3D게임프로그래밍1 과제3 설명문서

2025.06.20

게임공학과

2022184015

김해님

1. 접근방식

과제2로 제출했던 코드와 따라하기 15, 그리고 LabProject07-9-1의 구조와 구현 방식을 바탕으로, 각 단계별로 기능을 분석하고 필요한 부분을 선택적으로 확장 및 수정하는 방식으로 진행하였다. 처음에는 LabProject07-9-1의 프레임워크에 지형을 적용하려 했으나, 메시 클래스 구조가 따라하기 예제와 달라 호환이 되지 않았고, 이에 따라 따라하기 15의 코드에 계층구조를 추가하기로 계획을 변경했다. 모델 로드와 계층구조 구현은 LabProject07-9-1의 파일 파싱 및 메시 생성 코드를 바탕으로, 따라하기 15의 Mesh 코드에 맞게 메쉬 생성 부분을 일부 수정하였다.

폭발 이펙트는 이전 과제에서 구현한 CExplosiveObject와 CInstancingShader를 그대로 가져와 적용하려고 했으나, 인스턴싱 쉐이더를 사용하는 과정에서 문제가 있었다. 루트시그니처가 맞지 않는 오류였고 수정해서 해결했다.

충돌 처리는 처음에는 모든 오브젝트의 바운딩 박스를 충돌 검사하는 방법으로 접근했으나, 프레임 드랍이 너무 심해서 각 프레임('BODY', 'TURRET', 'cannon') 단위로 충돌 검사를 수행하는 방식으로 변경했다. 이를 위해 CollectBoundingBoxes 함수를 구현하여 각 게임 오브젝트의 주요 프레임 바운딩 박스를 벡터에 담아 반환하도록 했고, 이를 충돌 처리 함수에서 이용하도록 했다.

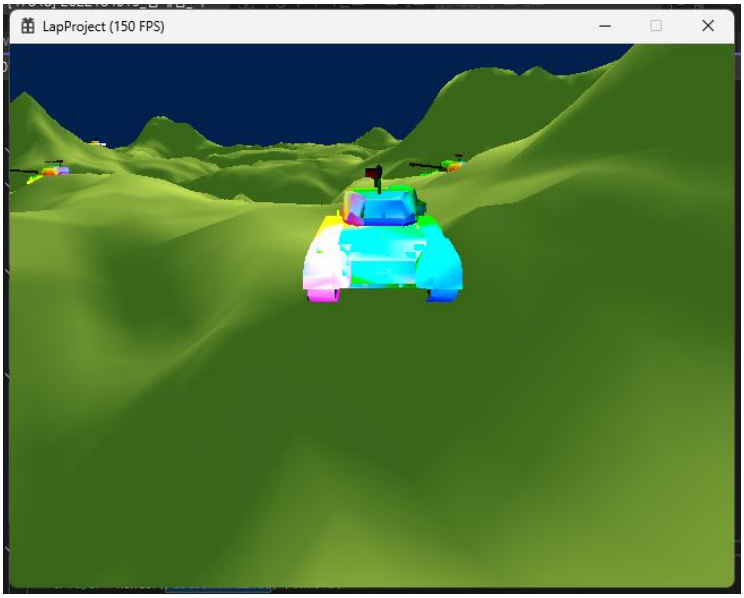
적 탱크의 AI는 플레이어와의 거리를 계산해 200.0f 이내로 접근하면 플레이어 쪽을 향해 이동 방향을 갱신하고, 그렇지 않을 때는 기존 방향을 유지하며, 맵 경계를 벗어나지 못하도록 구현했다. 플레이어는 CTerrainPlayer 클래스를 수정해, 생성자에서 위치와 카메라, 지형, 메시, 총알을 초기화하고, OnInitialize에서 머리의 초기 각도를 지정했다. 머리와 포신의 독립적인 회전을 위해 Rotate 함수를 재정의하고, FireBullet 함수에서는 목표 오브젝트를 향한 yaw와 pitch를 계산해 머리와 포신을 조준한 뒤, 비활성화된 총알을 찾아 발사 위치와 방향을 지정해주는 방식으로 무기 시스템을 완성했다.

기존에는 씬 여러 개를 만들어야 했기에 따라하기의 프레임워크를 수정해서 프레임워크 위에 씬 클래스를 만들어 상속받아 여러 씬을 구현했었다. 하지만 이번에는 한 개의 씬만 구현하면 돼서 프레임워크와 씬 구조를 그대로 사용했다.

std::vector 컨테이너를 사용하기 위해 iostream과 vector 헤더를 추가했다.

2. 구현

1) 지형

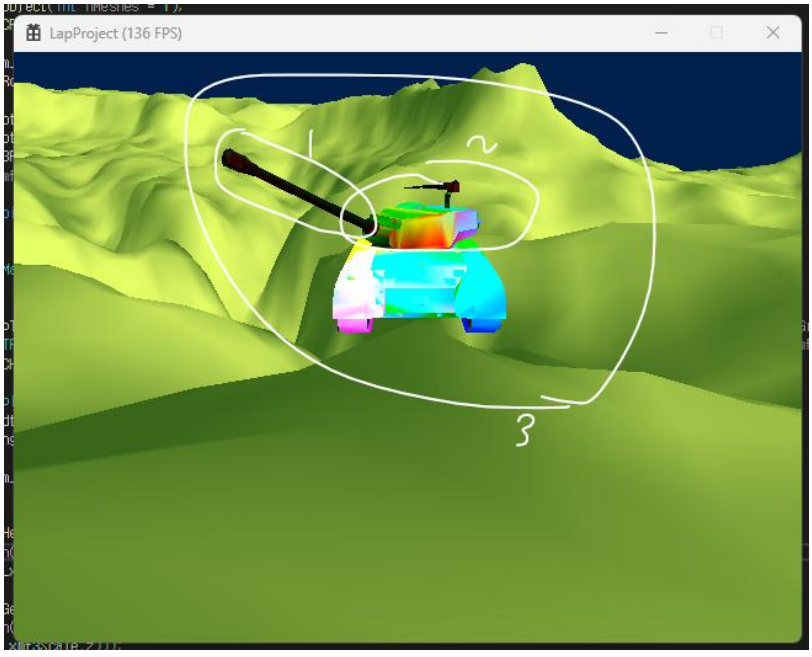


➔ 지형 위의 탱크 플레이어 모습

첫번째로 지형을 먼저 구현했다.

처음에는 LabProject07-9-1의 코드를 바탕으로 그 프레임워크에 지형을 띄우려고 계획했다. 하지만 코드를 뜯어보고 Mesh 클래스의 구조 자체가 따라하기의 코드와 다르게 구현되어 있어서 포기하고 따라하기 15를 수행한 코드에 계층구조를 추가하기로 계획을 수정했다. 지형 구현은 기존 코드를 그대로 이용했고 CMesh**로 되어있었던 메쉬만 자식으로 변경했다.

2) 모델 로드 및 계층구조 구현



➔ 계층구조를 통해 따로 회전할 수 있도록 구현했다.

모델 로드는 LabProject07-9-1의 코드에서 Material을 읽어오는 부분을 삭제하고 그대로 가져왔다. LoadMeshInfoFromFile, LoadFrameHierarchyFromFile, LoadGeometryFromFile함수를 이용해 바이너리 파일을 파싱하고 메쉬를 생성한다.

CMeshFromFile 클래스와 CMeshLoadInfo 클래스도 그대로 가져왔다.

CMeshFromFile 클래스의 생성자는 외부 파일로부터 전달받은 CMeshLoadInfo를 기반으로 정점 버퍼와 인덱스 버퍼를 생성하고 초기화한다. 먼저, 메쉬의 정점 개수를 pMeshInfo에서 받아 m_nVertices에 저장하고, 정점 하나의 크기를 XMFLOAT3 크기로 설정한다. 그 다음, CDiffusedVertex 배열을 할당하여 각 정점의 위치를 pMeshInfo의 m_pxmf3Positions 배열에서 복사하고, 기본 색상(RANDOM_COLOR)을 할당한다. 다음으로 서브메쉬(부분 메쉬)의 개수를 pMeshInfo에서 받아 m_nSubMeshes에 저장하고, 각 서브메쉬에 대한 인덱스 개수와 인덱스 배열을 복사하여 내부 멤버 변수에 저장한다. 이후 CreateBufferResource 함수를 호출해 GPU에 정점 버퍼를 생성하고, 정점 버퍼 뷰를 설정한다. 서브메쉬가 존재할 경우, 각 서브메쉬에 대해 별도의 인덱스 버퍼와 인덱스 버퍼 뷰를 생성하여 부분 렌더링이 가능하도록 준비한다. 마지막으로, 모든 정점의 위치를 순회하며 최소 및 최대 좌표를 계산해 메쉬의 바운딩 박스를 만든다.

계층구조는 LabProject07-9-1의 코드를 그대로 이용했다. Render, Animate등 모든 자식 오브젝트에 적용해야하는 함수는 재귀호출을 통해 작성했다.

3) 인스턴싱 쉐이더를 이용한 폭발 이펙트

➔ 적 탱크를 죽였을 때 나오는 폭발 이펙트

저번 과제에서 구현했던 CExplosiveObject 클래스를 그대로 가져왔다.

CExplosiveObject 클래스는 폭발 이펙트를 시각적으로 구현하기 위한 클래스다. 평상시에는 일반 게임 오브젝트처럼 동작하지만(주로 메쉬를 가진 다른 클래스의 부모 클래스로 사용한다), 폭발이 시작되면 오브젝트가 여러 개의 작은 조각으로 분해되어 사방으로 흩어지는 애니메이션을 보여준다. 이때 각 조각은 무작위 방향으로 이동하며, 일정 시간 동안 폭발 효과가 지속된다. 폭발이 끝나면 오브젝트의 bActive를 false로 바꾼다.

setExplosionMesh 함수에서 조각이 생성되고, 각 조각은 구 표면 위의 임의 방향으로 움직이도록 초기화된다. 이 조각들은 동일한 메쉬를 공유하며, 인스턴싱 셰이더를 통해 한 번의 드로우 콜로 효율적으로 렌더링된다. 폭발 중에는 Animate 함수가 매 프레임마다 호출되어 각 조각의 위치와 회전이 갱신되고, 경과 시간이 누적된다. 폭발 애니메이션이 지정된 시간(m_fDuration)만큼 진행되면 폭발 상태가 종료되고, 오브젝트와 조각의 상태가 초기화된다.

CInstancingShader 클래스는 GPU 인스턴싱 기법을 활용해, 여러 게임 오브젝트의 월드 변환 행렬과 색상 정보를 업로드 버퍼에 저장하고, 이를 셰이더에 전달해 한 번의 드로우 콜로 다수의 오브젝트를 효율적으로 렌더링한다.

그대로 가져와서 적용하면 당연히 제대로 동작할 거라고 생각했는데 제대로 실행되지 않아서 문제를 한참 찾았다. PSO가 제대로 생성되지 않는 문제였다. HIsl 코드가 문제인지, 복사하다가 잘못 복사한 코드가 있는 건지 한참 찾아 헤메다가 비주얼 스튜디오의 출력창을 봤는데,

D3D12 ERROR: ID3D12Device::CreateGraphicsPipelineState: Root Signature doesn't match Vertex Shader: Shader SRV descriptor range (BaseShaderRegister=0, NumDescriptors=1, RegisterSpace=0) is not fully bound in root signature

[STATE_CREATION ERROR #688: CREATEGRAPHICSPIPELINESTATE_VS_ROOT_SIGNATURE_MISMATCH]

라는 에러가 출력되어 있었다. 루트시그니처가 제대로 호환되지 않아서 PSO가 제대로 만들어지지 않던 것이었다. 루트시그니처를 변경했더니 제대로 실행되었다.

4) 충돌 처리

- 충돌 구현 방식

처음엔 모델을 읽어오는 코드가 어떤 방식으로 동작하는지 완벽하게 이해하지 못해서 전체 정점들에 대해서 하나의 커다란 바운딩 박스가 만들어지는 줄 알았다. 그래서 모든 자식 오브젝트들이 다 같은 바운딩 박스를 가지고 있는 줄 알았다. 디버그 모드로 하나

의 게임 오브젝트가 가진 모든 바운딩 박스의 값들을 확인하다가 하나의 부분 게임 오브젝트마다 다른 바운딩 박스를 가지고 있다는 사실을 알게 되었다.

이를 바탕으로 게임 오브젝트의 모든 바운딩 박스에 대해서 충돌검사를 진행하기로 계획했다. Player의 경우 오브젝트가 가지고 있는 모든 자식/형제 오브젝트들을 순회하면서 바운딩 박스를 std::vector에 담아서 반환하는 함수를 만들었다. 이렇게 Enemy와 Player의 충돌을 구현했더니 프레임이 거의 30 정도밖에 나오지 않았다. 뭔가 잘못됨을 느끼고 충돌 체크 로직을 변경해야겠다고 생각했다.

가장 간단한 충돌 방식은 게임 오브젝트의 전체 정점들을 이용해서 하나의 커다란 바운딩 박스를 만드는 것이라고 생각했다. 하지만 전체 정점을 다 이용해서 바운딩 박스 하나를 만드는 방법을 찾아내지 못했고(모든 정점을 한 번에 읽어서 게임 오브젝트를 생성하도록 동작하지 않는 것 같았다), Frame을 이용해서 객체의 큰 덩어리들에서만 바운딩 박스를 추출해 충돌 검사를 수행하는 방법을 채택했다.

Blender에서 FBX파일을 읽어서 어떤 오브젝트가 적당할지 확인했고, 그 결과 'BODY', 'TURRET', 'cannon' 정도의 덩어리만 충돌체크를 진행해도 괜찮을 것 같았다. 물론 정확도는 떨어지겠으나 이정도의 정확도라면 나쁘지 않다고 생각했다.

위의 결론을 바탕으로, TankObject 클래스와 TerrainPlayer 클래스에 CollectBoundingBoxes 함수를 만들었다.

```
void CollectBoundingBoxes(std::vector<BoundingOrientedBox>& boxes)
{
    boxes.push_back(m_pTurretFrame->GetBoundingBox());
    boxes.push_back(m_pCannonFrame->GetBoundingBox());
    boxes.push_back(m_pBodyFrame->GetBoundingBox());
}
```

이 함수를 통해 세개의 프레임의 바운딩 박스를 가져와 Scene에서 충돌 체크를 진행하도록 구현했다. 이렇게 수정하고 나니 Enemy-Bullet, Enemy-Player만 검사하는 상태에서 160~190프레임 정도가 나왔다.

- Bullet과 Enemy의 충돌: CheckEnemyByBulletCollisions 함수

이 함수는 플레이어가 발사한 총알이 적 오브젝트와 충돌하는지를 검사한다. 먼저 플레이어가 가진 모든 총알을 순회하면서, 각 총알의 bActive가 true일 때만 충돌 판정을 진행한다. 각 총알에 대해 바운딩 박스를 가져온 후, 게임 내의 모든 적 오브젝트에 대해 반복해서 확인한다. 적의 m_bBlowingUp가 true이거나 bActive가 false일 때는 체크를 건너뛴다. 적에 대해서 3개의 바운딩 박스(BODY, TURRET, cannon 프레임)를 가져와, 각 상자와 총알의 바운딩 박스가 교차하는지 검사한다. 만약 충돌이 감지되면 해당 적의 m_bBlowingUp를 true로 변경하고, 총알은 Reset 함수를 호출하여 bActive를 false로 변

경한다. 이 과정을 모든 총알과 모든 적 오브젝트에 대해 반복한다.

- Enemy와 Player의 충돌: CheckEnemyByPlayerCollisions 함수

이 함수는 플레이어와 적 오브젝트 간의 충돌을 검사한다. 먼저 Player의 CollectBoundingBoxes 함수를 호출하여 플레이어를 구성하는 바운딩 박스들을 playerBoxes 벡터에 저장한다. 이후 게임 내의 모든 적 오브젝트를 순회하면서, 적의 bActive가 false이거나 m_bBlowingUp이 true인 경우에는 충돌 체크를 건너뛴다. bActive가 true인 적에 대해서는 적의 CollectBoundingBoxes 함수를 호출하여 적 오브젝트를 구성하는 바운딩 박스들을 enemyBoxes 벡터에 저장한다. 그 다음, 플레이어의 모든 바운딩 박스와 적의 모든 바운딩 박스에 대해 이중 for문을 돌며, Intersects 함수를 통해 교차하는지 검사한다. 만약 교차가 감지되면, 적 오브젝트의 StartExplosion 함수를 호출하여 적을 폭발 상태로 만들고, bHit 플래그를 true로 설정한 뒤 반복문을 탈출한다. 이 과정을 모든 적 오브젝트에 대해 반복한다.

- Enemy와 Enemy의 충돌: CheckEnemyByEnemyCollisions 함수

이 함수는 적 오브젝트들끼리의 충돌을 검사한다. 먼저 모든 적 오브젝트를 순회하면서, 각 적의 m_bBlowingUp이 true이거나 bActive가 false인 경우에는 충돌 체크를 건너뛴다. bActive가 true인 적에 대해서는 CollectBoundingBoxes 함수를 호출하여 적을 구성하는 바운딩 박스들을 enemyBoxes1 벡터에 저장한다. 이후 현재 적 다음 인덱스부터 모든 적을 다시 순회한다. 마찬가지로 m_bBlowingUp이 true이거나 bActive가 false인 적은 건너뛴다. 활성화된 두 번째 적에 대해서도 CollectBoundingBoxes 함수를 호출하여 바운딩 박스들을 enemyBoxes2 벡터에 저장한다. 그 다음, 첫 번째 적의 모든 바운딩 박스와 두 번째 적의 모든 바운딩 박스에 대해 이중 for문을 돌며, Intersects 함수를 통해 교차하는지 검사한다. 만약 교차가 감지되면, 두 적 오브젝트의 m_xmf3MoveDirection 값을 std::swap 함수를 사용해 서로 교환한다.

- 피킹

기존에 사용했던 방식을 바탕으로 구현했다.

마우스로 클릭한 화면상의 2D 좌표를 카메라의 viewport와 projection행렬을 이용해 3D 뷰 공간 상의 레이로 변환한 뒤, 이 레이가 적 오브젝트의 메쉬와 교차하는지를 검사한다. 씬 내의 모든 적 오브젝트를 순회하면서, bActive가 true이고 폭발 중이 아닌 오브젝트만 대상으로 피킹 검사를 진행한다. 각 오브젝트에 대해 PickObjectByRayIntersection 함수를 호출하는데, 이 함수는 뷰 공간의 광선을 월드 공간으로 변환한 뒤, 해당 오브젝트의 메쉬와 자식 오브젝트들에 대해 재귀적으로 광선과 삼각형 사이의 충돌 검사를 수행한다. 자식 오브젝트까지 재귀적으로 탐색하여 계층 구조 전체에 대해 충돌 여부를 확인한다. 이 과정에서 충돌이 발생하면 광선의 시작점부터 충돌 지점까지의 거리를 반환

받고, 이 값이 현재까지 찾은 것 중 가장 가까운 경우 해당 오브젝트를 최종 선택한다.

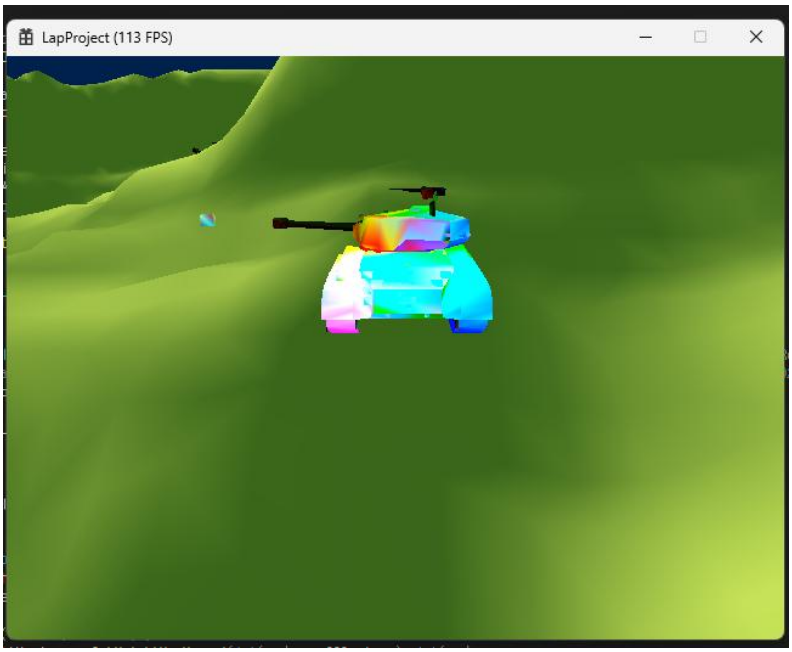
5) 적 탱크

적 탱크는 플레이어가 200.0f 이내의 근처에 있을 때 플레이어를 따라가고, 그렇지 않을 때는 m_xmf3MoveDirection방향으로 이동한다. 맵의 경계에 닿았을 때에는 m_xmf3MoveDirection의 x 또는 z값을 반전한다.

- 적이 플레이어를 따라오도록 하는 함수: CheckEnemyNearbyPlayer 함수

이 함수는 각 적 오브젝트가 플레이어와의 거리를 계산해, 플레이어가 200.0f 거리 이내에 있을 때 적이 플레이어 쪽을 향하도록 이동 방향 벡터를 설정한다. 먼저 bActive가 true인 모든 적 오브젝트를 순회하면서, 적과 플레이어의 위치를 가져온다. x, z 평면에서 두 위치의 차이를 계산해 거리를 구하고, 만약 이 거리가 200.0f이면, 방향 벡터를 구해 적의 m_xmf3MoveDirection에 할당한다. 방향 벡터는 플레이어 위치에서 적 위치를 뺀 벡터를 정규화해서 얻으며, 이를 통해 적이 플레이어 쪽으로 자연스럽게 움직이도록 유도한다. 이 과정은 매 프레임마다 반복되어, 근처에 있는 적들이 플레이어를 추적하는 AI 행동을 구현한다.

6) 플레이어



➔ 플레이어의 총알 발사

CTerrainPlayer 클래스를 수정하여 구현했다.

생성자에서 플레이어의 위치, 카메라, 지형 세팅, 메쉬 세팅, 총알 세팅을 수행한다. OnInitialize 함수를 이용해 플레이어 탱크의 포신이 정면을 바라보도록 초기 회전값을

지정한다. OnPlayerUpdateCallback과 OnCameraUpdateCallback 함수는 각각 플레이어와 카메라가 지형 위에 자연스럽게 위치하도록 y좌표를 실시간으로 보정한다. 플레이어가 지형보다 아래로 내려가면 y속도를 0으로 만들고, y좌표를 지형 표면 위로 올린다. 카메라도 마찬가지로 지형 표면보다 아래로 내려가지 않도록 한다.

머리와 포신을 따로 회전하기 위해 Rotate함수를 재정의하였다. 입력값에 따라 머리와 포신을 회전한다. 머리는 y축 회전을, 포신은 x축 회전을 담당하며, 포신의 회전 각도는 -20도~20도 사이 범위로 제한한다.

FireBullet 함수는 탱크가 목표 오브젝트를 향해 포신과 머리를 회전시키고 총알을 발사하는 기능을 담당한다. 목표 오브젝트의 위치와 포신 위치를 비교해 yaw와 pitch각도를 계산하고, 현재 포신의 방향과의 차이만큼 회전시킨다. 아직 발사되지 않은 총알을 찾아 발사할 수 있도록 세팅하고, 포신의 현재 위치와 방향을 기반으로 총알의 시작 위치와 이동 방향을 설정한다. 목표 오브젝트가 지정된 경우, 총알의 추적 대상 포인터에 할당한다.

3. 게임 플레이

- 조작법

방향키로 플레이어를 이동하고, 우클릭으로 적을 클릭해 피킹한다. 좌클릭 드래그로 머리와 포신을 회전할 수 있고, 우클릭 드래그로 탱크 전체를 회전할 수 있다. 적을 선택한 상태로 ctrl키를 누르면 적을 따라가는 총알을 발사한다. 적이 선택되지 않았으면 포신 방향으로 총알을 발사한다. 'w'키를 누르면 모든 적이 죽고 게임이 끝난다.

- 게임 진행

총 10마리의 적 탱크가 생성되며, 적 탱크는 맵을 돌아다닌다. 플레이어가 적 탱크와 충돌하면 적 탱크가 폭발하면서 플레이어가 죽는다(게임이 종료된다). 게임의 종료 조건은 적 탱크를 모두 죽이는 것과 플레이어가 죽는 것이 있다.

4. 느낀 점

처음에 게임 오브젝트의 계층 구조가 제대로 이식되지 않아서 많이 헤맸다. 그동안 CGameObject에서 상속받은 Tank 클래스에 Body와 Head를 선언해서 분리 회전이 가능하게끔 구현했는데, LabProject07-9-1의 트리 구조인 오브젝트 계층 구조를 가져와서 유사하게 구현했다. 계층 구조 이식 과정에서 자식 프레임이 부모의 변환을 제대로 상속받지 못하거나 변환 행렬이 갱신되지 않아 같은 위치에만 탱크가 그려지는 문제가 발생했다. 이 문제는 UpdateTransform 함수를 호출하지 않아서 발생했던 문제였고 이동/회전과 관련된 부분에 모두 저 함수를 추가함으로써 해결할 수 있었다.

또한 프러스텀 컬링을 적용하려고 관련 코드를 옮겼을 때, 탱크의 바디 윗부분이 아예 그려지지 않는 문제가 발생했다. 충돌은 제대로 확인되는 것으로 보아 바운딩 박스는 제대로 설정된 것 같은데 그려지지 않는 이유를 모르겠다. 그리고 계층구조를 다 타고 컬링을 해야하는지 확인하는 함수도 프레임을 많이 잡아먹을 것 같아서 컬링은 생략했다. 컬링 없어도 대략 110~130프레임정도 평균적으로 유지된다.

인스턴싱 쉐이더가 제대로 동작하지 않을 때는 많이 놀랐다. DirectX 12의 파이프라인과 리소스 바인딩 구조에 대한 기술적인 이해가 부족했던 탓에, 어디서 문제가 발생하는지 원인을 찾는 데 많은 시간이 걸렸다. 특히, 루트시그니처와 쉐이더 간의 바인딩 규칙, 그리고 인스턴스 데이터가 GPU에 올바르게 전달되는 방식 등 DirectX 12의 저수준 리소스 관리와 파이프라인 상태 객체(PSO) 생성 과정이 얼마나 까다로운지 직접 경험할 수 있었다. 실제로 루트시그니처와 쉐이더의 바인딩 방식이 일치하지 않으면 PSO 생성이 실패한다는 점을 알게 되었다. 이를 통해 그래픽 파이프라인의 각 단계와 리소스 바인딩, 그리고 HLSL과 루트 시그니처의 연결 방식에 대해 더 깊이 공부해야겠다는 필요성을 느꼈다.

플레이어 탱크의 머리를 회전할 때 카메라도 같이 회전하도록 구현하고 싶었는데(이전에는 Player 오브젝트에 헤드 메쉬를 넣고 바디를 따로 만들어서 플레이어 오브젝트가 회전하도록 만들었다), 잘 되지 않아서 우클릭 드래그로 전체 오브젝트와 카메라 회전을 구현했다.

과제를 진행하면서 계층 구조 이식, 프러스텀 컬링 및 충돌체크, 인스턴싱 쉐이더, 그리고 카메라 연동 등에서 다양한 시행착오와 기술적 문제를 직접 겪었고, 이를 해결하는 과정을 통해 3D 그래픽스 시스템의 구조와 DirectX 12 파이프라인에 대한 이해를 한층 더 깊이 쌓을 수 있었다. 특히 지형과 관련된 부분을 처음 배워서 신기했다. 지금 언리얼로 졸업작품을 진행하고 있지만, 이번 수업에서 배운 내용들을 상용 엔진에서 어떻게 사용하도록 만들어져 있는지 생각하면서 졸업작품을 진행해야겠다는 생각이 들었다. 개인적으로 이번 과제의 코드가 안예쁘다고 생각하는데, 종강 이후에 프로그램을 수정하면서 리팩토링도 해야겠다는 생각이 들었다.

보고서에 적었던 프레임은 모두 Debug모드로 실행했을 때의 결과이고, 마지막에 Release로 실행했을 때 평균적으로 330~360프레임(개발한 노트북이 360hz이다)정도가 나왔다.