**[3D게임 과제 02 보고서]**

[게임공학과]

최해성(2018180044)

**-과제에 대한 목표-**

저의 이번 과제에 대한 목표는 다음과 같습니다.

1. 교수님께서 제공해주신 자동차 3D 모델을 불러온다.

2. 다함께 차차차와 유사한 자동차 트랙을 생성한다.

3. 자동차 트랙은 무한하게 이어진다.

4. 자동차 트랙 위에 돌 모양의 장애물을 생성한다.

5. 자동차와 돌이 충돌하면 돌은 폭발한다(인스턴싱을 이용한 파티클 이펙트).

6. 플레이어는 키보드 왼쪽, 오른쪽 키로 트랙을 바꾸고 위, 아래키로 속도를 조절한다.

**-과제 실행 환경-**

과제의 실행 환경은 다음과 같습니다.

1. Visual Studio 2019 최신버전

2. 준수 모드 해제

3. Release/x64 모드

Etc) exe 파일 실행을 위해선 Model 폴더와 Shaders.hlsl이 필요합니다.

**-조작법-**

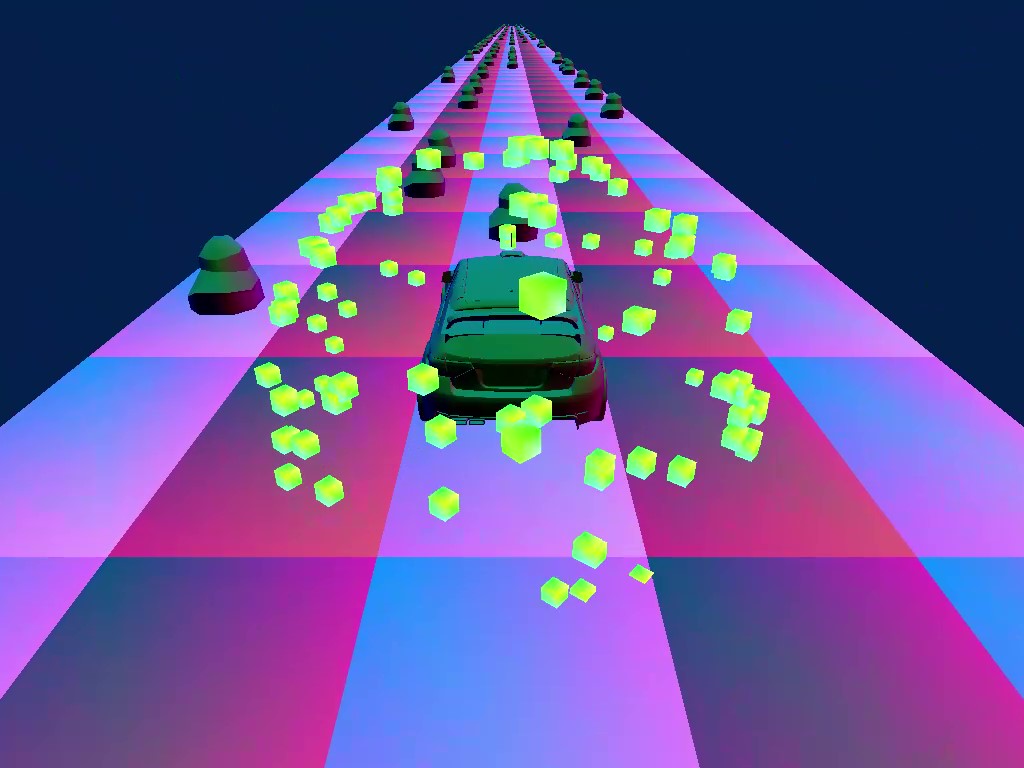
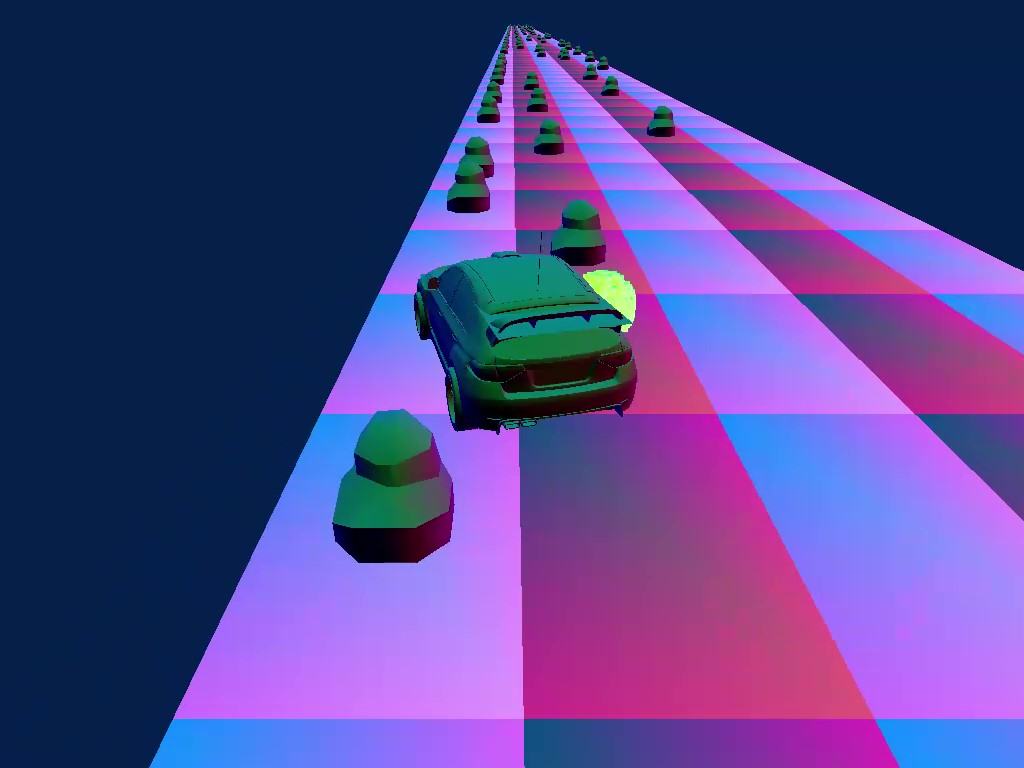
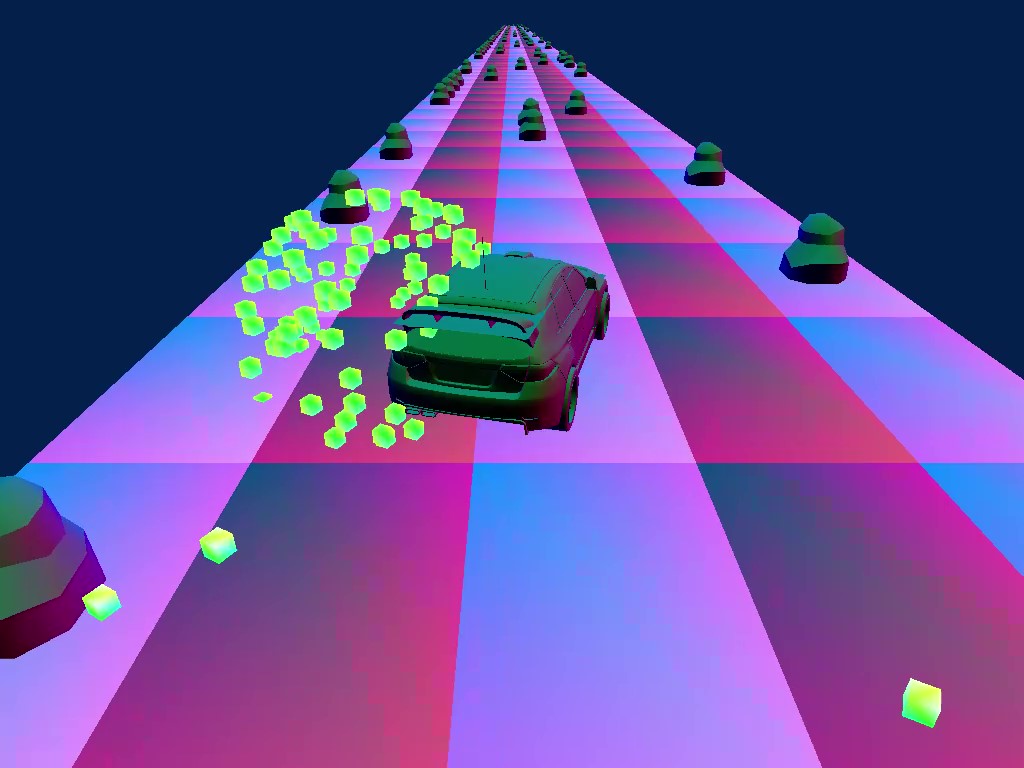
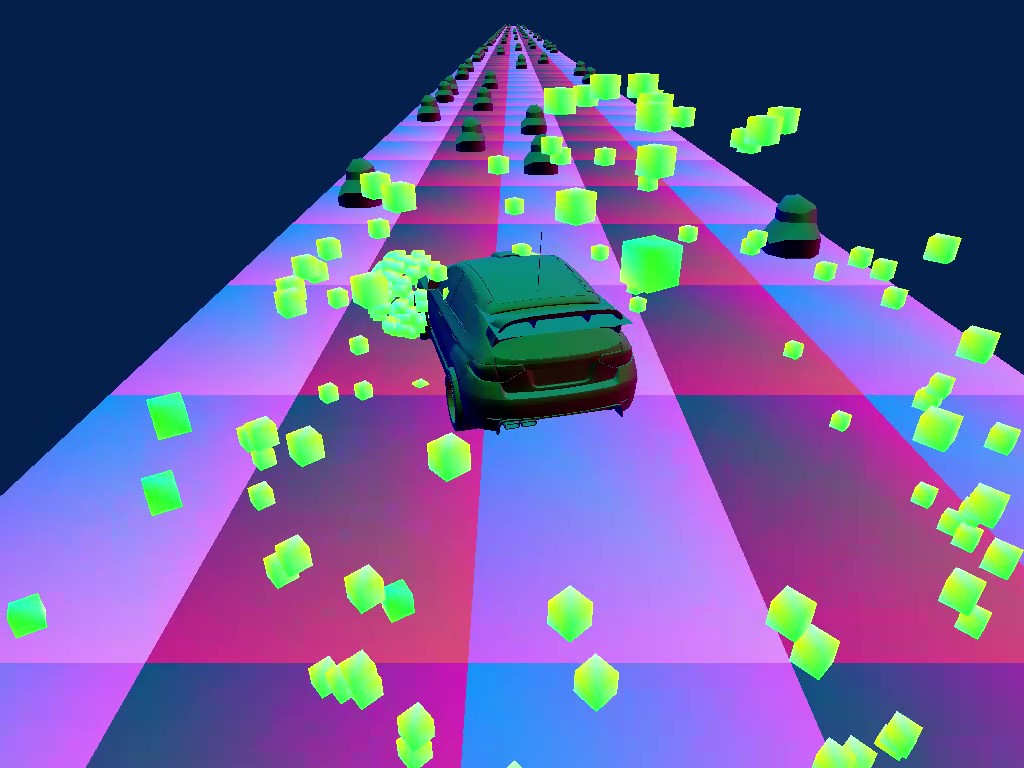
키보드 왼쪽, 오른쪽 키로 트랙을 바꿉니다.

키보드 위, 아래 키로 속도를 조절합니다.

키보드 Z/z로 플레이어 모델을 90도 회전합니다.

F1으로 1인칭 카메라 시점, F3으로 3인칭 카메라 시점으로 전환합니다.

**-실행결과-**

****

**-구현방법-**

**1. 오브젝트 생성 및 렌더링 준비**

외부에 저장된 자동차 모델을 불러오기 위해서 CHierarchyObject라는 클래스를 생성해 CGameObject를 부모로 가지게 하였습니다. CHierarchyObject는 계층구조를 가진 모델들을 관리하기위한 클래스입니다(모델 정보를 불러오는 코드는 교수님의 SoftwareRenderer를 참고해서 비슷하게 만들었습니다.). 프로그램이 시작하면 CGameFramework의 OnCreate 함수가 호출되며 Direct3D 디바이스, 명령 큐, 명령 리스트, 스왑체인 등을 생성합니다. 이때, BuildObject() 함수가 호출되게 되는데 BuildObject()에서 플레이어 오브젝트를 관리할 CCarPlayer 클래스와 플레이어 외의 나머지 객체를 관리할 CScene 클래스를 생성합니다. BuildObject()에서는 CScene을 생성하고 CCarPlayer를 생성하게 되는데 CScene을 생성하기위해 new로 메모리를 할당해주고 CScene::BuildObject()를 호출하게됩니다. CScene::BuildObject()에서는 CreateGraphicsRootSignature() 함수로 셰이더에 객체의 월드변환행렬과 카메라의 정보, 인스턴싱을 위한 셰이더 리소스를 전달할 수 있도록 루트시그니쳐를 생성합니다. CScene에서는 그 루트시그니쳐 정보를 m\_pd3dGraphicsRootSignature라는 이름을 가진 ID3D12RootSignature\* 변수에 저장하고 자동차 트랙과 장애물과 이펙트를 렌더링할 CInstancingShader 클래스를 생성해 동적할당해주고 CInstancingShader::CreateShader() 함수로 그래픽스 파이프라인 상태 객체를 생성합니다. 그리고 CInstancingShader::BuildObject() 함수로 객체들을 생성해주는데 여기에는 자동차 트랙, 장애물, 충돌 이펙트를 위한 객체들이 생성이 됩니다. 먼저, 자동차 트랙을 생성하기 위해서 CTrackObject 클래스를 1500번 동적 할당하고 생성해 그 위치를 가로(x축)길이 30.0f, 세로(y축)길이 50.0f 간격을 가지도록 3중for문으로 설정하고 m\_ppObjects라는 CGameObject\*\* 변수로 객체들을 관리할 수 있게 넣어주었습니다. 즉, 1500개의 트랙(x축으로 5개 z측으로 300개)을 관리할 객체를 생성해 m\_ppObjects 변수로 넣어주었습니다. 이 과정에서 std::random\_device와 std::default\_random\_engine, std::uniform\_int\_distribution을 사용해 하나의 z축 트랙에 랜덤하게 하나의 x축의 위치를 가진 장애물을 생성하였고 ObstacleObject라는 클래스로 이 장애물을 관리해주었습니다. 이때 외부의 돌 모양 메쉬를 가져왔습니다. 또한 장애물들은 CInstancingShader 클래스안에 protected 성격을 가진 v\_Obstacle이라는 vector 컨테이너에 push\_back 함수로 담아 CInstancingShader에서 트랙, 장애물을 관리할 수 있게 하였습니다. 장애물을 관리하는 ObstacleObject는 외부의 돌 모양의 모델을 가진 장애물 오브젝트를 관리하는 P\_Obs라는 CHierarchyObject\*와 플레이어와 장애물이 충돌했을 때 파티클 이펙트를 생성해줄 CGameObject\*를 담은 vector를 가지고 있습니다. 또한 파티클 이펙트를 생성할 때 인스턴싱을 이용하기 위해 m\_myGameObjects라는 ID3D12Resource\*와 버퍼를 가르킬 VS\_VB\_INSTANCE\*형 변수 m\_myMappedGameObjects를 가지며 플레이어 객체와 충돌했는지 여부를 판단하는 bool형 변수 IsExPlosing, 폭발하고 얼마나 지났는지 알 수 있는 float형 duration 변수를 가집니다. 트랙과 장애물을 생성한 뒤 트랙들의 정보를 담고 있는 m\_ppObjects의 첫번째만 메쉬를 가지게 하고 장애물들을 담고있는 v\_Obstacle을 순회하면서 각 ObstacleObject\*의 SetExplosion이라는 함수를 호출해주었습니다. ObstacleObject::SetExplosion()은 장애물의 위치에 160개의 CRotatingObject를 생성하고 각 CRotatingObject는 랜덤한 방향으로 움직이게 방향을 설정해주고 인스턴싱을 위한 준비들을 해주었습니다. 이 과정을 마치면 다시 CGameFramework::BuildObjects()로 빠져나와서 플레이어 객체를 생성하는데 플레이어 객체는 CCarPlayer라는 클래스로 관리해주었습니다. CCarPlayer는 생성자에서 카메라를 삼인칭 카메라로 생성하고 RallyCar 모델을 불러오고 Child로 설정한 뒤 OnInitialize()에서 앞 바퀴, 뒷 바퀴 프레임을 찾아 포인터 변수로 가지고 있게 했습니다. CCarPlayer는 m\_pScene::GetTrack()함수로 트랙의 정보들을 가져와 CCarPlayer::SetTrack() 함수로 트랙 정보들을 m\_Track이라는 포인터 변수 가르키게 했습니다. 반대로 Scene은 Scene::SetPlayer()함수로 플레이어의 정보를 m\_pPlayer라는 포인터 변수로 가르키게 했습니다.

**2-1. 플레이어 오브젝트 애니메이션**

플레이어 오브젝트를 관리하는 CCarPlayer 클래스는 생성자에서 OnInitialize() 함수로 바퀴에 해당하는 프레임을 찾아 CHierarchyObject\*로 저장했습니다. 그리고 CGameFramework::FrameAdvance()가 호출되어 프레임이 하나 전진할 때 CGameFramework::AnimateObjects()에서 CCarPlayer::Animate() 함수가 불리게 되고 이때 앞 왼쪽 바퀴, 앞 오른쪽 바퀴, 뒤 왼쪽 바퀴, 뒤 오른쪽 바퀴의 CHierarchyObject의 월드변환행렬에 Matrix4x4::Multiply로 X축을 기준으로 회전하는 행렬을 곱해주어 바퀴가 돌아가는 애니메이션을 구현하였습니다. 키보드 Z/z를 누르면 플레이어의 자동차 모델이 90도 회전해 그 모습을 볼 수 있습니다.

**2-2. 플레이어 이동**

프로그램 실행 중에 키보드를 조작하면 CGameFramework::OnProcessingKeyboardMessage()가 호출이 되고 키보드 왼쪽, 오른쪽키를 눌러서 호출이 됐을 경우 그 정보를 dwDirection이라는 이름을 가진 변수에 저장합니다. 그리고 매 프레임마다 CGameFramework::FrameAdvance() 함수가 호출되는데 CGameFramework::FrameAdvance()에서 CGameFramework::ProcessInput()이 호출이 되고 CGameFramework::ProcessInput()에서 CPlayer::Update() 함수가 호출이 됩니다. 플레이어는 현재 자신이 x축 상으로 몇 번째 트랙에 위치하는지를 xRail이라는 정수형 변수에 저장하고 있고 트랙 변경을 위한 xNextRail이라는 정수형 변수를 가지고 있으며 현재 자신이 z축 상으로 몇 번째 트랙에 위치하는지를 zRail이라는 정수형 변수에, 다음 가져야할 zRail 값을 zNextRail이라는 정수형 변수에, 그리고 플레이어의 좌우이동과 전진이동을 하기위한 보간 값 tx, tz를 가집니다. CPlayer::Update() 함수에서 플레이어 객체는 트랙을 교체하고 있는지 xRail과 xNextRail의 값을 비교해서 같으면 교체하고 있지 않고 다르면 교체하고 있다는 것으로 판단하는데 트랙을 교체하고 있지 않고 이전에 키가 눌려 dwDirection에 값이 있을 경우 dwDirection에 따라 xNextRail을 설정하고 해당 방향으로 모델을 15도 회전합니다. 그리고 플레이어의 보간값 tz, tx(트랙을 변경 중이라면)를 speed라는 플레이어의 속도를 정의하는 변수에 이전 프레임에서 지금 프레임까지 지난 시간 fTimeElapsed를 곱해 더 해줍니다. 이때 tx, tz가 1.0f보다 크다면 값을 초기화 해주고 xRail, zRail, xNextRail, zNextRail을 적절히 설정해주었습니다. 그 다음 tz의 값으로 플레이어가 가지고 있는 트랙의 정보 m\_Track에서 현재 트랙과 가져야할 트랙의 z 값을 보간해 resultPosition이라는 XMFLOAT3 변수의 z에 저장하고 xRail과 xNextRail을 비교해 트랙을 변경 중이라면 tx의 값으로 m\_Track의 x좌표를 보간해 resultPosition의 x에 저장하며 트랙을 변경 중이 아니라면 현재 x 좌표를 resultPosition에 저장하고 만들어진 resultPosition으로 플레이어의 위치를 정해주는 방식으로 플레이어를 이동시켰습니다.

**3-1. 플레이어 - 장애물 충돌처리**

플레이어 – 장애물의 충돌은 BoundingOrientedBox::Intersects() 함수로 처리해주었습니다. 플레이어와 장애물은 둘 다 외부 모델을 불러와 계층 구조를 가진 CHierarchyObject 클래스로 관리를 해주었는데 외부 모델을 불러오는 과정에서 바운딩 박스에 대한 정보를 가져와 m\_xmOOBB라는 BoundingOrientedBox 변수에 저장해 주었고 매 프레임 마다 CHierarchyObject::checkCollision() 함수에서 충돌을 확인해주었습니다. 충돌을 확인 할 때 계층구조를 가지고 있으므로 메쉬에 대한 정보가 있으면 BoundingOrientedBox::Intersects()로 충돌 여부를 검사하고 충돌 했을 경우 true를 리턴해주며 충돌하지 않았을 경우 m\_pChild,, m\_pSibling을 검사해 자식이나 형제들의 노드에 대해서도 충돌체크를 하게 재귀함수 비슷한 형식으로 구현을 해주었습니다.

**3-2. 충돌 이펙트**

ObstacleObject는 1에서 말한 ObstacleObject::SetExplosion() 함수로 파티클 이펙트에 필요한 파티클들의 정보를 가지고 있습니다. 또한 3-1에서 충돌여부를 확인하고 충돌했을 경우 해당 장애물의 IsExPlosing이라는 bool 변수를 true로 바꾸어주는데 장애물을 렌더링하는 과정에서 IsExPlosing 변수를 검사해 true라면 인스턴싱에 필요한 정점 정보를 설정하고 파티클 이펙트를 렌더링 해주며 아니라면 돌 모양의 장애물을 렌더링 해줍니다. 또한 ObstalceObject는 매 프레임 마다 ObstacleObject::Update() 함수가 불려 IsExPlosing이 true로 폭발하는 상태라면 duration이라는 폭발이 지속되고 있는 시간에 fTimeElapsed를 더해 얼마나 폭발이 지속되고 있는지 확인하여 일정이상 폭발이 지속된 이후에는 IsExPlosing 변수를 false로 바꾸어주고 파티클 이펙트를 초기화 시켜주었습니다. 만약 duration이 아직 충분하지 않아 폭발하는 중이라면 파티클 이펙트에 사용되는 큐브 메쉬들을 각 오브젝트가 가지고 있는 방향으로 이동시키면서 Y축을 기준으로 회전시켜주었습니다.

**4. 트랙 생성**

자동차 트랙은 5x300의 TrackObject들로 구성이 되어있습니다. 프로그램이 구동될 때 CGameFramework::BuildObjects() 함수에서 CScene()::BuildObjects() 함수가 불려 CInstancingShader::BuildObjects() 함수가 불릴 때 생성이 되는데 이 트랙들을 무한하게 이어주기 위해서 플레이어가 전진할 때 마다(tz라는 매개변수가 1.0f보다 커질 때 마다) UpdateCount라는 정수형 변수를 하나씩 더해주고 이 UpdateCount가 100이 되면 플레이어가 100칸 전진했다는 뜻이 되므로 CScene()::AnimateObjects() 에서 이를 확인해 CInstancingShader::UpdateTrack() 함수를 호출하는 식으로 트랙을 업데이트 시켜주었습니다. CInstancingShader::UpdateTrack()에서는 5x100개의 트랙(5칸 100줄)을 15000.0f(트랙의 크기는 50.0f이며 300칸 전진을 해야 트랙의 끝에 이어서 붙음)만큼 z값을 이동시켜줘서 트랙을 이어붙어주었고 그 위에 위치하는 장애물들도 ObstacleObject::UpdatePosition() 함수를 호출해 장애물들도 15000.0f만큼 z값을 이동시켜주는 방식으로 트랙을 무한하게 이어지도록 구현하였습니다.