**3D 게임프로그래밍 1**

**과제 01 설명 문서**

김태순 2021182009

# • 프로젝트의 구조

프로젝트의 구조는 소프트웨어 렌더러를 기반으로 하되 기존 프레임워크 중심의 구조를 씬 중심으로 바꾸고 각 씬에서 별도로 필요한 입력처리와 업데이트를 진행하게 했다.

# • 조작법

CScene: **ESC** – CMenuScene로 씬 전환

CStartScene/CMenuScene: **ESC** – CStartScene에서는 프로그램 종료, CMenuScene에서는 CStartScene으로 씬 전환**, 마우스 이동** – 마우스 위치에서 피킹 실행, **마우스 좌클릭** – 현재 피킹된 텍스트 오브젝트 폭발 및 씬 전환 시도

CLevel1: **ESC** – CMenuScene으로 씬 전환, **N/n** – Clevel2로 씬 전환

CLevel2: **ESC** - CMenuScene으로 씬 전환, **좌/우 방향키** -플레이어 좌우 회전, **앞/뒤 방향키** – 플레이어 앞/뒤 이동, **마우스 좌클릭 상태로 좌우 이동** – 카메라 및 플레이어의 총알 발사 방향 회전, **W** – “YOU WIN!”텍스트 객체 보여주기, **S** – 실드 토글, **A** – 자동 발사 토글

# • CSceneManager

이 객체는 모든 씬을 하나의 객체에서 관리하고 변경 가능하도록 만들기 위해 셜계하고 제작하였다. SceneManager는 CScene의 포인터로 이용하여 현재 씬 객체를 저장하고 있고 각 씬은 CScene에서 상속받아 오버라이딩한 업데이트(Animate,Render)함수들과 입력처리 함수들이 호출되어 각 씬의 내용을 실행한다.

씬을 변경할 때에는 우선 현재 씬의 종료 여부를 Animate에서 현재 씬의 IsFinished를 통해 bool값을 리턴받아 확인하고 이것이 true면 현재 씬에 저장되어 있는 다음 씬의 번호(int값)를 GetNextSceneID를 통해 확인하여 ChangeScene을 호출한다. 이때 각 번호에 해당하는 씬으로 SetCurrentScene을 호출하여 변경되고 번호가 -1이면 씬을 변경하지 않고 리턴된다. 이 과정에서 이전 씬의 객체들을 삭제하고 메모리를 해제하는 ReleaseObjects를 호출하고 현재 씬의 메모리도 해제한 이후에 변경될 씬을 새로 생성하고 BuildObjects를 통해 씬의 오브젝트들을 생성한다.

이 객체는 GameFrameWork의 소멸자, 즉 게임이 종료될 때 삭제되며 이때 ReleaseCurrentScene을 통해 현재 씬 또한 해제된다.

# • CScene

텍스트, 스크린샷, 스케치, 아동 미술이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

모든 씬이 공통적으로 가져야 하는 구조를 가진 원형 클래스로 모든 씬 객체들은 이 CScene을 상속받아 제작된다. 해당되는 구조로는 카메라를 가지고 생성되는 플레이어 객체와 현재 씬의 종료여부를 표시하는 bool값, 다음 씬의 번호를 저장하는 int값을 멤버 변수로 가지며 virtual로 선언된 함수들을 가지고 있다. 여기에 포함되는 함수들은 객체의 생성 및 소멸함수, Animate와 Render, 키보드와 마우스 입력처리 함수, 카메라 업데이트 함수와 멤버 변수를 얻어오는 IsFinished와 GetNextSceneID이 있다.

해당 씬은 Tutorial을 통해 확인할 수 있고 카메라와 플레이어만 생성될 뿐 어떠한 입력처리도 처리하지 않는다.

# • CTextCharacterObject

텍스트 오브젝트의 각 글자에 해당하는 객체로 글자를 표현하기 위한 큐브를 CGameObject\*를 통해 관리하며 이것을 벡터로 저장한다. 또한 모든 큐브는 가로 세로 높이가 1인 정육면체로 생성되므로 이것을 static CCubeMesh\*로 선언하여 여러 글자가 있어도 하나의 Mesh를 통해 생성되게 하였다.

각 객체는 BuildCharacterShape를 통해 큐브를 생성하고 배치하는데 이때 TextGlyphTable에 존재하는 GetGlyph를 통해 각 문자에 해당하는 7x7크기의 문자열을 얻고 해당 문자열의 #이 존재하는 자리에만 큐브를 생성하여 문자를 표현하였다.

# • CTextObject

텍스트를 3D객체로 표현하기 위해 제작. 회전에 필요한 각도와 텍스트 캐릭터 오브젝트의 포인터를 벡터로 저장하고 텍스트마다 다른 씬으로의 전환을 위해 다음 어떤 씬으로 전환될지에 대한 값이 저장되어 있다.

해당 오브젝트는 Animate를 통해 회전하는데 이때 오브젝트의 모델 좌표계 상의 원점이 오브젝트의 중앙이 아니기 때문에 중앙 글자의 위치를 원점으로 변환하는 행렬과 회전각도를 이용해 회전 변환 행렬을 만들어 중앙 글자의 위치를 기준으로 회전하게 행렬을 만들고 해당 행렬을 통해 변환된 각 글자의 위치를 각 글자 오브젝트에 저장하고 Render할 때 해당 위치에서 Render되도록 한다. 또한 피킹할 때에도 해당 위치를 기준으로 계산을 실시한다.

# • CStartScene / CMenuScene

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 스크린샷, 폰트, 로고이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

게임의 시작 씬과 메뉴 씬으로 다른 오브젝트 없이 텍스트 오브젝트와 폭발 오브젝트만을 관리하며 플레이어의 메쉬를 NULL로 설정하고 카메라를 플레이어의 -z방향으로 10만큼 떨어진 위치에 생성하고 회전과 이동에 관한 입력처리를 없애 UI처럼 동작하게 했다.

마우스가 움직일 때 매 프레임마다 피킹을 실행하여 피킹된 텍스트 객체가 있다면 해당 텍스트의 색을 파란색으로 바꾸어 표시하게 만들었다. 좌클릭을 입력하면 현재 피킹되어 있는 m\_pHitText가 있다면 해당 텍스트의 위치와 색을 이용해 폭발 오브젝트를 생성하고 해당 텍스트 객체는 비활성화 한다. 이때 폭발 오브젝트는 부모 오브젝트로써 텍스트 객체의 포인터를 저장한다.

이후 Animate에서 폭발 오브젝트의 폭발이 종료된 것을 확인하면 해당 폭발 오브젝트의 부모 오브젝트인 텍스트 오브젝트에서 타겟 씬 ID를 받아와 씬을 전환할 준비를 하고 해당 폭발 오브젝트는 삭제한다. 삭제 이후에 씬이 종료되었음을 알린다.

# • CRailMesh, CRailObject / RailSegment

Level1에서 롤러코스터를 구현하기 위한 객체.

CRailMesh와 CRailObject는 롤러코스터의 경로를 시작적으로 표현하기 위한 객체로 긴 직육면체의 형태로 만들어지며 롤러코스터의 경로를 따라 배치된다.

RailSegment는 실질적인 롤러코스터 경로의 위치와 방향을 저장하는 구조체로 각 세그먼트의 위치와 방향을 이용해 플레이어를 이동시키고 회전시킨다.

# • CLevel1

도표, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

롤러코스터를 구현한 씬으로 RailSegment, CRailObject\*를 벡터로 저장하고 있다. 또한 현재 레일 세그먼트의 인덱스와 현재 세그먼트와 다음 세그먼트의 진행도(보간에서 사용되는 t값), 진행도가 증가하는 속도와 카메라의 회전속도를 변수로 가지고 있다. 세그먼트는 BuildRailSegments에서 컨트롤 포인트를 캣멀-롬 스플라인을 통해 보간하여 생성하며 이렇게 생성된 세그먼트를 이용해 BuildRailObjects에서 CRailObject를 생성하고 UpdatePlayerOnRail에서 각 세그먼트의 위치와 방향(tangent)을 선형보간하여 플레이어의 위치와 방향을 변경한다. 이렇게 플레이어를 회전시키고 회전된 플레이어를 기준으로 카메라의 위치(offset)를 재설정하면 카메라도 플레이어를 따라 회전 및 이동이 자연스럽게 이루어진다. 마지막 세그먼트에 도착하면 m\_nNextSceneID을 CLevel2의 가리키는 값인 3으로 설정하고 m\_bSceneFinished을 true로 바꿔 씬이 바뀌게 만든다.

이 씬에서는 n/N과 ESC키를 제외한 입력처리는 하지 않고 있고 n/N키를 누르면 마지막 세그먼트에 도착한 것과 같은 과정을 거쳐 다음 씬으로 전환되며 ESC키도 같은 처리가 일어나지만 nNextSceneID의 값이 MenuScene을 가리키는 1로 설정된다.

# • CTankPlayer

CLevel2에서 사용될 탱크 플레이어 객체로 CPlayer를 상속받는다. 그러나 Player와는 다르게 3개의 메쉬를 이용해 아랫몸통, 윗몸통, 포신을 따로 계산 및 렌러링하여 더 탱크스러운 움직임을 구사하였다. 또한 이동 전의 위치를 저장하는 m\_xmf3PrePos를 가지고 있다.

메쉬를 설정할 때에는 SetTankMesh를 이용해 설정하는데 이때 윗몸통의 메쉬와 아랫몸통의 메쉬의 OBB를 이용하여 전체 탱크 오브젝트에 대한 OBB를 생성한다. 또한 바운딩 박스의 업데이트는 각 메쉬의 OBB를 월드 변환으로 변환한 뒤 BoundingBox::CreateMerged를 이용해 통합된 AABB를 만들고 이것을 CreateFromBoundingBox를 이용해 OBB로 변환하여 플레이어의 바운딩 박스를 업데이트 한다.

회전은 마우스를 이용한 윗몸통과 포신의 회전과 키보드를 이용한 아랫몸통의 회전이 나뉘어서 처리된다. Rotate에서는 키보드 입력에 의해 아랫몸통 및 플레이어의 이동 방향의 회전이 결정되는데, XMMatrixRotationY를 이용하여 플레이어의 Look, Right만을 변화시켜 회전에 의해 플레이어가 바닥 평면을 벗어나는 일이 일어나지 않게 하였다. RotateCameraOffset을 이용해 마우스에 의한 회전은 기존 카메라의 offset을 y값과 x,z값을 나누어 처리하여 같은 방법으로 y값은 변하지 않고 xz에 대한 회전만을 계산하고 기존 y값과 회전된 xz값을 새로운 offset으로 설정하여 카메라의 위치를 설정해서 카메라가 대각선으로 회전하는 문제를 방지하고 xz평면에 평행하게 회전되도록 만들었다. 또한 UpdateTopParts를 이용해 윗몸통의 Look을 카메라의 Look에서 y를 제외한 값으로 변경하여 카메라 회전에 따라 자연스럽게 윗몸통이 회전하되 위아래로는 회전하지 않도록 설정하였다.

플레이어의 이동은 좌우 방향키를 입력했을 때는 Rotate를, 앞뒤 방향키를 누르면 Move를 실행하게 만들어 실제 탱크와 유사한 움직임을 구현하였다.

Render과정에서 각 메쉬는 탱크의 월드변환 행렬을 기준으로 렌더링되며 윗몸통은 자신이 가진 회전에 관한 값(Right,Up,Look)을 이용하여 월드변환 행렬을 바꾸고 아랫몸통의 높이만큼 +y로 이동하여 렌더링된다. 포신은 윗몸통의 월드변환 행렬을 기준으로 아랫몸통의 높이와 윗몸통의 높이를 기준으로 하여 높이를 설정하고 윗몸통의 Look방향으로 윗몸통의 길이만큼 이동하여 앞쪽에 렌더링 되도록 계산한다.

총알 발사 같은 경우에는 플레이어의 위치에서 아랫몸통의 높이와 윗몸통의 높이를 이용해 y값을 조정하고 윗몸통의 길이를 더해 발사 위치를 정하고 윗몸통이 바라보는 방향을 기준으로 발사 방향을 지정해 포신에서 나가는 것처럼 보이게 제작하였다.

# • CTankEnemy

CTankPlayer를 상속받은 적 개체로 크게 달라진 것은 없지만 Animate에서 일정 시간마다 회전하게 만들어 다양한 방향으로 이동할 수 있도록 하였고, 또한 피킹에 대한 처리는 메쉬에서 이루어지기에 PickObjectByRayIntersection에서 각 메쉬(윗몸통과 아랫몸통)에 대한 피킹 검사를 실시하게 구현하였다.

# • CLevel2

텍스트, 스크린샷, 도표, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

탱크 게임을 구현한 씬으로 탱크 플레이어와 승리 텍스트(YOU WIN!), 피킹된 오브젝트는 포인터로 저장하며 적, 장애물, 폭발 오브젝트들은 포인터들의 벡터로 저장한다. 또한 승리 메시지를 보여줄지에 대한 여부는 bool값으로 씬에서 관리한다.

충돌검사 같은 경우에는 플레이어와 적 탱크, 총알과 적 탱크, 장애물과 모든 오브젝트에 대한 충돌검사를 실시하는데 모든 충돌은 오브젝트가 가진 OBB의 Intersects를 이용해 충돌 검사를 실시한다. 플레이어와 적 탱크가 충돌이 일어나면 플레이어의 실드 여부에 따라 적이 삭제되거나(이 경우 폭발이 일어나지 않음) 플레이어가 원점(0,0,0)으로 이동된다. 총알과 적 탱크의 충돌에서는 먼저 총알이 활성화되어 있는지 확인하고 활성화되어 있다면 충돌된 적의 위치와 색을 이용해 폭발 오브젝트를 생성하고 해당 적 객체를 삭제한 이후에 총알 역시 비활성화 한다. 마지막으로 장애물과의 충돌 검사는 모든 오브젝트가 실시하는데 플레이어는 이동이 실행되기 전의 위치를 m\_xmf3PrePos에 저장해 두었다가 장애물과 충돌되었으면 해당 PrePos로 위치를 변경하여 마치 벽에 막힌 것처럼 구현하였다. 총알은 장애물과 충돌되었으면 총알을 비활성화 시키고 적 탱크는 ReverseDirection를 이용해 현재 이동방향의 반대방향으로 이동하게 만들었다. 적 탱크의 이동은 현재 Look방향으로 m\_iMoveDirection \* 5.0f \* fElapsedTime만큼 이동하는 것이기에 ReverseDirection을 통해 m\_iMoveDirection의 값을 반전시키면 반대로 이동되는 것처럼 보이게된다.

승리 메시지는 Animate에서 CheckWinCondition을 통해 적 객체의 벡터가 비어있다면 m\_bShowWinMessage를 true로 만들고 이것이 true가 되면 Animate와 Render에서 해당 텍스트 객체에 대한 업데이트를 실행하여 화면에 보이게 제작하였다.