

## UNIT29: ADVANCED SKINNED MESH

---

### 【学習要項】

- ☐ Normal mapping
- ☐ Tangent vector
- ☐ Bounding box

### 【演習手順】

#### 1. 鏡面反射(スペキュラー)と法線マッピング表現を実装する

- ① 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¥plantune.fbx に変更する  
※ plantune.fbx は右手系・Y 軸アップ・センチメートル単位・三角形化済み

- ② skinned\_mesh クラスの vertex 構造体にメンバ変数を追加する

```
1: struct vertex
2: {
3:     DirectX::XMFLOAT3 position;
4:     DirectX::XMFLOAT3 normal{ 0, 1, 0 };
*5:     DirectX::XMFLOAT4 tangent{ 1, 0, 0, 1 };
6:     DirectX::XMFLOAT2 texcoord{ 0, 0 };
7:     FLOAT bone_weights[MAX_BONE_INFLUENCES]{ 1, 0, 0, 0 };
8:     INT bone_indices[MAX_BONE_INFLUENCES]{};
9: };
```

- ③ skinned\_mesh クラスの create\_com\_objects メンバ関数で D3D11\_INPUT\_ELEMENT\_DESC を変更する  
※セマンティックは”TANGENT”にすること

- ④ skinned\_mesh クラスの fetch\_meshes メンバ関数で法線ベクトルの値を取得する

※下記コードをテクスチャ座標取得のコードの後に挿入する

※ファイルに法線ベクトル情報を持たない場合は GenerateTangentsData 関数で生成する

```
1: if (fbx_mesh->GenerateTangentsData(0, false))
2: {
3:     const FbxGeometryElementTangent* tangent = fbx_mesh->GetElementTangent(0);
4:     vertex.tangent.x = static_cast<float>(tangent->GetDirectArray().GetAt(vertex_index)[0]);
5:     vertex.tangent.y = static_cast<float>(tangent->GetDirectArray().GetAt(vertex_index)[1]);
6:     vertex.tangent.z = static_cast<float>(tangent->GetDirectArray().GetAt(vertex_index)[2]);
7:     vertex.tangent.w = static_cast<float>(tangent->GetDirectArray().GetAt(vertex_index)[3]);
8: }
```

- ⑤ skinned\_mesh クラスの fetch\_materials メンバ関数で法線マップのファイル名を取得する

※material 構造体の texture\_filenames[1] に法線マップのファイル名を格納する

```
1: property = surface_material->FindProperty(FbxSurfaceMaterial::sNormalMap);
2: if (property.IsValid())
3: {
4:     const FbxFileTexture* file_texture{ property.GetSrcObject<FbxFileTexture>() };
5:     material.texture_filenames[1] = file_texture ? file_texture->GetRelativeFileName() : "";
6: }
```

- ⑥ skinned\_mesh クラスの create\_com\_objects メンバ関数で法線マップのシェーダーリソースビューを生成する

※UNIT.19 1. ④で実装したコードを変更する

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

## UNIT29: ADVANCED SKINNED MESH

---

```
18:         texture_index == 1 ? 0xFFFF7F7F : 0xFFFFFFFF, 16);
19:     }
20: }
21: }
```

⑦skinned\_mesh クラスの render メンバ関数で法線マップのシェーダーリソースビューをバインドする

⑧シェーダヘッダ(skinned\_mesh.hlsl)のコードを変更する

```
1: struct VS_IN
2: {
3:     float4 position : POSITION;
4:     float4 normal : NORMAL;
* 5:     float4 tangent : TANGENT;
6:     float2 texcoord : TEXCOORD;
7:     float4 bone_weights : WEIGHTS;
8:     uint4 bone_indices : BONES;
9: };
10:
11: struct VS_OUT
12: {
13:     float4 position : SV_POSITION;
14:     float4 world_position : POSITION;
15:     float4 world_normal : NORMAL;
*16:     float4 world_tangent : TANGENT;
17:     float2 texcoord : TEXCOORD;
18:     float4 color : COLOR;
19: };
```

⑨頂点シェーダ(skinned\_mesh\_vs.hlsl) のコードを変更する

```
1: VS_OUT main(VS_IN vin)
2: {
3:     vin.normal.w = 0;
* 4:     float sigma = vin.tangent.w;
* 5:     vin.tangent.w = 0;
6:
7:     float4 blended_position = { 0, 0, 0, 1 };
8:     float4 blended_normal = { 0, 0, 0, 0 };
* 9:     float4 blended_tangent = { 0, 0, 0, 0 };
10:     for (int bone_index = 0; bone_index < 4; ++bone_index)
11:     {
12:         blended_position += vin.bone_weights[bone_index] *
13:             mul(vin.position, bone_transforms[vin.bone_indices[bone_index]]);
14:         blended_normal += vin.bone_weights[bone_index] *
15:             mul(vin.normal, bone_transforms[vin.bone_indices[bone_index]]);
*16:         blended_tangent += vin.bone_weights[bone_index] *
*17:             mul(vin.tangent, bone_transforms[vin.bone_indices[bone_index]]);
18:     }
19:     vin.position = float4(blended_position.xyz, 1.0f);
20:     vin.normal = float4(blended_normal.xyz, 0.0f);
*21:     vin.tangent = float4(blended_tangent.xyz, 0.0f);
22:
23:     VS_OUT vout;
24:     vout.position = mul(vin.position, mul(world, view_projection));
25:
26:     vout.world_position = mul(vin.position, world);
27:     vout.world_normal = normalize(mul(vin.normal, world));
*28:     vout.world_tangent = normalize(mul(vin.tangent, world));
*29:     vout.world_tangent.w = sigma;
30:
31:     vout.texcoord = vin.texcoord;
32:     vout.color = material_color;
33:
34:     return vout;
35: }
```

## ⑩ピクセルシェーダ(skinned\_mesh\_ps.hlsl)のコードを変更する

```
1: #include "skinned_mesh.hlsl"
2: #define POINT 0
3: #define LINEAR 1
4: #define ANISOTROPIC 2
5: SamplerState sampler_states[3] : register(s0);
6: Texture2D texture_maps[4] : register(t0);
7: float4 main(VS_OUT pin) : SV_TARGET
8: {
9:     float4 color = texture_maps[0].Sample(sampler_states[ANISOTROPIC], pin.texcoord);
10:    float alpha = color.a;
11:    float3 N = normalize(pin.world_normal.xyz);
12:    float3 T = normalize(pin.world_tangent.xyz);
13:    float sigma = pin.world_tangent.w;
14:    T = normalize(T - N * dot(N, T));
15:    float3 B = normalize(cross(N, T) * sigma);
16:
17:    float4 normal = texture_maps[1].Sample(sampler_states[LINEAR], pin.texcoord);
18:    normal = (normal * 2.0) - 1.0;
19:    N = normalize((normal.x * T) + (normal.y * B) + (normal.z * N));
20:
21:    float3 L = normalize(-light_direction.xyz);
22:    float3 diffuse = color.rgb * max(0, dot(N, L));
23:    float3 V = normalize(camera_position.xyz - pin.world_position.xyz);
24:    float3 specular = pow(max(0, dot(N, normalize(V + L))), 128);
25:    return float4(diffuse + specular, alpha) * pin.color;
26: }
```

## 2. バウンディングボックス(境界ボックス)情報を保持する

## ①skinned\_mesh::mesh 構造体にメンバ変数に追加する

```
1: DirectX::XMFLOAT3 bounding_box[2]
2: {
3:     { +D3D11_FLOAT32_MAX, +D3D11_FLOAT32_MAX, +D3D11_FLOAT32_MAX },
4:     { -D3D11_FLOAT32_MAX, -D3D11_FLOAT32_MAX, -D3D11_FLOAT32_MAX }
5: };
```

## ②skinned\_mesh クラスの fetch\_meshes メンバ関数に下記コードを追加する

```
1: for (const vertex& v : mesh.vertices)
2: {
3:     mesh.bounding_box[0].x = std::min<float>(mesh.bounding_box[0].x, v.position.x);
4:     mesh.bounding_box[0].y = std::min<float>(mesh.bounding_box[0].y, v.position.y);
5:     mesh.bounding_box[0].z = std::min<float>(mesh.bounding_box[0].z, v.position.z);
6:     mesh.bounding_box[1].x = std::max<float>(mesh.bounding_box[1].x, v.position.x);
7:     mesh.bounding_box[1].y = std::max<float>(mesh.bounding_box[1].y, v.position.y);
8:     mesh.bounding_box[1].z = std::max<float>(mesh.bounding_box[1].z, v.position.z);
9: }
```

## 3. これまでにテストで使ったすべての FBX ファイルを描画できるようにする

```
1: void skinned_mesh::render(ID3D11DeviceContext* immediate_context,
2:    const XMFLOAT4X4& world, const XMFLOAT4& material_color,
3:    const animation::keyframe* keyframe)
4: {
5:     for (mesh& mesh : meshes)
6:     {
7:         uint32_t stride{ sizeof(vertex) };
8:         uint32_t offset{ 0 };
9:         immediate_context->IASetVertexBuffers(0, 1, mesh.vertex_buffer.GetAddressOf(), &stride, &offset);
10:        immediate_context->IASetIndexBuffer(mesh.index_buffer.Get(), DXGI_FORMAT_R32_UINT, 0);
11:        immediate_context->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
12:        immediate_context->IASetInputLayout(input_layout.Get());
```

```
13:
14:     immediate_context->VSSetShader(vertex_shader.Get(), nullptr, 0);
15:     immediate_context->PSSetShader(pixel_shader.Get(), nullptr, 0);
16:
17:     constants data;
18:     if (keyframe && keyframe->nodes.size() > 0)
19:     {
20:         const animation::keyframe::node& mesh_node{ keyframe->nodes.at(mesh.node_index) };
21:         XMStoreFloat4x4(&data.world, XMLoadFloat4x4(&mesh_node.global_transform) * XMLoadFloat4x4(&world));
22:
23:         const size_t bone_count{ mesh.bind_pose.bones.size() };
24:         _ASSERT_EXPR(bone_count < MAX_BONES, L"The value of the 'bone_count' has exceeded MAX_BONES.");
25:
26:         for (size_t bone_index = 0; bone_index < bone_count; ++bone_index)
27:         {
28:             const skeleton::bone& bone{ mesh.bind_pose.bones.at(bone_index) };
29:             const animation::keyframe::node& bone_node{ keyframe->nodes.at(bone.node_index) };
30:             XMStoreFloat4x4(&data.bone_transforms[bone_index],
31:                 XMLoadFloat4x4(&bone.offset_transform) *
32:                 XMLoadFloat4x4(&bone_node.global_transform) *
33:                 XMMatrixInverse(nullptr, XMLoadFloat4x4(&mesh.default_global_transform))
34:             );
35:         }
36:     }
37:     else
38:     {
39:         XMStoreFloat4x4(&data.world,
40:             XMLoadFloat4x4(&mesh.default_global_transform) * XMLoadFloat4x4(&world));
41:         for (size_t bone_index = 0; bone_index < MAX_BONES; ++bone_index)
42:         {
43:             data.bone_transforms[bone_index] = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 };
44:         }
45:     }
46:     for (const mesh::subset& subset : mesh.subsets)
47:     {
48:         const material& material{ materials.at(subset.material_unique_id) };
49:         XMStoreFloat4(&data.material_color, XMLoadFloat4(&material_color) * XMLoadFloat4(&material.Kd));
50:         immediate_context->UpdateSubresource(constant_buffer.Get(), 0, 0, &data, 0, 0);
51:         immediate_context->VSSetConstantBuffers(0, 1, constant_buffer.GetAddressOf());
52:
53:         immediate_context->PSSetShaderResources(0, 1, material.shader_resource_views[0].GetAddressOf());
54:         immediate_context->PSSetShaderResources(1, 1, material.shader_resource_views[1].GetAddressOf());
55:
56:         immediate_context->DrawIndexed(subset.index_count, subset.start_index_location, 0);
57:     }
58: }
59: }
```

#### 4. スキンメッシュを使った3Dゲームを制作する

##### 【評価項目】

- ☐ スキンメッシュクラスの改良
- ☐ 3Dゲーム制作