**3D 게임프로그래밍 1**

**과제 02 설명 문서**

김태순 2021182009

# • 프로젝트의 구조

프로젝트의 전체적인 구조는 과제01의 구조와 동일하고 그래픽스 파이프라인의 구조는 따라하기와 동일하지만 인스턴싱은 사용하지 않고 제작하였다. 프레임워크에는 현재 씬을 관리하는 씬 매니저를 저장 및 관리하며 씬 매니저가 현재 씬의 업데이트와 렌더링, 입력처리를 호출하는 형식으로 구성 되어있다.

CScene과 CShader를 상속받아 각 레벨별 Scene과 Shader를 따로 구현하였고 virtual함수의 특성을 최대한 이용하기 위해 다른 오브젝트들도 CGameObject를 상속받아 Shader에서 CGameObject 포인터의 배열로 관리할 수 있도록 제작하였다.

# • 조작법

**ESC** – 각 씬에서 CMenuScene으로 전환, CMenuScene에서는 CStartScene으로 CStartScene에서는 프로그램 종료

**방향키** – 캐릭터 이동, CStartScene/CMenuScene에서는 이동하지 않음. CLevel2(탱크)에서는 좌우로 캐릭터 회전 앞뒤로 캐릭터 이동

**마우스 좌클릭** – 오브젝트 선택(피킹), 선택한 오브젝트 폭발 시도

**마우스 클릭 후 드래그** – 플레이어 및 카메라 회전, CStartScene/CMenuScene에서는 회전하지 않음. CLevel2에서는 탱크 상단 회전 및 발사 방향 회전

CLevel1: **N/n** – Clevel2로 씬 전환

CLevel2: **W** – “YOU WIN!” 텍스트 객체 보여주기, **S** – 실드 토글, **A** – 자동 발사 토글

# • 렌더링 과정

렌더링은 우선 씬에서 루트 시그너처를 생성, 루트 시그너처는 0번 슬롯에 오브젝트의 월드변환 행렬을 넘겨주기 위한 16개의 상수를 지정하고 1번 슬롯에는 카메라에 저장된 뷰변환 행렬과 투영변환 행렬을 넘겨주기 위한 32개의 상수를 지정한다. 이후 셰이더에서 루트 시그너처를 받아와 PSO를 생성, 이때 래스터라이저, 깊이 – 스텐실, Blend, InputLayout의 상태들도 생성 및 설정한다. 여기서 InputLayout은 위치와 색상 정보를 각각 RGB, RGBA 32비트 포맷으로 “POSITION”, “COLOR”라는 시맨틱으로 넘긴다. 프리미티브 토폴로지는 세개의 정점으로 하나의 삼각형을 그리는 D3D12\_PRIMITIVE\_TOPOLOGY\_TYPE\_TRIANGLE로 설정하고 하나의 윈도우에 하나의 화면을 그리기 때문에 렌더 타겟은 1로 설정하고 렌더타겟 뷰 포맷은 RGBA각 8비트씩 총 32비트를 사용하고 깊이-스텐실 포맷은 깊이 24비트 스텐실 8비트의 32비트를 사용한다. 이렇게 생성한 PSO는 셰이더에 저장된다. 이후에 오브젝트를 생성하고 Animate를 거치며 오브젝트의 월드변환 행렬이 변화하고 이후 렌더링 과정에서 사용한다. Render는 우선 CGameFramework에서 FrameAdvance를 통해 커맨드 리스트를 리셋하고 ResourceBarrier를 통해 렌더 타겟의 상태를 PRESENT에서 RENDER\_TARGET으로 바꾼다. 이후 렌더 타겟뷰를 지정한 색상으로 Clear하고 깊이-스텐실 뷰도 Clear한다. 이후 OMSetRenderTargets를 통해 Clear한 렌더 타켓뷰와 깊이-스탠실 뷰를 1번 렌더 타겟에 Set하고 Scene의 Render를 호출한다. Scene에서는 우선 카메라를 통해 뷰포트와 시저 렉트를 설정한다. 이후 씬에 저장되어 있는 루트 시그너처로 Set하고 카메라의 뷰변환 행렬과 투영변환 행렬을 전치행렬로 변환한 다음 SetGraphicsRoot32BitConstants를 통해 파이프라인에 넘겨준다. 그 이후 셰이더의 Redner를 호출하는데 먼저 셰이더에 저장되어 있는 PSO를 Set하고 셰이더가 관리하는 오브젝트들의 Render를 호출한다. 오브젝트에서는 월드변환 행렬을 전치행렬로 변환하고 SetGraphicsRoot32BitConstants를 통해 파이프라인에 넘겨준다. 이후 Mesh의 Render를 호출하고 Mesh에서는 입력조립기에 정점 버퍼 뷰와 프리미티브 토폴로지ㅡ 인덱스 버퍼를 Set하고 Draw를 호출한다. 오브젝트의 Render가 끝나면 Player의 Render를 호출하고 동일한 과정을 거쳐 Draw한다. 이후 다시 ResourceBarrier를 통해 렌더 타겟의 상태를 RENDER\_TARGET에서 PRESENT로 변경하고 커맨드 리스트를 Close하고 ExecuteCommandLists를 통해 GPU에 전달한다. 이후 WaitForGpuComplete에서 Fence오브젝트를 이용해 GPU가 모든 명령어를 실행할 때까지 대기하고 이후에 SwapChain에 Present를 명령한다. 이후 MoveToNextFrame을 통해 현재 백 버퍼의 Fence값을 증가시키고 동일한 방법으로 모든 명령어를 실행할 때까지 대기하고 이 과정이 모두 끝나면 FrameRate를 측정하여 출력한다.

# • 셰이더 코드

셰이더 코드에서는 루트 시그너처를 통해 넘어온 오브젝트의 월드변환 행렬을 b0레지스터에 gmtxWorld로 저장하고 카메라의 뷰 변환 행렬과 투영 변환 행렬을 b1레지스터에 각각 gmtxView와 gmtxProjection으로 저장한다. InputLayout으로 설정한 Mesh의 위치정보와 색상정보가 버텍스 셰이더에 입력으로 들어오고 버텍스 셰이더에서는 위치 정보에 루트 시그너처로 넘어온 월드변환행렬과 뷰, 투영 변환 행렬을 mul을 통해 곱하여 최종 위치를 구한다. 픽셀 셰이더에서는 버텍스 셰이더에서 넘어온 색상 정보를 그대로 출력정보로 리턴한다.

# • 씬 전환

씬 전환은 GameFramework의 AnimateObjects에서 씬 매니저를 통해 씬의 종료 여부를 확인하고 씬에서는 m\_bSceneFinished를 통해 씬의 종료 여부를 반환한다. 씬이 종료되었으면 m\_nNextSceneID를 통해 변경할 씬의 종류를 알아내고 GameFramework의 ChangeScene을 호출한다. 해당 함수에서는 커맨드 리스트를 리셋 한 이후에 씬 매니저의 ChangeScene을 호출, 현재 씬을 Release하고 씬 ID에 맞는 새로운 씬을 생성하고 BuildObjects를 호출하여 루트 시그너처와 Shader, Player를 생성한다. Shader를 생성할 때 각 Scene에 맞는 Shader가 생성되고 Shader의 BuildObjects를 호출하여 해당 씬에 맞는 오브젝트들을 생성한다. 이때 각 Mesh에서는 정점 버퍼와 인덱스 버퍼를 업로드 힙을 통해 GPU에 복사한다. 이후 커맨드 리스트를 Close하고 ExecuteCommandLists를 통해 해당 명령어들을 GPU에 넘긴다. WaitForGpuComplete를 통해 GPU의 모든 명령이 실행될 때까지 대기하고 모든 명령이 실행되었다면 정점 정보와 인덱스 정보가 GPU에 복사된 것이므로 업로드 힙을 삭제한다.

# • 실행결과

텍스트, 스크린샷, 폰트, 로고이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.스크린샷, 컴퓨터, 운영 체제이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.