3DGP

과제 3

Rollercoaster Application

후안
2016180046

구현한 기능

Category	Feature	Details
HLSL	Instancing Shaders 하고, Diffused Shaders 를 파일 따로 분리	VSInstancing, PSInstancing “InstancingShader.hlsl”에 넣고 VSDiffused, PSDiffused 는 “DiffusedShader.hlsl”에 넣었습니다.
	Include 클래스 (ID3DInclude 의 Child 클래스 생성)	Shader 들이 파일 따로 했지만 VSDiffused 하고 VSInstancing Vertex Shader 들이 둘다 Root Signature 가 정한 Constant Buffer 를 필요합니다. 그래서 중복성을 없애기 위해서 Constant Buffer 정의를 “CBStruct.hlsl” 파일에 넣고 InstancingShader 와 DiffusedShader HLSL 파일들이 둘다 CBStruct HLSL 파일 Include 합니다. Shader Compile 할때 ShaderInclude 클래서를 통해서 CBStruct.hlsl Compile 데이터에 include 할 수 있습니다.
Instancing	Column Instancing	정해진 시간마다 레일 생성하면서 Instancing 기능을 쓰는 Column Object 를 생성해서 Terrain 이나 y = 0 플레인에 붙이는 겁니다. 이런식으로 해서 실제 물리적인 Rollercoaster 느낌이 납니다.
Macro	_3DGP_BEGIN_, _3DGP_END_ (namespace _3DGP_ 범위 정하는 Macro)	앞으로 이 구현한 프레임워크를 쓰기 위해서 모든 해당하는 클래스를 3DGP namespace 에 있습니다. 그래서 앞으로 새로운 Camera 나 Object 클래스 만들면 namespace 다르니까 안 겹칠 겁니다.
Frustum Culling	Terrain 의 Bounding Box 제대로 설정	과제 3 에서 Terrain Height Mesh 높이가 0 부터 1 까지인 줄 알아서 Bounding Box 설정을 제대로 안 해서 Frustum Culling 이 문제가 있었습니다. 이제 높이 범위를 계산해서 맞는 Bounding Box 만들게 됩니다.

Controls

(Rollercoaster Scene 의 Process Input 함수에서 처리)

Key	Action
‘W’	생성할 Rail 들이 Pitch 양수 회전
‘S’	생성할 Rail 들이 Pitch 음수 회전
‘D’	생성할 Rail 들이 Yaw 양수 회전
‘A’	생성할 Rail 들이 Yaw 음수 회전
'Q'	생성할 Rail 들이 Roll 양수 회전
'E'	생성할 Rail 들이 Roll 음수 회전

Spacebar	Spacebar 계속 누르고 있으면 게임 시간이 10 배로 더 느리게 됩니다. Spacebar release 하면 게임 원래 속도로 됩니다.
F1	1 인칭 카메라. 이 카메라는 Player 회전에 따라서 회전을 합니다.
F2	3 인칭 카메라. 이 카메라는 Player 의 위치를 따라하지만 Player 의 회전 정보 상관없고 Player 위치만 바라보는 겁니다.
F3	3 인칭 카메라. 이 카메라는 Player 회전에 따라서 회전을 합니다.
F4	위치를 고정되어 있는 카메라. 이 카메라로 계속 Wagon 을 보이지만 위치를 안 바꿔서 Rollercoaster 를 멀리 모습을 보일 수 있습니다.

Column Spawn 로직 설명

각 Column 이 Mesh 두개 있을 겁니다. Column A 라는 Mesh 와 Column B 라는 Mesh 있을 겁니다. Column A 는 생성하는 Rail 과 함께 똑같은 World 변환 가지고 있지만 Column B(기둥) World 의 Up 벡터와 평행해야 해서 따로 World 변환 배열로 만들었습니다. 다음은, SpawnColumn 함수입니다:

RailObjectShader.cpp (244~253 줄)

```
void RailObjectShader::SpawnColumn(const RailObject& ConnectedRail)
{
if (!m_Terrain) return;
	ColumnObject Column;

	Column.SetWorldTransform(ConnectedRail.GetWorldTransform());
	Column.CalculateColumnTransform(ConnectedRail.GetPosition(),
	ConnectedRail.GetLookVector(), *m_Terrain);

	m_ColumnObjects.emplace_back(std::move(Column));
}
```

ColumnObject 라는 Object 클래스는 World 변환 4X4 Float 변수가지고 있습니다. GameObject 의 World 변환 변수는 Column A Mesh 와 같이 쓸 거라서 생성한 Rail 의 World 변환 Matrix 로 설정합니다. CalculateColumnTransform 는 다음과 같습니다:

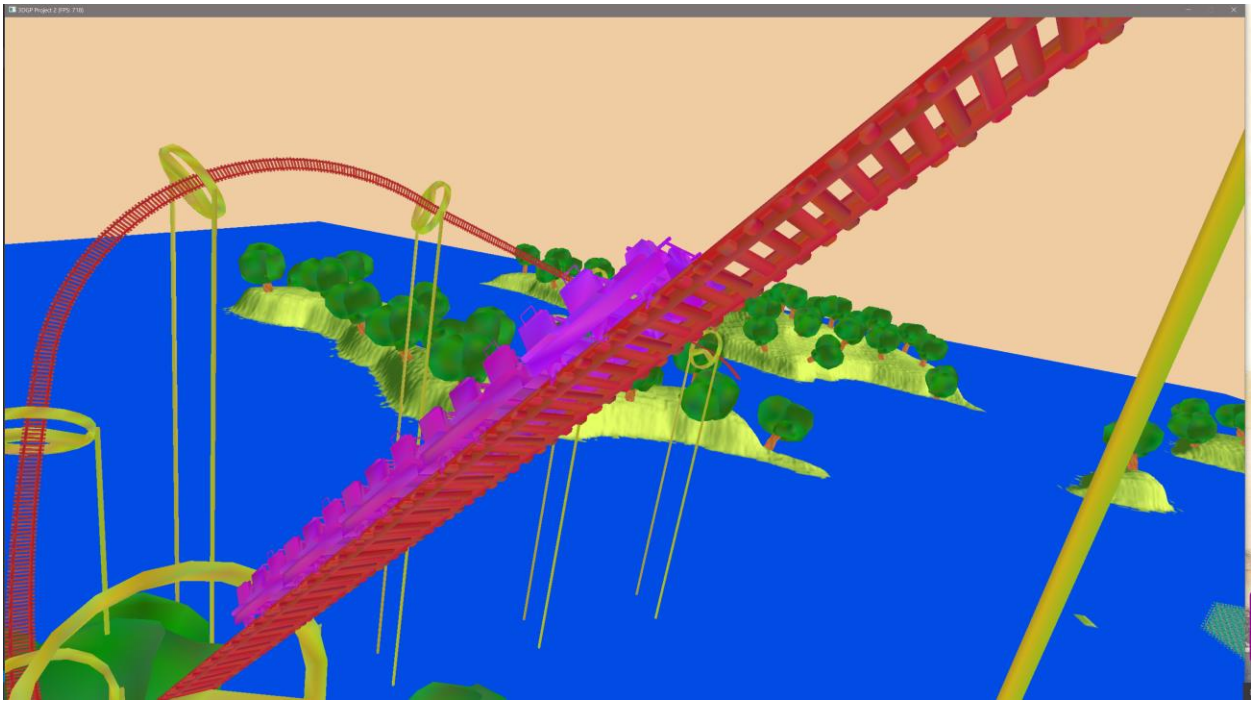
```
void XM_CALLCONV ColumnObject::CalculateColumnTransform(const DX XMFLOAT3& Pos, DX
XMVECTOR_P0 LookVector, HeightMapTerrain* pTerrain)
{
	DX XMFLOAT3 Look;
	XMStoreFloat3(&Look, LookVector);

	DX XMFLOAT3 Up;
	XMStoreFloat3(&Up, gWorldUp);

	DX XMFLOAT3 Right;
	XMStoreFloat3(&Right, XMVector3Normalize(XMVector3Cross(gWorldUp,
			LookVector)));

	float Height = pTerrain->GetHeight(Pos.x, Pos.z);
	float DeltaHeight = Pos.y - Height;


	m_ColWorld._11 = Right.x;
	m_ColWorld._12 = Right.y;
	m_ColWorld._13 = Right.z;

	m_ColWorld._21 = Up.x;
	m_ColWorld._22 = Up.y;
	m_ColWorld._23 = Up.z;

	m_ColWorld._31 = Look.x;
	m_ColWorld._32 = Look.y;
	m_ColWorld._33 = Look.z;

	m_ColWorld._41 = Pos.x;
	m_ColWorld._42 = Pos.y;
	m_ColWorld._43 = Pos.z;

	XMStoreFloat4x4(&m_ColWorld, XMMatrixMultiply(XMMatrixScaling(1.f,
	DeltaHeight,  1.f), XMLoadFloat4x4(&m_ColWorld)));
}
```

CalculateColumnTransform 함수는 Rail의 위치와, Rail Look Vector를 받아서 World Rotation하고 World Location 계산 할 수 있습니다. World Location는 바로 Rail의 위치이고, World Rotation는 각 Right, Up하고 Look 직접 계산합니다. Look 벡터는 Player의 바라보는 방향으로 같아야 돼서 Player Look벡터와 같고, Colum(기둥)의 로컬 Y출하고 World Up 벡터 똑같게 하고 Right는 이미 구현한 Up벡터 하고 Look벡터를 써서 외적 곱셈으로 계산합니다.
마지막으로 기둥이 Default 높이가 1.f 인데 바닥에 딱 맞추기 위해서 Terrain Height 알아보고 지금 Player 의 높이와 차이를 계산합니다. 높이 차이가 Scaling 하는 걸로 했는데 결과가 좋았습니다.

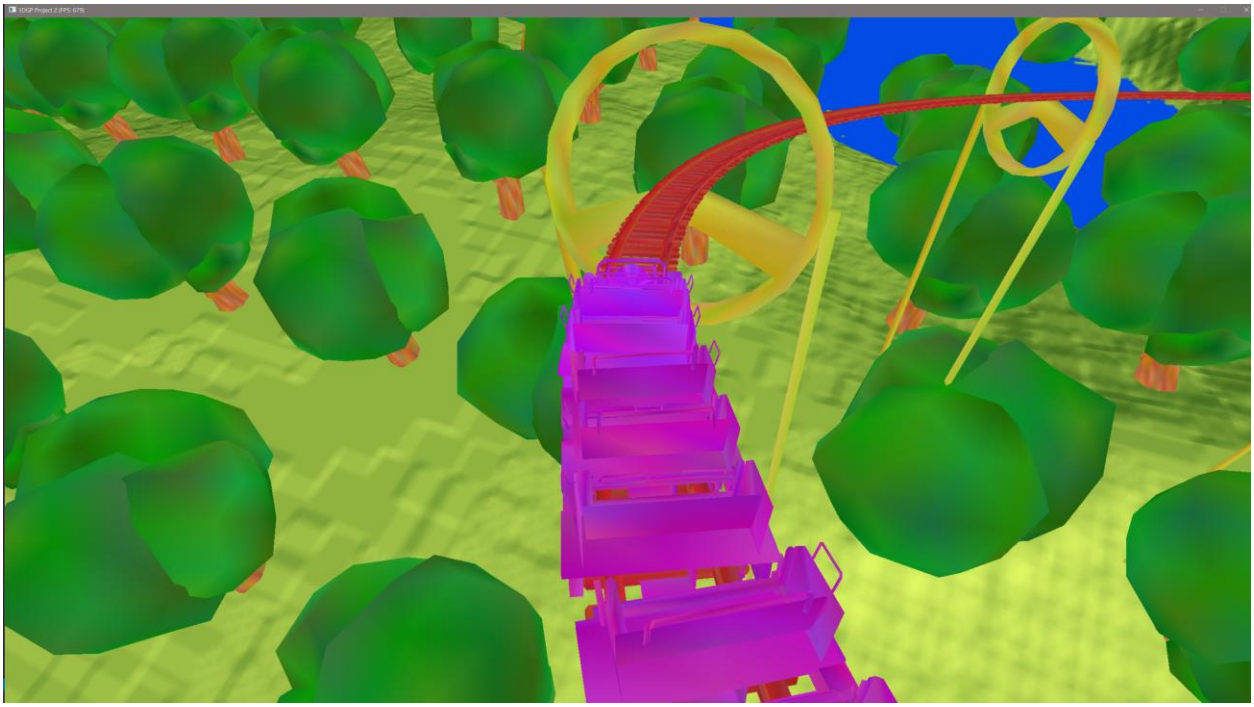
Gameplay Screenshots

Camera Mode F1 (1 인칭 카메라, 플레이어 회전 의존성이 있습니다)

Camera Mode F2 (Orbital Camera)



Camera Mode 3 (3 인칭 카메라)



Camera Mode 4 (위치 고정된 카메라)