FBX

- ☒ FBX フォーマットの内部構造
- ☒ Node Transforms（ノード変換）

☒ FBX フォーマットの内部構造

このセクションでは、**FBX ファイルフォーマットの内部仕様** について詳しく説明します。
これらの情報は、主に **高度な利用ケース** においてのみ必要となります。
なぜなら、**ufbx** が FBX フォーマット特有の挙動や癖を
内部で自動的に処理するよう設計されているためです。

⚠ 注意

FBX フォーマットは非常に複雑で、
あまり使われないが存在する多くの機能 を持っています。
そのため、FBX ファイルを直接扱う場合は次の点に注意が必要です。

> ☒ 新しい FBX ファイルを読み込むたびに、
> そのファイルが利用している未対応の機能に対応する必要があるかもしれません。

もし自分自身で **インポータと出力ファイルの両方を管理している** 場合は、
必要な機能だけをサポートすれば十分ですが、
そうでない場合は広範囲な仕様への対応が求められます。

☒ **補足**

☒ Node Transforms（ノード変換）

FBX のノード変換は、**一連の変換のチェーン**として構成されています。

もし特別な理由がなければ、FBX 固有の変換モデルを直接扱うよりも、
ufbx が提供する変換表現を使用することを強く推奨します。
詳細は → Elements / Nodes / Transforms を参照してください。

☒ 基本構造（Transform Properties）

FBX 内でのノード変換は、いくつかの**プロパティ**で構成されています。

プロパティ名	意味
`"Lcl Translation"`	親ノードに対する平行移動
`"Lcl Scaling"`	親ノードに対する非一様スケール
`"Lcl Rotation"`	親ノードに対する回転（オイラー角）
`"RotationOrder"`	`"Lcl Rotation"` のオイラー回転順序（`ufbx_rotation_order`）

☒ ピボット・オフセット関連

以下のプロパティによって、回転やスケールの中心・ずれが定義されます。

プロパティ名	内容
`"ScalingPivot"`	スケーリングの中心点
`"ScalingOffset"`	スケール後のオフセット移動
`"RotationPivot"`	回転の中心点
`"RotationOffset"`	回転後のオフセット移動

さらに `"Lcl Rotation"` 以外に、**2種類の補助回転**が存在します：

プロパティ名	内容
`"PreRotation"`	`"Lcl Rotation"` の前に適用される回転（常に XYZ 順）
`"PostRotation"`	`"Lcl Rotation"` の後に適用される**逆回転**（常に XYZ 順）

⚙ 変換チェーンの計算例

以下は、**ufbx の内部補正（adjust transform）を考慮しない**
基本的なノード変換の計算例です。

```
// 手動で `ufbx_node.node_to_parent` を計算
// ※ ufbx固有の調整変換（adjust transform）は考慮しない
Matrix4 get_transform(ufbx_node *node)
{
    ufbx_props *props = &node->props;

    // 主要プロパティを取得（必要に応じてカーブから評価）
    int64_t rotation_order  = ufbx_find_int(props, "RotationOrder", 0);
    Vector3 lcl_translation = ufbx_find_vec3(props, "Lcl Translation", ufbx_zero_vec3);
```

```
    Vector3 lcl_scaling     = ufbx_find_vec3(props, "Lcl Scaling", ufbx_zero_vec3);
    Vector3 lcl_rotation    = ufbx_find_vec3(props, "Lcl Rotation", ufbx_zero_vec3);
    Vector3 rotation_pivot  = ufbx_find_vec3(props, "RotationPivot", ufbx_zero_vec3);
    Vector3 scaling_pivot   = ufbx_find_vec3(props, "ScalingPivot", ufbx_zero_vec3);
    Vector3 rotation_offset = ufbx_find_vec3(props, "RotationOffset", ufbx_zero_vec3);
    Vector3 scaling_offset  = ufbx_find_vec3(props, "ScalingOffset", ufbx_zero_vec3);
    Vector3 pre_rotation    = ufbx_find_vec3(props, "PreRotation", ufbx_zero_vec3);
    Vector3 post_rotation   = ufbx_find_vec3(props, "PostRotation", ufbx_zero_vec3);

    // オイラー角 → クォータニオン変換
    Quaternion lcl_quat = Quaternion_euler(lcl_rotation, (EulerOrder)rotation_order);
    Quaternion pre_quat = Quaternion_euler(pre_rotation, EulerOrder_XYZ);
    Quaternion post_quat = Quaternion_euler(post_rotation, EulerOrder_XYZ);

    Matrix4 m = Matrix4_identity;

    // スケール適用（ピボット・オフセット含む）
    m = Matrix4_translate(-scaling_pivot) * m;
    m = Matrix4_scale_nonuniform(lcl_scaling) * m;
    m = Matrix4_translate(scaling_pivot) * m;
    m = Matrix4_translate(scaling_offset) * m;

    // 回転適用（PostRotation は反転）
    m = Matrix4_translate(-rotation_pivot) * m;
    m = Matrix4_rotate(Quaternion_inverse(post_quat)) * m;
    m = Matrix4_rotate(lcl_quat) * m;
    m = Matrix4_rotate(pre_quat) * m;
    m = Matrix4_translate(rotation_pivot) * m;
    m = Matrix4_translate(rotation_offset) * m;

    // 最後に平行移動
    m = Matrix4_translate(lcl_translation) * m;

    return m;
}
```

☒ Adjust Transforms（調整変換）

FBX の座標系やピボットの扱いに応じて、
ufbx は内部でいくつかの「調整変換（adjust transform）」を自動的に付与します。
これらを考慮しないと、見た目上の座標がずれることがあります。

主な adjust transform の対応表

機能	対応するフィールド
`UFBX_SPACE_CONVERSION_ADJUST_TRANSFORMS`	`adjust_pre_rotation`, `adjust_pre_scale`
`UFBX_SPACE_CONVERSION_MODIFY_GEOMETRY`	`adjust_pre_rotation`, `adjust_translation_scale`
`UFBX_PIVOT_HANDLING_ADJUST_TO_PIVOT`	`adjust_pre_translation`
`UFBX_INHERIT_MODE_HANDLING_COMPENSATE`	`adjust_post_scale`
`ufbx_load_opts.target_camera_axes`	`adjust_post_rotation`
`ufbx_load_opts.target_light_axes`	`adjust_post_rotation`

`ufbx_node.has_adjust_transform` が `true` の場合は、
ノードが非単位（≠ identity）の調整変換を持っています。
ただし、常に適用しても安全です。

☒ Adjust 変換対応版の完全計算例

```
// ufbx固有の adjust transform を考慮した `ufbx_node.node_to_parent` の計算
Matrix4 get_transform(ufbx_node *node)
{
```

```
    ufbx_props *props = &node->props;

    // プロパティの取得
    int64_t rotation_order  = ufbx_find_int(props, "RotationOrder", 0);
    Vector3 lcl_translation = ufbx_find_vec3(props, "Lcl Translation", ufbx_zero_vec3);
    Vector3 lcl_scaling     = ufbx_find_vec3(props, "Lcl Scaling", ufbx_zero_vec3);
    Vector3 lcl_rotation    = ufbx_find_vec3(props, "Lcl Rotation", ufbx_zero_vec3);
    Vector3 rotation_pivot  = ufbx_find_vec3(props, "RotationPivot", ufbx_zero_vec3);
    Vector3 scaling_pivot   = ufbx_find_vec3(props, "ScalingPivot", ufbx_zero_vec3);
    Vector3 rotation_offset = ufbx_find_vec3(props, "RotationOffset", ufbx_zero_vec3);
    Vector3 scaling_offset  = ufbx_find_vec3(props, "ScalingOffset", ufbx_zero_vec3);
    Vector3 pre_rotation    = ufbx_find_vec3(props, "PreRotation", ufbx_zero_vec3);
    Vector3 post_rotation   = ufbx_find_vec3(props, "PostRotation", ufbx_zero_vec3);

    // ufbx 特有の調整パラメータ
    Vector3 adjust_pre_translation = node->adjust_pre_translation;
    Quaternion adjust_pre_rotation = node->adjust_pre_rotation;
    Quaternion adjust_post_rotation = node->adjust_post_rotation;
    float adjust_pre_scale = (float)node->adjust_pre_scale;
    float adjust_post_scale = (float)node->adjust_post_scale;
    float adjust_translation_scale = (float)node->adjust_translation_scale;

    // クォータニオン化
    Quaternion lcl_quat = Quaternion_euler(lcl_rotation, (EulerOrder)rotation_order);
    Quaternion pre_quat = Quaternion_euler(pre_rotation, EulerOrder_XYZ);
    Quaternion post_quat = Quaternion_euler(post_rotation, EulerOrder_XYZ);

    Matrix4 m = Matrix4_identity;

    // 後処理（post-adjust）
    m = Matrix4_rotate(adjust_post_rotation) * m;
    m = Matrix4_scale(adjust_post_scale) * m;

    // スケール
    m = Matrix4_translate(-scaling_pivot) * m;
    m = Matrix4_scale_nonuniform(lcl_scaling) * m;
    m = Matrix4_translate(scaling_pivot) * m;
    m = Matrix4_translate(scaling_offset) * m;

    // 回転
    m = Matrix4_translate(-rotation_pivot) * m;
    m = Matrix4_rotate(Quaternion_inverse(post_quat)) * m;
    m = Matrix4_rotate(lcl_quat) * m;
    m = Matrix4_rotate(pre_quat) * m;
    m = Matrix4_translate(rotation_pivot) * m;
    m = Matrix4_translate(rotation_offset) * m;

    // 平行移動
    m = Matrix4_translate(lcl_translation) * m;

    // 前処理（pre-adjust）
    m = Matrix4_translate(adjust_pre_translation) * m;
    m = Matrix4_rotate(adjust_pre_rotation) * m;
    m = Matrix4_scale(adjust_pre_scale) * m;

    // translation のみスケーリング
    m.m03 *= adjust_translation_scale;
    m.m13 *= adjust_translation_scale;
    m.m23 *= adjust_translation_scale;

    return m;
}
```

☒ 実際の実装例：
`ufbxi_get_transform()` on GitHub

☒ Geometric Transforms（ジオメトリ変換）

ジオメトリ変換（*geometry transforms*）は、
ノードの**子ノードには影響せず、ノード自身の内容（例：メッシュ）だけを変換**します。

詳細は → Elements / Nodes / Geometry Transforms

プロパティ名	内容
`"GeometricTranslation"`	ノード内コンテンツの平行移動
`"GeometricScaling"`	ノード内コンテンツのスケール
`"GeometricRotation"`	ノード内コンテンツの回転（常に XYZ 順）

☒ Geometry Transform の計算例

```
// 手動で `ufbx_node.geometry_to_node` を計算
Matrix4 get_geometry_transform(ufbx_node *node)
{
    ufbx_props *props = &node->props;

    Vector3 geo_translation = ufbx_find_vec3(props, "GeometricTranslation", ufbx_zero_vec3);
    Vector3 geo_scaling     = ufbx_find_vec3(props, "GeometricScaling", ufbx_zero_vec3);
    Vector3 geo_rotation    = ufbx_find_vec3(props, "GeometricRotation", ufbx_zero_vec3);

    Quaternion geo_quat = Quaternion_euler(geo_rotation, EulerOrder_XYZ);

    Matrix4 m = Matrix4_identity;
    m = Matrix4_scale_nonuniform(geo_scaling) * m;
    m = Matrix4_rotate(geo_quat) * m;
    m = Matrix4_translate(geo_translation) * m;

    return m;
}
```

☒ **備考**
このドキュメントは MIT / Public Domain ライセンスのもとで公開された
ufbx (c) 2020 Samuli Raivio の内容を翻訳・整形しています。