☒ Animation（アニメーション）

FBX ファイル内のアニメーションは、**スタック (`ufbx_anim_stack`)** と呼ばれる **レイヤー (`ufbx_anim_layer`)** の集合として表現されます。
各スタックは、ファイル内の 1 つのアニメーションクリップ（または「テイク」）に対応します。

ufbx ではアニメーションを `ufbx_anim` デスクリプタを通して扱います。
これにより、「複数レイヤーを合成したスタック」か「単一レイヤー」かを
統一されたインターフェースで選択・評価できます。

☒ `ufbx_anim` インスタンスを取得できる場所

- `ufbx_scene.anim`：シーンのデフォルトアニメーションスタック
- `ufbx_anim_stack.anim`：合成済みアニメーションスタック
- `ufbx_anim_layer.anim`：個別のアニメーションレイヤー
- `ufbx_create_anim()`：カスタムアニメーションデスクリプタを作成

☒ 評価（Evaluation）

ufbx はファイル内のアニメーションカーブを直接扱えますが、
それらを手動で解釈するのは非常に複雑です。

主な理由は次の通りです：

- ☒ 時間が非線形な **キュービック補間カーブ**
- ☒ 回転順序を持つ **オイラー角回転カーブ**
- ⚙ **プリ／ポスト回転、ピボット、オフセット** などの複雑なノード変換
- ☒ **アニメーションレイヤーの合成**

こうした複雑さを避けるため、
これらを内部で処理してくれる *ufbx* の「評価ユーティリティ」を使用することが推奨されます。

特に、下記の「アニメーションのベイク（baking）」は、
複雑な FBX アニメーションを扱いやすい形式に変換する良い出発点です。

☒ シーン全体の評価（Scene Evaluation）

最も簡単な方法は **`ufbx_evaluate_scene()`** を使うことです。
これは指定した時刻におけるアニメーションをすべて適用し、
新しい `ufbx_scene` を生成する「重い」関数です。

結果のシーンは通常の `ufbx_scene` と同様に扱えます。

☒ アニメーションのベイク（Animation Baking）

FBX ファイル内のアニメーションは **`ufbx_bake_anim()`** を使って
より単純な形式に「ベイク」できます。

この関数は、トランスフォームアニメーションを
線形補間のトラック（translation／quaternion rotation／scale）に変換します。
トランスフォーム以外のプロパティも線形補間キーとしてベイクされます。

ベイクアルゴリズムは単純な再サンプリングではなく、
キーフレームの頻度などを考慮して効率的に処理します。
ただし、**キュービック補間**や**オイラー回転**は
クォータニオンに再サンプリングする必要があります。

☒ サンプリング設定

- `ufbx_bake_opts.resample_rate`：再サンプリングレートの設定
- `ufbx_bake_opts.minimum_sample_rate`：高頻度キーをスキップして二重サンプリングを防止

☒ C 言語例

```
void bake_animation(ufbx_scene *scene, ufbx_anim *anim)
{
    ufbx_baked_anim *bake = ufbx_bake_anim(scene, anim, NULL, NULL);
    assert(bake);

    for (size_t i = 0; i < bake->nodes.count; i++) {
        ufbx_baked_node *bake_node = &bake->nodes.data[i];
        ufbx_node *scene_node = scene->nodes.data[bake_node->typed_id];

        printf("  node %s:\n", scene_node->name.data);
        printf("    translation: %zu keys\n", bake_node->translation_keys.count);
        printf("    rotation: %zu keys\n", bake_node->rotation_keys.count);
        printf("    scale: %zu keys\n", bake_node->scale_keys.count);
    }

    ufbx_free_baked_anim(bake);
}

void bake_animations(ufbx_scene *scene)
{
    for (size_t i = 0; i < scene->anim_stacks.count; i++) {
        ufbx_anim_stack *stack = scene->anim_stacks.data[i];
        printf("stack %s:\n", stack->name.data);
        bake_animation(scene, stack->anim);
    }
}
```

☒ C++ 例

```
void bake_animation(ufbx_scene *scene, ufbx_anim *anim)
{
    ufbx_baked_anim *bake = ufbx_bake_anim(scene, anim, NULL, NULL);
    assert(bake);

    for (const ufbx_baked_node &bake_node : bake->nodes) {
        ufbx_node *scene_node = scene->nodes[bake_node.typed_id];
        printf("  node %s:\n", scene_node->name.data);
        printf("    translation: %zu keys\n", bake_node.translation_keys.count);
        printf("    rotation: %zu keys\n", bake_node.rotation_keys.count);
        printf("    scale: %zu keys\n", bake_node.scale_keys.count);
    }

    ufbx_free_baked_anim(bake);
}

void bake_animations(ufbx_scene *scene)
{
    for (ufbx_anim_stack *stack : scene->anim_stacks) {
        printf("stack %s:\n", stack->name.data);
        bake_animation(scene, stack->anim);
    }
}
```

☒ Rust 例

```
fn bake_animation(scene: &ufbx::Scene, anim: &ufbx::Anim) {
    let bake = ufbx::bake_anim(scene, anim, ufbx::BakeOpts::default())
        .expect("expected to bake animation");

    for bake_node in &bake.nodes {
        let scene_node = &scene.nodes[bake_node.typed_id as usize];
```

```
        println!("  node {}:", scene_node.element.name);
        println!("    translation: {} keys", bake_node.translation_keys.len());
        println!("    rotation: {} keys", bake_node.rotation_keys.len());
        println!("    scale: {} keys", bake_node.scale_keys.len());
    }
}

fn bake_animations(scene: &ufbx::Scene) {
    for stack in &scene.anim_stacks {
        println!("stack {}:", stack.element.name);
        bake_animation(scene, &stack.anim);
    }
}
```

⚙ トランスフォーム・プロパティの評価

ufbx は個々の要素を特定の時刻で評価するための低レベル API も提供しています。

関数名	内容
`ufbx_evaluate_transform()`	ノードの位置・回転（クォータニオン）・スケールを評価
`ufbx_evaluate_blend_weight()`	ブレンドシェイプのウェイトを評価
`ufbx_evaluate_prop()`	任意の FBX プロパティ値を評価
`ufbx_evaluate_props()`	要素全体のプロパティをまとめて評価

さらに低レベルの関数：

関数名	内容
`ufbx_evaluate_anim_value_real()` / `ufbx_evaluate_anim_value_vec3()`	`ufbx_anim_value` を評価
`ufbx_evaluate_curve()`	単一の `ufbx_anim_curve` を評価

Revision #1
★Created 28 October 2025 23:55:46 by youe2
Updated 16 June 2026 09:33:13 by youe2