UNIT19: SKINNED MESH - MATERIAL

【学習要項】
☐Materials
☐Textures

【演習手順】
1．FBX ファイルからメッシュが使用するマテリアル情報（色・テクスチャ）を抽出する
①skinned_mesh クラスにマテリアル構造体を定義する
※必要なヘッダファイルをインクルードする

```
struct material
{
    uint64_t unique_id{ 0 };
    std::string name;

    DirectX::XMFLOAT4 Ka{ 0.2f, 0.2f, 0.2f, 1.0f };
    DirectX::XMFLOAT4 Kd{ 0.8f, 0.8f, 0.8f, 1.0f };
    DirectX::XMFLOAT4 Ks{ 1.0f, 1.0f, 1.0f, 1.0f };

    std::string texture_filenames[4];
    Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> shader_resource_views[4];
};
std::unordered_map<uint64_t, material> materials;
```

②skinned_mesh クラスに fetch_materials メンバ関数を実装する
※スペキュラ(Ks)・アンビエント(Ka)のカラー情報も取得しなさい

```
void skinned_mesh::fetch_materials(FbxScene* fbx_scene,
    std::unordered_map<uint64_t, material>& materials)
{
    const size_t node_count{ scene_view.nodes.size() };
    for (size_t node_index = 0; node_index < node_count; ++node_index)
    {
        const scene::node& node{ scene_view.nodes.at(node_index) };
        const FbxNode* fbx_node{ fbx_scene->FindNodeByName(node.name.c_str()) };

        const int material_count{ fbx_node->GetMaterialCount() };
        for (int material_index = 0; material_index < material_count; ++material_index)
        {
            const FbxSurfaceMaterial* fbx_material{ fbx_node->GetMaterial(material_index) };

            material material;
            material.name = fbx_material->GetName();
            material.unique_id = fbx_material->GetUniqueID();
            FbxProperty fbx_property;
            fbx_property = fbx_material->FindProperty(FbxSurfaceMaterial::sDiffuse);
            if (fbx_property.IsValid())
            {
                const FbxDouble3 color{ fbx_property.Get<FbxDouble3>() };
                material.Kd.x = static_cast<float>(color[0]);
                material.Kd.y = static_cast<float>(color[1]);
                material.Kd.z = static_cast<float>(color[2]);
                material.Kd.w = 1.0f;

                const FbxFileTexture* fbx_texture{ fbx_property.GetSrcObject<FbxFileTexture>() };
                material.texture_filenames[0] =
                    fbx_texture ? fbx_texture->GetRelativeFileName() : "";
            }
            materials.emplace(material.unique_id, std::move(material));
        }
    }
}
```

③skinned_mesh コンストラクタで fetch_materials メンバ関数を呼び出す

```
fetch_materials(fbx_scene, materials);
```

④ create_com_objects メンバ関数にシェーダーリソースビューオブジェクト生成のコードを追加する

※必要なヘッダファイルをインクルードする
※UNIT.10 に掲載した load_texture_from_file 関数を使用した実装例
※UNIT.16 に掲載した make_dummy_texture 関数を使用した実装例

```
 1:  for (std::unordered_map<uint64_t, material>::iterator iterator = materials.begin();
 2:      iterator != materials.end(); ++iterator)
 3:  {
 4:      if (iterator->second.texture_filenames[0].size() > 0)
 5:      {
 6:          std::filesystem::path path(fbx_filename);
 7:          path.replace_filename(iterator->second.texture_filenames[0]);
 8:          D3D11_TEXTURE2D_DESC texture2d_desc;
 9:          load_texture_from_file(device, path.c_str(),
10:              iterator->second.shader_resource_views[0].GetAddressOf(), &texture2d_desc);
11:      }
12:      else
13:      {
14:          make_dummy_texture(device, iterator->second.shader_resource_views[0].GetAddressOf(),
15:              0xFFFFFFFF, 16);
16:      }
17:  }
```

2．skinned_mesh の render メンバ関数でシェーダーリソースビューオブジェクトをピクセルシェーダーにバインドする
※今回はマテリアルは１つだけしかない前提で実装する

```
immediate_context->PSSetShaderResources(0, 1, materials.cbegin()->second.shader_resource_views[0].GetAddressOf());
```

3．ピクセルシェーダー（skinned_mesh_ps.hlsl）を変更する

```
 1:  #define POINT 0
 2:  #define LINEAR 1
 3:  #define ANISOTROPIC 2
 4:  SamplerState sampler_states[3] : register(s0);
 5:  Texture2D texture_maps[4] : register(t0);
 6:  float4 main(VS_OUT pin) : SV_TARGET
 7:  {
 8:      float4 color = texture_maps[0].Sample(sampler_states[ANISOTROPIC], pin.texcoord);
 9:      float3 N = normalize(pin.world_normal.xyz);
10:      float3 L = normalize(-light_direction.xyz);
11:      float3 diffuse = color.rgb * max(0, dot(N, L));
12:      return float4(diffuse, color.a) * pin.color;
13:  }
```

4．framework クラスの initialize メンバ関数で skinned_mesh コンストラクタ引数を.¥¥resources¥¥cube.001.0.fbx に変更する
※cube.001.0.fbx はマテリアルあり、テクスチャ（black-metal-texture.jpg）あり
5．実行しテクスチャが貼られていることを確認する
6．framework クラスの initialize メンバ関数で skinned_mesh コンストラクタ引数を.¥¥resources¥¥cube.001.1.fbx に変更する
※cube.001.1.fbx はマテリアルあり、埋め込みテクスチャ（cube.001.1.fbm¥¥ue.png）あり
※cube.001.1.fbx と同じ階層に cube.001.1.fbm フォルダが自動生成される
7．framework クラスの initialize メンバ関数で skinned_mesh コンストラクタ引数を.¥¥resources¥¥cube.001.2.fbx に変更する
※cube.001.2.fbx はマテリアル（赤）あり、テクスチャなし
①material 構造体のメンバ変数 Kd の値と render メンバ関数引数の material_color を合成する

```
XMStoreFloat4(&data.material_color,
    XMLoadFloat4(&material_color) * XMLoadFloat4(&materials.cbegin()->second.Kd));
```

8．framework クラスの initialize メンバ関数で skinned_mesh コンストラクタ引数を.¥¥resources¥¥cube.000.fbx に変更する
※cube.000.fbx はマテリアルなし、テクスチャなし
①skinned_mesh クラスの render メンバ関数でクラッシュする（materials の要素数が 0 であることが原因）
②この問題を解決する（例えば、ダミーのマテリアルを materials に追加する）

【評価項目】
□テクスチャマッピングされたメッシュの描画（cube.001.0.fbx、cube.001.1.fbx）
□マテリアル色（Kd）がセットされたメッシュの描画（cube.001.2.fbx）
□マテリアル情報がないメッシュの描画（cube.000.fbx）