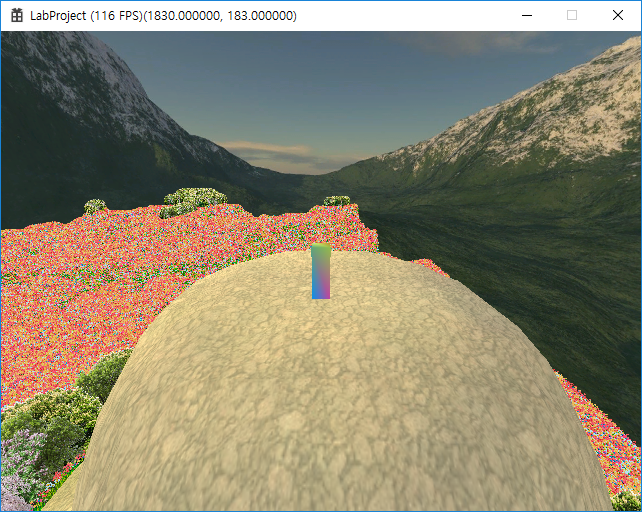
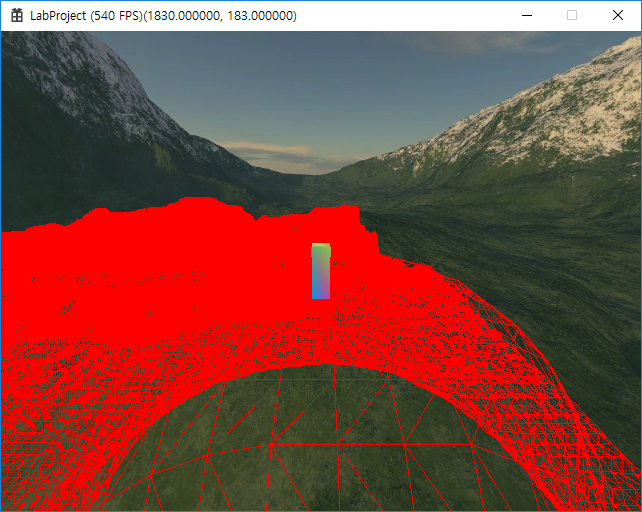
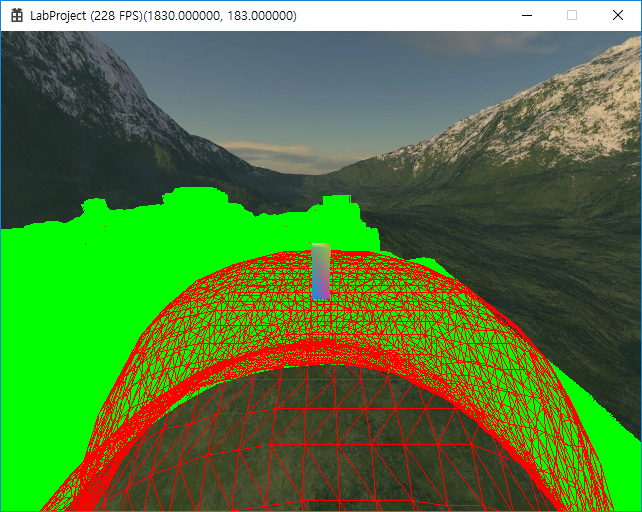
**3D 게임프로그래밍2**

**과제 2**

**게임공학과 2013180013 박진수**





**<실행 결과>**

**와이어프레임으로 씬을 렌더링할 때 지형은 빨간색, 빌보드 객체는 연두색으로 렌더링하였음.**

**단, 기하 쉐이더를 적용하지 않은 상태에서는 지형이나 빌보드 객체나 모두 빨간색으로 렌더링함.  
빌보드 오브젝트는 대략 100만개 정도 생성하였음.**

* **과제 목표**
  + **과제1에 기하 쉐이더를 적용한다.**
  + **기하 쉐이더를 이용하여 지형을 렌더링한다.**
  + **기하 쉐이더를 이용하여 빌보드 오브젝트를 렌더링한다.**
  + **기하 쉐이더를 이용하여 렌더링한 것을 증명하기 위해 적용하지 않은 씬과 적용한 씬을 와이어프레임으로 볼 수 있도록 한다.**
* **조작법** 
  + **방향키 : 플레이어 이동**
  + **Z : 기하 쉐이더를 적용한 씬을 렌더링**
  + **X : 기하 쉐이더를 적용한 씬을 와이어프레임으로 렌더링**
  + **C : 기하 쉐이더를 적용하지 않은 씬을 렌더링**
  + **V : 기하 쉐이더를 적용하지 않은 씬을 와이어프레임으로 렌더링**
  + **S : 빌보드 오브젝트를 렌더링하지 않는다.**
  + **R : 플레이어를 초기 위치로 이동**
* **코드 설명**
  + **그래픽스 파이프라인 상태를 생성할 때 기하 쉐이더 적용, 기하 쉐이더 적용 와이어 프레임, 기하 쉐이더 미적용, 기하 쉐이더 미적용 와이어 프레임 이렇게 총 4종류의 그래픽스 파이프라인 상태를 생성하였다.**

|  |
| --- |
| CompileShaderFromFile(L"Shaders.hlsl", "VS", "vs\_5\_1", ppd3dShaderBlob) |
| CompileShaderFromFile(L"Shaders.hlsl", "VSWire", "vs\_5\_1", ppd3dShaderBlob) |
| CompileShaderFromFile(L"Shaders.hlsl", "GS", "gs\_5\_1", ppd3dShaderBlob) |
| CompileShaderFromFile(L"Shaders.hlsl", "GSWire", "gs\_5\_1", ppd3dShaderBlob) |
| CompileShaderFromFile(L"Shaders.hlsl", "PS", "ps\_5\_1", ppd3dShaderBlob) |
| CompileShaderFromFile(L"Shaders.hlsl", "PSWire", "ps\_5\_1", ppd3dShaderBlob) |

* + **키보드 입력으로 그래픽스 파이프라인 상태 배열의 인덱스를 결정하는 변수를 바꿔주고 씬에서 Render 함수를 호출할 때 해당 변수를 각 쉐이더 Render함수의 매개변수로 보내주고 각 쉐이더에서 해당 그래픽스 파이프라인 상태 배열의 인덱스로 Set해주고 렌더링하도록 했다.**

|  |
| --- |
| class CScene |
| private:  UINT m\_nSceneMode = 1; |

|  |
| --- |
| CScene::Render(…) |
| …  m\_ppShaders[i]->Render(pd3dCommandList, m\_nSceneMode);  … |

* **코드 설명(Terrain)**
  + **기하 쉐이더에서 지형의 삼각형을 구성하는 정점 3개를 받아 각 두 정점들의 중간값(위치, 색상, uv좌표)을 갖는 새로운 정점 3개를 생성한 후 정점 6개를 이용하여 새로운 삼각형 프리미티브 4개를 만들어 다음 단계로 넘겨주었다.**

|  |
| --- |
| [maxvertexcount(8)]  void GSTerrain(triangle VS\_TERRAIN\_OUTPUT input[3], inout TriangleStream<VS\_TERRAIN\_OUTPUT> outStream) |
| VS\_TERRAIN\_OUTPUT v[6];  v[0].position = input[0].position;  …  v[3].position = (input[0].position + input[1].position) \* 0.5f;  v[3].color = (input[0].color + input[1].color) \* 0.5f;  v[3].uv0 = (input[0].uv0 + input[1].uv0) \* 0.5f;  v[3].uv1 = (input[0].uv1 + input[1].uv1) \* 0.5f;  v[3].ntex = input[0].ntex; // 디테일 텍스쳐 배열의 인덱스  …  v[5].position = (input[0].position + input[2].position) \* 0.5f;  …  outStream.Append(v[0]), (v[3]), (v[5]), (v[4]), (v[2])  outStream.RestartStrip();  outStream.Append(v[3]), (v[1]), (v[4]); |

* + **와이어 프레임으로 렌더링할 때는 색상, uv좌표, 텍스쳐 인덱스 변수가 필요 없기 때문에 기하 쉐이더에서도 위치 정보만을 사용하여 정점을 만들어 렌더링하였다.**

|  |
| --- |
| [maxvertexcount(8)]  void GSTerrainWire(triangle VS\_WIRE\_OUTPUT input[3], inout TriangleStream<VS\_WIRE\_OUTPUT> outStream) |

struct VS\_WIRE\_OUTPUT

float4 position : SV\_POSITION;

* **코드 설명(Billboard)**
  + **빌보드는 4개의 정점으로 렌더링한 기존의 방법과는 달리 정점 하나만을 이용하여 기하 쉐이더로 정점 4개를 만들어 렌더링하도록 쉐이더 클래스가 오브젝트를 갖게 하지 않고 정점들의 포인터를 갖도록 했다.**
  + **정점에는 위치, 크기, 빌보드 텍스처 배열의 인덱스를 갖는다.**

class CVertex

{

public:

XMFLOAT3 m\_xmf3Position;

|  |
| --- |
| class CBillboardVertex : public CVertex  {  public:  XMFLOAT2 m\_xmf2Size;  UINT m\_nTexture; |

* + **정점만을 이용하여 렌더링하는 클래스를 만들어서 이 클래스에 정점들의 포인터, 정점 개수, 정점 버퍼 등 쉐이더에서 정점을 이용하여 렌더링할 수 있게 변수들을 선언했다.**

|  |
| --- |
| class CVerticesObjectsShader : public CTexturedShader  {  protected:  UINT m\_nVertices;  CBillboardVertex \*m\_pVertices;  ID3D12Resource \*m\_pd3dVertexBuffer = NULL;  ID3D12Resource \*m\_pd3dVertexUploadBuffer = NULL;  D3D12\_VERTEX\_BUFFER\_VIEW m\_d3dVertexBufferView; |

* + **정점만 이용하기 때문에 그와 관련된 그래픽스 파이프라인 상태를 생성하였다.**

|  |
| --- |
| d3dPipelineStateDesc.VS = CreateGeometryVertexShader(&pd3dVertexShaderBlob);  d3dPipelineStateDesc.GS = CreateGeometryShader(&pd3dGeometryShaderBlob);  d3dPipelineStateDesc.PS = CreateGeometryPixelShader(&pd3dPixelShaderBlob);  d3dPipelineStateDesc.InputLayout = CreateInputLayout();  d3dPipelineStateDesc.PrimitiveTopologyType = D3D12\_PRIMITIVE\_TOPOLOGY\_TYPE\_POINT; |
| CompileShaderFromFile(L"Shaders.hlsl", "VSGeometryBillboard", "vs\_5\_1", ppd3dShaderBlob); |
| CompileShaderFromFile(L"Shaders.hlsl", "GSBillboard", "gs\_5\_1", ppd3dShaderBlob); |
| CompileShaderFromFile(L"Shaders.hlsl", "PSGeometryBillboard", "ps\_5\_1", ppd3dShaderBlob) |
| pd3dInputElementDescs[0] = { "POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0, … }  pd3dInputElementDescs[1] = { "SIZE", 0, DXGI\_FORMAT\_R32G32\_FLOAT, 0, 12, … }  pd3dInputElementDescs[2] = { "TEXTURE", 0, DXGI\_FORMAT\_R32\_UINT, 0, 20, … } |

* + **쉐이더의 BuildObjects 함수에서 오브젝트 클래스를 생성하는 대신 정점들을 생성하여 정점 버퍼와 정점 버퍼 뷰를 생성하고 Render 함수에서 직접 정점 버퍼를 이용하여 렌더링 하였다.**
  + **정점 쉐이더에서는 입력 조립기에서 받은 정보를 그대로 기하 쉐이더로 보내도록 했다.**

|  |
| --- |
| VS\_GEOMETRY\_BILLBOARD\_OUT VSGeometryBillboard(VS\_BILLBOARD\_IN input)  {  struct VS\_GEOMETRY\_BILLBOARD\_OUT  {  float3 centerW : POSITION;  float2 sizeW : SIZE;  uint ntexture : TEXTURE;  };  VS\_GEOMETRY\_BILLBOARD\_OUT output;  struct VS\_BILLBOARD\_IN  {  float3 posW : POSITION;  float2 sizeW : SIZE;  uint ntexture : TEXTURE;  };  output.centerW = input.posW;  output.sizeW = input.sizeW;  output.ntexture = input.ntexture;  return(output);  } |

* + **기하 쉐이더에서는 정점 1개를 받아 정점 4개를 생성한 후 정점 4개를 이용하여 새로운 삼각형 프리미티브 2개를 다음 단계로 넘겨주었다.**
  + **정점을 생성할 때 빌보드 오브젝트가 카메라를 향한 Look 벡터를 카메라의 위치와 정점의 위치를 빼서 구하고 Up 벡터와 Look 벡터에 수직인 Right 벡터를 외적으로 구한 후 각 정점의 위치를 계산하도록 했다.**

|  |
| --- |
| [maxvertexcount(4)]  void GSBillboard(point VS\_GEOMETRY\_BILLBOARD\_OUT input[1], uint primID : SV\_PrimitiveID,  inout TriangleStream<GS\_BILLBOARD\_OUT> outStream)  struct GS\_BILLBOARD\_OUT  {  float4 posH : SV\_POSITION;  float2 uv : TEXCOORD;  uint ntexture : TEXTURE;  }; |
| float3 vUp = float3(0.0f, 1.0f, 0.0f);  float3 vLook = gvCameraPosition.xyz - input[0].centerW;  vLook = normalize(vLook);  vLook.y = 0.0f;  float3 vRight = normalize(cross(vUp, vLook));  float fHalfW = input[0].sizeW.x \* 0.5f;  float fHalfH = input[0].sizeW.y;  float4 fVertices[4];  fVertices[0] = float4(input[0].centerW + fHalfW \* vRight, 1.0f);  fVertices[1] = float4(input[0].centerW + fHalfW \* vRight + fHalfH \* vUp, 1.0f);  fVertices[2] = float4(input[0].centerW - fHalfW \* vRight, 1.0f);  fVertices[3] = float4(input[0].centerW - fHalfW \* vRight + fHalfH \* vUp, 1.0f);  … |

* + **와이어 프레임을 위한 쉐이더 코드에서는 지형과 마찬가지로 위치값만을 이용하였다.**
* **코드 설명(Billboard-Flower Object)**
  + **나무 오브젝트는 위와 같은 방식으로 렌더링했지만 꽃 오브젝트는 바람에 살랑살랑거리는 연출을 위해 약간 다른 방식으로 코딩하였다.**
  + **먼저 쉐이더 클래스에서 변위를 위한 행렬과 프레임당 회전해야하는 각도, 델타값을 멤버 변수로 선언하여 사용했다.**

|  |
| --- |
| class CFlowerObjectsShader : public CVerticesObjectsShader |
| protected:  ID3D12Resource \*m\_pd3dsbRotations = NULL;  ROTATIONINFO \*m\_psbMappedRotations = NULL;  #define NUMROTATEMATRIX 10  ROTATIONINFO m\_Rotation[NUMROTATEMATRIX];  float m\_fRotationAngle[NUMROTATEMATRIX];  float m\_fRotationDelta[NUMROTATEMATRIX]; |

|  |
| --- |
| CFlowerObjectsShader::BuildObjects(…) |
| …  for (int i = 0; i < NUMROTATEMATRIX; i++)  m\_Rotation[i].xmf4x4Rotate = Matrix4x4::Identity();  m\_fRotationAngle[i] = rand() % 150 / 100.0f;  m\_fRotationDelta[i] = i % 2 ? 1.5f : -1.5f;  **…** |

* + **회전을 위한 행렬은 구조화된 버퍼로 쉐이더에 넘겨주도록 하였다.**

|  |
| --- |
| CFlowerObjectsShader::CreateShaderVariables(…) |
| m\_pd3dsbRotations = ::CreateBufferResource(…)  StructuredBuffer<ROTATIONINFO> gRotationInfo : register(t0);  m\_pd3dsbRotations->Map(…) |
| CFlowerObjectsShader::UpdateShaderVariables(…) |
| pd3dCommandList->SetGraphicsRootShaderResourceView(…)  for (int j = 0; j < NUMROTATEMATRIX; j++)  XMStoreFloat4x4(&m\_psbMappedRotations[j].xmf4x4Rotate, XMMatrixTranspose(XMLoadFloat4x4(&m\_Rotation[j].xmf4x4Rotate))); |

* + **쉐이더 클래스의 AnimateObjects 함수에서 회전 행렬을 갱신시켰다.**

|  |
| --- |
| CFlowerObjectsShader::AnimateObjects(…) |
| for (int i = 0; i < NUMROTATEMATRIX; i++)  if (m\_fRotationAngle[i] <= -1.5f) m\_fRotationDelta[i] = 1.5f;  if (m\_fRotationAngle[i] >= +1.5f) m\_fRotationDelta[i] = -1.5f;  m\_fRotationAngle[i] += m\_fRotationDelta[i] \* fTimeElapsed;  XMStoreFloat4x4(&m\_Rotation[i].xmf4x4Rotate, XMMatrixRotationRollPitchYaw(0.0f, 0.0f, XMConvertToRadians(m\_fRotationAngle[i]))); |

* + **기하 쉐이더 함수에서 이 회전 행렬을 이용하여 정점의 위치를 바꿔주어야 하는데 그러기 위해서는 정점의 위치 자체가 이미 월드 좌표이기 때문에 정점 4개를 생성할 때 정점의 위치를 계산식에 넣지 않고 생성하고 그 후 카메라를 바라보기 위한 행렬을 만들어 응용 프로그램에서 넘겨준 회전 행렬을 곱한 행렬을 새로 만들어 그 행렬을 월드 변환 행렬처럼 사용하여 4개의 정점들의 위치 벡터와 곱하여 회전을 적용시켰다.**

|  |
| --- |
| void GSRotateBillboard(point VS\_GEOMETRY\_BILLBOARD\_OUT input[1], uint primID : SV\_PrimitiveID, inout TriangleStream<GS\_BILLBOARD\_OUT> outStream) |
| …  float4 fVertices[4];  fVertices[0] = float4(+fHalfW \* vRight, 1.0f);  fVertices[1] = float4(+fHalfW \* vRight + fHalfH \* vUp, 1.0f);  fVertices[2] = float4(-fHalfW \* vRight, 1.0f);  fVertices[3] = float4(-fHalfW \* vRight + fHalfH \* vUp, 1.0f);  …  matrix mtxWorld = CalculateWorld(input[0].centerW, primID);  …  output.posH = mul(mul(mul(fVertices[i], mtxWorld), gmtxView), gmtxProjection);  … |

float3 vUp = float3(0.0f, 1.0f, 0.0f);

float3 vLook = gvCameraPosition.xyz - center;

vLook = normalize(vLook);

vLook.y = 0.0f;

float3 vRight = normalize(cross(vUp, vLook));

matrix mtxWorld;

mtxWorld.\_11\_12\_13\_14 = float4(vRight, 0.0f);

mtxWorld.\_21\_22\_23\_24 = float4(vUp, 0.0f);

mtxWorld.\_31\_32\_33\_34 = float4(vLook, 0.0f);

mtxWorld.\_41\_42\_43\_44 = float4(center, 1.0f);

mtxWorld = mul(gRotationInfo[primID % 5].xmf4x4Rotate, mtxWorld);

return(mtxWorld);

* + **프리미티브 ID로 어떤 회전 행렬을 결정할지 정해주었다.**
  + **이렇게 해서 꽃 오브젝트가 월드 변환 행렬을 가지고 있지 않으면서 바람에 살랑살랑거리도록 만들었다.**
* **부족한 점**
  + **블렌딩을 사용하지 않고 discard 명령어를 사용해서 투명도를 표현했기 때문에 다음 과제에서는 블렌딩을 사용하여 투명도를 표현하도록 한다.**