100倍スケールを打ち消し、不要なスケーリングを取り除けます。
  一方、`UFBX_SPACE_CONVERSION_MODIFY_GEOMETRY` はジオメトリ自体を0.01倍にスケールして正しい形状を維持しますが、中間ノードにスケール要素が残ります。
- **Maya**：多くの場合 cm 単位がネイティブです。
  こちらでは `UFBX_SPACE_CONVERSION_MODIFY_GEOMETRY` の方が適しており、ノードスケールを変更せずにシーンを正しいスケールへ変換できます。

結論として、**すべてのケースで完全に一貫した変換方法は存在しません。**
ユーザーに変換方法を選択させるオプションを提供するのが望ましいです。

また、軽量ロード（`ufbx_load_opts.ignore_all_content = true`）を使ってシーンを一度読み込み、
`ufbx_metadata.exporter` を確認することで、エクスポータ（Blender / Maya 等）を特定し、
最適な変換方法を事前設定することも可能です。

> ☒ Blender でエクスポートする場合は、**「Apply Scalings」を “FBX Units Scale” に設定**するのが推奨です。
> これにより、追加スケールなしのメートル単位（unit_meters = 1.0）でエクスポートされます。

サンプルコード

```
// シーンをメートル単位・右手Y-upに変換してロード
ufbx_load_opts opts = { 0 };
opts.target_axes = ufbx_axes_right_handed_y_up;
opts.target_unit_meters = 1.0f;
opts.target_camera_axes = ufbx_axes_right_handed_y_up;
opts.target_light_axes = ufbx_axes_right_handed_y_up;

if (prefer_blender) {
    opts.space_conversion = UFBX_SPACE_CONVERSION_ADJUST_TRANSFORMS;
} else {
    opts.space_conversion = UFBX_SPACE_CONVERSION_MODIFY_GEOMETRY;
}
```

Geometry Transforms（ジオメトリ変換）

FBX では **ジオメトリ変換（geometry transform）** と呼ばれる特殊な変換をサポートしています。
これはノード直下のメッシュなどにのみ適用され、子ノードには継承されません。
多くのシーングラフではこの仕組みを直接サポートしていないため、ufbx は代替方法を提供します。

ジオメトリ変換の利用

ジオメトリ変換は `ufbx_node.geometry_transform` に格納されます。
また、補助的な行列：

- `ufbx_node.geometry_to_node`
- `ufbx_node.geometry_to_world`

もあり、特に静的メッシュをワールド座標で扱う際に便利です。

ジオメトリ変換を除去する

非静的シーンでのジオメトリ変換は扱いが難しいため、
`ufbx_load_opts.geometry_transform_handling` によりロード時に削除することも可能です。

定数名	内容

`UFBX_GEOMETRY_TRANSFORM_HANDLING_HELPER_NODES`	補助ノードを挿入して対応（確実だがノード数が増える）
`UFBX_GEOMETRY_TRANSFORM_HANDLING_MODIFY_GEOMETRY`	ジオメトリ変換を頂点データに焼き込み（きれいだが制約あり）
`UFBX_GEOMETRY_TRANSFORM_HANDLING_MODIFY_GEOMETRY_NO_FALLBACK`	補助ノードを絶対に作らない（ただし変換誤差が発生する）

Inherit Modes（継承モード）

FBX では非標準的な変換継承も可能です。
これは `ufbx_node.inherit_mode` によって示されます。

モード	内容
`UFBX_INHERIT_MODE_IGNORE_PARENT_SCALE`	親のスケールを無視
`UFBX_INHERIT_MODE_COMPONENTWISE_SCALE`	スケールと回転を独立して合成

ufbx では、ロード時にこれらを標準的なシーングラフへ変換するオプションを用意しています：

定数	内容
`UFBX_INHERIT_MODE_HANDLING_PRESERVE`	継承モードをそのまま保持（正確だが複雑）
`UFBX_INHERIT_MODE_HANDLING_HELPER_NODES`	補助ノードを追加して対応
`UFBX_INHERIT_MODE_HANDLING_COMPENSATE`	子ノードのスケールを逆補正（できない場合は補助ノード）
`UFBX_INHERIT_MODE_HANDLING_IGNORE`	非標準継承をすべて無視（単純だが不正確）

Pivots（ピボット）

FBX ノードでは、**回転ピボット** と **スケールピボット** を個別に設定できます。
ufbx では、デフォルトでこれらの効果をノードの平行移動へ焼き込みます。

回転ピボットとスケールピボットが同一である場合、
`ufbx_load_opts.pivot_handling = UFBX_PIVOT_HANDLING_ADJUST_TO_PIVOT`
を設定することで、ピボットをジオメトリ変換へ変換することも可能です。
この場合は、`ufbx_load_opts.geometry_transform_handling` も併せて指定します
（例：`UFBX_GEOMETRY_TRANSFORM_HANDLING_MODIFY_GEOMETRY`）。

その他のプロパティ

ノードには変換以外にも以下のプロパティがあります：

- **可視性**：`ufbx_node.visible` でノードを非表示にできる
- **マテリアルの上書き**：`ufbx_node.materials[]` でインスタンスごとに別マテリアルを適用できる

備考

FBX 仕様ではこれらを **“Geometric” transforms（ジオメトリック変換）** と呼びますが、
ufbx では明確化のため **geometry transforms** と呼んでいます。

Revision #2
★Created 28 October 2025 07:31:58 by youe2
Updated 2 June 2026 21:40:01 by youe2