UNIT20: SKINNED MESH - SUBSET

【学習要項】
☐Multimaterials
☐Subsets

【演習手順】
1．複数のマテリアル情報をもつメッシュをロードし描画する
2．マテリアル単位で描画を行うためにの subset 構造体を skinned_mesh::mesh クラスに定義する

```
 1:  struct subset
 2:  {
 3:      uint64_t material_unique_id{ 0 };
 4:      std::string material_name;
 5:
 6:      uint32_t start_index_location{ 0 };
 7:      uint32_t index_count{ 0 };
 8:  };
 9:  std::vector<subset> subsets;
```

3．skinned_mesh クラスの fetch_meshes メンバ関数にコードを追加・変更する

```
 1:  void skinned_mesh::fetch_meshes(FbxScene* fbx_scene, std::vector<mesh>& meshes)
 2:  {
 3:      for (const scene::node& node : scene_view.nodes)
 4:      {
 5:          if (node.attribute != FbxNodeAttribute::EType::eMesh)
 6:          {
 7:              continue;
 8:          }
 9:          FbxNode* fbx_node{ fbx_scene->FindNodeByName(node.name.c_str()) };
10:          FbxMesh* fbx_mesh{ fbx_node->GetMesh() };
11:
12:          mesh& mesh{ meshes.emplace_back() };
13:          mesh.unique_id = fbx_mesh->GetNode()->GetUniqueID();
14:          mesh.name = fbx_mesh->GetNode()->GetName();
15:          mesh.node_index = scene_view.indexof(mesh.unique_id);
16:
*17:          std::vector<mesh::subset>& subsets{ mesh.subsets };
*18:          const int material_count{ fbx_mesh->GetNode()->GetMaterialCount() };
*19:          subsets.resize(material_count > 0 ? material_count : 1);
*20:          for (int material_index = 0; material_index < material_count; ++material_index)
*21:          {
*22:              const FbxSurfaceMaterial* fbx_material{ fbx_mesh->GetNode()->GetMaterial(material_index) };
*23:              subsets.at(material_index).material_name = fbx_material->GetName();
*24:              subsets.at(material_index).material_unique_id = fbx_material->GetUniqueID();
*25:          }
*26:          if (material_count > 0)
*27:          {
*28:              const int polygon_count{ fbx_mesh->GetPolygonCount() };
*29:              for (int polygon_index = 0; polygon_index < polygon_count; ++polygon_index)
*30:              {
*31:                  const int material_index
*32:                      { fbx_mesh->GetElementMaterial()->GetIndexArray().GetAt(polygon_index) };
*33:                  subsets.at(material_index).index_count += 3;
*34:              }
*35:              uint32_t offset{ 0 };
*36:              for (mesh::subset& subset : subsets)
*37:              {
*38:                  subset.start_index_location = offset;
*39:                  offset += subset.index_count;
*40:                  // This will be used as counter in the following procedures, reset to zero
*41:                  subset.index_count = 0;
*42:              }
*43:          }
 44:
 45:          const int polygon_count{ fbx_mesh->GetPolygonCount() };
```

```
 46:         mesh.vertices.resize(polygon_count * 3LL);
 47:         mesh.indices.resize(polygon_count * 3LL);
 48:
 49:         FbxStringList uv_names;
 50:         fbx_mesh->GetUVSetNames(uv_names);
 51:         const FbxVector4* control_points{ fbx_mesh->GetControlPoints() };
 52:         for (int polygon_index = 0; polygon_index < polygon_count; ++polygon_index)
 53:         {
*54:             const int material_index{ material_count > 0 ?
*55:                 fbx_mesh->GetElementMaterial()->GetIndexArray().GetAt(polygon_index) : 0 };
*56:             mesh::subset& subset{ subsets.at(material_index) };
*57:             const uint32_t offset{ subset.start_index_location + subset.index_count };
 58:
 59:             for (int position_in_polygon = 0; position_in_polygon < 3; ++position_in_polygon)
 60:             {
 61:                 const int vertex_index{ polygon_index * 3 + position_in_polygon };
 62:
 63:                     :
 64:                     :
 65:                     :
 66:
 67:                 mesh.vertices.at(vertex_index) = std::move(vertex);
*68:                 mesh.indices.at(static_cast<size_t>(offset) + position_in_polygon) = vertex_index;
*69:                 subset.index_count++;
 70:             }
 71:         }
 72:     }
 73: }
```

4．skinned_mesh クラスの render メンバ関数をサブセット単位で描画するように変更する

```
 1: void skinned_mesh::render(ID3D11DeviceContext* immediate_context,
 2:     const XMFLOAT4X4& world, const XMFLOAT4& material_color)
 3: {
 4:     for (mesh& mesh : meshes)
 5:     {
 6:         uint32_t stride{ sizeof(vertex) };
 7:         uint32_t offset{ 0 };
 8:         immediate_context->IASetVertexBuffers(0, 1, mesh.vertex_buffer.GetAddressOf(), &stride, &offset);
 9:         immediate_context->IASetIndexBuffer(mesh.index_buffer.Get(), DXGI_FORMAT_R32_UINT, 0);
10:         immediate_context->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
11:         immediate_context->IASetInputLayout(input_layout.Get());
12:
13:         immediate_context->VSSetShader(vertex_shader.Get(), nullptr, 0);
14:         immediate_context->PSSetShader(pixel_shader.Get(), nullptr, 0);
15:
16:         constants data;
17:         data.world = world;
18:
*19:         for (const mesh::subset& subset : mesh.subsets)
*20:         {
*21:             const material& material{ materials.at(subset.material_unique_id) };
22:
*23:             XMStoreFloat4(&data.material_color, XMLoadFloat4(&material_color) * XMLoadFloat4(&material.Kd));
24:             immediate_context->UpdateSubresource(constant_buffer.Get(), 0, 0, &data, 0, 0);
25:             immediate_context->VSSetConstantBuffers(0, 1, constant_buffer.GetAddressOf());
26:
*27:             immediate_context->PSSetShaderResources(0, 1, material.shader_resource_views[0].GetAddressOf());
28:
*29:             immediate_context->DrawIndexed(subset.index_count, subset.start_index_location, 0);
*30:         }
31:     }
32: }
```

5．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.002.0.fbx に変更する

※cube.002.0.fbx は３つのマテリアルがあり、各々に埋め込みテクスチャがある

６．実行し３種類のテクスチャが貼られていることを確認する

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.002.1.fbx に変更する

※cube.002.1.fbx は３つのマテリアルがあり、うち１つは埋め込みテクスチャ、残り２つは色情報（白・赤）のみ

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

【評価項目】

□複数のマテリアル情報を持ったメッシュの描画（cube.002.0.fbx、cube.002.1.fbx）

□マテリアル情報がないメッシュの描画（cube.000.fbx）