

UNIT28: SKINNED MESH – BLENDING ANIMATION

【学習要項】

- ☐ Blending animation
- ☐ Appending animations

【演習手順】

1. skinned_mesh オブジェクトをアニメーション情報を持たない。¥¥resources¥¥AimTest¥¥MNK_Mesh.fbx に変更する
※MNK_Mesh.fbx は右手系・Z 軸アップ・センチメートル単位・三角形化済み

2. skinned_mesh クラスに append_animations メンバ関数を追加する

```
1: bool skinned_mesh::append_animations(const char* animation_filename, float sampling_rate)
2: {
3:     FbxManager* fbx_manager{ FbxManager::Create() };
4:     FbxScene* fbx_scene{ FbxScene::Create(fbx_manager, "") };
5:
6:     FbxImporter* fbx_importer{ FbxImporter::Create(fbx_manager, "") };
7:     bool import_status{ false };
8:     import_status = fbx_importer->Initialize(animation_filename);
9:     _ASSERT_EXPR_A(import_status, fbx_importer->GetStatus().GetErrorString());
10:    import_status = fbx_importer->Import(fbx_scene);
11:    _ASSERT_EXPR_A(import_status, fbx_importer->GetStatus().GetErrorString());
12:
13:    fetch_animations(fbx_scene, animation_clips, sampling_rate);
14:
15:    fbx_manager->Destroy();
16:
17:    return true;
18: }
```

3. skinned_mesh オブジェクト生成後 append_animations メンバ関数を呼び出す

※Aim_Space.fbx はアニメーションデータだけを持ったファイルである
skinned_meshes[0] = make_unique<skinned_mesh>(device.Get(), "¥¥resources¥¥AimTest¥¥MNK_Mesh.fbx");
skinned_meshes[0]->append_animations("¥¥resources¥¥AimTest¥¥Aim_Space.fbx", 0);

4. 実行し、アニメーションが再生されていることを確認する

※10 フレーム間隔で正面・上・下・左・左上・左下・右・右上・右下の9方向を狙っているポーズを持っている

5. skinned_mesh クラスに blend_animations メンバ関数を追加する

```
1: void skinned_mesh::blend_animations(const animation::keyframe* keyframes[2], float factor,
2:     animation::keyframe& keyframe)
3: {
4:     size_t node_count{ keyframes[0]->nodes.size() };
5:     keyframe.nodes.resize(node_count);
6:     for (size_t node_index = 0; node_index < node_count; ++node_index)
7:     {
8:         XMVECTOR S[2]{
9:             XMLoadFloat3(&keyframes[0]->nodes.at(node_index).scaling),
10:            XMLoadFloat3(&keyframes[1]->nodes.at(node_index).scaling) };
11:         XMStoreFloat3(&keyframe.nodes.at(node_index).scaling, XMVectorLerp(S[0], S[1], factor));
12:
13:         XMVECTOR R[2]{
14:             XMLoadFloat4(&keyframes[0]->nodes.at(node_index).rotation),
15:            XMLoadFloat4(&keyframes[1]->nodes.at(node_index).rotation) };
16:         XMStoreFloat4(&keyframe.nodes.at(node_index).rotation, XMQuaternionSlerp(R[0], R[1], factor));
17:
18:         XMVECTOR T[2]{ XMLoadFloat3(&keyframes[0]->nodes.at(node_index).translation),
19:            XMLoadFloat3(&keyframes[1]->nodes.at(node_index).translation) };
20:         XMStoreFloat3(&keyframe.nodes.at(node_index).translation, XMVectorLerp(T[0], T[1], factor));
21:     }
22: }
```

6. framework クラスの render メンバ関数を変更する

※10 フレーム毎に正面(0)・上(10)・下(20)・左(30)・左上(40)・左下(50)・右(60)・右上(70)・右下(80)の9のポーズがある
※フレーム番号:40 は左上、フレーム番号:80 は右下のポーズである

```
* 1:  #if 0
2:    int clip_index{ 0 };
3:    int frame_index{ 0 };
4:    static float animation_tick{ 0 };
5:
6:    animation& animation{ skinned_meshes[0]->animation_clips.at(clip_index) };
7:    frame_index = static_cast<int>(animation_tick * animation.sampling_rate);
8:    if (frame_index > animation.sequence.size() - 1)
9:    {
10:        frame_index = 0;
11:        animation_tick = 0;
12:    }
13:    else
14:    {
15:        animation_tick += elapsed_time;
16:    }
17:    animation::keyframe& keyframe{ animation.sequence.at(frame_index) };
*18: #else
*19:    animation::keyframe keyframe;
*20:    const animation::keyframe* keyframes[2]{
*21:        &skinned_meshes[0]->animation_clips.at(0).sequence.at(40),
*22:        &skinned_meshes[0]->animation_clips.at(0).sequence.at(80)
*23:    };
*24:    skinned_meshes[0]->blend_animations(keyframes, 0.5f, keyframe);
*25:    skinned_meshes[0]->update_animation(keyframe);
*26: #endif
27:    skinned_meshes[0]->render(immediate_context.Get(), world, material_color, &keyframe);
```

7. 実行し、正面を狙っているポーズであることを確認する

8. 7. 24:行目の定数値を変数に変更し、ImGui から実行時に変更できるようにする

【評価項目】

- ☐ モーションブレンド
- ☐ アニメーションデータの追加