UNIT33: GLTF MODEL - SCENE NODES & MESHES

【学習要項】
□glTF
□tinygltf : Header only C++ tiny glTF library(loader/saver)
□Fetching meshes

【演習手順】
１．tinygltf ライブラリの導入
※https://www.khronos.org/registry/glTF/specs/2.0/glTF-2.0.html
※https://github.com/javagl/gltfOverview/blob/master/gltfOverview2.0.svg
① GitHub から tinygltf ライブラリをダウンロードする
※https://github.com/syoyo/tinygltf

②ダウンロードしたファイルをプロジェクトフォルダに展開する
※以下のコードは展開したフォルダが「tinygltf-release」の場合

２．gltf_model クラスを定義・実装する
① gltf_model クラスの定義（プロジェクトに gltf_model.h を新規追加する）

```
#pragma once
#define NOMINMAX
#include <d3d11.h>
#include <wrl.h>
#include <directxmath.h>
#include "tinygltf-release/tiny_gltf.h"

class gltf_model
{
   std::string filename;
public:
  gltf_model(ID3D11Device* device, const std::string& filename);
  virtual ~gltf_model() = default;
  struct scene
  {
    std::string name;
    std::vector<int> nodes; // Array of 'root' nodes
  };
  std::vector<scene> scenes;
};
```

② gltf_model クラスの実装（プロジェクトに gltf_model.cpp を新規追加する）

```
#include "gltf_model.h"

#define TINYGLTF_IMPLEMENTATION
#define TINYGLTF_NO_EXTERNAL_IMAGE
#define STB_IMAGE_IMPLEMENTATION
#define STB_IMAGE_WRITE_IMPLEMENTATION
#define STBI_MSC_SECURE_CRT
#include "tinygltf-release/tiny_gltf.h"

#include "misc.h"

gltf_model::gltf_model(ID3D11Device* device, const std::string& filename) : filename(filename)
{
  tinygltf::Model gltf_model;

  tinygltf::TinyGLTF tiny_gltf;
  std::string error, warning;
  bool succeeded{ false };
  if (filename.find(".glb") != std::string::npos)
  {
       succeeded = tiny_gltf.LoadBinaryFromFile(&gltf_model, &error, &warning, filename.c_str());
  }
  else if (filename.find(".gltf") != std::string::npos)
  {
       succeeded = tiny_gltf.LoadASCIIFromFile(&gltf_model, &error, &warning, filename.c_str());
  }

  _ASSERT_EXPR_A(warning.empty(), warning.c_str());
  _ASSERT_EXPR_A(error.empty(), warning.c_str());
  _ASSERT_EXPR_A(succeeded, L"Failed to load glTF file");
  for (std::vector<tinygltf::Scene>::const_reference gltf_scene : gltf_model.scenes)
  {
    scene& scene{ scenes.emplace_back() };
    scene.name = gltf_model.scenes.at(0).name;
    scene.nodes = gltf_model.scenes.at(0).nodes;
  }
}
```

３．サンプルモデルの取得

① GitHub から glTF サンプルモデルをダウンロードする

※https://github.com/KhronosGroup/glTF-Sample-Models

②ダウンロードしたファイルをプロジェクトフォルダと同じ階層に展開する

４．gltf_model クラスのインスタンス生成

①framework クラスにメンバ変数を定義する

```
std::unique_ptr<gltf_model> gltf_models[8];
```

②framework クラスの initialize メンバ関数で gltf_model オブジェクトを生成する

```
gltf_models[0] = std::make_unique<gltf_model>(device.Get(),
     "..¥¥glTF-Sample-Models-master¥¥2.0¥¥2CylinderEngine¥¥glTF¥¥2CylinderEngine.gltf");
```

③ビルド・実行してアサーションがでないことを確認する

５．ノードデータの取得

① gltf_model クラスに node 構造体とコンテナ変数を定義する

```
struct node
{
    std::string name;
    int skin{ -1 };  // index of skin referenced by this node
    int mesh{ -1 };  // index of mesh referenced by this node

    std::vector<int> children; // An array of indices of child nodes of this node

    // Local transforms
    DirectX::XMFLOAT4 rotation{ 0, 0, 0, 1 };
    DirectX::XMFLOAT3 scale{ 1, 1, 1 };
    DirectX::XMFLOAT3 translation{ 0, 0, 0 };

    DirectX::XMFLOAT4X4 global_transform{ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 };
};
std::vector<node> nodes;
```

② gltf_model クラスに fetch_nodes メンバ関数を実装する

```
void gltf_model::fetch_nodes(const tinygltf::Model& gltf_model)
{
  for (std::vector<tinygltf::Node>::const_reference gltf_node : gltf_model.nodes)
  {
    node& node{ nodes.emplace_back() };
    node.name = gltf_node.name;
    node.skin = gltf_node.skin;
    node.mesh = gltf_node.mesh;
    node.children = gltf_node.children;
    if (!gltf_node.matrix.empty())
    {
      DirectX::XMFLOAT4X4 matrix;
      for (size_t row = 0; row < 4; row++)
      {
        for (size_t column = 0; column < 4; column++)
        {
          matrix(row, column) = static_cast<float>(gltf_node.matrix.at(4 * row + column));
        }
      }

      DirectX::XMVECTOR S, T, R;
      bool succeed = DirectX::XMMatrixDecompose(&S, &R, &T, DirectX::XMLoadFloat4x4(&matrix));
      _ASSERT_EXPR(succeed, L"Failed to decompose matrix.");

      DirectX::XMStoreFloat3(&node.scale, S);
      DirectX::XMStoreFloat4(&node.rotation, R);
      DirectX::XMStoreFloat3(&node.translation, T);
    }
    else
    {
      if (gltf_node.scale.size() > 0)
      {
        node.scale.x = static_cast<float>(gltf_node.scale.at(0));
        node.scale.y = static_cast<float>(gltf_node.scale.at(1));
        node.scale.z = static_cast<float>(gltf_node.scale.at(2));
      }
      if (gltf_node.translation.size() > 0)
      {
        node.translation.x = static_cast<float>(gltf_node.translation.at(0));
        node.translation.y = static_cast<float>(gltf_node.translation.at(1));
        node.translation.z = static_cast<float>(gltf_node.translation.at(2));
      }
      if (gltf_node.rotation.size() > 0)
      {
        node.rotation.x = static_cast<float>(gltf_node.rotation.at(0));
        node.rotation.y = static_cast<float>(gltf_node.rotation.at(1));
        node.rotation.z = static_cast<float>(gltf_node.rotation.at(2));
        node.rotation.w = static_cast<float>(gltf_node.rotation.at(3));
      }
    }
  }
  cumulate_transforms(nodes);
}
```

③ gltf_model クラスに cumulate_transforms メンバ関数を実装する

```
void gltf_model::cumulate_transforms(std::vector<node>& nodes)
{
  using namespace DirectX;

  std::stack<XMFLOAT4X4> parent_global_transforms;
  std::function<void(int)> traverse{ [&](int node_index)->void
  {
    node& node{nodes.at(node_index)};
    XMMATRIX S{ XMMatrixScaling(node.scale.x, node.scale.y, node.scale.z) };
    XMMATRIX R{ XMMatrixRotationQuaternion(
      XMVectorSet(node.rotation.x, node.rotation.y, node.rotation.z, node.rotation.w)) };
    XMMATRIX T{ XMMatrixTranslation(node.translation.x, node.translation.y, node.translation.z) };
    XMStoreFloat4x4(&node.global_transform, S * R * T * XMLoadFloat4x4(&parent_global_transforms.top()));
    for (int child_index : node.children)
    {
      parent_global_transforms.push(node.global_transform);
      traverse(child_index);
      parent_global_transforms.pop();
    }
  } };
  for (std::vector<int>::value_type node_index : scenes.at(0).nodes)
  {
    parent_global_transforms.push({ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 });
    traverse(node_index);
    parent_global_transforms.pop();
  }
}
```

６．メッシュデータの取得

① gltf_model クラスに mesh 構造体とコンテナ変数を定義する

```
struct buffer_view
{
    DXGI_FORMAT format = DXGI_FORMAT_UNKNOWN;
    Microsoft::WRL::ComPtr<ID3D11Buffer> buffer;
    size_t stride_in_bytes{ 0 };
    size_t size_in_bytes{ 0 };
    size_t count() const
    {
        return size_in_bytes / stride_in_bytes;
    }
};
struct mesh
{
    std::string name;
    struct primitive
    {
        int material;
        std::map<std::string, buffer_view> vertex_buffer_views;
        buffer_view index_buffer_view;
    };
    std::vector<primitive> primitives;
};
std::vector<mesh> meshes;
```

② gltf_model クラスに make_buffer_view メンバ関数を実装する

```
gltf_model::buffer_view gltf_model::make_buffer_view(const tinygltf::Accessor& accessor)
{
  buffer_view buffer_view;
  switch (accessor.type)
  {
  case TINYGLTF_TYPE_SCALAR:
    switch (accessor.componentType)
    {
    case TINYGLTF_COMPONENT_TYPE_UNSIGNED_SHORT:
      buffer_view.format = DXGI_FORMAT_R16_UINT;
      buffer_view.stride_in_bytes = sizeof(USHORT);
      break;
    case TINYGLTF_COMPONENT_TYPE_UNSIGNED_INT:
      buffer_view.format = DXGI_FORMAT_R32_UINT;
      buffer_view.stride_in_bytes = sizeof(UINT);
      break;
    default:
      _ASSERT_EXPR(FALSE, L"This accessor component type is not supported.");
      break;
    }
    break;
  case TINYGLTF_TYPE_VEC2:
    switch (accessor.componentType)
    {
    case TINYGLTF_COMPONENT_TYPE_FLOAT:
      buffer_view.format = DXGI_FORMAT_R32G32_FLOAT;
      buffer_view.stride_in_bytes = sizeof(FLOAT) * 2;
      break;
    default:
      _ASSERT_EXPR(FALSE, L"This accessor component type is not supported.");
      break;
    }
    break;
  case TINYGLTF_TYPE_VEC3:
    switch (accessor.componentType)
    {
    case TINYGLTF_COMPONENT_TYPE_FLOAT:
      buffer_view.format = DXGI_FORMAT_R32G32B32_FLOAT;
      buffer_view.stride_in_bytes = sizeof(FLOAT) * 3;
      break;
    default:
      _ASSERT_EXPR(FALSE, L"This accessor component type is not supported.");
      break;
    }
    break;
  case TINYGLTF_TYPE_VEC4:
    switch (accessor.componentType)
    {
    case TINYGLTF_COMPONENT_TYPE_UNSIGNED_SHORT:
      buffer_view.format = DXGI_FORMAT_R16G16B16A16_UINT;
      buffer_view.stride_in_bytes = sizeof(USHORT) * 4;
      break;
    case TINYGLTF_COMPONENT_TYPE_UNSIGNED_INT:
      buffer_view.format = DXGI_FORMAT_R32G32B32A32_UINT;
      buffer_view.stride_in_bytes = sizeof(UINT) * 4;
      break;
    case TINYGLTF_COMPONENT_TYPE_FLOAT:
      buffer_view.format = DXGI_FORMAT_R32G32B32A32_FLOAT;
      buffer_view.stride_in_bytes = sizeof(FLOAT) * 4;
      break;
    default:
      _ASSERT_EXPR(FALSE, L"This accessor component type is not supported.");
      break;
    }
    break;
  default:
    _ASSERT_EXPR(FALSE, L"This accessor type is not supported.");
    break;
  }
  buffer_view.size_in_bytes = static_cast<UINT>(accessor.count * buffer_view.stride_in_bytes);
  return buffer_view;
}
```

③gltf_model クラスに fetch_meshes メンバ関数を実装する

```
void gltf_model::fetch_meshes(ID3D11Device* device, const tinygltf::Model& gltf_model)
{
  HRESULT hr;
  for (std::vector<tinygltf::Mesh>::const_reference gltf_mesh : gltf_model.meshes)
  {
    mesh& mesh{ meshes.emplace_back() };
    mesh.name = gltf_mesh.name;
    for (std::vector<tinygltf::Primitive>::const_reference gltf_primitive : gltf_mesh.primitives)
    {
      mesh::primitive& primitive{ mesh.primitives.emplace_back() };
      primitive.material = gltf_primitive.material;

      // Create index buffer
      const tinygltf::Accessor& gltf_accessor{ gltf_model.accessors.at(gltf_primitive.indices) };
      const tinygltf::BufferView& gltf_buffer_view{ gltf_model.bufferViews.at(gltf_accessor.bufferView) };

      primitive.index_buffer_view = make_buffer_view(gltf_accessor);

      D3D11_BUFFER_DESC buffer_desc{};
      buffer_desc.ByteWidth = static_cast<UINT>(primitive.index_buffer_view.size_in_bytes);
      buffer_desc.Usage = D3D11_USAGE_DEFAULT;
      buffer_desc.BindFlags = D3D11_BIND_INDEX_BUFFER;
      D3D11_SUBRESOURCE_DATA subresource_data{};
      subresource_data.pSysMem = gltf_model.buffers.at(gltf_buffer_view.buffer).data.data()
        + gltf_buffer_view.byteOffset + gltf_accessor.byteOffset;
      hr = device->CreateBuffer(&buffer_desc, &subresource_data,
        primitive.index_buffer_view.buffer.ReleaseAndGetAddressOf());
      _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));

      // Create vertex buffers
      for (std::map<std::string, int>::const_reference gltf_attribute : gltf_primitive.attributes)
      {
        const tinygltf::Accessor& gltf_accessor{ gltf_model.accessors.at(gltf_attribute.second) };
        const tinygltf::BufferView& gltf_buffer_view{ gltf_model.bufferViews.at(gltf_accessor.bufferView) };

        buffer_view vertex_buffer_view{ make_buffer_view(gltf_accessor) };

        D3D11_BUFFER_DESC buffer_desc{};
        buffer_desc.ByteWidth = static_cast<UINT>(vertex_buffer_view.size_in_bytes);
        buffer_desc.Usage = D3D11_USAGE_DEFAULT;
        buffer_desc.BindFlags = D3D11_BIND_VERTEX_BUFFER;
        D3D11_SUBRESOURCE_DATA subresource_data{};
        subresource_data.pSysMem = gltf_model.buffers.at(gltf_buffer_view.buffer).data.data()
          + gltf_buffer_view.byteOffset + gltf_accessor.byteOffset;
        hr = device->CreateBuffer(&buffer_desc, &subresource_data,
          vertex_buffer_view.buffer.ReleaseAndGetAddressOf());
        _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));

        primitive.vertex_buffer_views.emplace(std::make_pair(gltf_attribute.first, vertex_buffer_view));
      }

      // Add dummy attributes if any are missing.
      const std::unordered_map<std::string, buffer_view> attributes{
        { "TANGENT", { DXGI_FORMAT_R32G32B32A32_FLOAT } },
        { "TEXCOORD_0", { DXGI_FORMAT_R32G32_FLOAT } },
        { "JOINTS_0", { DXGI_FORMAT_R16G16B16A16_UINT } },
        { "WEIGHTS_0", { DXGI_FORMAT_R32G32B32A32_FLOAT } },
      };
      for (std::unordered_map<std::string, buffer_view>::const_reference attribute : attributes)
      {
        if (primitive.vertex_buffer_views.find(attribute.first) == primitive.vertex_buffer_views.end())
        {
          primitive.vertex_buffer_views.insert(std::make_pair(attribute.first, attribute.second));
        }
      }

    }
  }
}
```

④ gltf_model クラスのコンストラクタから fetch_meshes メンバ関数を呼び出す

⑤ビルド・実行してアサーションがでないことを確認する
※デバッガで gltf_model:: meshes の内容を確認しなさい

【評価項目】
□ノードデータの取得
□メッシュデータの取得