[3D 게임 2 과제 02 보고서]

[게임공학과]
최해성(2018180044)

-과제에 대한 목표-

저의 이번 과제에 대한 목표는 다음과 같습니다.

1. 플레이어 1인칭 화면을 전환하면 헬기 내부에서 유리창을 통해 씬을 바라보게한다.

2. 특정 키를 눌렀을 때, 지형에 가려져 있든 없든 적 헬기에 윤곽선을 그려 볼 수 있게 한다.

3. 스트림 출력 단계를 사용한 파티클 효과를 구현한다.

4. 환경매핑으로 씬을 반사하는 거대한 구를 렌더링한다.

-과제 실행 환경-

과제의 실행 환경은 다음과 같습니다.

1. Visual Studio 2019 최신버전

3. Release/x64 모드

4. C++ 언어 표준: 최신 C++ 초안의 기능

-조작법-

W/A/S/D – 플레이어를 앞/왼쪽/뒤/오른쪽으로 이동시킵니다.

Ctrl – 총알을 발사합니다.

Q/E – 플레이어가 위/아래로 이동합니다.

F1 – 플레이어 1인칭 시점으로 변합니다.

F3 – 플레이어 3인칭 시점으로 변합니다.

F – 적 헬기에 윤곽선을 입혀 보이게 합니다.

P – 폭죽 위치를 플레이어의 위치로 옮깁니다.

1인칭 – 마우스 왼클릭 후 움직이면 주위를 둘러볼 수 있습니다.

3인칭 - 마우스 왼클릭 후 움직이면 플레이어가 회전합니다.

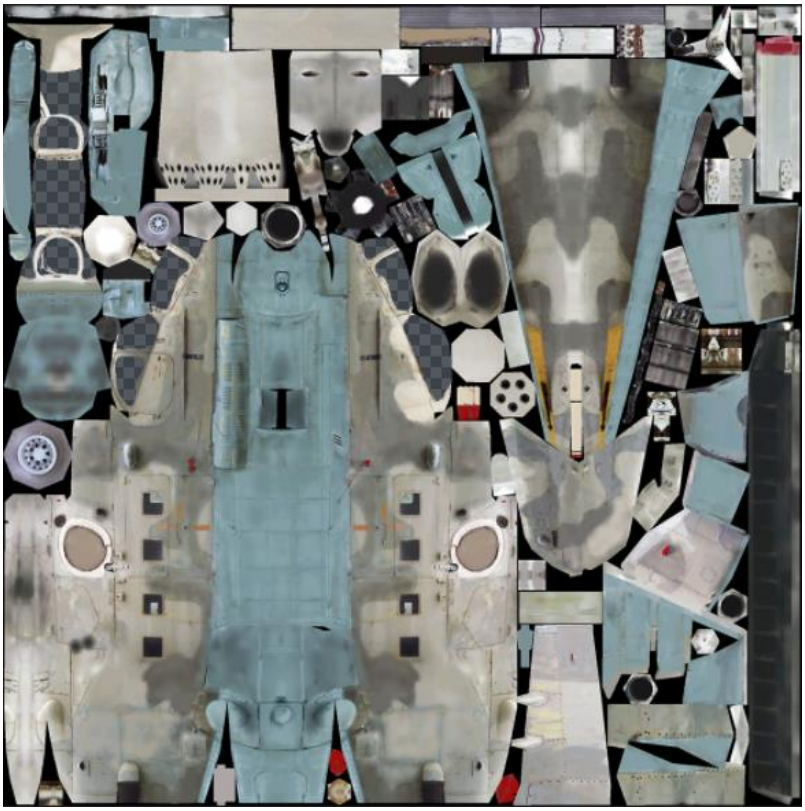
-구현방법-

1-0. 헬기 안 조종석에 카메라를 위치하게 하기

1인칭 화면을 전환했을 때, 헬기 조종석에 카메라가 위치하게 하기 위해서 플레이어의 위치에서 카메라가 얼만큼 떨어져 있는지를 나타내는 CCamera 클래스가 가지고 있는 XMFLOAT3형 m_xmf3Offset을 적절하게 설정해줍니다. 또한 카메라의 투영변환행렬에 사용하는 근평면의 거리가 너무 멀면 카메라가 헬기 모델 내부에 있어도 헬기 모델이 잘 그려지지 않는 경우가 생기기 때문에 근평면의 거리를 적절히 가깝게 해줍니다.

1-1. 헬기 모델의 유리창을 통해 씬을 바라보게 하기

헬기 모델의 조종석에 위치한 카메라가 유리창을 통해 씬을 바라보게 하기 위해서 저는 플레이어의 모델로 사용되고 있는 Mi24의 정보가 담긴 Mi24.bin 파일에서 유리창이 따로 분류가 되어있는지 확인을 해보았으나 Mi24 모델 같은 경우에는 따로 유리창 프레임이 설정되어 있지 않고, 기체의 몸을 표현하는 프레임에 텍스쳐로 유리창이 표현된 형태였습니다. 따라서 저는 몸체를 표현하는 1K_Mi24_TXTR.dds 파일을 아래 사진과 같이 비쥬얼 스튜디오 이미지 편집기를 이용해 유리창에 해당하는 부분의 텍스쳐의 색상을 적당한 알파 값을 가진 투명한 하늘색으로 표현하고 렌더링하게 한 뒤, 플레이어를 렌더링하는 파이프라인 상태 객체를 생성할 때, D3D12_RASTERIZER_DESC::CullMode를 D3D12_CULL_MODE_NONE으로 설정해 헬기 모델이 컬링되지 않게 하고 D3D12_BLEND_DESC를 알파값으로 블렌딩 될 수 있게 적절하게 설정하게 하는 방법으로 카메라가 헬기 모델 조종석에 위치한 상태로 투명한 유리창을 통해 씬을 바라보는 듯한 효과를 내주었습니다.



2-0. 여러 개의 렌더타겟 준비하기

윤곽선을 그리기 위해서 여러 개의 렌더타겟을 여러 개 사용할 필요가 있어 CGameFramework::CreateRtvAndDsvDescriptorHeaps()가 호출될 때, 기존에는 더블 버퍼링을 위한 2개의 렌더타겟 서술자 힙을 만들었지만 이번 코드에서는 6개를 추가해 총 8개의 렌더 타겟 서술자 힙을 생성하고 생성된 메모리 블록을 m_pd3dRtvDescriptorHeap라는 이름을 가진 ComPtr<ID3D12DescriptorHeap>형 변수로 가르킵니다. 그 다음 CGameFramework::BuildObjects()에서 윤곽선을 그리는 CLaplacianEdgeShader 클래스를 생성해 CreateShader() 함수를 호출함으로써 윤곽선을 그리는 파이프라인 상태 객체를 생성합니다. 그리고 만들어 놓은 렌더 타겟 서술자 힙에서 3번째 CPU 주소와 각 6개의 렌더타겟의 포맷들, 화면의 해상도 등의 인자들로 CLaplacianEdgeShader::CreateResourcesAndViews() 함수를 호출합니다. CreateResourcesAndViews() 함수에서는 6개의 텍스쳐를 저장할 수 있는 std::unique_ptr<CTexture> 형 변수 m_pTexture를 선언해 6개의 화면 크기의 텍스쳐를 생성해 서술자 힙을 생성하고 SRV를 만들어줍니다. 이때, 6개의 텍스쳐는 D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET를 Flag로 해서 생성했으며 초기 상태는 D3D12_RESOURCE_STATE_COMMON입니다. 이 SRV는 이후 윤곽선 그릴 때, RootSignature로 셰이더에 연결되어 사용됩니다. 이후 만들어진 6개의 텍스쳐 리소스로 CreateRenderTargetView를 호출해 6개의 렌더타겟 뷰를 생성하는 과정으로 6개의 추가 렌더타겟을 생성해주었습니다.

2-1. 여러 개의 렌더타겟에 씬을 구성하기 위한 렌더링하기

여러 개의 렌더타겟을 사용한 씬 렌더를 하기 위해서 그래픽스 파이프라인 상태 객체를 생성할 때, D3D12_GRAPHICS_PIPELINE_STATE_DESC::NumRenderTargets을 7로 해주고

D3D12_GRAPHICS_PIPELINE_STATE_DESC::RTVFormats[]에 렌더타겟의 포맷들을 설정해주었고 CGameFramework::FrameAdvance() 함수가 호출되어 렌더링 과정이 진행되기 전에 CLaplacianEdgeShader::OnPrepareRenderTarget() 함수를 호출해주어 현재 렌더링 될 렌더타겟의 색상을 검정으로 초기화 해주고 CLaplacianEdgeShader에서 생성한 6개의 추가 렌더타겟을 리소스 베리어를 이용해 D3D12_RESOURCE_STATE_COMMON 상태에서

D3D12_RESOURCE_STATE_RENDER_TARGET로 전환시켜주고 파란색으로 렌더타겟의 색을 초기화 시켜준뒤 렌더링 될 렌더타겟 + CLaplacianEdgeShader의 렌더타겟 6개로 OMSetRenderTargets() 함수를 호출해주어 렌더타겟을 연결, 설정해줍니다. 이후 CScene::Render()를 호출하면 씬에 생성된 객체들을 렌더링하는데 이때, 적 객체들을 제외한 윤곽선이 그려지지 않을 객체들은 픽셀 쉐이더에서 스왑체인 버퍼, 색상을 표시하는 버퍼인 SV_TARGET0, SV_TARGET1에 이전과 똑같이 그려주고 적 객체들을 렌더링하는 픽셀 쉐이더의 경우 SV_TARGET0, SV_TARGET1뿐만 아니라 객체의 노말 정보를 표시하는 SV_TARGET2, 카메라 좌표계에서의 객체의 노말 정보를 표시하는 SV_TARGET6에 해당 정보들을 써줍니다. 렌더링 과정이 끝나고 나서는 CLaplacianEdgeShader::OnPostRenderTarget() 함수를 호출해 사용한 6개의 렌더타겟(텍스쳐)들을 D3D12_RESOURCE_STATE_RENDER_TARGET에서 D3D12_RESOURCE_STATE_COMMON로 리소스 베리어를 이용해 상태를 전환시켜줍니다.

2-2. 평범한 Scene or 윤곽선 Scene 그리기

CGameFramework 클래스에 m_nDrawOptions이라는 int형 변수는 초기에는 DRAW_SCENE_EDGE라는 'F'(70)의 값을 가지고 있고 사용자가 F를 누르게 되면 ChangeShowEdge()라는 함수가 호출되어 m_nDrawOption이 DRAW_SCENE_EDGE면 DRAW_SCENE_COLOR라는 'S'(83) 의 값을 가지게 되고 DRAW_SCENE_COLOR라면 DRAW_SCENE_EDGE로 전환됩니다. 이 m_nDrawOption은 렌더링하는 과정에서 CGameFramework::UpdateShaderVariables() 함수가 호출되면 CBV의 형태로 루트시그니쳐로 연결되어 쉐이더로 전달됩니다. 2-1의 방법으로 렌더링을 마치면 CLaplacianShader의 m_pTexture에는 씬을 렌더링하기위한 정보들이 모두 담겨있기 때문에 씬을 렌더링한 뒤에 . CLaplacianShader::Render() 함수를 호출하여 씬을 렌더링하기위한 정보가 담긴 m_pTexture를 루트시그니쳐로 쉐이더로 전달해주고 CLaplacianShader는 화면의 픽셀좌표를 정점쉐이더로부터 전달받아 해당 좌표로 m_nDrawOption의 값이 'S'이면 전체 씬의 색상 정보가 담긴 0번 텍스쳐에서 샘플링해 최종 색상으로 렌더링하고 'F'이면 LaplacianEdge() 함수를 통해 윤곽선을 가진 적 객체 + 씬을 렌더링하게 됩니다. LaplacianEdge() 함수는 1번 텍스쳐에 담겨있는 적 객체들의 노말 정보, 6번 텍스쳐에 담겨있는 적 객체의 카메라 좌표계에서의 노말정보를 가지고 적 객체에는 윤곽선이 있는 Scene을 렌더링합니다.

지형에서 가려진 적 객체들도 윤곽선을 그리기 위해서 적 객체를 그리는 그래픽스 파이프라인 상태 객체의 경우 깊이 검사를 해서 그리는 파이프라인 상태 객체와 깊이 검사를 안하면서 그리는 파이프 라인 상태 객체 두 개를 생성해 m_nDrawOption == DRAW_SCENE_EDGE를 검사해 윤곽선을 그려야 한다면 깊이 검사를 안하면서 그리는 파이프라인 상태 객체를 Set하고 윤곽선을 그리지 않아도 된다면 깊이 검사를 해서 그리는 파이프라인 상태 객체로 Set하게 해서 지형에 가려져 있더라도 윤곽선을 그리려고 하면 적 객체가 렌더링이 되게 했습니다.

3-0. 스트림 출력 단계를 사용하기 위한 오브젝트 & 버퍼 준비

파티클 효과를 표현하는 CParticleObject 객체는 CScene의 BuildObject()에서 생성되어 m_ppParticleObject라는 이름의 std::vector<std::unique_ptr<CParticleObject>> 형 변수에 저장됩니다. CParticleObject는 위치, 속도, 수명을 전달받아 생성자가 호출되면 위치, 속도, 수명 변수를 가지고 CParticleMesh라고하는 클래스를 생성하게 되는 데 CParticleMesh는 폭죽이 시작되는 최초 점의 위치에 해당하는 위치, 속도, 수명을 표현하는 정점 버퍼를 만들고 스트림 출력에 사용할 스트림 출력 버퍼 m_pd3dStreamOutputBuffer, 그리기 위한 정점들을 담을 m_pd3dDrawBuffer를 최대 파티클 개수를 담을 수 있을 만큼의 크기로 생성하며 스트림 출력 버퍼들을 생성해줍니다. 그리고 폭죽 파티클을 표현할 텍스쳐와 1024개의 x, y, z, w를 랜덤하게 -1.0f ~ 1.0f를 가지는 pxmf4RandomValues라는 std::vector<XMFLOAT4>형 변수를 만들어 1024개의 랜덤한 값을 가지는 텍스쳐, 256개의 랜덤한 값을 가지는 텍스쳐를 하나씩 생성해 m_pRandowmValueTexture, m_pRandowmValueOnSphereTexture라는 이름의 변수로 저장합니다. 파티클 효과에 사용할 CParticleShader를 생성해 3개의 텍스쳐를 저장

할 서술자 힙을 만들어 3개의 텍스쳐를 SRV로 연결해주고 CParticleObject가 CParticelShader를 사용하도록 연결해주어 스트림 출력 단계를 사용하기 위한 오브젝트 & 버퍼를 준비합니다.

3-1. 파티클 렌더링

파티클 객체는 두 번의 파이프라인에 거쳐 렌더링 되는데, CGameFramework에서 FrameAdvance()가 불릴 때, CScene의 RenderParticle을 통해 렌더링이 됩니다. RenderParticle에서는 CScene의 BuildObject()에서 생성한 CParticleObject 클래스의 m_ppParticleObject의 Render를 호출합니다 CParticleObject의 Render()에서는 연결되어있는 CParticleShader로 생성된 첫번째 파이프라인 상태 객체를 설정해주고 서술자 힙을 설정해준 뒤 파티클을 렌더링하기 위한 텍스쳐, 랜덤 텍스쳐를 루트시그니쳐로 연결해줍니다. 첫 번째 파이프라인은 스트림 출력 버퍼를 설정하고 렌더링을 해 조건이 만족하면 새로운 파티클들을 생성해 스트림 출력 버퍼에 출력하는 파이프라인을 진행하고 이어서 두번째 파이프라인으로 스트림 출력 버퍼에 써진 파티클들을 실제로 그리기 위한 DrawBuffer로 옮겨 픽셀 쉐이더를 거쳐서 렌더링을 하는 과정을 가져 실제로 씬에 렌더링이 되게 됩니다. 사용자가 'P'를 누르면 플레이어 오브젝트의 위치로 파티클 객체의 위치를 설정해 주고 월드 변환 행렬을 렌더링할 때, 루트시그니쳐로 넘겨주어 정점 쉐이더에서 위치를 옮겨주는 방식으로 사용자가 'P'를 눌렀을 때, 폭죽 파티클의 위치를 변화시켜주었습니다.

4-0. 환경매핑을 할 거대한 구를 준비하기

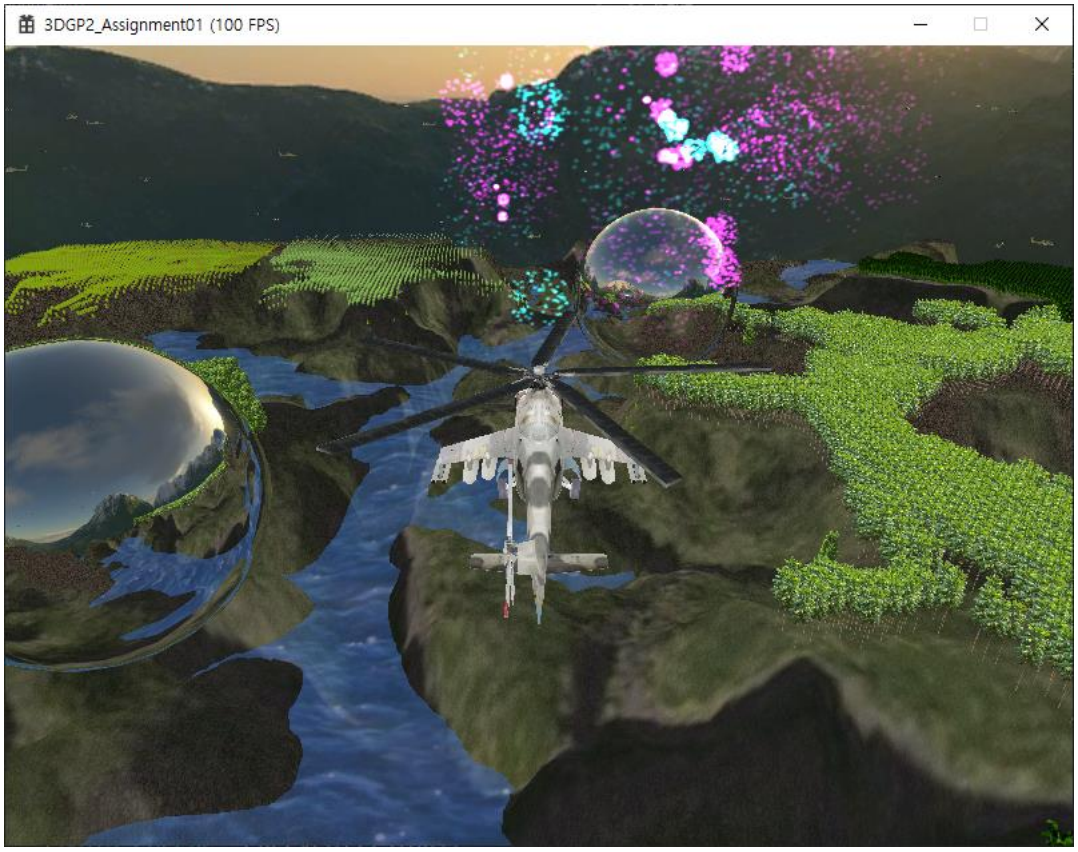
환경 매핑을 하기 위한 쉐이더는 CDynamicCubeMappingShader 클래스로 설정됩니다. CDynamicCubeMappingShader는 CScene의 BuildObjects()가 호출될 때, 싱글톤 객체로 호출되어 CreateShader(), BuildObjects() 함수가 호출이 되는데 CreateShader()에서는 다른 쉐이더 클래스들과 같이 파이프라인 상태 객체를 생성하며 BuildObjects()에서는 렌더링에 사용하기 위한 m_hFenceEvent라는 이름을 가진 펜스 객체와 m_pd3dCommandAllocator라는 이름을 가진 명령 할당자 객체, m_pd3dCommandList라는 이름을 가진 커맨드 리스트 객체를 만들고 2개의 반사하는 거대한 구를 생성하기 위한 2개의 깊이-스텐실 뷰와 환경 매핑을 위한 12개의 렌더타겟 뷰를 만들어 그 주소를 각각 m_pd3dDsvDescriptorHeap이라는 이름과 m_pd3dRtvDescriptorHeap이라는 이름을 가진 ComPtr<ID3D12DescriptorHeap>형 변수에 저장합니다. 이후 2개의 CDynamicCubeMappingObject라는 이름을 가진 클래스 객체 2개를 unique_ptr로 생성해 m_ppObjects[]라는 2개의 std::unique_ptr<CGameObject>를 담을 std::vector에 저장하게되는데 CDynamicCubeMappingObject는 이전에 생성한 큐브 매핑할 크기를 나타내는 nCubeMapSize와 pd3dDsvDescriptorHeap와 pd3dRtvDescriptorHeap를 인자로 전달 받아 객체를 기준으로 상하좌우 앞뒤에 비치는 씬을 담기 위한 6개의 카메라를 가로 세로가 넘어온 nCubeMapSize의 값 만큼을 Viewport로 가지게 생성하고 이 6개의 카메라에 담은 씬의 모습을 저장할 큐브텍스쳐를 생성한 뒤 넘어온 큐브텍스쳐의 6개의 면에 렌더타겟 뷰를 생성해 연결합니다. 거대한 구의 위치는

CreateShader()를 호출할 때, 터레인의 정보를 전달해주어 터레인의 중앙에서 x, z 좌표가 +150, +150인 곳에 하나, -150, -150인 곳에 하나를 배치해주었습니다.

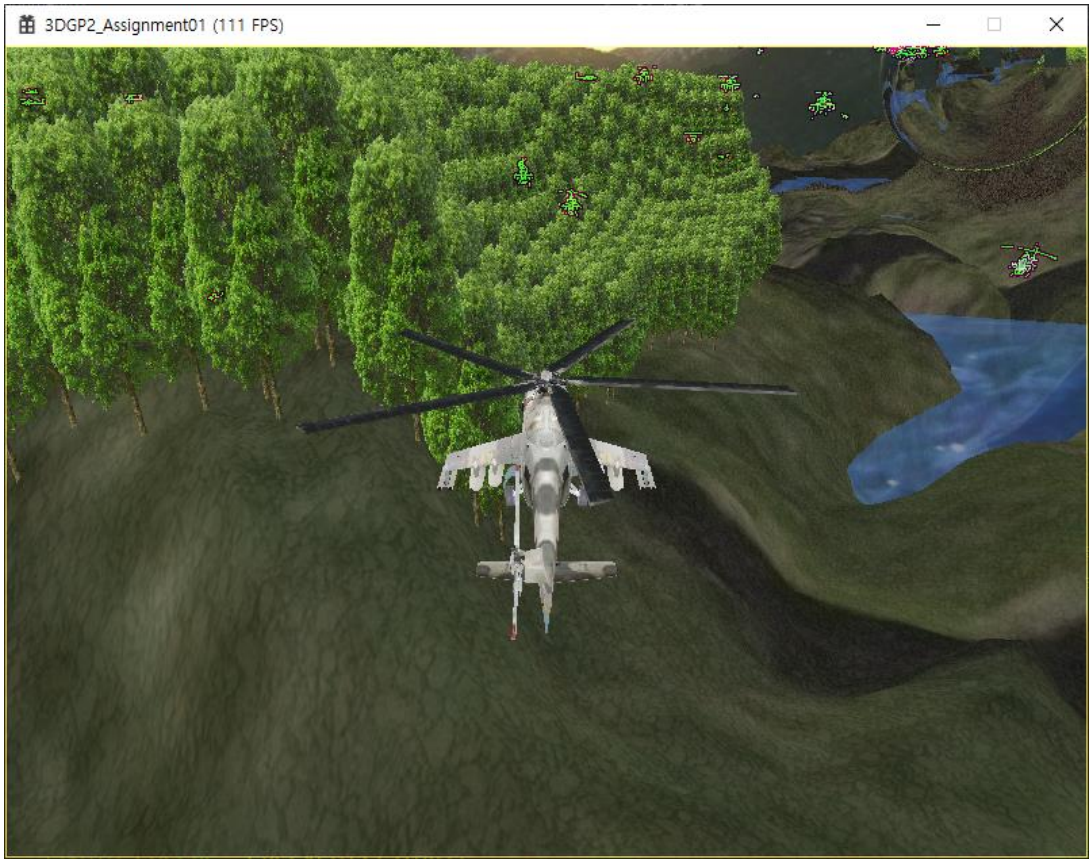
4-1. 씬을 반사하는 거대한 구를 렌더링하기

씬을 반사하는 거대한 구를 렌더링하기 위해서 우선 구체를 기준으로 상하좌우앞뒤의 씬을 렌더타겟에 렌더링해 그 정보를 가지고 있을 필요가 있는 데, 이를 CGameFramework의 FrameAdvance()가 호출되어 렌더링이 되기 전에 CScene의 OnPrepareRender 함수가 호출되면 CDynamicCubeMappingShader를 싱글톤 객체로 호출해 OnPreRender() 함수를 호출해서 진행하게 됩니다. CDynamicCubeMappingShader의 OnPreRender()에서는 앞서 말한 것을 하기 위해 명령할당자와 커맨드 리스트를 초기화 해주고 두 개의 CDynamicCubeMappingObject의 OnPreRender()를 호출합니다. CDynamicCubeMappingObject의 OnPreRender()에서는 루트시그니쳐를 Set 해주고 리소스 베리어를 이용해 큐브매핑할 텍스쳐의 상태를 D3D12_RESOURCE_STATE_GENERIC_READ에서 D3D12_RESOURCE_STATE_RENDER_TARGET으로 바꿔준 뒤, 생성했던 6개의 카메라로 각각 뷰 행렬을 설정해 루트시그니쳐로 연결하고 씬을 렌더링해서 렌더타겟에 각 카메라로 봤을 때의 씬의 모습이 담겨 텍스쳐에 저장되게 합니다. 렌더링을 마치면 다시 리소스 베리어를 이용해 원래대로 D3D12_RESOURCE_STATE_GENERIC_READ 상태로 돌아오게 해줍니다. 이 작업은 실제 씬을 반사하는 거대한 구가 렌더링이 되기 이전에 행해져야하는 작업이기 때문에 CDynamicCubeMappingShader의 CreateShader에서 생성한 m_pd3dFence라는 이름의 펜스 객체를 이용해 Execute하고 작업이 완료되기를 기다리게 해서 작업이 완료되어 각 2개의 구체를 기준으로 각각 6개의 씬이 담기는 작업이 완료되었다면, CScene의 Render() 함수가 호출될 때, CDynamicCubeMappingShader의 Render()가 호출되어 거대한 구가 렌더링되게 됩니다. 이때 정점쉐이더는 다른 오브젝트를 그릴 때와 별반 차이가 없이 위치를 객체의 월드변환행렬에 따라 움직여주고 노말 값을 바꿔주고 하는 통상적인 작업을 하지만 픽셀쉐이더에서는 구체에 반사되어지는 씬을 그려야하기 때문에 카메라에서 구체를 바라보는 벡터를 구한뒤 이를 구체의 노말 값으로 반사벡터를 구해 그 반사벡터를 이용해 텍스쳐를 샘플링을 하는 방법으로 카메라를 기준으로 구체에 반사되어 보여지는 부분을 그리게 해줬습니다.

실행결과




3DGP2_Assignment01 (90 FPS)


3DGP2_Assignment01 (111 FPS)