**3D Game Programming 과제2 보고서**

게임공학과 2013180028 이우상

**[ 목표 ]**

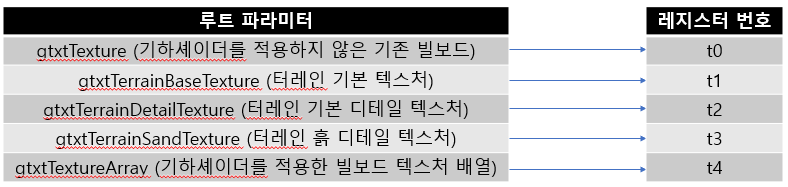
정점 하나만 생성하여, 기하셰이더로 사각형을 만들고, 빌보드 나무, 풀, 꽃 텍스처를 매핑한다. 또한 터레인도 기하셰이더를 적용시켜 만든다. 키 버튼 입력으로 기하셰이더를 이용해서 만든 빌보드, 터레인과, 기존에 만들었던 빌보드 오브젝트, 터레인의 차이를 WIREFRAME으로 보여준다.

**[ 조작법 ]**

1. s/S 키를 이용하여, RasterizerState의 FillMode를 SOLID와 WIREFRAME으로 바꾸어 다시 시작하게 한다.
2. g/G 키를 이용하여 기하셰이더를 적용하고, 적용하지 않은 것을 보여준다.

**[ 구현 방법 ]**

**[루트 파라미터와 연결된 레지스터 번호]**



1. **기하셰이더를 적용하기 위해 루트시그니처 플래그에서 D3D12\_ROOT\_SIGNATURE\_FLAG\_DENY\_GEOMETRY\_SHADER\_ROOT\_ACCESS 를 지워준다.**
2. **s/S, 키를 이용하여 FillMode를 구분 짓게 하는 변수 g\_FillMode를 전역변수로 생성한다.**

**또한 g/G 키를 이용하여 기하셰이더를 적용시킨 것과 적용시키지 않은 것을 보이기 위해, 전역변수 g\_OnGeometry를 생성한다.**

1. **enum으로 타입을 각각 정의하여, FillMode의 SOLID와 WIREFRAME, 기하셰이더를 사용한 유무의 USE, UNUSE를 만든다.**

**// 0번이면 FillMode를 SOLID로 생성, 1번이면 WIREFRAME으로 생성한다.**

enum FILL\_MODE { SOLID, WIRE};

**// false이면 기하셰이더를 적용하지 않고, true이면 기하셰이더를 적용**

enum ON\_GEOMETRY { USE, UNUSE };

**// 처음에는 FillMode를 SOLID, 기하셰이더를 적용 X로 한다.**

int g\_FillMode = SOLID;

bool g\_OnGeometry = UNUSE;

1. CGameFramework의 OnProcessingKeyboardMessage( )에서 s/S 와 g/G 키를 눌렀을 때, 각각 FillMode를 변경시키고, 기하셰이더를 적용시킬 수 있도록 만든다.

case 's':

case 'S':

**// s/S 키를 눌렀을 때, 솔리드 모드이면, 와이어프레임으로 바꿔주고,**

**// 와이어 모드이면, 솔리드 모드로 바꾼다.**

if (g\_FillMode == SOLID)

{

g\_FillMode = WIRE;

cout << "와이어 모드" << endl;

}

else

{

g\_FillMode = SOLID;

cout << "솔리드 모드" << endl;

}

break;

case 'g':

case 'G':

**// g/G 키를 눌렀을 때, 기하셰이더가 적용되었을 때는, 적용을 하지 않겠다고**

**// g\_OnGeometry를 UNUSE로 바꿔주고, 기하셰이더가 적용되지 않았을 때는**

**// USE로 바꾸어 기하셰이더를 적용시킨다.**

if (g\_OnGeometry == USE)

{

g\_OnGeometry = UNUSE;

cout << "기하셰이더 적용 X" << endl;

}

else

{

g\_OnGeometry = USE;

cout << "기하셰이더 적용 O" << endl;

}

break;

1. **CScene의 BuildObject( )에서 빌보드 객체의 위치를 텍스트 파일의 값을 읽어들인다. 이 파일은 빌보드 객체의 위치를 특정 위치에만 설정되도록 하기 위함이다. 또한 기하셰이더를 적용시킨 것과, 적용 시키지 않는 셰이더로 구분해서 만들어 준다.**

**// 2차원 배열 257 \* 257 크기만큼 만들어, 텍스트 파일에 있는 값을 읽는다.**

**// 만약 값이 0이면 빌보드가 위치하는 곳이 아니고,**

**// 1이면 빌보드 객체가 그려질 터레인의 위치이다.**

int PositionArray[PIXELCOUNT][PIXELCOUNT];

ifstream in("Position.txt");

for (int i = 0; i < PIXELCOUNT; ++i)

{

for (int j = 0; j < PIXELCOUNT; ++j)

{

in >> PositionArray[i][j];

}

}

**// 기하셰더를 적용시킨다면, CGeometryShader 객체를 생성하여 셰이더를 생성**

if (g\_OnGeometry == USE)

{

CGeometryShader\* pGeometryShader = new CGeometryShader;

pGeometryShader->CreateShader(pd3dDevice, m\_pd3dGraphicsRootSignature);

pGeometryShader->setPositionArray(PositionArray);

pGeometryShader->BuildObjects(pd3dDevice, pd3dCommandList, m\_pTerrain);

m\_ppShaders[0] = pGeometryShader;

}

**// 기하셰더를 적용시키지 않는다면, CBillboardShader 객체를 생성하여 셰이더를 생성**

else if(g\_OnGeometry == UNUSE)

{

CBillboardShader\* pBillboardShader = new CBillboardShader;

pBillboardShader->CreateShader(pd3dDevice, m\_pd3dGraphicsRootSignature);

**// 나무, 꽃, 풀들이 위치할 텍스처 픽셸의 위치**

pBillboardShader->setPositionArray(PositionArray);

pBillboardShader->BuildObjects(pd3dDevice, pd3dCommandList, m\_pTerrain);

m\_ppShaders[0] = pBillboardShader;

}

1. **Terrain에서 기하셰이더를 적용시킨것과 적용시키지 않은 것을 만들기 위해 파이프라인 스테이트를 2개를 만든다. 1개는 기하셰이더를 적용시킨 것, 1개는 적용시키지 않은 것**

**// CTerrainShader의 CreateShader에서 파이프라인 개수를 2개로 하고,**

void CTerrainShader::CreateShader(ID3D12Device \*pd3dDevice, ID3D12RootSignature \*pd3dGraphicsRootSignature)

{

m\_nPipelineStates = 2;

…

**// 0번째 파이프라인이라면, 기하셰이더 코드를 컴파일한다. 1번째이면, 기하셰이더 코드를 컴파일하지 않음**

**// 적용시키**

If(i == 0)

{

d3dPipelineStateDesc[i].GS = CreateGeometryShader(&pd3dGeometryShaderBlob);

}

}

// **CGeometryShader의 CreateShader에서도 추가해준다.**

**void CGeometryShader::CreateShader(ID3D12Device \*pd3dDevice, ID3D12RootSignature \*pd3dGraphicsRootSignature)**

**{**

**…**

**d3dPipelineStateDesc.GS = CreateGeometryShader(&pd3dGeometeryShaderBlob);**

**…**

**// 또한 PrimitiveTopology를 점으로 만들기위해 POINT로 설정**

d3dPipelineStateDesc.PrimitiveTopologyType = D3D12\_PRIMITIVE\_TOPOLOGY\_TYPE\_POINT;

1. **기하셰이더 코드의 함수를 리턴하는 함수를 만들어준다.**

D3D12\_SHADER\_BYTECODE CTerrainShader::CreateGeometryShader(ID3DBlob \*\*ppd3dShaderBlob)

{

**// 기하셰이더를 적용시킨다면, GS\_Terrain 이라는 함수를 컴파일한다.**

return(CShader::CompileShaderFromFile(L"Shaders.hlsl", "GS\_Terrain", "gs\_5\_1", ppd3dShaderBlob));

}

1. **기하셰이더를 적용시켰으면 그에 맞는 픽셸셰이더 함수도 원래의 터레인 픽셸셰이더와 다르게 한 개 더 생성해준다.**

D3D12\_SHADER\_BYTECODE CTerrainShader::CreatePixelShader(ID3DBlob \*\*ppd3dShaderBlob)

{

**// 기하셰이더가 적용되었을 때, PS\_GSTerrain 픽셸셰이더 함수를 실행**

return(CShader::CompileShaderFromFile(L"Shaders.hlsl", "PS\_GSTerrain", "ps\_5\_1", ppd3dShaderBlob));

}

1. **레스터라이저 스테이트의 FillMode도 g\_FillMode 변수에 맞게 변경해준다.**

D3D12\_RASTERIZER\_DESC CTerrainShader::CreateRasterizerState()

{

D3D12\_RASTERIZER\_DESC d3dRasterizerDesc;

::ZeroMemory(&d3dRasterizerDesc, sizeof(D3D12\_RASTERIZER\_DESC));

**// 키보드로 s 버튼에 따라 g\_FillMode가 SOLID이면, 레스터라이저 스테이트를**

**// SOLID로, WIRE이면, WIREFRAME으로 변경**

if (g\_FillMode == SOLID)

d3dRasterizerDesc.FillMode = D3D12\_FILL\_MODE\_SOLID;

else if (g\_FillMode == WIRE)

d3dRasterizerDesc.FillMode = D3D12\_FILL\_MODE\_WIREFRAME;

. . .

}

1. **CTerrainShader의 Onprepare( )도 기하셰이더를 적용시키는 파이프라인 스테이트와, 기하셰이더를 적용시키지 않은 파이프라인 스테이트를 나눠 각각 랜더할 때 다르게**

**랜더 되도록 별도로 처리한다.**

void CTerrainShader::OnPrepareRender(ID3D12GraphicsCommandList \*pd3dCommandList)

{

if (m\_ppd3dPipelineStates)

{

**// g\_OnGeometry가 USE이면, 0번째(기하셰이더 적용)를,**

**// 1번째이면 기하셰이더를 적용시키지 않는다.**

if(g\_OnGeometry == USE)

pd3dCommandList->SetPipelineState(m\_ppd3dPipelineStates[0]);

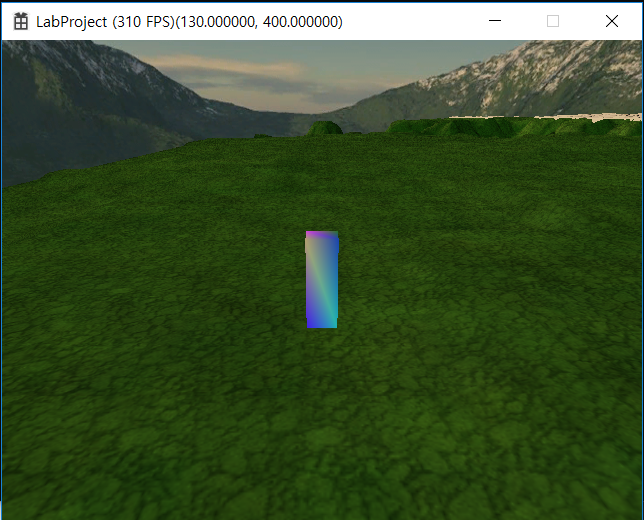
else

pd3dCommandList->SetPipelineState(m\_ppd3dPipelineStates[1]);

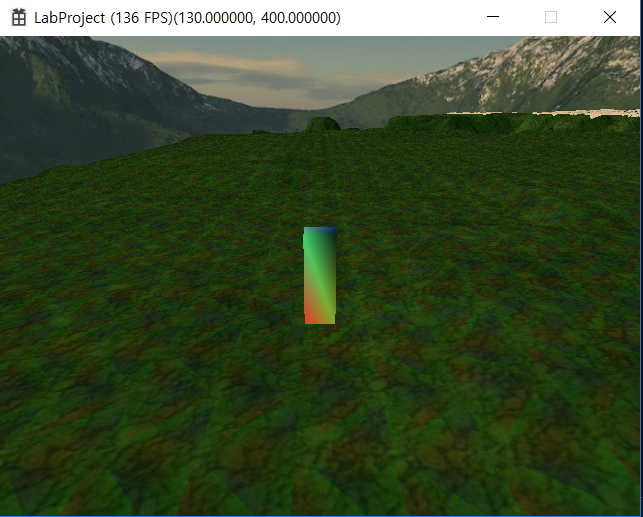
}

. . .

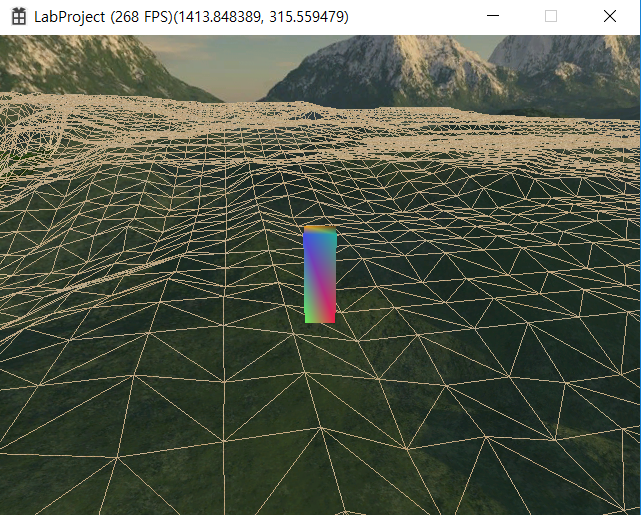
}



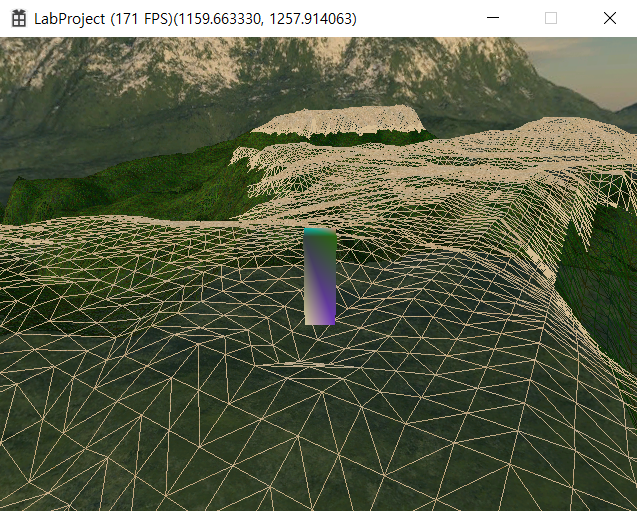
**기하셰이더를 적용하지 않고, 솔리드 모드일 때,**



**기하셰이더를 적용하고, 솔리드 모드일 때**



**기하셰이더를 적용하지 않고, 흙 위에서 와이어프레임 모드일 때**



**기하셰이더를 적용하고, 흙위에서 와이어프레임 모드일 때**

**빌보드 나무, 꽃, 풀을 과제1때는 셰이더 코드에서 discard를 사용하여, 텍스처의 검은 부분을 출력 되지 않도록 했지만, 이번에는 AlphaToCoverage를 사용하여 처리하였다.**

D3D12\_BLEND\_DESC CGeometryShader::CreateBlendState()

{

. . .

**// CGeometryShader와 CBillboardShader의 CreateBlendState( )에서**

**// AlphaToCoverageEnable을 true로 변경, BlendEnable을 true로 변경하여,**

**// 블랜드를 활성화하고, AlphaToCoverage를 사용하게 한다.**

**// 그래서 텍스처의 알파값을 이용하여, 검은 부분이 출력되지 않게 변경하였다.**

d3dBlendDesc.AlphaToCoverageEnable = true;

d3dBlendDesc.RenderTarget[0].BlendEnable = true;

. . .

}

**과제1때는 텍스처 배열을 사용하지 않았지만, 이번에는 CGeometryShader BuildObjects( )에서 텍스처 배열을 이용하여 텍스처를 매핑하였다.**

void CGeometryShader::BuildObjects(ID3D12Device \*pd3dDevice, ID3D12GraphicsCommandList \*pd3dCommandList, void \*pContext)

{

. . .

**// 텍스처 배열을 사용하기위해, CTexture의 생성자의 2번째 인자로**

**// RESOURCE\_TEXTURE2DARRAY를 준다.**

CTexture\* pTree\_Texture2DArray = new CTexture(1, RESOURCE\_TEXTURE2DARRAY, 0);

**// 텍스처 배열로 된 dds파일을 로드**

pTree\_Texture2DArray->LoadTextureFromFile(pd3dDevice, pd3dCommandList, L"Image/TreeArray.dds", 0);

**// 텍스처배열은 루트파라미터 7번에 연결 되어있으므로**

**// CreateShaderResourceView의 인자를 텍스처 배열 객체와 7을 입력한다.**

CreateShaderResourceViews(pd3dDevice, pd3dCommandList, pTree\_Texture2DArray, 7, false);

**// 만들어 놓은 텍스처 배열 객체를 재질에 Set해준다.**

m\_pMaterial = new CMaterial();

m\_pMaterial->SetTexture(pTree\_Texture2DArray);

**// 과제 1에서는 빌보드 객체의 위치를 터레인의 높이를 비교하여 설정하였지만,**

**// 이번에는 텍스트 파일에서 터레인의 텍스처 크기 257 \* 257 만큼 값을 만들어 1인곳만**

**// 빌보드 객체의 위치가 되도록 하였다.**

for (int i = 0; i < m\_nVertices;)

{

**// 터레인의 크기는 8배 스케일 된 것이므로, 랜덤으로 뽑힌 값을 8로 나누어,**

**// 배열의 있는 크기(257 \* 257)이 되도록한다.**

**// 그래서 나오는 값을 각각 x와 z에 대입하고, 파일에서 읽어 저장한 배열**

**// m\_PositionArray[x][z]의 값이 1이면 x와 z에 8배를 다시 스케일하여**

**// 객체의 위치 x, y, z를 구해준다.**

**// 이렇게하면, 파일에서 있는 값 1인 부분만 객체가 위치해서 그려지게 된다.**

x = urd\_x(dre) / 8;

z = urd\_z(dre) / 8;

if (m\_PositionArray[x][z] == 1)

{

x = x \* 8.;

z = z \* 8.;

y = pTerrain->GetHeight(x, z);

xmf3Position.x = x;

xmf3Position.z = z;

xmf3Position.y = pTerrain->GetHeight(x, z) + 10.0f;

pBillboardVertex[i].m\_xmf3Position = xmf3Position;

pBillboardVertex[i].m\_xmf2Size = XMFLOAT2(50, 70);

++i;

}

else

continue;

}

**// 과제1때는 빌보드 나무, 꽃, 풀의 위치를 랜덤**

**// 기하셰이더로 만들 정점을 정점 버퍼로 생성해준다.**

m\_pd3dVertexBuffer = ::CreateBufferResource

(

pd3dDevice,

pd3dCommandList,

pBillboardVertex,

nStride \* m\_nVertices,

D3D12\_HEAP\_TYPE\_DEFAULT,

D3D12\_RESOURCE\_STATE\_VERTEX\_AND\_CONSTANT\_BUFFER,

&m\_pd3dVertexUploadBuffer

);

m\_d3dVertexBufferView.BufferLocation = m\_pd3dVertexBuffer->GetGPUVirtualAddress();

m\_d3dVertexBufferView.StrideInBytes = nStride;

m\_d3dVertexBufferView.SizeInBytes = nStride \* m\_nVertices;

**// 정점버퍼를 생성했다면, 정점객체를 지워준다.**

if (pBillboardVertex)

delete[] pBillboardVertex;

}

**CGeometry의 BuildObjects( )가 끝나서 정점을 생성하려는 빌보드 개수만큼 만들었으면, Render( )부분에서는 GameObject( )에서 랜더하는 부분을 직접 적어 재질을 랜더하게 한다.**

**왜냐하면, 과제1에서는 BillboardObject를 만들어 게임오브젝트를 생성하여 랜더하게 했지만, 기하셰이더의 빌보드는 GameObject로 생성하지 않았기 때문에 직접 정점을 바로 랜더한다.**

void CGeometryShader::Render(ID3D12GraphicsCommandList \*pd3dCommandList, CCamera \*pCamera)

{

OnPrepareRender(pd3dCommandList);

if (m\_pMaterial)

{

if (m\_pMaterial->m\_pShader)

{

m\_pMaterial->m\_pShader->Render(pd3dCommandList, pCamera);

m\_pMaterial->m\_pShader->UpdateShaderVariables(pd3dCommandList);

UpdateShaderVariables(pd3dCommandList);

}

if (m\_pMaterial->m\_pTexture)

m\_pMaterial->m\_pTexture->UpdateShaderVariables(pd3dCommandList);

}

**// PrimitiveTopology를 POINTLIST로 해준다.**

pd3dCommandList->IASetPrimitiveTopology(D3D\_PRIMITIVE\_TOPOLOGY\_POINTLIST);

pd3dCommandList->IASetVertexBuffers(0, 1, &m\_d3dVertexBufferView);

pd3dCommandList->DrawInstanced(m\_nVertices, 1, 0, 0);

}

**기하셰이더 빌보드의 픽셸 셰이더에서는 텍스처 배열에 나무 텍스처가 3개가 있으므로 w**

**값을 primID % 3을 이용하여 매핑하게 한다.**

float4 PS\_Geometry(GS\_OUT input) : SV\_Target

{

**// 텍스처 배열에 텍스처가 3개있음**

float3 uvw = float3(input.uv, (input.primID % 3));

float4 cTexture = gtxtTextureArray.Sample(gWrapSamplerState, uvw);

float4 cColor = cTexture;

return (cColor);

}