UNIT18: SKINNED MESH - GEOMETRY

【学習要項】
☐Scene
☐Node
☐Mesh

【演習手順】
1．シェーダーファイルの追加
①HLSL ヘッダーファイル(skinned_mesh.hlsli)

```
struct VS_IN
{
    float4 position : POSITION;
    float4 normal : NORMAL;
    float2 texcoord : TEXCOORD;
};
struct VS_OUT
{
    float4 position : SV_POSITION;
    float4 world_position : POSITION;
    float4 world_normal : NORMAL;
    float2 texcoord : TEXCOORD;
    float4 color : COLOR;
};
cbuffer OBJECT_CONSTANT_BUFFER : register(b0)
{
    row_major float4x4 world;
    float4 material_color;
};
cbuffer SCENE_CONSTANT_BUFFER : register(b1)
{
    row_major float4x4 view_projection;
    float4 light_direction;
    float4 camera_position;
};
```

②頂点シェーダーファイル(skinned_mesh_vs.hlsl)

```
#include "skinned_mesh.hlsli"
VS_OUT main(VS_IN vin)
{
    VS_OUT vout;
    vout.position = mul(vin.position, mul(world, view_projection));
    vout.world_position = mul(vin.position, world);
    vin.normal.w = 0;
    vout.world_normal = normalize(mul(vin.normal, world));
    vout.texcoord = vin.texcoord;
    vout.color = material_color;
    return vout;
}
```

③ピクセルシェーダーファイル(skinned_mesh_ps.hlsl)

```
#include "skinned_mesh.hlsli"
float4 main(VS_OUT pin) : SV_TARGET
{
  float3 N = normalize(pin.world_normal.xyz);
  float3 L = normalize(-light_direction.xyz);
  float3 diffuse = max(0, dot(N, L));
  return float4(diffuse, 1) * pin.color;
}
```

2．skinned_mesh クラスの実装
①skinned_mesh クラスにメッシュ構造体を定義する

```
struct mesh
```

```
{
    uint64_t unique_id{ 0 };
    std::string name;
    // 'node_index' is an index that refers to the node array of the scene.
    int64_t node_index{ 0 };

    std::vector<vertex> vertices;
    std::vector<uint32_t> indices;

private:
    Microsoft::WRL::ComPtr<ID3D11Buffer> vertex_buffer;
    Microsoft::WRL::ComPtr<ID3D11Buffer> index_buffer;
    friend class skinned_mesh;
};
std::vector<mesh> meshes;
```

② skinned_mesh クラスに fetch_meshes メンバ関数を定義・実装する

```
void skinned_mesh::fetch_meshes(FbxScene* fbx_scene, std::vector<mesh>& meshes)
{
    for (const scene::node& node : scene_view.nodes)
    {
        if (node.attribute != FbxNodeAttribute::EType::eMesh)
        {
            continue;
        }

        FbxNode* fbx_node{ fbx_scene->FindNodeByName(node.name.c_str()) };
        FbxMesh* fbx_mesh{ fbx_node->GetMesh() };

        mesh& mesh{ meshes.emplace_back() };
        mesh.unique_id = fbx_mesh->GetNode()->GetUniqueID();
        mesh.name = fbx_mesh->GetNode()->GetName();
        mesh.node_index = scene_view.indexof(mesh.unique_id);

        const int polygon_count{ fbx_mesh->GetPolygonCount() };
        mesh.vertices.resize(polygon_count * 3LL);
        mesh.indices.resize(polygon_count * 3LL);

        FbxStringList uv_names;
        fbx_mesh->GetUVSetNames(uv_names);
        const FbxVector4* control_points{ fbx_mesh->GetControlPoints() };
        for (int polygon_index = 0; polygon_index < polygon_count; ++polygon_index)
        {
            for (int position_in_polygon = 0; position_in_polygon < 3; ++position_in_polygon)
            {
                const int vertex_index{ polygon_index * 3 + position_in_polygon };

                vertex vertex;
                const int polygon_vertex{ fbx_mesh->GetPolygonVertex(polygon_index, position_in_polygon) };
                vertex.position.x = static_cast<float>(control_points[polygon_vertex][0]);
                vertex.position.y = static_cast<float>(control_points[polygon_vertex][1]);
                vertex.position.z = static_cast<float>(control_points[polygon_vertex][2]);

                if (fbx_mesh->GetElementNormalCount() > 0)
                {
                    FbxVector4 normal;
                    fbx_mesh->GetPolygonVertexNormal(polygon_index, position_in_polygon, normal);
                    vertex.normal.x = static_cast<float>(normal[0]);
                    vertex.normal.y = static_cast<float>(normal[1]);
                    vertex.normal.z = static_cast<float>(normal[2]);
                }
                if (fbx_mesh->GetElementUVCount() > 0)
                {
                    FbxVector2 uv;
```

```
48:                     bool unmapped_uv;
49:                     fbx_mesh->GetPolygonVertexUV(polygon_index, position_in_polygon,
50:                         uv_names[0], uv, unmapped_uv);
51:                     vertex.texcoord.x = static_cast<float>(uv[0]);
52:                     vertex.texcoord.y = 1.0f - static_cast<float>(uv[1]);
53:                 }
54:
55:                 mesh.vertices.at(vertex_index) = std::move(vertex);
56:                 mesh.indices.at(vertex_index) = vertex_index;
57:             }
58:
59:         }
60:     }
61: }
```

③skinned_mesh クラスに create_com_objects メンバ関数を定義・実装する

```
 1: void skinned_mesh::create_com_objects(ID3D11Device* device, const char* fbx_filename)
 2: {
 3:     for (mesh& mesh : meshes)
 4:     {
 5:         HRESULT hr{ S_OK };
 6:         D3D11_BUFFER_DESC buffer_desc{};
 7:         D3D11_SUBRESOURCE_DATA subresource_data{};
 8:         buffer_desc.ByteWidth = static_cast<UINT>(sizeof(vertex) * mesh.vertices.size());
 9:         buffer_desc.Usage = D3D11_USAGE_DEFAULT;
10:         buffer_desc.BindFlags = D3D11_BIND_VERTEX_BUFFER;
11:         buffer_desc.CPUAccessFlags = 0;
12:         buffer_desc.MiscFlags = 0;
13:         buffer_desc.StructureByteStride = 0;
14:         subresource_data.pSysMem = mesh.vertices.data();
15:         subresource_data.SysMemPitch = 0;
16:         subresource_data.SysMemSlicePitch = 0;
17:         hr = device->CreateBuffer(&buffer_desc, &subresource_data,
18:             mesh.vertex_buffer.ReleaseAndGetAddressOf());
19:         _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
20:
21:         buffer_desc.ByteWidth = static_cast<UINT>(sizeof(uint32_t) * mesh.indices.size());
22:         buffer_desc.Usage = D3D11_USAGE_DEFAULT;
23:         buffer_desc.BindFlags = D3D11_BIND_INDEX_BUFFER;
24:         subresource_data.pSysMem = mesh.indices.data();
25:         hr = device->CreateBuffer(&buffer_desc, &subresource_data,
26:             mesh.index_buffer.ReleaseAndGetAddressOf());
27:         _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
28: #if 1
29:         mesh.vertices.clear();
30:         mesh.indices.clear();
31: #endif
32:     }
33:
34:     HRESULT hr = S_OK;
35:     D3D11_INPUT_ELEMENT_DESC input_element_desc[]
36:     {
37:         { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, D3D11_APPEND_ALIGNED_ELEMENT },
38:         { "NORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, D3D11_APPEND_ALIGNED_ELEMENT },
39:         { "TEXCOORD", 0, DXGI_FORMAT_R32G32_FLOAT, 0, D3D11_APPEND_ALIGNED_ELEMENT },
40:     };
41:     create_vs_from_cso(device, "skinned_mesh_vs.cso", vertex_shader.ReleaseAndGetAddressOf(),
42:         input_layout.ReleaseAndGetAddressOf(), input_element_desc, ARRAYSIZE(input_element_desc));
43:     create_ps_from_cso(device, "skinned_mesh_ps.cso", pixel_shader.ReleaseAndGetAddressOf());
44:
45:     D3D11_BUFFER_DESC buffer_desc{};
46:     buffer_desc.ByteWidth = sizeof(constants);
47:     buffer_desc.Usage = D3D11_USAGE_DEFAULT;
48:     buffer_desc.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
```

```
    hr = device->CreateBuffer(&buffer_desc, nullptr, constant_buffer.ReleaseAndGetAddressOf());
    _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
}
```

④skinned_mesh コンストラクタのコードを変更する

```
skinned_mesh::skinned_mesh(ID3D11Device* device, const char* fbx_filename, bool triangulate)
{

        :
        :
        :

    traverse(fbx_scene->GetRootNode());

    fetch_meshes(fbx_scene, meshes);

    fbx_manager->Destroy();

    create_com_objects(device, fbx_filename);
}
```

⑤skinned_mesh クラスに render メンバ関数を定義・実装する

```
void skinned_mesh::render(ID3D11DeviceContext* immediate_context,
    const XMFLOAT4X4& world, const XMFLOAT4& material_color)
{
    for (const mesh& mesh : meshes)
    {
        uint32_t stride{ sizeof(vertex) };
        uint32_t offset{ 0 };
        immediate_context->IASetVertexBuffers(0, 1, mesh.vertex_buffer.GetAddressOf(), &stride, &offset);
        immediate_context->IASetIndexBuffer(mesh.index_buffer.Get(), DXGI_FORMAT_R32_UINT, 0);
        immediate_context->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
        immediate_context->IASetInputLayout(input_layout.Get());

        immediate_context->VSSetShader(vertex_shader.Get(), nullptr, 0);
        immediate_context->PSSetShader(pixel_shader.Get(), nullptr, 0);

        constants data;
        data.world = world;
        data.material_color = material_color;
        immediate_context->UpdateSubresource(constant_buffer.Get(), 0, 0, &data, 0, 0);
        immediate_context->VSSetConstantBuffers(0, 1, constant_buffer.GetAddressOf());

        D3D11_BUFFER_DESC buffer_desc;
        mesh.index_buffer->GetDesc(&buffer_desc);
        immediate_context->DrawIndexed(buffer_desc.ByteWidth / sizeof(uint32_t), 0, 0);
    }
}
```

3．framework クラスの render メンバ関数で　sknned_mesh クラスの render メンバ関数を呼び出す
①拡大縮小（S）・回転（R）・平行移動（T）行列を計算する
②上記３行列を合成しワールド変換行列（world）を作成する
③sknned_mesh クラスの render メンバ関数を呼び出す

```
skinned_meshes[0]->render(immediate_context.Get(), world, { 1, 0, 0, 1 });
```

4．実行し、正立方体が描画される事を確認する

【評価項目】
□頂点位置・法線情報のみを持つ FBX ファイルのロードと描画
□描画色の変更