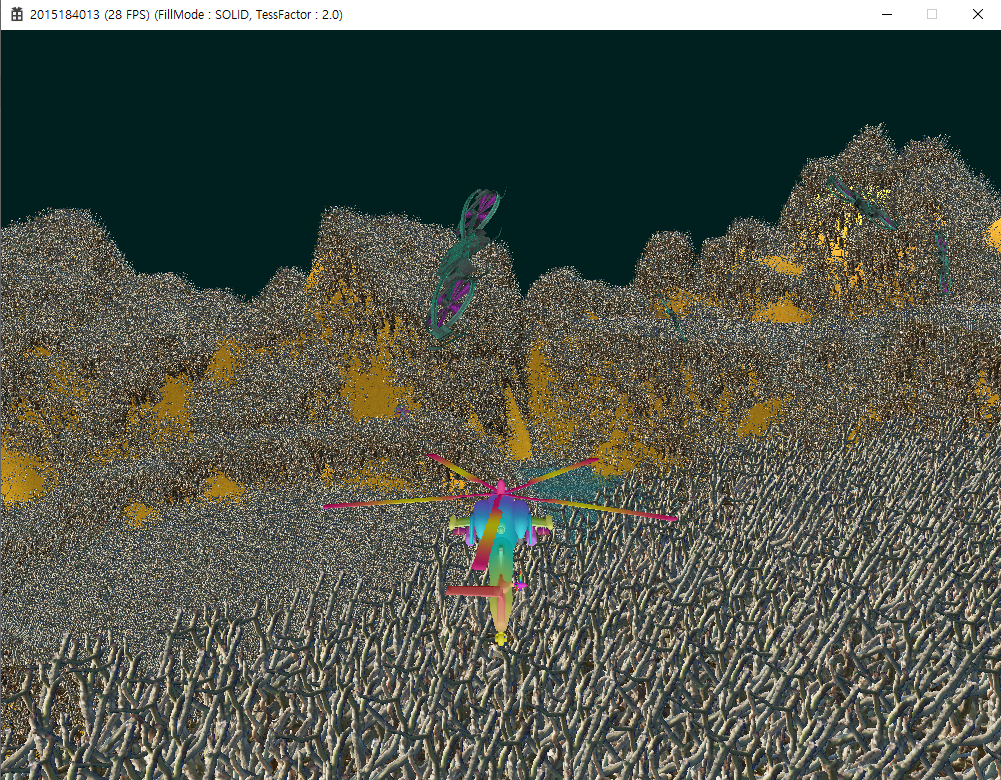
화성의 풍성해진 북극



1. 게임 개요

- 제목 : 화성의 풍성해진 북극

- 컨셉 : 화성의 북극이 풍성해졌다.

2. 목표

- 빌보드를 최대한 많이 배치하기

- 지형에 테셀레이션 적용하기

3. 조작

전진/후진 : W/S

Y축 자전 : A/D

(플레이어와 카메라가 함께 회전)

(카메라 방향과 같지 않을 경우 카메라가 알아서 따라 감)

상승/하강 : SPACE/SHIFT + SPACE

카메라 Y축 공전 : 마우스 왼쪽 클릭 + 드래그

미사일 발사 : 마우스 오른쪽 클릭

그래픽 설정

( 테셀레이션을 잘 확인하기 위함)

F1 : Tesselation Factor 조절

F2 : GRID/SOLID 모드 전환

F3 : 빌보드 렌더 ON/OFF

4. 과제 1에서 추가된 내용

**(가) 테셀레이션**

**- 목표**

카메라와의 거리에 따른 테셀레이션

사용자가 임의로 테셀레이션 레벨을 조종할수 있게 하기

**테셀리에션 적용을 위한 준비**

(1) Scene에서 루트 시그니쳐를 생성할 때, d3cdRootSignatureDesc의 Flags를 수정한다.

D3D12\_ROOT\_SIGNATURE\_FLAG\_DENY\_HULL\_SHADER\_ROOT\_ACCESS와 D3D12\_ROOT\_SIGNATURE\_FLAG\_DENY\_DOMAIN\_SHADER\_ROOT\_ACCESS을 지워주었다.

(2) CShader의 CreateShader()에서 헐쉐이더와 도메인 쉐이더도 set해주도록 한다.

m\_d3dPipelineStateDesc.HS = CreateHullShader();

m\_d3dPipelineStateDesc.DS = CreateDomainShader();

CreateHullShader()와 CreateDomainShader() 둘다 CreateVertexShader() 와 같은 내용의 가상함수이다.

(3) CterrainShader의 m\_d3dPipelineStateDesc의 PrimitiveTopologyType를 D3D12\_PRIMITIVE\_TOPOLOGY\_TYPE\_PATCH로 바꾼다.

(4) 그리드 모드로 확인할 수 있도록 m\_d3dPipelineStateDesc를 2개 만들었다.

m\_d3dPipelineStateDesc.RasterizerState.FillMode = D3D12\_FILL\_MODE\_WIREFRAME;

hResult = pd3dDevice->CreateGraphicsPipelineState(&m\_d3dPipelineStateDesc, \_\_uuidof(ID3D12PipelineState), (void\*\*)&m\_ppd3dPipelineStates[1]);

onPrepareRender() 에서 render할 PSO를 인자 nPipelineState를 받아 인덱스로 접근한다.

(5) CHeightMapGridMesh

원래는 터레인 오브젝트 하나가 커다란 메쉬를 통째로 하나 가지고 있는 방식이었으나, LabProject09-1-0를 보고 이에 맞게 바꿀 필요가 있음을 느꼈다. 그래서 Terrain Opject 만 m\_ppMeshes 배열을 가지고 있기로 하였다.

m\_nVertices는 25개, m\_d3dPrimitiveTopology는 D3D\_PRIMITIVE\_TOPOLOGY\_25\_CONTROL\_POINT\_PATCHLIST로 설정하였다.

**테셀레이션 상수 함수**

(1) 사용자가 테셀레이션 인자를 임의로 조절할수 있게 하기 위해 FLOAT2의 쉐이더 상수버퍼를 만들었다.

cbuffer cbTessFactor : register(b6)

{

float2 gfTessFactor : packoffset(c0);

//x : tessfactor , y : tessinsidefactor

};

(2) CheightMapTerrain이 ID3D12Resource\* m\_pd3dcbTessFactor를 가지고 있고, 생성하고 업데이트하는 방식은 Cwater의 m\_pcbfWave을 다루는 방식과 동일하다.

Edge와 insde를 따로 조절할수 있게 float2 자료형으로 만들었지만, 실제로는 인수가 둘다 같게 하였다.

void CHeightMapTerrain::CreateShaderVariables(ID3D12Device\* pd3dDevice, ID3D12GraphicsCommandList\* pd3dCommandList)

{

UINT ncbElementBytes = sizeof(float)\*2; //256의 배수

m\_pd3dcbTessFactor = ::CreateBufferResource(pd3dDevice, pd3dCommandList, NULL, ncbElementBytes, D3D12\_HEAP\_TYPE\_UPLOAD, D3D12\_RESOURCE\_STATE\_VERTEX\_AND\_CONSTANT\_BUFFER, NULL);

m\_pd3dcbTessFactor->Map(0, NULL, (void\*\*)&m\_pxmf2TessFactor);

CGameObject::CreateShaderVariables(pd3dDevice, pd3dCommandList);

}

void CHeightMapTerrain::UpdateShaderVariables(ID3D12GraphicsCommandList\* pd3dCommandList)

{

D3D12\_GPU\_VIRTUAL\_ADDRESS d3dcbGpuVirtualAddress = m\_pd3dcbTessFactor->GetGPUVirtualAddress();

pd3dCommandList->SetGraphicsRootConstantBufferView(9, d3dcbGpuVirtualAddress);

}

(3) F1키를 누르면 m\_pxmf2TessFactor를 변환해서 레벨을 조절할 수 있게 하였다.

void CHeightMapTerrain::SetTessellationMode() {

if ((\*m\_pxmf2TessFactor).x > 20)

\*m\_pxmf2TessFactor = XMFLOAT2(2, 2);

else

{

(\*m\_pxmf2TessFactor).x += 2; (\*m\_pxmf2TessFactor).y += 2;

} }

(4) VSTerrainTessellationConstant

강의자료 PPT를 참고하여 카메라와의 거리에 따라 테셀레이션 레벨이 달라지고, 또 사용자가 조절한 값이 적용 가능해지게 하였다.

거리의 최대는 1000이 예상되고 최소는 10 정도가 예상되었다. 그래서 거리(100 기준)에 따라 레벨이 나눠지게 하였다. 거리를 반비례한 값에 100을 곱하면, 레벨은 0에서 10까지 구분될 것이다. 이에 사용자가 조절한 gfTessFactor를 곱했다.

[partitioning("integer")]을 해두었으니 값이 0이 나와도 1로 자동 보정 될 것이다.

Edge와 inside가 모두 똑같아야 이쁜 네모 모양이 나오길래 그렇게 되게 하였다.

HS\_TERRAIN\_TESSELLATION\_CONSTANT **VSTerrainTessellationConstant**(InputPatch<VS\_TERRAIN\_OUTPUT, 25> input)

{

**// 제어점들의 위치 평균 으로 패치의 위치를 구한다.**

float3 vCenter;

for (int i = 0; i < 25; i++)

vCenter += input[i].position;

vCenter = vCenter / 25.f;

**//패치의 위치와 카메라와의 거리를 구한다.**

float fDistanceToCamera = distance(vCenter, gvCameraPosition);

float fTessFactor = (1.f/fDistanceToCamera)\*100.f \* gfTessFactor.x;

HS\_TERRAIN\_TESSELLATION\_CONSTANT output;

output.fTessEdges[0] =fTessFactor;

**…**

output.fTessInsides[0] = fTessFactor;

output.fTessInsides[1] = fTessFactor;

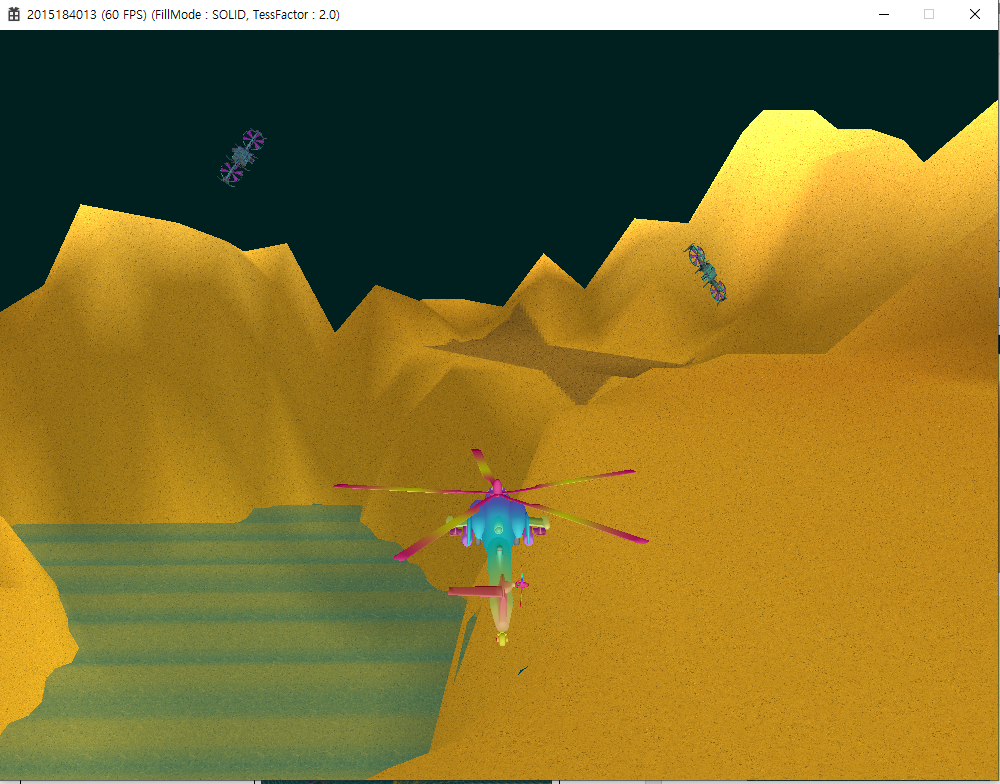
return(output);

}

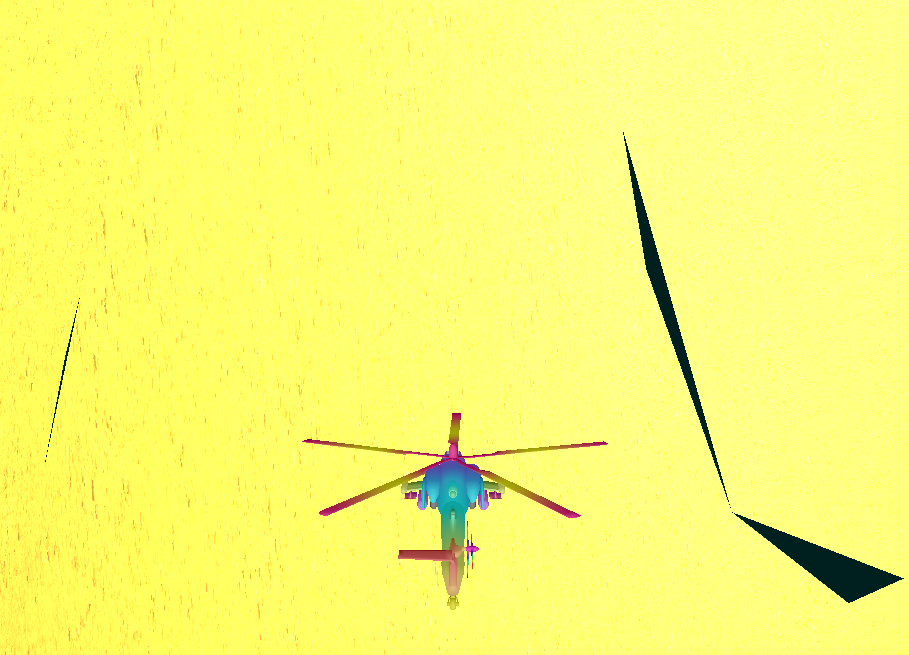
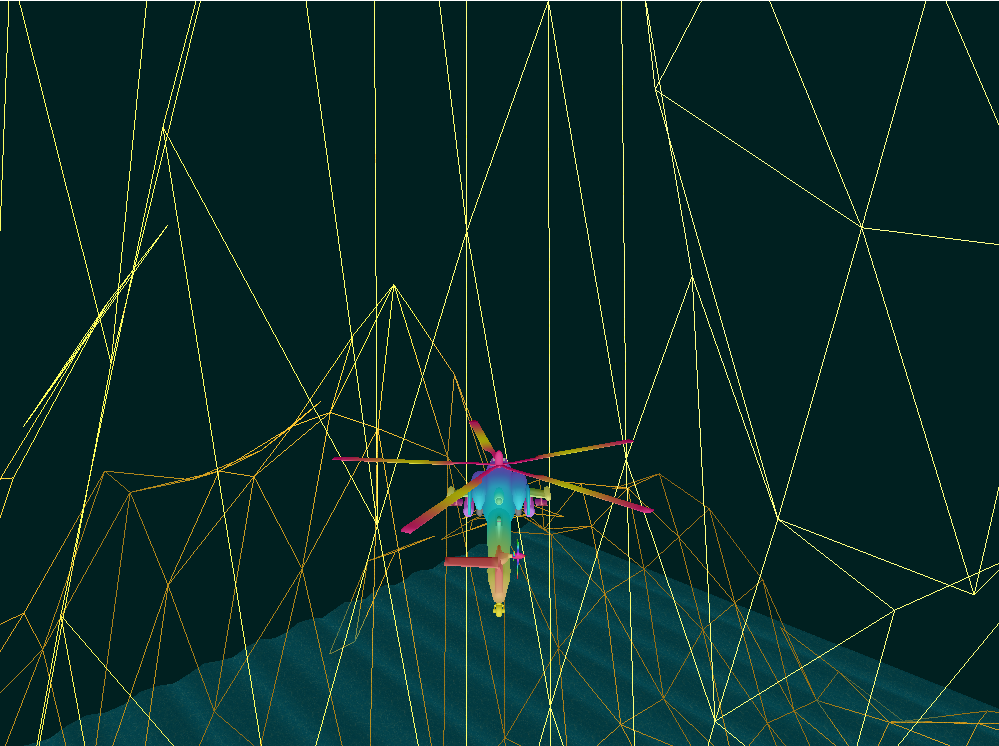
**적용결과**

|  |  |
| --- | --- |
| **gfTessFactor** |  |
| 2 |  |
| 10 |  |
| 20 |  |

**Solid mode를 적용해도 차이가 확연히 보인다.**

****

**문제점**



레벨이 다른 두 패치 사이의 경계면에, 디테일의 정도가 다른 차이가 빈 공간으로 나타났다. Solid 모드로 확인하면 눈에 너무 잘 띄었다. 이는 테셀레이션 레벨이 너무 낮을때 나타나는 문제로, 레벨이 높아질수록 개선되었다.

**(나) 빌보드**

**목표**

**- 10만개 빌보드를 띄우고 평균 fps가 30 이상 나오기**

**인스턴싱 방식으로 넘기기**

- LabProject080-0-1을 참고하며 CbillboardObjectsShader를 인스턴싱방식으로 바꾸었다.

(1) 빌보드의 종류는 하나만 되도록 하였다. 따라서 텍스쳐가 배열일 필요도, info로 종류를 구분할 필요도 없었다. 기하셰이더를 적용할 것이기 때문에 인스턴스 버퍼로 넘겨줄 VS\_VB\_BILLBOARD\_INSTANCE 구조체도 FLOAT3의 위치값, FLOAT2의 이미지 크기 값만 가지게 했다.

(2) 처음 BuildObjects를 할 때, 랜덤 포지션을 구하고, 랜덤포지션에 getheight()를 하는 연산을 100000번 하니까 처음 로딩하는 시간이 길어지는것처럼 느껴졌다. 그래서 따로 100000개의 Billboards를 따로 binary로 저장하고 이를 한번에 읽으면 로딩시간이 줄어들 줄 알았다. 결과는 차이가 1초 미만정도 밖에 나지 않았다.

m\_nInstances = 1'00000;

VS\_VB\_BILLBOARD\_INSTANCE\* pInstanceInfos = new VS\_VB\_BILLBOARD\_INSTANCE[m\_nInstances];

ifstream input("billboadInfos.bin", ios::in | ios::binary);

input.read((char\*)pInstanceInfos, sizeof(VS\_VB\_BILLBOARD\_INSTANCE)\* 100000);

input.close();

그 다음 m\_d3dInstancingBuffer 리소스를 만들고, 리소스 뷰를 만들었다.

m\_pd3dInstancesBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, pInstanceInfos, sizeof(VS\_VB\_BILLBOARD\_INSTANCE) \* m\_nInstances, D3D12\_HEAP\_TYPE\_DEFAULT, D3D12\_RESOURCE\_STATE\_VERTEX\_AND\_CONSTANT\_BUFFER, &m\_pd3dInstanceUploadBuffer);

m\_d3dInstancingBufferView.BufferLocation = m\_pd3dInstancesBuffer->GetGPUVirtualAddress();

**…**

**CreateInputLayout()**

Pd3dInputElementDescs의 원소들을 수정하였다.

struct VS\_BILLBOARD\_INSTANCING\_INPUT

{

float3 instancePosition : INSTANCEPOSITION;

float2 billboardInfo : BILLBOARDINFO; //(cx, cy)

};

pd3dInputElementDescs[0] = { "INSTANCEPOSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0, D3D12\_INPUT\_CLASSIFICATION\_PER\_INSTANCE\_DATA, 1 };

pd3dInputElementDescs[1] = { "BILLBOARDINFO", 0, DXGI\_FORMAT\_R32G32\_FLOAT, 0, 12, D3D12\_INPUT\_CLASSIFICATION\_PER\_INSTANCE\_DATA, 1 };

둘 다 InputSloatClass는 D3D12\_INPUT\_CLASSIFICATION\_PER\_INSTANCE\_DATA, InstanceDataStepRate는 1로 변경하였다.

**기하셰이더 적용하기**

기하셰이더 강의자료를 참고하였다.

(1) Scene에서 루트 시그니쳐를 생성할 때, d3cdRootSignatureDesc의 Flags를 수정한다. D3D12\_ROOT\_SIGNATURE\_FLAG\_DENY\_GEOMTRY\_SHADER\_ROOT\_ACCESS를 지워주었다.

(2) PipelineState의 PrimitiveTopologyType을 D3D12\_PRIMITIVE\_TOPOLOGY\_TYPE\_POINT로 변경한다.

(3) Render()

CShader::Render(pd3dCommandList, pCamera);

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

D3D12\_VERTEX\_BUFFER\_VIEW pVertexBufferViews[] = { m\_d3dInstancingBufferView };

pd3dCommandList->IASetVertexBuffers(0, \_countof(pVertexBufferViews), pVertexBufferViews);

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

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

**// 정점 하나를, m\_instance개수만큼 그린다.**

(4) CShader의 CreateShader()에서 기하쉐이더도 set해주도록 한다.

m\_d3dPipelineStateDesc.GS = CreateGeometryShader();

[maxvertexcount(4)] //강의자료 내용과 유사함

void GSBillboardInstancing(point VS\_BILLBOARD\_INSTANCING\_OUTPUT input[1], uint primID : SV\_PrimitiveID, inout TriangleStream <GS\_OUT> outStream)

{

**…**

float fHalfW = input[0].billboardInfo.x ; //가로길이

float fHalfH = input[0].billboardInfo.y ; //세로길이

float4 pVertices[4];

pVertices[0] = float4(input[0].position + fHalfW \* vRight - fHalfH \* vUp, 1.0f);

pVertices[1] = float4(input[0].position + fHalfW \* vRight + fHalfH \* vUp, 1.0f);

pVertices[2] = float4(input[0].position - fHalfW \* vRight - fHalfH \* vUp, 1.0f);

pVertices[3] = float4(input[0].position - fHalfW \* vRight + fHalfH \* vUp, 1.0f);

float2 pUVs[4] = { float2(0,1),float2(0,0),float2(1,1),float2(1,0) };

GS\_OUT output;

for (int i = 0; i < 4; i++) {

**…**

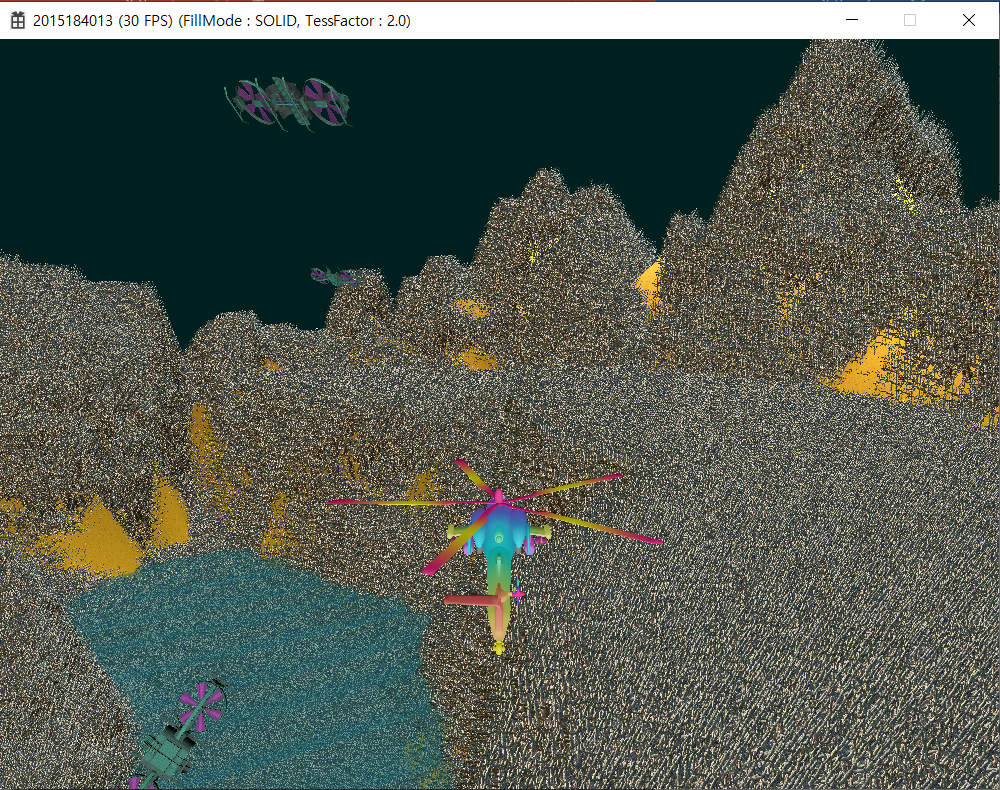
outStream.Append(output);

}

}

(5) PipelineState의 RasterizerDesc에서 CULLMODE를 D3D12\_CULL\_MODE\_NONE로 변경하여 컬링이 되지 않도록 했다.

적용 결과 : 평균 FPS가 빌보드 10만 개일 때 52, 20만개일 때 30이 나왔다. (Release 모드 기준)



**(다) 기타**

(1) FPS를 높이기 위해 빌보드에 적용되는 샘플러의 필터링 방법을 D3D12\_FILTER\_MIN\_MAG\_MIP\_LINEAR에서 D3D12\_FILTER\_MIN\_POINT\_MAG\_LINEAR\_MIP\_POINT으로 변경했다. 확대했을때도 점 필터링을 했을때는 너무 모자이크처럼 보여서 확대할 경우에만 선형 필터링을 하도록 했다. 결과는 별차이 없었지만 필요해 보였다.

(2) SetWindowModeText() 함수를 만들어서 윈도우 텍스트에 적용중인 터레인 fillmode와 tessfactor가 표시되게 했다.

m\_GameTimer.GetFrameRate(m\_pszFrameRate + 12, 37);

XMFLOAT2 mode = m\_pTerrain->GetPipelineMode(); //x : Tessfactor, y : fillmode(bool)

if (mode.y)

wcscat\_s(m\_pszFrameRate, \_T(" (FillMode : GRID, "));

else

wcscat\_s(m\_pszFrameRate, \_T(" (FillMode : SOLID, "));

size\_t nLength = \_tcslen(m\_pszFrameRate);

\_stprintf\_s(m\_pszFrameRate + nLength, 70 - nLength, \_T("TessFactor : %1.1f)"), mode.x);

xmf3Position.x, xmf3Position.y, xmf3Position.z);

::SetWindowText(m\_hWnd, m\_pszFrameRate);

5. 후기

이번에도 성선미 학우와 같이 연구하면서 진행하였다. 이번 과제는 과제 1보다 훨씬 지루하지 않고 재밌었다. 특히 테셀레이션을 변화를 눈으로 계속 확인해가면서 직접 다뤄보니까 너무 신기했다. 아쉬운건 빌보드를 다양하게 하지도 못한 것과 터레인도 아주 넓은 걸로 여러개를 새로 배치하고 싶었는데 시간 상 그러지 못한 것이다. 할수 있을 것 같은데 아쉽다. 그리고 가장 고민한 건 FPS를 높이는 것이었다. 인스턴싱도 하고 기하셰이더 까지 적용했는데 생각보다 FPS가 높게 나오지 않아서 실망스러웠다.