1. 기본 도형

1-1. Cube

1-2. Sphere

2. Light

2-1. Ambient

2-2. Diffuse

2-3. Specular

2-4. Emissive

2-5. Light

2-6. Normal Mapping

1. 기본 도형

1-1. Cube

Cube의 앞면, 뒷면, 윗면, 아랫면, 왼쪽면, 오른쪽 면을 0.5f의 크기로 총 24개 정점 생성

**void** GeometryHelper::CreateCube(shared\_ptr<Geometry<VertexTextureNormalTangentData>> geometry)

{

float w2 = 0.5f;

float h2 = 0.5f;

float d2 = 0.5f;

vector<VertexTextureNormalTangentData> vtx(24);

*// 앞면*

vtx[0] = VertexTextureNormalTangentData(Vec3(-w2, -h2, -d2), Vec2(0.0f, 1.0f), Vec3(0.0f, 0.0f, -1.0f), Vec3(1.0f, 0.0f, 0.0f));

vtx[1] = VertexTextureNormalTangentData(Vec3(-w2, +h2, -d2), Vec2(0.0f, 0.0f), Vec3(0.0f, 0.0f, -1.0f), Vec3(1.0f, 0.0f, 0.0f));

vtx[2] = VertexTextureNormalTangentData(Vec3(+w2, +h2, -d2), Vec2(1.0f, 0.0f), Vec3(0.0f, 0.0f, -1.0f), Vec3(1.0f, 0.0f, 0.0f));

vtx[3] = VertexTextureNormalTangentData(Vec3(+w2, -h2, -d2), Vec2(1.0f, 1.0f), Vec3(0.0f, 0.0f, -1.0f), Vec3(1.0f, 0.0f, 0.0f));

*// 뒷면*

vtx[4] = VertexTextureNormalTangentData(Vec3(-w2, -h2, +d2), Vec2(1.0f, 1.0f), Vec3(0.0f, 0.0f, 1.0f), Vec3(-1.0f, 0.0f, 0.0f));

vtx[5] = VertexTextureNormalTangentData(Vec3(+w2, -h2, +d2), Vec2(0.0f, 1.0f), Vec3(0.0f, 0.0f, 1.0f), Vec3(-1.0f, 0.0f, 0.0f));

vtx[6] = VertexTextureNormalTangentData(Vec3(+w2, +h2, +d2), Vec2(0.0f, 0.0f), Vec3(0.0f, 0.0f, 1.0f), Vec3(-1.0f, 0.0f, 0.0f));

vtx[7] = VertexTextureNormalTangentData(Vec3(-w2, +h2, +d2), Vec2(1.0f, 0.0f), Vec3(0.0f, 0.0f, 1.0f), Vec3(-1.0f, 0.0f, 0.0f));

*// 윗면*

vtx[8] = VertexTextureNormalTangentData(Vec3(-w2, +h2, -d2), Vec2(0.0f, 1.0f), Vec3(0.0f, 1.0f, 0.0f), Vec3(1.0f, 0.0f, 0.0f));

vtx[9] = VertexTextureNormalTangentData(Vec3(-w2, +h2, +d2), Vec2(0.0f, 0.0f), Vec3(0.0f, 1.0f, 0.0f), Vec3(1.0f, 0.0f, 0.0f));

vtx[10] = VertexTextureNormalTangentData(Vec3(+w2, +h2, +d2), Vec2(1.0f, 0.0f), Vec3(0.0f, 1.0f, 0.0f), Vec3(1.0f, 0.0f, 0.0f));

vtx[11] = VertexTextureNormalTangentData(Vec3(+w2, +h2, -d2), Vec2(1.0f, 1.0f), Vec3(0.0f, 1.0f, 0.0f), Vec3(1.0f, 0.0f, 0.0f));

*// 아랫면*

vtx[12] = VertexTextureNormalTangentData(Vec3(-w2, -h2, -d2), Vec2(1.0f, 1.0f), Vec3(0.0f, -1.0f, 0.0f), Vec3(-1.0f, 0.0f, 0.0f));

vtx[13] = VertexTextureNormalTangentData(Vec3(+w2, -h2, -d2), Vec2(0.0f, 1.0f), Vec3(0.0f, -1.0f, 0.0f), Vec3(-1.0f, 0.0f, 0.0f));

vtx[14] = VertexTextureNormalTangentData(Vec3(+w2, -h2, +d2), Vec2(0.0f, 0.0f), Vec3(0.0f, -1.0f, 0.0f), Vec3(-1.0f, 0.0f, 0.0f));

vtx[15] = VertexTextureNormalTangentData(Vec3(-w2, -h2, +d2), Vec2(1.0f, 0.0f), Vec3(0.0f, -1.0f, 0.0f), Vec3(-1.0f, 0.0f, 0.0f));

*// 왼쪽면*

vtx[16] = VertexTextureNormalTangentData(Vec3(-w2, -h2, +d2), Vec2(0.0f, 1.0f), Vec3(-1.0f, 0.0f, 0.0f), Vec3(0.0f, 0.0f, -1.0f));

vtx[17] = VertexTextureNormalTangentData(Vec3(-w2, +h2, +d2), Vec2(0.0f, 0.0f), Vec3(-1.0f, 0.0f, 0.0f), Vec3(0.0f, 0.0f, -1.0f));

vtx[18] = VertexTextureNormalTangentData(Vec3(-w2, +h2, -d2), Vec2(1.0f, 0.0f), Vec3(-1.0f, 0.0f, 0.0f), Vec3(0.0f, 0.0f, -1.0f));

vtx[19] = VertexTextureNormalTangentData(Vec3(-w2, -h2, -d2), Vec2(1.0f, 1.0f), Vec3(-1.0f, 0.0f, 0.0f), Vec3(0.0f, 0.0f, -1.0f));

*// 오른쪽면*

vtx[20] = VertexTextureNormalTangentData(Vec3(+w2, -h2, -d2), Vec2(0.0f, 1.0f), Vec3(1.0f, 0.0f, 0.0f), Vec3(0.0f, 0.0f, 1.0f));

vtx[21] = VertexTextureNormalTangentData(Vec3(+w2, +h2, -d2), Vec2(0.0f, 0.0f), Vec3(1.0f, 0.0f, 0.0f), Vec3(0.0f, 0.0f, 1.0f));

vtx[22] = VertexTextureNormalTangentData(Vec3(+w2, +h2, +d2), Vec2(1.0f, 0.0f), Vec3(1.0f, 0.0f, 0.0f), Vec3(0.0f, 0.0f, 1.0f));

vtx[23] = VertexTextureNormalTangentData(Vec3(+w2, -h2, +d2), Vec2(1.0f, 1.0f), Vec3(1.0f, 0.0f, 0.0f), Vec3(0.0f, 0.0f, 1.0f));

geometry->SetVertices(vtx);

vector<uint32> idx(36);

*// 앞면*

idx[0] = 0; idx[1] = 1; idx[2] = 2;

idx[3] = 0; idx[4] = 2; idx[5] = 3;

*// 뒷면*

idx[6] = 4; idx[7] = 5; idx[8] = 6;

idx[9] = 4; idx[10] = 6; idx[11] = 7;

*// 윗면*

idx[12] = 8; idx[13] = 9; idx[14] = 10;

idx[15] = 8; idx[16] = 10; idx[17] = 11;

*// 아랫면*

idx[18] = 12; idx[19] = 13; idx[20] = 14;

idx[21] = 12; idx[22] = 14; idx[23] = 15;

*// 왼쪽면*

idx[24] = 16; idx[25] = 17; idx[26] = 18;

idx[27] = 16; idx[28] = 18; idx[29] = 19;

*// 오른쪽면*

idx[30] = 20; idx[31] = 21; idx[32] = 22;

idx[33] = 20; idx[34] = 22; idx[35] = 23;

geometry->SetIndices(idx);

}

1-2. Sphere

북극 정점, 고리 정점, 남극 정점을 생성한 후 Index를 설정하여 구 생성

**void** GeometryHelper::CreateSphere(shared\_ptr<Geometry<VertexTextureNormalTangentData>> geometry)

{

float radius = 0.5f; *// 구의 반지름*

uint32 stackCount = 20; *// 가로 분할*

uint32 sliceCount = 20; *// 세로 분할*

vector<VertexTextureNormalTangentData> vtx;

VertexTextureNormalTangentData v;

*// 북극*

v.position = Vec3(0.0f, radius, 0.0f);

v.uv = Vec2(0.5f, 0.0f);

v.normal = v.position;

v.normal.Normalize();

v.tangent = Vec3(1.0f, 0.0f, 0.0f);

v.tangent.Normalize();

vtx.push\_back(v);

float stackAngle = XM\_PI / stackCount;

float sliceAngle = XM\_2PI / sliceCount;

float deltaU = 1.f / static\_cast<float>(sliceCount);

float deltaV = 1.f / static\_cast<float>(stackCount);

*// 고리마다 돌면서 정점을 계산한다 (북극/남극 단일점은 고리가 X)*

**for** (uint32 y = 1; y <= stackCount - 1; ++y)

{

float phi = y \* stackAngle;

*// 고리에 위치한 정점*

**for** (uint32 x = 0; x <= sliceCount; ++x)

{

float theta = x \* sliceAngle;

v.position.x = radius \* sinf(phi) \* cosf(theta);

v.position.y = radius \* cosf(phi);

v.position.z = radius \* sinf(phi) \* sinf(theta);

v.uv = Vec2(deltaU \* x, deltaV \* y);

v.normal = v.position;

v.normal.Normalize();

v.tangent.x = -radius \* sinf(phi) \* sinf(theta);

v.tangent.y = 0.0f;

v.tangent.z = radius \* sinf(phi) \* cosf(theta);

v.tangent.Normalize();

vtx.push\_back(v);

}

}

*// 남극*

v.position = Vec3(0.0f, -radius, 0.0f);

v.uv = Vec2(0.5f, 1.0f);

v.normal = v.position;

v.normal.Normalize();

v.tangent = Vec3(1.0f, 0.0f, 0.0f);

v.tangent.Normalize();

vtx.push\_back(v);

geometry->SetVertices(vtx);

vector<uint32> idx(36);

*// 북극 인덱스*

**for** (uint32 i = 0; i <= sliceCount; ++i)

{

*// [0]*

*// | \*

*// [i+1]-[i+2]*

idx.push\_back(0);

idx.push\_back(i + 2);

idx.push\_back(i + 1);

}

*// 몸통 인덱스*

uint32 ringVertexCount = sliceCount + 1;

**for** (uint32 y = 0; y < stackCount - 2; ++y)

{

**for** (uint32 x = 0; x < sliceCount; ++x)

{

*// [y, x]-[y, x+1]*

*// | /*

*// [y+1, x]*

idx.push\_back(1 + (y)\*ringVertexCount + (x));

idx.push\_back(1 + (y)\*ringVertexCount + (x + 1));

idx.push\_back(1 + (y + 1) \* ringVertexCount + (x));

*// [y, x+1]*

*// / |*

*// [y+1, x]-[y+1, x+1]*

idx.push\_back(1 + (y + 1) \* ringVertexCount + (x));

idx.push\_back(1 + (y)\*ringVertexCount + (x + 1));

idx.push\_back(1 + (y + 1) \* ringVertexCount + (x + 1));

}

}

*// 남극 인덱스*

uint32 bottomIndex = static\_cast<uint32>(vtx.size()) - 1;

uint32 lastRingStartIndex = bottomIndex - ringVertexCount;

**for** (uint32 i = 0; i < sliceCount; ++i)

{

*// [last+i]-[last+i+1]*

*// | /*

*// [bottom]*

idx.push\_back(bottomIndex);

idx.push\_back(lastRingStartIndex + i);

idx.push\_back(lastRingStartIndex + i + 1);

}

geometry->SetIndices(idx);

}

2. Light

공통 내용

VS : 정점 위치를 월드 공간과 뷰 공간으로 변환

VertexOutput VS(VertexTextureNormal input)

{

VertexOutput output;

output.position = mul(input.position, W);

output.position = mul(output.position, VP);

output.uv = input.uv;

output.normal = mul(input.normal, (float3x3)W);

**return** output;

}

2-1. Ambient

Ambient 강도와 MaterialAmbient를 곱하여 Ambient Light 적용

float4 LightAmbient;

float4 MaterialAmbient;

Texture2D Texture0;

float4 PS(VertexOutput input) : SV\_TARGET

{

float4 color = LightAmbient \* MaterialAmbient;

**return** Texture0.Sample(LinearSampler, input.uv) \* color;

}

2-2. Diffuse

텍스처 매핑을 통해 픽셀의 색상을 가져온 후 광원과 표면의 법선벡터 사이의 내적을 계산하여 조명 강도 결정

텍스처 색상에 내적값과 광원의 Diffuse강도와 Material의 Diffuse 속성을 곱하여 계산

float4 LightAmbient;

float4 MaterialAmbient;

Texture2D Texture0;

float4 PS(VertexOutput input) : SV\_TARGET

{

float4 color = LightAmbient \* MaterialAmbient;

**return** Texture0.Sample(LinearSampler, input.uv) \* color;

}

2-3. Specular

빛의 반사 방향 벡터를 계산, 뷰어와 표면의 법선 벡터 사이의 반사 방향 벡터 계산 후 반사 방향 벡터와 뷰어 방향 벡터 사이의 내적을 계산하여 반다 강도 결정

반사 강도에 10승을 적용하여 Specular 강도 계산 후 Specular 색상과 MaterialSpecular을 곱하여 색상 계산

float3 LightDir;

float4 LightSpecular;

float4 MaterialSpecular;

Texture2D DiffuseMap;

float4 PS(MeshOutput input) : SV\_TARGET

{

*//빛의 반대 방향으로 반사되는 것*

*//float3 R = reflect(LightDir, input.normal);*

float3 R = LightDir - (2 \* input.normal \* dot(LightDir, input.normal));

R = normalize(R);

float3 cameraPosition = -V.\_41\_42\_43;

float3 E = normalize(cameraPosition - input.worldPosition);

float **value** = saturate(dot(R, E)); *//clamp(0~1) 사이로 지정*

float specular = pow(**value**, 10);

float4 color = LightSpecular \* MaterialSpecular \* specular;

**return** color;

}

2-4. Emissive

카메라 위치 계산 후 Viewer와 Pixel 사이의 벡터 계산

Viewer와 Pixel 법선 벡터 사이의 내적을 계산하여 발광 강도 결정 후 발광 강도에서 1을 빼 발광 효과 계산

발광 색상과 발광 효과를 곱하여 최종 색상 계산

float4 MaterialEmissive;

float4 PS(MeshOutput input) : SV\_TARGET

{

float3 cameraPostion = -V.\_41\_42\_43;

float3 E = normalize(cameraPostion - input.worldPosition);

float **value** = saturate(dot(E, input.normal));

float emissive = 1.0f - **value**;

emissive = smoothstep(0.0f, 1.0f, emissive);

emissive = pow(emissive, 3);

float4 color = MaterialEmissive \* emissive;

**return** color;

}

2-5. Light

Ambient, Diffuse, Specular, Emissive를 모두 합친 함수

float4 ComputeLight(float3 normal, float2 uv, float3 worldPosition)

{

float4 ambientColor = 0;

float4 diffuseColor = 0;

float4 specularColor = 0;

float4 emissiveColor = 0;

*//ambient*

{

float4 color = GlobalLight.ambient\* Material.ambient;

ambientColor = DiffuseMap.Sample(LinearSampler, uv) \* color;

}

*// diffuse*

{

float4 color = DiffuseMap.Sample(LinearSampler, uv);

float **value** = dot(-GlobalLight.direction, normalize(normal));

diffuseColor = color \* **value** \* GlobalLight.diffuse \* Material.diffuse;

}

*// Specular*

{

*//float3 R = reflect(GlobalLight.direction, normal);*

float3 R = GlobalLight.direction - (2 \* normal \* dot(GlobalLight.direction, normal));

R = normalize(R);

float3 cameraPosition = CameraPosition();

float3 E = normalize(cameraPosition - worldPosition);

float **value** = saturate(dot(R, E)); *// clamp(0~1)*

float specular = pow(**value**, 10);

specularColor = GlobalLight.specular \* Material.specular \* specular;

}

*//Emissive*

{

float3 E = normalize(CameraPosition() - worldPosition);

float **value** = saturate(dot(E, normal));

float emissive = 1.0f - **value**;

*// min, max, x 넣으면 부드럽게 보간해주는 함수 : smoothstep*

emissive = smoothstep(0.0f, 1.0f, emissive);

emissive = pow(emissive, 10);

emissiveColor = GlobalLight.emissive \* Material.emissive \* emissive;

}

**return** ambientColor + diffuseColor + emissiveColor + specularColor;

}

2-6. Normal Mapping

Normal Map에서의 샘플링한 값을 이용하여 tangent space에서의 법선을 계산하고 world space로 변환하여 표면의 normal 벡터를 업데이트

**void** ComputeNormalMapping(inout float3 normal, float3 tangent, float2 uv)

{

float4 map = NormalMap.Sample(LinearSampler, uv);

**if** (any(map.rgb) == **false**) **return**;

float3 N = normalize(normal);

float3 T = normalize(tangent);

float3 B = normalize(cross(N, T));

float3x3 TBN = float3x3(T, B, N);

*// [0, 1] 범위에서 [-1, 1] 범위로 변환*

float3 tangentSpaceNormal = (map.rgb \* 2.0f - 1.0f);

float3 worldNormal = mul(tangentSpaceNormal, TBN);

normal = worldNormal;

}