UNIT35: GLTF MODEL - MATERIALS

【学習要項】
☐Materials
☐StructuredBuffer

【演習手順】
1．マテリアルの取得
① gltf_model クラスに material 構造体を定義する

```
 1:  struct texture_info
 2:  {
 3:    int index = -1;
 4:    int texcoord = 0;
 5:  };
 6:  struct normal_texture_info
 7:  {
 8:    int index = -1;
 9:    int texcoord = 0;.
10:    float scale = 1;
11:  };
12:  struct occlusion_texture_info
13:  {
14:    int index = -1;
15:    int texcoord = 0;
16:    float strength = 1;
17:  };
18:  struct pbr_metallic_roughness
19:  {
20:    float basecolor_factor[4] = { 1, 1, 1, 1 };
21:    texture_info basecolor_texture;
22:    float metallic_factor = 1;
23:    float roughness_factor = 1;
24:    texture_info metallic_roughness_texture;
25:  };
26:  struct material {
27:    std::string name;
28:    struct cbuffer
29:    {
30:      float emissive_factor[3] = { 0, 0, 0 };
31:      int alpha_mode = 0; // "OPAQUE" : 0, "MASK" : 1, "BLEND" : 2
32:      float alpha_cutoff = 0.5f;
33:      bool double_sided = false;
34:
35:      pbr_metallic_roughness pbr_metallic_roughness;
36:
37:      normal_texture_info normal_texture;
38:      occlusion_texture_info occlusion_texture;
39:      texture_info emissive_texture;
40:    };
41:    cbuffer data;
42:  };
43:  std::vector<material> materials;
44:  Microsoft::WRL::ComPtr<ID3D11ShaderResourceView> material_resource_view;
```

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

```
 1:  void gltf_model::fetch_materials(ID3D11Device* device, const tinygltf::Model& gltf_model)
 2:  {
 3:    for (std::vector<tinygltf::Material>::const_reference gltf_material : gltf_model.materials)
 4:    {
 5:      std::vector<material>::reference material = materials.emplace_back();
 6:
 7:      material.name = gltf_material.name;
 8:
 9:      material.data.emissive_factor[0] = static_cast<float>(gltf_material.emissiveFactor.at(0));
10:      material.data.emissive_factor[1] = static_cast<float>(gltf_material.emissiveFactor.at(1));
```

```
    material.data.emissive_factor[2] = static_cast<float>(gltf_material.emissiveFactor.at(2));

    material.data.alpha_mode = gltf_material.alphaMode == "OPAQUE" ?
      0 : gltf_material.alphaMode == "MASK" ? 1 : gltf_material.alphaMode == "BLEND" ? 2 : 0;
    material.data.alpha_cutoff = static_cast<float>(gltf_material.alphaCutoff);
    material.data.double_sided = gltf_material.doubleSided ? 1 : 0;

    material.data.pbr_metallic_roughness.basecolor_factor[0] =
      static_cast<float>(gltf_material.pbrMetallicRoughness.baseColorFactor.at(0));
    material.data.pbr_metallic_roughness.basecolor_factor[1] =
      static_cast<float>(gltf_material.pbrMetallicRoughness.baseColorFactor.at(1));
    material.data.pbr_metallic_roughness.basecolor_factor[2] =
      static_cast<float>(gltf_material.pbrMetallicRoughness.baseColorFactor.at(2));
    material.data.pbr_metallic_roughness.basecolor_factor[3] =
      static_cast<float>(gltf_material.pbrMetallicRoughness.baseColorFactor.at(3));
    material.data.pbr_metallic_roughness.basecolor_texture.index =
      gltf_material.pbrMetallicRoughness.baseColorTexture.index;
    material.data.pbr_metallic_roughness.basecolor_texture.texcoord =
      gltf_material.pbrMetallicRoughness.baseColorTexture.texCoord;
    material.data.pbr_metallic_roughness.metallic_factor =
      static_cast<float>(gltf_material.pbrMetallicRoughness.metallicFactor);
    material.data.pbr_metallic_roughness.roughness_factor =
      static_cast<float>(gltf_material.pbrMetallicRoughness.roughnessFactor);
    material.data.pbr_metallic_roughness.metallic_roughness_texture.index =
      gltf_material.pbrMetallicRoughness.metallicRoughnessTexture.index;
    material.data.pbr_metallic_roughness.metallic_roughness_texture.texcoord =
      gltf_material.pbrMetallicRoughness.metallicRoughnessTexture.texCoord;

    material.data.normal_texture.index = gltf_material.normalTexture.index;
    material.data.normal_texture.texcoord = gltf_material.normalTexture.texCoord;
    material.data.normal_texture.scale = static_cast<float>(gltf_material.normalTexture.scale);

    material.data.occlusion_texture.index = gltf_material.occlusionTexture.index;
    material.data.occlusion_texture.texcoord = gltf_material.occlusionTexture.texCoord;
    material.data.occlusion_texture.strength =
      static_cast<float>(gltf_material.occlusionTexture.strength);

    material.data.emissive_texture.index = gltf_material.emissiveTexture.index;
    material.data.emissive_texture.texcoord = gltf_material.emissiveTexture.texCoord;
  }

  // Create material data as shader resource view on GPU
  std::vector<material::cbuffer> material_data;
  for (std::vector<material>::const_reference material : materials)
  {
    material_data.emplace_back(material.data);
  }

  HRESULT hr;
  Microsoft::WRL::ComPtr<ID3D11Buffer> material_buffer;
  D3D11_BUFFER_DESC buffer_desc{};
  buffer_desc.ByteWidth = static_cast<UINT>(sizeof(material::cbuffer) * material_data.size());
  buffer_desc.StructureByteStride = sizeof(material::cbuffer);
  buffer_desc.Usage = D3D11_USAGE_DEFAULT;
  buffer_desc.BindFlags = D3D11_BIND_SHADER_RESOURCE;
  buffer_desc.MiscFlags = D3D11_RESOURCE_MISC_BUFFER_STRUCTURED;
  D3D11_SUBRESOURCE_DATA subresource_data{};
  subresource_data.pSysMem = material_data.data();
  hr = device->CreateBuffer(&buffer_desc, &subresource_data, material_buffer.GetAddressOf());
  _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
  D3D11_SHADER_RESOURCE_VIEW_DESC shader_resource_view_desc{};
  shader_resource_view_desc.Format = DXGI_FORMAT_UNKNOWN;
  shader_resource_view_desc.ViewDimension = D3D11_SRV_DIMENSION_BUFFER;
  shader_resource_view_desc.Buffer.NumElements = static_cast<UINT>(material_data.size());
  hr = device->CreateShaderResourceView(material_buffer.Get(),
```

```
76:     &shader_resource_view_desc, material_resource_view.GetAddressOf());
77:   _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
78: }
79:
```

③ gltf_model クラスのコンストラクタから fetch_materials メンバ関数を呼び出す
④ gltf_model クラスの render メンバ関数の先頭で material_resource_view オブジェクトをバインドする

```
immediate_context->PSSetShaderResources(0, 1, material_resource_view.GetAddressOf());
```

2．シェーダーの実装（変更）
① ピクセルシェーダー(gltf_model_ps.hlsl)にマテリアル構造体と構造化バッファを定義する

```
 1: struct texture_info
 2: {
 3:   int index;
 4:   int texcoord;
 5: };
 6: struct normal_texture_info
 7: {
 8:   int index;
 9:   int texcoord;
10:   float scale;
11: };
12: struct occlusion_texture_info
13: {
14:   int index;
15:   int texcoord;
16:   float strength;
17: };
18: struct pbr_metallic_roughness
19: {
20:   float4 basecolor_factor;
21:   texture_info basecolor_texture;
22:   float metallic_factor;
23:   float roughness_factor;
24:   texture_info metallic_roughness_texture;
25: };
26: struct material_constants
27: {
28:   float3 emissive_factor;
29:   int alpha_mode; // "OPAQUE" : 0, "MASK" : 1, "BLEND" : 2
30:   float alpha_cutoff;
31:   bool double_sided;
32:
33:   pbr_metallic_roughness pbr_metallic_roughness;
34:
35:   normal_texture_info normal_texture;
36:   occlusion_texture_info occlusion_texture;
37:   texture_info emissive_texture;
38: };
39: StructuredBuffer<material_constants> materials : register(t0);
```

②ピクセルシェーダー(gltf_model_ps.hlsl)の main 関数を変更する

```
 1: float4 main(VS_OUT pin) : SV_TARGET
 2: {
 3:   material_constants m = materials[material];
 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)) * m.pbr_metallic_roughness.basecolor_factor.rgb;
 9:   return float4(color, 1);
```

```
10: }
```

3．ビルド・実行し正しく描画されていることを確認する

【評価項目】
□マテリアルの取得
□マテリアルの描画