UNIT34: GLTF MODEL - RENDER MESHES

【学習要項】
□Render meshes
□SoA vs AoS

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

```
 1:  struct VS_IN
 2:  {
 3:    float4 position : POSITION;
 4:    float4 normal : NORMAL;
 5:    float4 tangent : TANGENT;
 6:    float2 texcoord : TEXCOORD;
 7:    uint4 joints : JOINTS;
 8:    float4 weights : WEIGHTS;
 9:  };
10:
11:  struct VS_OUT
12:  {
13:    float4 position : SV_POSITION;
14:    float4 w_position : POSITION;
15:    float4 w_normal : NORMAL;
16:    float4 w_tangent : TANGENT;
17:    float2 texcoord : TEXCOORD;
18:  };
19:
20:  cbuffer PRIMITIVE_CONSTANT_BUFFER : register(b0)
21:  {
22:    row_major float4x4 world;
23:    int material;
24:    bool has_tangent;
25:    int skin;
26:    int pad;
27:  };
28:
29:  cbuffer SCENE_CONSTANT_BUFFER : register(b1)
30:  {
31:    row_major float4x4 view_projection;
32:    float4 light_direction;
33:    float4 camera_position;
34:  };
```

② 頂点シェーダーファイルの追加（gltf_model_vs.hlsl）

```
 1:  #include "gltf_model.hlsli"
 2:
 3:  VS_OUT main(VS_IN vin)
 4:  {
 5:    VS_OUT vout;
 6:
 7:    vin.position.w = 1;
 8:    vout.position = mul(vin.position, mul(world, view_projection));
 9:    vout.w_position = mul(vin.position, world);
10:
11:    vin.normal.w = 0;
12:    vout.w_normal = normalize(mul(vin.normal, world));
13:
14:    float sigma = vin.tangent.w;
15:    vin.tangent.w = 0;
16:    vout.w_tangent = normalize(mul(vin.tangent, world));
17:    vout.w_tangent.w = sigma;
18:
19:    vout.texcoord = vin.texcoord;
20:
```

```
21:   return vout;
22: }
```

③ ピクセルシェーダーファイルの追加（gltf_model_ps.hlsl）

```
 1: #include "gltf_model.hlsli"
 2:
 3: float4 main(VS_OUT pin) : SV_TARGET
 4: {
 5:   float3 N = normalize(pin.w_normal.xyz);
 6:   float3 L = normalize(-light_direction.xyz);
 7:
 8:   float3 color = max(0, dot(N, L));
 9:   return float4(color, 1);
10: }
```

2．シェーダーオブジェクトの定義
① gltf_model クラスのメンバ変数を定義する

```
 1: Microsoft::WRL::ComPtr<ID3D11VertexShader> vertex_shader;
 2: Microsoft::WRL::ComPtr<ID3D11PixelShader> pixel_shader;
 3: Microsoft::WRL::ComPtr<ID3D11InputLayout> input_layout;
 4: struct primitive_constants
 5: {
 6:   DirectX::XMFLOAT4X4 world;
 7:   int material{ -1 };
 8:   int has_tangent{ 0 };
 9:   int skin{ -1 };
10:   int pad;
11: };
12: Microsoft::WRL::ComPtr<ID3D11Buffer> primitive_cbuffer;
```

②gltf_model クラスのコンストラクタでシェーダーオブジェクトの生成を行う
※頂点バッファは SoA (Structure of Array)で構成される（従来は AoS (Array of Structures)）

```
 1: // TODO: This is a force-brute programming, may cause bugs.
 2: const std::map<std::string, buffer_view>& vertex_buffer_views{
 3:   meshes.at(0).primitives.at(0).vertex_buffer_views };
 4: D3D11_INPUT_ELEMENT_DESC input_element_desc[]
 5: {
 6:   { "POSITION", 0, vertex_buffer_views.at("POSITION").format, 0, 0, D3D11_INPUT_PER_VERTEX_DATA, 0},
 7:   { "NORMAL", 0, vertex_buffer_views.at("NORMAL").format, 1, 0, D3D11_INPUT_PER_VERTEX_DATA, 0 },
 8:   { "TANGENT", 0, vertex_buffer_views.at("TANGENT").format, 2, 0, D3D11_INPUT_PER_VERTEX_DATA, 0 },
 9:   { "TEXCOORD", 0, vertex_buffer_views.at("TEXCOORD_0").format, 3, 0, D3D11_INPUT_PER_VERTEX_DATA, 0 },
10:   { "JOINTS", 0, vertex_buffer_views.at("JOINTS_0").format, 4, 0, D3D11_INPUT_PER_VERTEX_DATA, 0 },
11:   { "WEIGHTS", 0,vertex_buffer_views.at("WEIGHTS_0").format, 5, 0, D3D11_INPUT_PER_VERTEX_DATA, 0 },
12: };
13: create_vs_from_cso(device, "gltf_model_vs.cso", vertex_shader.ReleaseAndGetAddressOf(),
14:   input_layout.ReleaseAndGetAddressOf(), input_element_desc, _countof(input_element_desc));
15: create_ps_from_cso(device, "gltf_model_ps.cso", pixel_shader.ReleaseAndGetAddressOf());
16:
17: D3D11_BUFFER_DESC buffer_desc{};
18: buffer_desc.ByteWidth = sizeof(primitive_constants);
19: buffer_desc.Usage = D3D11_USAGE_DEFAULT;
20: buffer_desc.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
21: HRESULT hr;
22: hr = device->CreateBuffer(&buffer_desc, nullptr, primitive_cbuffer.ReleaseAndGetAddressOf());
23: _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
```

3．gltf_model クラスに render メンバ関数を実装する

```
 1: void gltf_model::render(ID3D11DeviceContext* immediate_context, const DirectX::XMFLOAT4X4& world)
 2: {
 3:   using namespace DirectX;
 4:
```

```
    immediate_context->VSSetShader(vertex_shader.Get(), nullptr, 0);
    immediate_context->PSSetShader(pixel_shader.Get(), nullptr, 0);
    immediate_context->IASetInputLayout(input_layout.Get());
    immediate_context->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);

    std::function<void(int)> traverse{ [&](int node_index)->void {
      const node& node{nodes.at(node_index)};
      if (node.mesh > -1)
      {
        const mesh& mesh{ meshes.at(node.mesh) };
        for (std::vector<mesh::primitive>::const_reference primitive : mesh.primitives)
        {
          ID3D11Buffer* vertex_buffers[]{
            primitive.vertex_buffer_views.at("POSITION").buffer.Get(),
            primitive.vertex_buffer_views.at("NORMAL").buffer.Get(),
            primitive.vertex_buffer_views.at("TANGENT").buffer.Get(),
            primitive.vertex_buffer_views.at("TEXCOORD_0").buffer.Get(),
            primitive.vertex_buffer_views.at("JOINTS_0").buffer.Get(),
            primitive.vertex_buffer_views.at("WEIGHTS_0").buffer.Get(),
          };
          UINT strides[]{
            static_cast<UINT>(primitive.vertex_buffer_views.at("POSITION").stride_in_bytes),
            static_cast<UINT>(primitive.vertex_buffer_views.at("NORMAL").stride_in_bytes),
            static_cast<UINT>(primitive.vertex_buffer_views.at("TANGENT").stride_in_bytes),
            static_cast<UINT>(primitive.vertex_buffer_views.at("TEXCOORD_0").stride_in_bytes),
            static_cast<UINT>(primitive.vertex_buffer_views.at("JOINTS_0").stride_in_bytes),
            static_cast<UINT>(primitive.vertex_buffer_views.at("WEIGHTS_0").stride_in_bytes),
          };
          UINT offsets[_countof(vertex_buffers)]{ 0 };
          immediate_context->IASetVertexBuffers(0, _countof(vertex_buffers), vertex_buffers, strides, offsets);
          immediate_context->IASetIndexBuffer(primitive.index_buffer_view.buffer.Get(),
            primitive.index_buffer_view.format, 0);

          primitive_constants primitive_data{};
          primitive_data.material = primitive.material;
          primitive_data.has_tangent = primitive.vertex_buffer_views.at("TANGENT").buffer != NULL;
          primitive_data.skin = node.skin;
          XMStoreFloat4x4(&primitive_data.world,
            XMLoadFloat4x4(&node.global_transform) * XMLoadFloat4x4(&world));
          immediate_context->UpdateSubresource(primitive_cbuffer.Get(), 0, 0, &primitive_data, 0, 0);
          immediate_context->VSSetConstantBuffers(0, 1, primitive_cbuffer.GetAddressOf());
          immediate_context->PSSetConstantBuffers(0, 1, primitive_cbuffer.GetAddressOf());

          immediate_context->DrawIndexed(static_cast<UINT>(primitive.index_buffer_view.count()), 0, 0);
        }
      }
      for (std::vector<int>::value_type child_index : node.children)
      {
        traverse(child_index);
      }
    } };
    for (std::vector<int>::value_type node_index : scenes.at(0).nodes)
    {
      traverse(node_index);
    }
  }
```

4．gltf_model クラスの render メンバ関数を呼び出し、モデルが描画されることを確認する
※glTF のシーンは基本的に右手系・Y 軸アップで記録されている
※別のモデルデータでもテストする

【評価項目】
□モデルの描画