DirectX12 구현 내용 설명

조작법

헬기 이동

- 마우스 클릭 후 조이스틱처럼 사용
- 좌클릭 : y 축 x 축 회전
- 우클릭 : z 축 회전

미사일 발사

- z : 화면의 중앙의 물체를 표적으로 설정하고 유도탄 발사
- x : 뷰 프러스텀에 있는 적 오브젝트 중 가장 가까운 물체를 목표로 하고 유도탄 발사

구현 내용

플레이어

- 이동
  - 헬기의 y 축의 기울기에 따라 헬기가 오르락내리락하도록 만들었습니다.
  - 미사일에 적중되면 깜박거리며 이동이 느려지게 만들었습니다.
- 유도탄 2 종
  - 뷰프러스텀에 있는 가장 가까운 적 오브젝트를 표적으로 따라가도록 만들었습니다.
  - 플레이어가 바라보고 있는 객체를 표적으로 해 따라가도록 만들었습니다.

적 오브젝트

- 이동
  - 이동하다 플레이어를 발견하면 플레이어를 바라보도록 만들었습니다.
- 유도탄
  - 플레이어를 발견하면 플레이어의 방향으로 유도탄을 발사합니다.
  - 미사일에 적중되면 파티클이 나옵니다.
  - 유도탄은 지형을 피해가도록 만들었습니다.

구현 내용 설명

플레이어 이동

- 목표 -> 실제 헬기가 이동하듯이 만들어 보자
- 구현

  y 축과 플레이어 up 벡터의 각도를 구해 기울기를 구하고 기울어진 방향을 이동 방향으로 사용하였고 기울기 각도를 중력 벡터에 곱해주어 기울기가 작으면 중력도 같이 작아지고 기울기가 크면 중력 벡터도 같이 커지도록 만들었습니다.

유도탄

- 목표 1 -> 내가 보고 있는 화면에 오브젝트가 있으면 가장 가까운 적이 표적으로 하자
- 목표 2 -> Lock-on 방식으로 표적을 지정해보자

- 목표 1 구현

  x 버튼을 누르면 뷰 프러스텀에 있는 오브젝트들 중 플레이어와 가장 가까운 거리에 있는 오브젝트를 표적으로 정해 위치를 받아와 플레이어의 위치벡터에서 빼 표적을 향하는 타겟 벡터를 만들고 미사일 오브젝트의 Look 벡터와 외적을 해 회전축을 구한 다음 그 회전축 각 성분을 이용해 회전했습니다. 또 미사일은 항상 Look 벡터 방향으로 가도록 해 회전하는 방향으로 이동하도록 만들어 주었습니다. 만약 표적이 없다면 플레이어가 보는 방향으로 미사일이 발사됩니다.

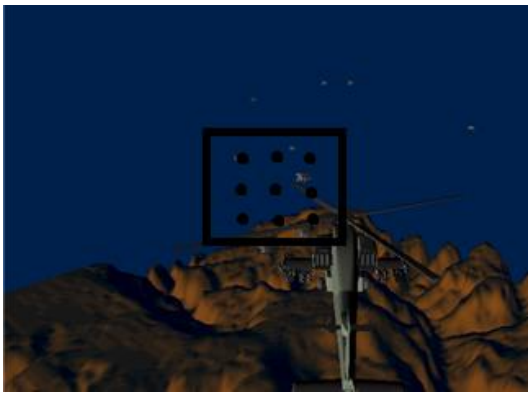
```
if (m_pTarget) {

    XMFLOAT3 xmf3Target = m_pTarget->GetPosition();
    XMFLOAT3 xmf3ToTarget = Vector3::Subtract(xmf3Target, xmf3Position);
    XMFLOAT3 xmf3Look = GetLook();

    xmf3ToTarget = Vector3::Normalize(xmf3ToTarget);

    XMFLOAT3 xmf3CrossProduct = Vector3::CrossProduct(xmf3Look, xmf3ToTarget);


    xmf3CrossProduct.x *= (xmf3Look.z > 0.0f) ? 1.0f : -1.0f;

    //각 x,y,z 값 만큼 pitch, yaw, roll 회전
    Rotate(xmf3CrossProduct.x, xmf3CrossProduct.y, xmf3CrossProduct.z);
    MoveForward(fDistance);

}
else {
    MoveForward(fDistance);
}
```

미사일이 타겟을 바라보며 날아가는 코드 Object.cpp

- 목표 2 실행결과

  화면의 중앙을 기준으로 25 개의 광선을 쏩니다.

  z 버튼을 누르면 화면 중앙을 기준으로 25 개의 광선을 쏘고 해당 광선에 맞는 적 중 가장 번호가 빠른 오브젝트를 표적으로 정합니다. 표적이 된 적의 위치로 조명을 옮겨 주어 표적으로 설정된 것을 알 수 있도록 하였습니다. 만약 표적이 없다면 플레이어가 보는 방향으로 미사일이 발사됩니다.



```cpp
void CGameObject::GenerateRayForPicking(XMVECTOR& xmvPickPosition, XMMATRIX& xmmtxView, XMVECTOR& xmvPickRayOrigin, XMVECTOR& xmvPickRayDirection)
{
    XMMATRIX xmmtxToModel = XMMatrixInverse(NULL, XMLoadFloat4x4(&m_xmf4x4World) * xmmtxView);

    XMFLOAT3 xmf3CameraOrigin(0.0f, 0.0f, 0.0f);
    xmvPickRayOrigin = XMVector3TransformCoord(XMLoadFloat3(&xmf3CameraOrigin), xmmtxToModel);
    xmvPickRayDirection = XMVector3TransformCoord(xmvPickPosition, xmmtxToModel);
    xmvPickRayDirection = XMVector3Normalize(xmvPickRayDirection - xmvPickRayOrigin);
}

bool CGameObject::PickObjectByRayIntersection(XMVECTOR& xmvPickPosition, XMMATRIX& xmmtxView, float* pfHitDistance)
{
    bool nIntersected = false;

    XMVECTOR xmvPickRayOrigin, xmvPickRayDirection;
    GenerateRayForPicking(xmvPickPosition, xmmtxView, xmvPickRayOrigin, xmvPickRayDirection);
    nIntersected = m_xmOOBB.Intersects(xmvPickRayOrigin, xmvPickRayDirection, *pfHitDistance);

    return(nIntersected);
}
```

화면의 점에서 선을 만들어 충돌 체크를 하는 코드 Object.cpp

행렬 M 이 변환 T 를 나타내는 행렬일 때 행렬 M 의 역행렬은 변환 T 의 역변환을 나타내는 행렬입니다. 이를 이용하여 화면에서 찍어준 점 A 를 월드 좌표계 상에서의 광선으로 생성하려면 월드 변환과 뷰 변환이 적용 되어있는 행렬의 역행렬을 구해 A 에 곱해주면 화면 좌표계의 위치에서 월드 좌표계의 위치로 바뀌게 됩니다. 이 위치를 카메라의 원점과 뺄셈을 해주어 카메라 원점에서 점 A 까지 가는 선을 하나 구해주고 그 선과 적 오브젝트들의 바운딩 박스와 충돌이 되는지 검사합니다.

적 오브젝트

- 목표 -> 플레이어가 가까이 다가오면 플레이어를 바라보며 유도 미사일을 발사하자
- 실행결과

XMMatrixLookAtLH 함수를 이용하여 표적을 바라보는 뷰 행렬을 만들어 주고 해당 행렬을 전치한 뒤 선형변환 부분을 월드 행렬에 적용해 적 오브젝트가 플레이어를 바라볼 수 있도록 하였습니다. 그 다음 오브젝트의 Right 방향으로 이동을 해주어 적 오브젝트가 플레이어를 중심으로 회전하면서 유도 미사일을

발사하도록 하였습니다. 아래의 코드는 뷰 행렬을 만들고 전치해 적 오브젝트의 행렬에 적용하는 코드입니다.

```
XMFLOAT3 xmf3Target = pPlayer->GetPosition();
XMFLOAT3 xmf3Mpos = GetPosition();
XMFLOAT3 xmf3ToTarget = Vector3::Subtract(xmf3Target, GetPosition());

if (Vector3::Length(xmf3ToTarget) < 500.f) {

    XMFLOAT3 xmf3WorldUp{ 0.0f,1.0f,0.0f };

    // 플레이어를 바라보는 행렬 생성
    XMFLOAT4X4 xmf4x4View = Matrix4x4::LookAtLH(xmf3Mpos, xmf3Target, xmf3WorldUp);

    //XMMatrixLookAtLH의 결과는 행 우선 표기법을 사용한다.
    //m_xmf4x4Transform는 열 우선 표기법을 사용하기 때문에 전치한 뒤 사용해야한다.
    XMStoreFloat4x4(&m_xmf4x4Transform,DirectX::XMMatrixTranspose(DirectX::XMMatrixLookAtLH(XMLoadFloat3(&xmf3Mpos), XMLoadFloat3(&xmf3Target), XMLoadFloat3(&xmf3WorldUp))));

    m_xmf4x4Transform._41 = xmf3Mpos.x; m_xmf4x4Transform._42 = xmf3Mpos.y; m_xmf4x4Transform._43 = xmf3Mpos.z;

    SetScale(2.f, 2.f, 2.f);
```

플레이어를 바라보는 적 오브젝트 Object.cpp

적 오브젝트 파괴 효과

- 목표 -> 미사일에 충돌하면 파티클이 나오게 하자
- 실행결과

  파티클로 사용할 큐브 매쉬의 정점 버퍼와 인덱스 버퍼와 노말 버퍼를 만들어 둡니다. 그 다음 미사일과 적 오브젝트가 충돌하면 기존에 렌더링하던 물체 말고 미리 만들어 준 큐브 매쉬와 재질을 이용해 파티클을 출력합니다. 박스 메쉬의 정점의 노말 벡터는 3 개의 평면의 평균이 되도록 지정해주었습니다.

실행화면

미사일에 적중된 적의 파티클

플레이어를 확인하고 유도탄을 발사하는 적

광선에 충돌 후 표적이 되어 조명이 들어간 오브젝트