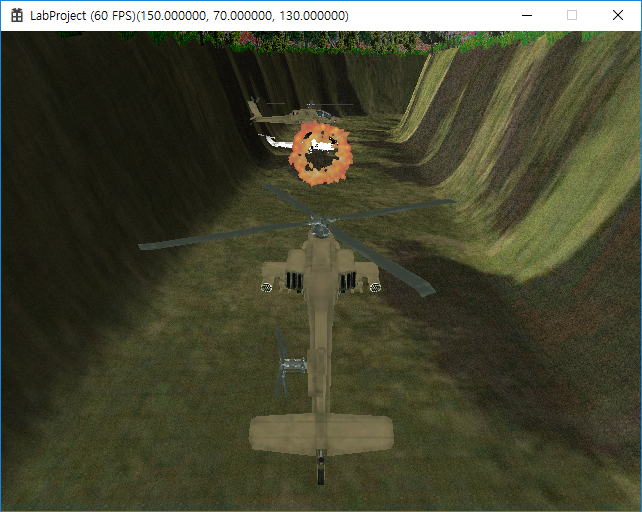
**3D 게임프로그래밍2**

**과제 3**

**게임공학과 2013180013 박진수**

**<실행 결과>**

**해당 프로젝트는 LabProject07-9-4를 기반으로 만들었다. 과제2에서 만든 모든 오브젝트, 머테리얼, 메쉬 등 LabProject07-9-4의 형태로 바꾸었고 조명 처리도 같이 적용하였다.**

* **과제 목표**
  + **과제2의 지형을 계곡의 형태로 만들고 계곡 위에 꽃과 나무를 배치한다. (빌보드) 그리고 계곡 부분에는 계곡의 느낌을 준다.**
  + **플레이어를 계층 구조 형태의 헬리콥터 오브젝트로 만들고, 적도 같은 계층 구조 형태의 오브젝트로 만든다.**
  + **플레이어는 미사일을 발사할 수 있으며 미사일의 꼬리 부분에는 텍스처 애니메이션을 이용해 화염을 그리고 적이 미사일에 맞으면 폭발하도록 한다. 이때 폭발하는 것도 텍스처 애니메이션으로 표현한다.**
  + **적은 계곡에서만 등장한다.**
  + **플레이어가 1인칭 모드로 시점을 변경할 때 조종석이 보이도록 UI를 만든다.**
  + **플레이어가 발사 버튼을 누르면 정조준을 할 수 있게 하고 조준점의 바깥 화면은 흐리게 한다.**
  + **조종석에는 백미러가 있어서 뒤에 적을 볼 수 있다.**
* **구현한 사항**
  + **지형을 계곡 형태로 만들고 계곡 위에 꽃과 나무를 배치함.**
  + **계곡 부분에 계곡의 느낌을 줌.**
  + **플레이어와 적을 계층 구조 형태의 헬리콥터 오브젝트로 만듦.**
  + **플레이어는 미사일을 발사할 수 있고 미사일의 꼬리 부분에는 화염 같은 게 그려짐.**
  + **적이 미사일에 맞으면 텍스처 애니메이션으로 폭발을 표현함.**
  + **플레이어가 1인칭 모드로 시점을 변경했을 때 조종석이 보이도록 UI를 만듦.**
* **구현하지 못한 사항**
  + **적이 계곡에서 플레이어 정면에 생성되도록 하는 것을 구현하지 못했음.**
  + **정조준 표현을 구현하지 못했음.**
  + **조종석의 백미러를 구현하지 못했음.**
* **조작법** 
  + **W, S, A, D, Q, E: 플레이어 앞뒤좌우상하 이동**
  + **F1 : 1인칭 모드로 전환**
  + **F2 : 3인칭 모드로 전환**
  + **SPACE : 미사일 발사**
* **코드 설명**
  + **지형을 계곡으로 만들고 계곡 위에 꽃과 나무를 배치, 계곡 부분에 계곡 느낌을 주는 부분은 과제1에서 쓴 방식으로 똑같이 로우맵 이미지를 사용했다.**
  + **플레이어와 적을 계층 구조 형태의 오브젝트로 만드는 부분은 LabProject07-9-4를 그대로 사용했다.**
* **코드 설명( 텍스처 애니메이션 )**
  + **미사일의 꼬리 부분에 화염, 미사일과 적 오브젝트 충돌시 폭발을 텍스처 애니메이션으로 구현하기 위해 씬에서 모든 이펙트를 관리하는 쉐이더 클래스를 생성했다. 이펙트 쉐이더 클래스의 BuildObjects 함수에서 화염과 폭발 이펙트의 메쉬와 텍스처를 갖는 오브젝트를 미리 만들어 두고 나중에 이펙트 오브젝트가 생성될 때마다 미리 생성한 오브젝트를 자식으로 갖게 해서 이펙트 오브젝트를 생성할 때마다 메쉬와 텍스처를 새로 만들어 내는 걸 피했다. 또한 이펙트 오브젝트는 빌보드이기 때문에 매 프레임마다 카메라를 바라보도록 만들었다.**

|  |
| --- |
| void CEffectShader::BuildObjects(…) |
| …  m\_nEffectObjects = 2;  m\_pEffectObjects = new CGameObject\*[m\_nEffectObjects];  …  CMaterial \*pMaterial = new CMaterial(1);  …  CRectMesh \*pMesh = new CRectMesh(pd3dDevice, pd3dCommandList, 20.0f, 20.0f);  m\_pEffectObjects[0] = new CGameObject(1);  m\_pEffectObjects[0]->SetMesh(pMesh);  m\_pEffectObjects[0]->SetMaterial(0, pMaterial);  …  m\_pEffectObjects[1] = new CGameObject(1);  m\_pEffectObjects[1]->SetMesh(pMesh);  m\_pEffectObjects[1]->SetMaterial(0, pMaterial); |

* + **이펙트 오브젝트는 AddObject() 함수에서 위치와 부가 정보가 Set된 오브젝트와 타입을 매개변수로 받아 이펙트 쉐이더 클래스에 오브젝트로 추가하도록 했다.**

|  |
| --- |
| void CEffectShader::AddObject(CGameObject\* pObject, EFFECT\_TYPE efType) |
| switch (efType)  enum EFFECT\_TYPE  {  EFFECT\_TYPE\_EXPLOSION,  EFFECT\_TYPE\_BOOSTER  };  {  case EFFECT\_TYPE\_EXPLOSION:  ((CEffect\*)pObject)->SetMaxSprite(5, 3, 12);  pObject->SetChild(m\_pEffectObjects[0]);  …  for (int i = 0; i < m\_nObjects; i++)  {  if (!m\_ppObjects[i])  {  m\_ppObjects[i] = pObject;  … |

* + **이펙트 오브젝트는 텍스처 애니메이션을 구현하기 위해 스프라이트 이미지의 최대 길이와 각 행과 열의 길이, 그리고 현재 행과 열의 위치를 멤버 변수로 갖고 이것들을 이용해 쉐이더로 XMFLOAT4 자료형의 스프라이트 데이터(x, y : 현재 행과 열의 위치, z, w : 1 / 행과 열의 최대 길이)를 보내주고 이 데이터로 행렬을 만들어 uv좌표에 곱해서 텍스처와 매핑되는 위치를 정해주도록 했다. 현재 행과 열의 위치는 프레임마다 열의 위치를 1씩 올리고 열의 위치가 최대 열의 길이가 되면 행의 위치를 1씩 올려주도록 해서 무한 루프 혹은 한 루프만 돌고 사라지도록 했다.**

|  |
| --- |
| m\_nMaxSpriteX = 열 최대 길이 , m\_nMaxSpriteY = 행 최대 길이 , m\_nMaxSprite = 스프라이트 이미지 최대 길이  **m\_xmf4Sprite.x = 1 / 열 최대 길이 , m\_xmf4Sprite.y = 1 / 행 최대 길이**  **m\_xmf4Sprite.z = 현재 열의 위치 , m\_xmf4Sprite.w = 현재 행의 위치** |
| pd3dCommandList->SetGraphicsRoot32BitConstants(14, 4, &m\_xmf4Sprite, 0); |
| void CEffect::SpriteAnimate() |
| if (m\_nSpritePosX + m\_nSpritePosY \* m\_nMaxSpriteX == m\_nMaxSprite) Delete();  else  {  m\_nSpritePosX++;  if (m\_nSpritePosX == m\_nMaxSpriteX)  {  m\_nSpritePosX = 0;  m\_nSpritePosY++;  }  if (m\_nSpritePosY == m\_nMaxSpriteY)  m\_nSpritePosY = 0;  } |
| VS\_TEXTURED\_OUTPUT VSEffect(VS\_TEXTURED\_INPUT input) |
| VS\_TEXTURED\_OUTPUT output;  float4 gfTextureSpriteInfo : packoffset(c0);  …  float3x3 f3x3Sprite = float3x3(**gfTextureSpriteInfo.x**, 0.0f, 0.0f, 0.0f, **gfTextureSpriteInfo.y**, 0.0f, **input.uv.x \* gfTextureSpriteInfo.x**, **input.uv.y \* gfTextureSpriteInfo.y, 1**.0f);  float3 f3Sprite = float3(gfTextureSpriteInfo.zw, 1.0f);  output.uv = (float2)mul(f3Sprite, f3x3Sprite);  return(output); |

* + **쉐이더에서는 3x3 행렬을 만들 때 11, 22는 그대로 두되 31, 32에는 각 행과 열 증가치(?)로 바꾸고 따로 float3 변수에 현재 행과 열의 위치를 저장하여 uv 좌표를 계산했다.**
  + **충돌 시 폭발 이펙트는 충돌된 위치에서만 있기 때문에 상관없지만 미사일의 꼬리 부분을 따라다니며 화염(?)을 표현하기 위해서 CObjectEffect 클래스를 CEffect 클래스로부터 상속받아 미사일을 따라다닐 수 있게 Follow 함수를 구현했다.**

|  |
| --- |
| void CObjectEffect::Follow()  XMFLOAT4X4 \*m\_xmf4x4Parent; // 따라다닐 오브젝트의 위치  XMFLOAT3 m\_xmf3Offset; |
| if (m\_xmf4x4Parent)  {  XMFLOAT4X4 xmf4x4Rotate = Matrix4x4::Identity();  XMFLOAT3 xmf3Right = XMFLOAT3(m\_xmf4x4Parent->\_11, m\_xmf4x4Parent->\_12, m\_xmf4x4Parent->\_13);  …  xmf4x4Rotate.\_11 = xmf3Right.x; xmf4x4Rotate.\_21 = xmf3Up.x; xmf4x4Rotate.\_31 = xmf3Look.x;  …  XMFLOAT3 xmf3Offset = Vector3::TransformCoord(m\_xmf3Offset, xmf4x4Rotate);  XMFLOAT3 xmf3LookAt = XMFLOAT3(m\_xmf4x4Parent->\_41, m\_xmf4x4Parent->\_42, m\_xmf4x4Parent->\_43);  XMFLOAT3 xmf3Position = Vector3::Add(xmf3LookAt, xmf3Offset);  m\_xmf4x4Transform.\_41 = xmf3Position.x; m\_xmf4x4Transform.\_42 = xmf3Position.y; m\_xmf4x4Transform.\_43 = xmf3Position.z;  } |

* **코드 설명( 미사일 )**
  + **미사일 오브젝트는 GunShip 모델의 Hellfire\_Missile를 이용하여 미사일 오브젝트를 생성할 때마다 자식으로 만들어 렌더링하도록 했다.**

|  |
| --- |
| void CAirplanePlayer::PrepareAnimate() |
| **…**  **m\_pHellfireMissileFrame** = FindFrame("Hellfire\_Missile");  m\_pHellfireMissileFrame->AddRef();  **…** |

* + **플레이어를 생성할 때 미사일에 해당하는 프레임을 따로 저장한 후 SPACE를 누를 때마다 Shot 함수를 호출해 미사일 오브젝트를 플레이어 클래스에서 멤버 변수로 갖고 있는 m\_pMissileObjects에 하나씩 추가시키도록 했다.**

|  |
| --- |
| void CAirplanePlayer::Shot() |
| if (m\_bShotable)  #define NUMOFMissile 2000  {  for (int i = 0; i < NUMOFMissile; i++)  {  CGameObject \*m\_pMissileObjects[NUMOFMissile] = { 0 };  if (!m\_pMissileObjects[i])  {  m\_pMissileObjects[i] = new CMissileObject();  m\_pMissileObjects[i]->SetPosition(m\_xmf3Position);  …  m\_pMissileObjects[i]->SetChild(**m\_pHellfireMissileFrame**);  …  CObjectEffect \*pEffectObject = new CObjectEffect();  pEffectObject->SetPosition(m\_xmf3Position);  pEffectObject->SetParent(&m\_pMissileObjects[i]->m\_xmf4x4World);  pEffectObject->SetOffset(XMFLOAT3(0.0f, 0.0f, -2.0f));  m\_pEffectShader->AddObject(pEffectObject, EFFECT\_TYPE\_BOOSTER);  ((CMissileObject\*)m\_pMissileObjects[i])->m\_MyEffect = pEffectObject;  m\_bShotable = false;  CObjectEffect \*m\_MyEffect;  **…** |

* + **미사일 오브젝트를 생성할 때 꼬리 부분에 화염을 표현하는 이펙트 오브젝트를 만들기 위해 미리 씬의 이펙트 쉐이더 클래스를 저장해두고 이펙트 오브젝트를 만들고 위치와 부가정보를 Set하고 이펙트 쉐이더에 추가하도록 만들었다. 그리고 미사일 오브젝트가 소멸될 때 이펙트 오브젝트도 같이 사라질 수 있도록 미사일 오브젝트의 m\_MyEffect 변수에도 저장해 두었다.**
  + **연사 속도를 구현하기 위해 플레이어 클래스 멤버 변수로 m\_bShotable 변수와 m\_fShotCoolTime 변수를 사용하여 Animate() 함수를 호출할 때마다 시간을 누적시켜 발사 가능, 불가능으로 만들도록 했다.**

* **코드 설명( 충돌 검사 )**
  + **적 오브젝트와 미사일 충돌을 구현하기 위해 계층 구조 형태의 메쉬들의 바운딩박스를 UpdateAABB() 함수를 통해 갱신시키고 CollisionObject() 함수와 CollisionBoundingBox() 함수를 오브젝트 클래스에 멤버 함수로 추가시켰다.**

|  |
| --- |
| void CGameObject::UpdateAABB() |
| if (m\_pMesh) m\_pMesh->m\_xmAABB.Transform(m\_xmAABB, XMLoadFloat4x4(&m\_xmf4x4World));  if (m\_pSibling) m\_pSibling->UpdateAABB();  if (m\_pChild) m\_pChild->UpdateAABB(); |

|  |
| --- |
| bool CGameObject::CollisionObject(CGameObject \*pObject) |
| if (m\_pMesh) if (pObject->CollisionBoundingBox(&m\_xmAABB)) return true;  if (m\_pSibling) return m\_pSibling->CollisionObject(pObject);  if (m\_pChild) return m\_pChild->CollisionObject(pObject);  return false; |

|  |
| --- |
| bool CGameObject::CollisionBoundingBox(BoundingBox \*pxmAABB) |
| if (m\_pMesh) if (m\_xmAABB.Intersects(\*pxmAABB)) return true;  if (m\_pSibling) return m\_pSibling->CollisionBoundingBox(pxmAABB);  if (m\_pChild) return m\_pChild->CollisionBoundingBox(pxmAABB);  return false; |

* + **각 함수에서는 오브젝트의 모든 형제와 자식들의 바운딩 박스를 갱신하고 각각이 매개변수로 넘겨받은 오브젝트의 바운딩 박스와 충돌했는지 확인할 수 있게 만들었다. 그리고 루트 노드에는 메쉬가 없을 수도 있기 때문에 항상 확인하도록 했다.**
  + **충돌 검사는 씬의 CheckCollision() 함수에서 수행하도록 했다.**

|  |
| --- |
| void CScene::CheckCollision() |
| CGameObject\*\* Enemy = ((CObjectsShader\*)m\_ppShaders[2])->m\_ppObjects; // 적 오브젝트 쉐이더의 오브젝트들  CGameObject\*\* Missile = ((CAirplanePlayer\*)m\_pPlayer)->m\_pMissileObjects; // 플레이어 미사일 오브젝트들  CEffectShader\* pEffectShader = (CEffectShader\*)m\_ppShaders[3]; // 이펙트 쉐이더  …  Enemy[i]->UpdateAABB();  Missile[j]->UpdateAABB();  if (Missile[j]->CollisionObject(Enemy[i]))  {  Missile[j]->Delete();  CEffect \*pEffect = new CEffect();  pEffect->SetPosition(Missile[j]->GetPosition());  pEffectShader->AddObject(pEffect, EFFECT\_TYPE\_EXPLOSION);  **…** |

* + **충돌 했을 때 해당 위치에 폭발 효과를 나타내기 위해 충돌 시 해당 위치에 이펙트 오브젝트를 만들었다.**
* **코드 설명 ( 조종석 UI )**
  + **1인칭 모드가 됐을 때 조종석이 보일 수 있도록 플레이어 클래스에 UI를 위한 쉐이더를 멤버 변수로 만들었다.**

|  |
| --- |
| CAirplanePlayer::CAirplanePlayer(ID3D12Device \*pd3dDevice, ID3D12GraphicsCommandList \*pd3dCommandList, ID3D12RootSignature \*pd3dGraphicsRootSignature, void \*pContext) |
| …  CUserInterface \*m\_pUserInterface = NULL;  m\_pUserInterface = new CUserInterface();  m\_pUserInterface->CreateShader(pd3dDevice, pd3dCommandList, pd3dGraphicsRootSignature);  m\_pUserInterface->Initialize(pd3dDevice, pd3dCommandList, pContext);  **…** |

* + **UI는 화면 좌표만을 이용해서 렌더링하기 때문에 그것과 관련된 파이프라인 상태를 설정하고 그래픽스 파이프라인 상태를 만들어 사용했다. UI는 기하 쉐이더를 이용해 점 위치 하나만으로 그릴 수 있게 했다.**

|  |
| --- |
| CompileShaderFromFile(L"Shaders.hlsl", "VS\_UI", "vs\_5\_1", &m\_pd3dVertexShaderBlob) |
| CompileShaderFromFile(L"Shaders.hlsl", "GS\_UI", "gs\_5\_1", &m\_pd3dGeometryShaderBlob) |
| CompileShaderFromFile(L"Shaders.hlsl", "PS\_UI", "ps\_5\_1", &m\_pd3dPixelShaderBlob) |
| pd3dInputElementDescs[0] = { "POSITION", 0, DXGI\_FORMAT\_R32G32\_FLOAT, 0, 0, … }  pd3dInputElementDescs[1] = { "SIZE", 0, DXGI\_FORMAT\_R32G32\_FLOAT, 1, 0, … } |

|  |
| --- |
| void CUserInterface::Initialize(…) |
| …  CTexture\* pTextures = new CTexture(1, RESOURCE\_TEXTURE2D, 0);  pTextures->LoadTextureFromFile(pd3dDevice, pd3dCommandList, L"./Cockpit/cockpit\_ui.dds", 0);  m\_ppTextures[1] = pTextures;  …  m\_nUIRects = 2;  m\_ppUIRects = new CUIRect\*[m\_nUIRects];  // Base UI  XMFLOAT2 xmf2Center = CalculateCenter(-1.0f, 1.0f, 1.0f, -1.0f);  XMFLOAT2 xmf2Size = CalculateSize(-1.0f, 1.0f, 1.0f, -1.0f);  m\_ppUIRects[1] = new CUIRect(pd3dDevice, pd3dCommandList, xmf2Center, xmf2Size);  **…** |

* + **CalculateCenter() 함수와 CalculateSize() 함수는 단순히 사각형 가로 세로 위치를 받아 중심점과 크기를 구하도록 만들었다.**
  + **UI는 과제2에서 빌보드를 기하 쉐이더를 이용해 렌더링하는 방식과 똑같이 만들었다.**

|  |
| --- |
| VS\_UI\_OUTPUT VS\_UI(VS\_UI\_INPUT input) |
| VS\_UI\_OUTPUT output;  output.center = input.center;  output.size = input.size;  return(output); |

* + **정점 쉐이더에서는 월드 변환 행렬, 뷰포트 변환 행렬, 투영 변환 행렬이 필요 없기 때문에 기하 쉐이더로 고스란히 넘겨주었다.**