

# 소개하기 앞서,

---

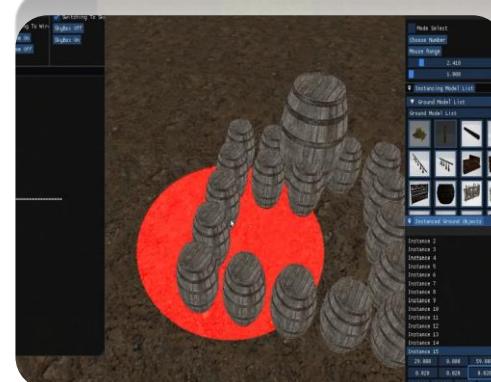
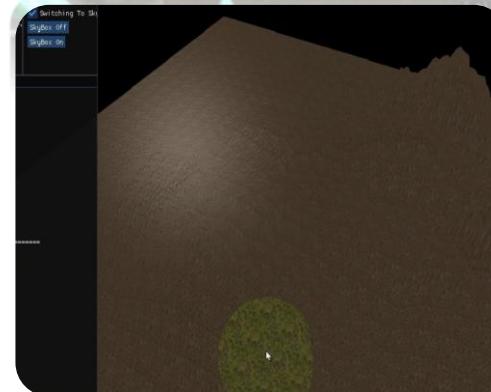
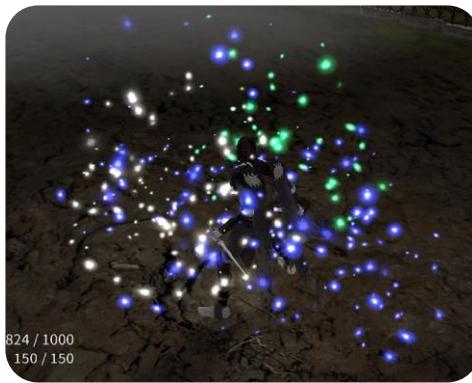
본 기술 문서에서는 제가 가장 많은 기여를 한 **THYMESIA** 프로젝트를 중심으로 설명하며,  
기타 프로젝트들은 보조적인 포트폴리오 소개 형식으로 간단히 정리했습니다.

---

GitHub : <https://github.com/HanUIn123>

Notion : [프로젝트 시연 영상 모음](#)

# THYMEΣΙΑ



제작 기간

2025.02 ~ 2025.04

개발 환경

DirectX11 C++ Windows API

개발 인원

6인

담당 파트  
&  
구현 내용

- [맵 툴]
- [네비게이션 시스템]
- [게임 내 쉐이더 연출]
- [아이템 시스템]
- [상호작용 시스템]
- [인스턴싱\_최적화]

영상 링크

<https://youtu.be/T3UJbHMiLYw>

## 1. Navigation System

- 1-1 픽셀 피킹
- 1-2 피킹한 점 처리
- 1-3 셀들의 인접, 경계 판정
- 1-4 오브젝트 이동체크

## 2. Geometry Shader 연출

- 2-1 포자 움직임 연출
- 2-2 오브젝트 파괴 연출
- 2-3 돌 파편 복제 연출

## 3. Item System

- 3-1 Item Manager 관리 구조
- 3-2 아이템 드롭 연출
- 3-3 아이템 렌더링 연출
- 3-4 맵 전환 시, 아이템 정보 처리

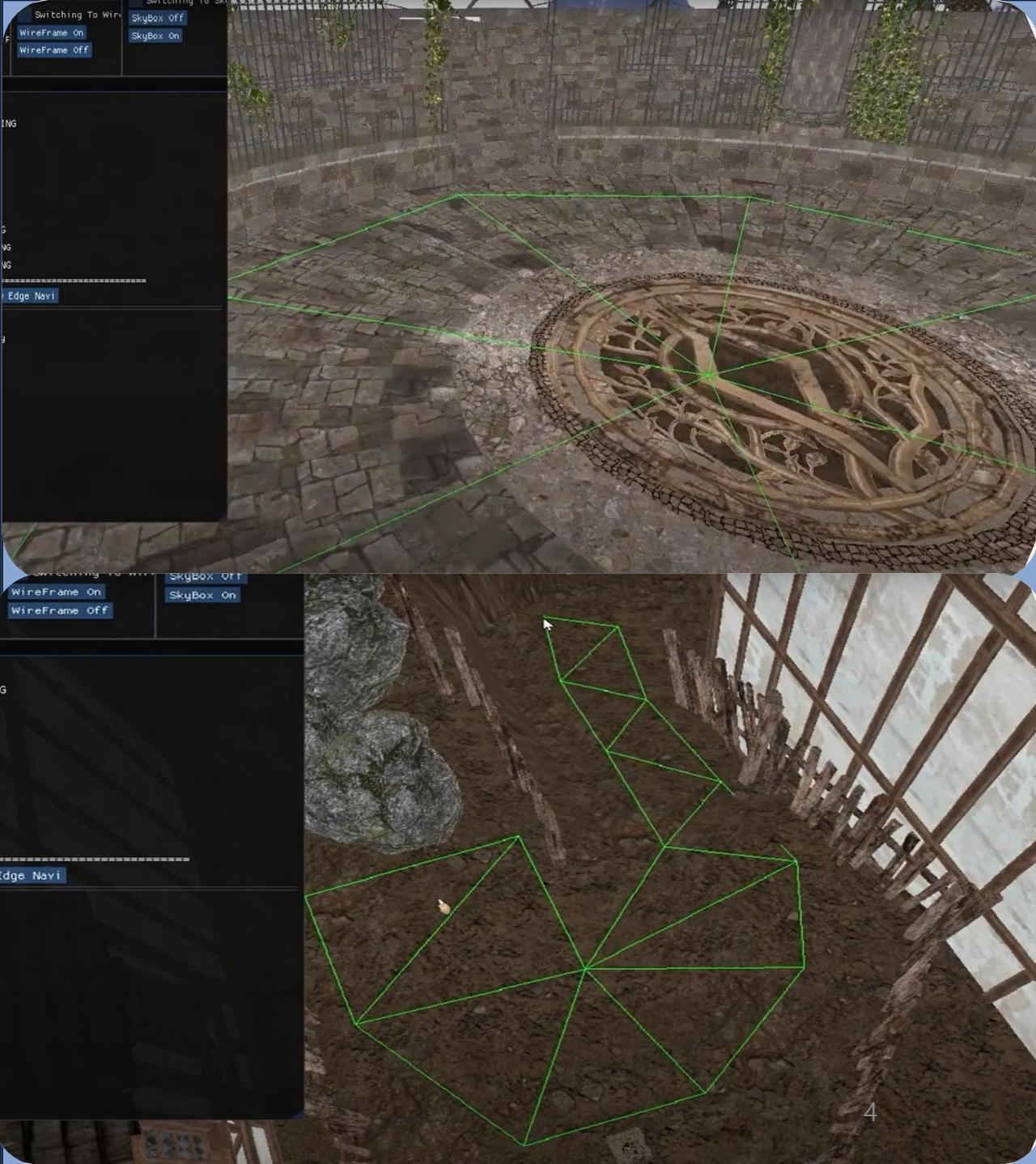
## 4. 상호작용 System

- 4-1 림라이트 셰이더 효과
- 4-2 락온 타깃 라인 표시
- 4-3 상호작용 UI 버튼

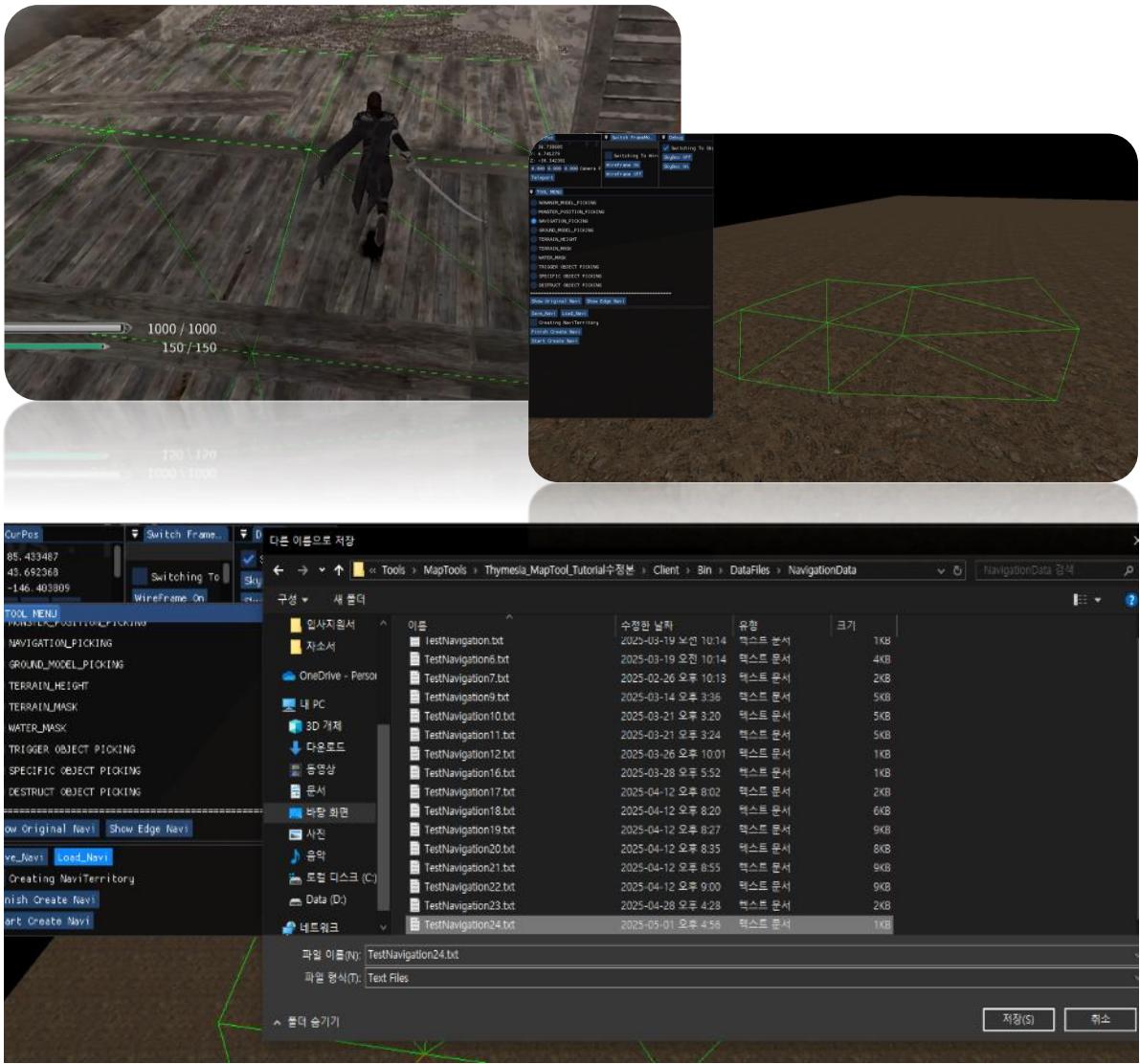
## 5. 메쉬 인스턴싱

- 5-1 인스턴싱 구조 설계
- 5-2 인스턴싱 구동 흐름
- 5-3 인스턴싱 절두체 컬링

Part 1 Navigation 시스템



# THYMESIA → Navigation System 작동 원리



각각, 인 게임 속 Navigation 영역이 설정된 모습, 그리고 툴 상에서, 지형에 Navigation 영역을 지정한 그림입니다.  
Navigation 시스템은 다음과 같은 과정으로 작동합니다.

맵 툴

[1] 마우스 피킹 입력  
(픽셀 피킹 방식)

[2] 클릭된 위치 저장 및  
셀 생성

[3] 설정한 네비게이션  
영역 데이터파일로  
save

인 게임

[1] 네비게이션 데이터  
파일 Load

[2] 생성된 셀 간 인접  
관계 및 경계 판정

[3] 오브젝트 이동 시, 셀  
판정 및 이동 가능 여부  
체크

# THYMEΣIA → Navigation System → 픽셀 피킹

```

m_pContext->Map(m_pTexture2D, 0, D3D11_MAP_READ_WRITE, 0, &SubResource);

_uint iPixelIndex = (_uint)(ptMouse.y * m_fViewportWidth + ptMouse.x);

if ((m_fViewportHeight * m_fViewportWidth) < iPixelIndex || 0.f > iPixelIndex)
    return false;

_float4 vDepthDesc = static_cast<_float4*>(SubResource.pData)[iPixelIndex];

m_pContext->Unmap(m_pTexture2D, 0);

_vector vWorldPos = XMVectorSet((float)ptMouse.x, (float)ptMouse.y, 0.f, 0.f);

vWorldPos = XMVectorSetX(vWorldPos, ptMouse.x / (m_fViewportWidth * 0.5f) - 1.f);
vWorldPos = XMVectorSetY(vWorldPos, ptMouse.y / (m_fViewportHeight * -0.5f) + 1.f);
vWorldPos = XMVectorSetZ(vWorldPos, vDepthDesc.x);
vWorldPos = XMVectorSetW(vWorldPos, 1.f);

vWorldPos = XMVector3TransformCoord(vWorldPos,
    m_pGameInstance->Get_Transform_Matrix_Inverse(CPipeline::D3DTS_PROJ));

vWorldPos = XMVector3TransformCoord(vWorldPos,
    m_pGameInstance->Get_Transform_Matrix_Inverse(CPipeline::D3DTS_VIEW));

XMStoreFloat3(_pOut, vWorldPos);

```

```

struct PS_OUT
{
    float4 vDiffuse : SV_TARGET0;
    float4 vNormal : SV_TARGET1;
    float4 vDepth : SV_TARGET2;
    float fSpecular : SV_TARGET3;
    float4 vPickDepth : SV_TARGET4;
};

```

```

PS_OUT PS_MAIN(PS_IN In)
{
    PS_OUT Out = (PS_OUT) 0;
    Out.vPickDepth = vector(In.vProjPos.z /
        In.vProjPos.w, In.vProjPos.w, 1.f, 0.f);
    return Out;
}

```

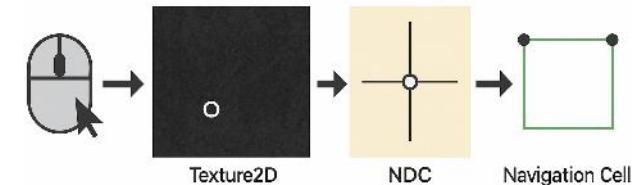
마우스 클릭 시, 픽셀 좌표를 기반으로 렌더 타겟(Texture2D)에서 해당 픽셀의 PickDepth 값을 읽어옵니다.

쉐이더에서 출력한 픽셀의 깊이(z/w) 및 w 값을 활용하여 정확한 NDC 좌표(-1에서 1 사이)를 계산합니다.

계산된 좌표는 Projection, View 역변환 행렬을 통해 3D 월드 좌표로 복원됩니다.

복원된 월드 좌표는 Navigation Cell의 꼭짓점(Vertex)으로 사용됩니다.

즉, 툴에서 클릭한 위치가 실제 게임 내 이동 가능 영역과 정확히 일치하도록 구성됩니다.



# THYMESIA → Navigation System → 피킹한 위치의 점 처리

```

HRESULT CLevel_GamePlay::Picking_Points()
{
    float3 vPickPos;
    if (_m_pGameInstance->Compute_PickPos(&vPickPos))
    {
        m_fWholePickPos = vPickPos;

        if (m_fWholePickPos.y == -0.5f)
            return S_OK;

        cout << "X : " << " " << m_fWholePickPos.x << endl;
        cout << "Y : " << " " << m_fWholePickPos.y << endl;
        cout << "Z : " << " " << m_fWholePickPos.z << endl;

        m_fWholePickPos += 0.07f;
    }
}

```

피킹한 위치를 월드 좌표로 복원하고,  
y좌표를 보정 후 실제 위치로 설정합니다.

```

if (_m_mapFloorPickedPoints[iFloorNumber].size() == 3 && m_bFirstPick && !m_bConnectingMode)
{
    m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[0] = _m_mapFloorPickedPoints[iFloorNumber][0];
    m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[1] = _m_mapFloorPickedPoints[iFloorNumber][1];
    m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[2] = _m_mapFloorPickedPoints[iFloorNumber][2];

    m_navigation->Create_Cell(m_mapTagWholeCellPoints[iFloorNumber].fCellPoints);
    m_mapWholeCellPoints[iFloorNumber].push_back(m_mapTagWholeCellPoints[iFloorNumber]);

    m_mapTagWholeCellPoints[iFloorNumber].fPrevPoints[0] = m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[0];
    m_mapTagWholeCellPoints[iFloorNumber].fPrevPoints[1] = m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[1];

    m_iNumCellCount++;
    m_bFirstPick = false;
    m_mapFloorPickedPoints[iFloorNumber].clear();
}

else if (_m_mapFloorPickedPoints[iFloorNumber].size() < 3 && !m_bFirstPick && !m_bConnectingMode)
{
    auto NearPoints = Compute_NearPoints(m_mapWholeCellPoints[iFloorNumber], _m_mapFloorPickedPoints[iFloorNumber][0]);
    m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[0] = NearPoints.first;
    m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[1] = NearPoints.second;
    m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[2] = _m_mapFloorPickedPoints[iFloorNumber][0];

    XMVECTOR vNewCellPoint1 = XMLoadFloat3(&m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[0]);
    XMVECTOR vNewCellPoint2 = XMLoadFloat3(&m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[1]);
    XMVECTOR vNewCellPoint3 = XMLoadFloat3(&m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[2]);

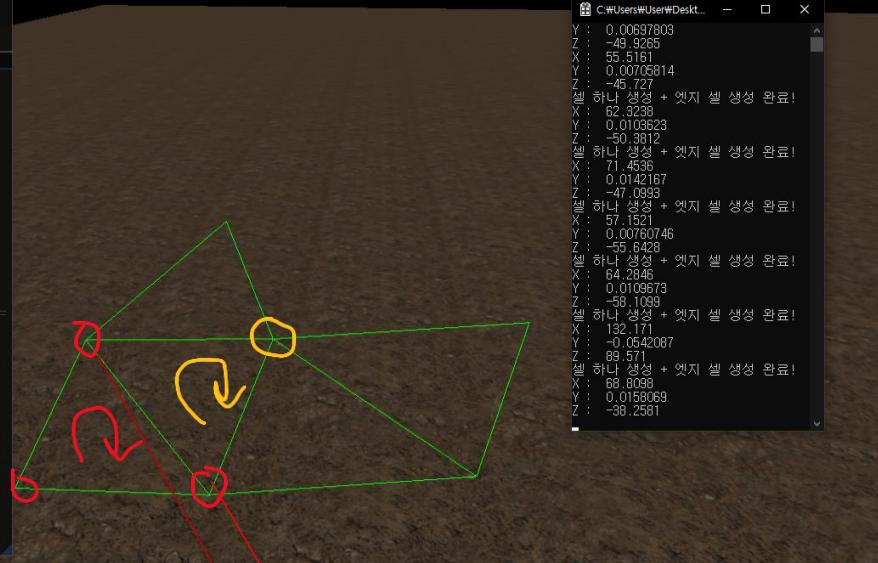
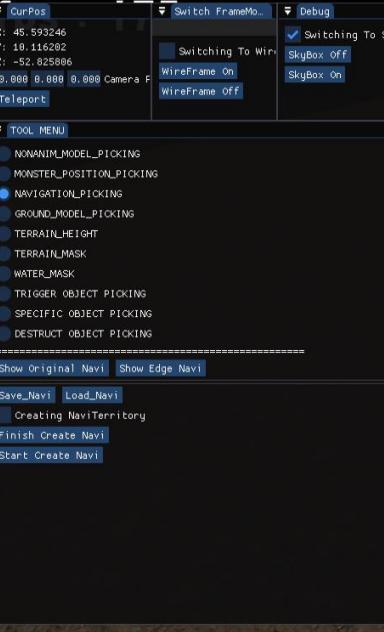
    if (!Is_CWTriangle(vNewCellPoint1, vNewCellPoint2, vNewCellPoint3))
    {
        swap(m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[1], m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[2]);

        m_navigation->Create_Cell(m_mapTagWholeCellPoints[iFloorNumber].fCellPoints);
        m_mapWholeCellPoints[iFloorNumber].push_back(m_mapTagWholeCellPoints[iFloorNumber]);

        m_mapTagWholeCellPoints[iFloorNumber].fPrevPoints[0] = m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[1];
        m_mapTagWholeCellPoints[iFloorNumber].fPrevPoints[1] = m_mapTagWholeCellPoints[iFloorNumber].fCellPoints[2];
    }

    m_iNumCellCount++;
    m_mapFloorPickedPoints[iFloorNumber].clear();
}

```



- ▶ 툴을 이용하는 사용자가 직접 3개의 점을 피킹하여, 삼각형을 구성합니다.
- ▶ Create\_Cell() 함수로 셀을 생성하고, 이전에 피킹한 점의 정보도 저장하여, 다음 셀을 만들 때 자동으로 삼각형이 만들어 질 수 있도록 활용했습니다.
- ▶ 기존 셀에서 가까운 두 점을 찾아 자동으로 셀을 구성할 수 있게 했습니다.
- ▶ 삼각형(Cell)을 구성할 때는 반드시 정점이 시계방향으로 배치되도록 처리했습니다.
- ▶ 반시계방향으로 정점이 배치된 경우에는 자동으로 두 점을 swap하여 시계방향으로 교정한 후 셀이 생성되도록 했습니다.

# THYMESIA → Navigation System → 셀 간 인접, 경계 판정

```

for (auto& pSourCell : m_Cells)
{
    for (auto& pDestCell : m_Cells)
    {
        if (pSourCell == pDestCell)
            continue;

        if (true == pDestCell->Compare_Points(pSourCell->Get_Point(CCell::POINT_A),
                                                pSourCell->Get_Point(CCell::POINT_B)))
            pSourCell->Set_Neighbor(CCell::LINE_AB, pDestCell);

        if (true == pDestCell->Compare_Points(pSourCell->Get_Point(CCell::POINT_B),
                                                pSourCell->Get_Point(CCell::POINT_C)))
            pSourCell->Set_Neighbor(CCell::LINE_BC, pDestCell);

        if (true == pDestCell->Compare_Points(pSourCell->Get_Point(CCell::POINT_C),
                                                pSourCell->Get_Point(CCell::POINT_A)))
            pSourCell->Set_Neighbor(CCell::LINE_CA, pDestCell);
    }
}

```

```

for (auto& pCell : m_Cells)
{
    for (int i = 0; i < 3; ++i)
    {
        if (pCell->Get_NeighborIndex(CCell::LINE)i) != -1)
            continue;

        uint iNext = (i + 1) % 3;

        XMVector vDir = XMVectorSetW(pCell->Get_Point(CCell::POINT)iNext) - pCell->Get_Point(CCell::POINT)i, 0.f);
        XMStoreFloat4(&m_Edge_Cell_Line.EDGE_1) + i, vDir);

        float3 v0, v1;
        XMStoreFloat3(&v0, pCell->Get_Point(CCell::POINT)i));
        XMStoreFloat3(&v1, pCell->Get_Point(CCell::POINT)iNext));

        float3 line[2] = { v0, v1 };

        CEdge_Cell* pEdgeCell = CEdge_Cell::Create(m_pDevice, m_pContext, line);
        if (!pEdgeCell)
            return E_FAIL;

        m_RenderEdgeLine.push_back(pEdgeCell);
    }

    m_Edge_Cell_Line.NeighborIndex[0] = pCell->Get_NeighborIndex(CCell::LINE_AB);
    m_Edge_Cell_Line.NeighborIndex[1] = pCell->Get_NeighborIndex(CCell::LINE_BC);
    m_Edge_Cell_Line.NeighborIndex[2] = pCell->Get_NeighborIndex(CCell::LINE_CA);

    m_EdgeCells.push_back(m_Edge_Cell_Line);
    ZeroMemory(&m_Edge_Cell_Line, sizeof(EDGE_CELL_LINE));
}

cout << "셀 하나 생성 + 엣지 셀 생성 완료!" << endl;

```

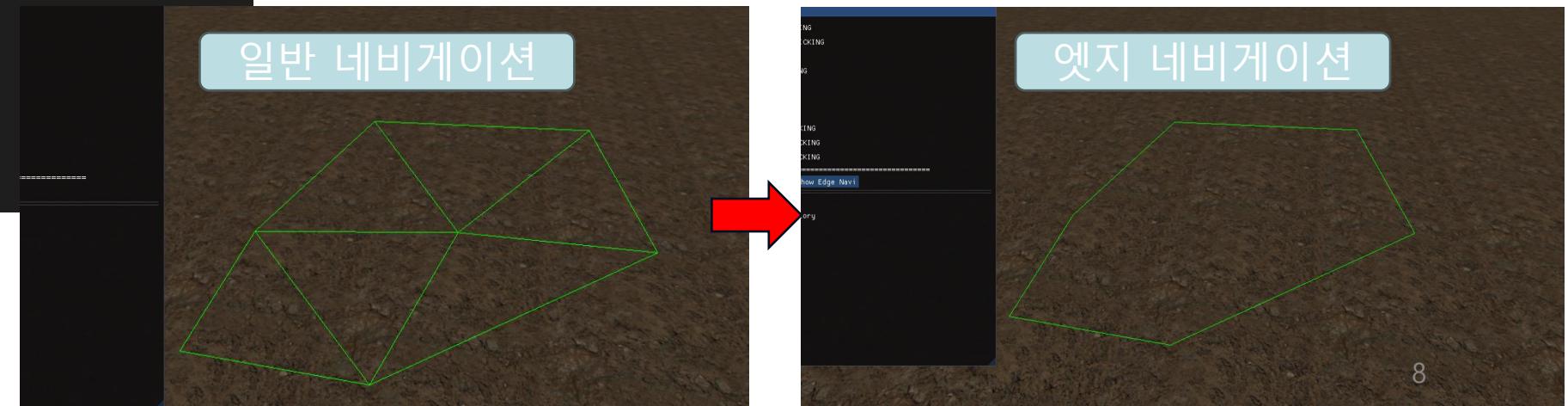
전체 셀을 이중 반복문으로 비교하여, 각 셀의 AB, BC, CA 세 라인을 기준으로 같은 라인을 공유하는 셀을 이웃(Neighbor)으로 설정합니다.

정점의 순서까지 일치해야 인접 셀로 판단되므로, 셀은 반드시 시계방향으로 구성되어야 정상적으로 이웃 관계가 형성됩니다.

한 셀의 특정 라인에 이웃 셀이 존재하지 않는 경우, 해당 라인은 외곽 경계선(Edge Line)으로 판단합니다.

이 경우 라인의 두 점 정보를 기반으로 CEdge\_Cell 객체를 생성하고, 렌더링 대상 목록(m\_RenderEdgeLine)에 등록하여 시각화에 활용합니다.

동시에, 이웃 관계 정보(NeighborIndex)를 m\_EdgeCells에 저장하여 추후 탐색 및 처리에 사용합니다.



# THYMESIA → Navigation System → 오브젝트 이동 체크

```

for (size_t i = 0; i < LINE_END; i++)
{
    _vector vDir = XMVector3Normalize(vPosition - XMLoadFloat3(&m_vPoints[i]));
    // 시계방향 ( 0이 맨 위 삼각형 꼭지점 기준)
    _vector vLine = XMLoadFloat3(&m_vPoints[(i + 1) % 3]) - XMLoadFloat3(&m_vPoints[i]);
    _vector vNormal = XMVector3Normalize(XMVectorSet(
        XMVectorGetZ(vLine) * -1.f, 0.f, XMVectorGetX(vLine), 0.f));
    // 내적의 값이 0보다 작으면 내부 0보다 크면 외부
    if (XMVectorGetX(XMVector3Dot(vDir, vNormal)) > 0)
    {
        // i = 0 일 때 직선AB가 해당 플레이어의 방향벡터 와 내적해서 0보다 작으면 이웃
        *pNeighborIndex = m_iNeighborIndices[i];
        return false;
    }
}
return true;
// ax+ by + cz + d =0;
// y = (-ax - cz - d ) / b

return (-m_vPlane.x * vPosition.m128_f32[0] -
    m_vPlane.z * vPosition.m128_f32[2] - m_vPlane.w) / m_vPlane.y;

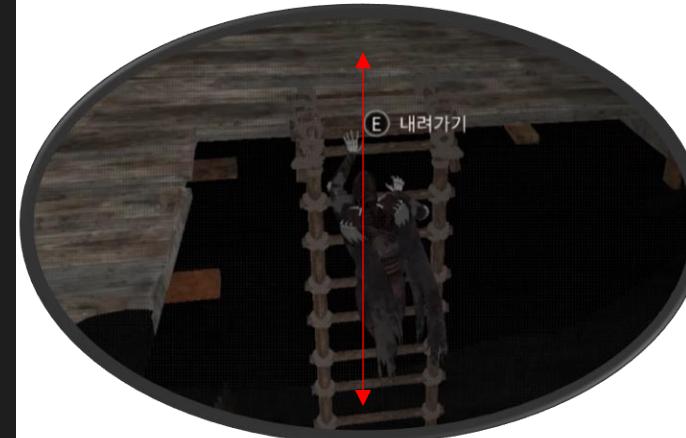
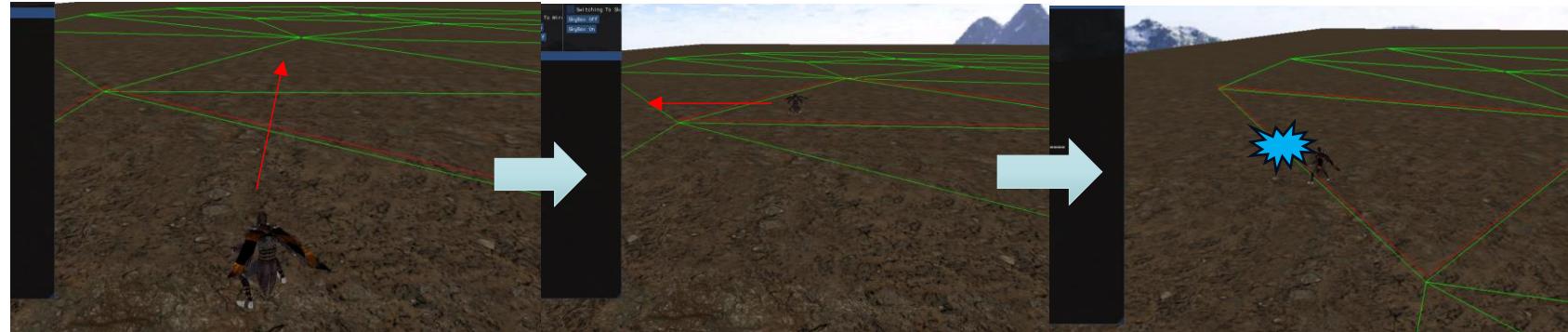
uint CNavigation::Find_Closest_Cell(_vector _vWorldPos)
{
    float fMinDistance = FLT_MAX;
    uint iClosestCellIndex = 0;
    float fPosX = XMVectorGetX(_vWorldPos);
    float fPosY = XMVectorGetY(_vWorldPos);
    float fPosZ = XMVectorGetZ(_vWorldPos);

    for (uint i = 0; i < m_Cells.size(); ++i)
    {
        XMFLOAT3 vCellCenter = m_Cells[i]->Get_Center();
        float fDistance = sqrt(
            powf(fPosX - vCellCenter.x, 2) +
            powf(fPosY - vCellCenter.y, 2) +
            powf(fPosZ - vCellCenter.z, 2));
    }
    if (fDistance < fMinDistance)
    {
        fMinDistance = fDistance;
        iClosestCellIndex = i;
    }
}
return iClosestCellIndex;
}

```

셀의 각 라인(AB, BC, CA)을 기준으로 외적을 구해 방향을 판별하고, 법선 방향의 내적 결과가 양수일 경우 셀 외부로 판단합니다.

평면 방정식을 이용해, 만들어진 현재 위치해 있는 셀의 y값을 플레이어의 y값으로 사용하여, 최종적으로 네비게이션 영역 내부만을 움직이게끔 합니다.



층간 이동의 경우, 플레이어의 현재 월드 좌표와 가장 근접한 셀의 인덱스를 찾아냅니다.

플레이어의 이동 로직이 완료되면, 해당 셀 인덱스의 y값을 플레이어의 y값으로 사용하여, 층간 이동을 구현했습니다.

Part 2

## Geometry 쉐이더 연출



# THYMESIA → GS기반 연출효과 → 포자 애니메이션 연출



```

float3 vMoveDir = float3(0, 1, 0);
switch (iPartIndex)
{
    case 0:
        vMoveDir = float3(-1, -1, -1); // 대각 1
        break;
    case 1:
        vMoveDir = float3(-1, -1, 0); // 옆 1
        break;
    case 2:
        vMoveDir = float3(1, -1, 0); // 옆 2
        break;
    case 3:
        vMoveDir = float3(1, -1, -1); // 대각 2
        break;
    case 4:
        vMoveDir = float3(-1, 1, 1); // 대각 3
        break;
    case 5:
        vMoveDir = float3(-1, 1, 0); // 옆 3
        break;
    case 6:
        vMoveDir = float3(1, 1, 0); // 옆 4
        break;
    case 7:
        vMoveDir = float3(1, 1, 1); // 대각 4
        break;
}
vMoveDir = normalize(vMoveDir);

```

// 로드리게스 회전 행렬  
<https://mathworld.wolfram.com/RodriguesRotationFormula.html>

```

float fCosAngle = cos(fAngle);
float fSinAngle = sin(fAngle);

float fRotationX = vRotationAxis.x;
float fRotationY = vRotationAxis.y;
float fRotationZ = vRotationAxis.z;

float3x3 matRotation =
[
    fCosAngle + ((fRotationX * fRotationX) * (1 - fCosAngle)),
    ((1 - fCosAngle) * fRotationX * fRotationY) - (fSinAngle * fRotationZ),
    ((1 - fCosAngle) * fRotationX * fRotationZ) + (fSinAngle * fRotationY),

    ((1 - fCosAngle) * fRotationX * fRotationY) + (fSinAngle * fRotationZ),
    fCosAngle + (fRotationY * fRotationY * (1 - fCosAngle)),
    (-fRotationX * fSinAngle) + (fRotationY * fRotationZ * (1 - fCosAngle)),

    ((1 - fCosAngle) * fRotationX * fRotationZ) - (fRotationY * fSinAngle),
    (fRotationX * fSinAngle) + (fRotationY * fRotationZ * (1 - fCosAngle)),
    fCosAngle + ((fRotationZ * fRotationZ) * (1 - fCosAngle))
];

```

```

float3 vWorldPos = input[i].vWorldPos.xyz;
float3 vLocalOffset = vWorldPos - g_ModelPosition.xyz;

float fFarOffsetValue = (vLocalOffset.x + vLocalOffset.z) * 2.0f;
float fWiggleValue = sin(fWiggleTime * 10.0f + fFarOffsetValue) * 0.02f;

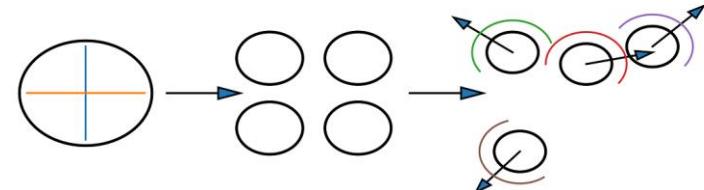
vLocalOffset += input[i].vNormal.xyz * fWiggleValue;

float3 vFinalPos = g_ModelPosition.xyz + vLocalOffset;

Out.vWorldPos = float4(vFinalPos, 1);
Out.vPosition = mul(Out.vWorldPos, mul(g_ViewMatrix, g_ProjMatrix));

```

포자의 Idle 상태에서는 시간에 따라 사인 파동을 생성해, 정점을 노멀 방향으로 미세하게 흔들어 주어 꿈틀거리는 효과를 연출합니다.



피격 시에는 오브젝트를 8방향으로 분할하고, 각 조각에 서로 다른 회전축과 이동 방향을 적용합니다. 여기에 로드리게스 회전 행렬을 사용하여, 파편이 날아갈 때 보다 극적인 연출을 구현하였습니다.

# THYMESEA → GS기반 연출효과 → 수정 파편화 연출



```
float3 HashDirection(float3 seed)
{
    float hashX = frac(sin(dot(seed, float3(12.9898, 78.233, 45.543))) * 43758.5453);
    float hashY = frac(sin(dot(seed, float3(33.989, 67.345, 12.345))) * 12458.372);
    float hashZ = frac(sin(dot(seed, float3(45.123, 12.345, 78.234))) * 32513.6521);

    return normalize(float3(hashX * 2 - 1, hashY * 2 - 1, hashZ * 2 - 1));
}
```

수정 파편들이 항상 동일한 방향으로 이동하도록 월드 좌표를 기반으로 해시 난수 벡터를 생성했습니다.

dot → sin → frac 연산에 임의의 상수를 직접 지정하여 GPU 환경에 관계없이 동일한 결과를 보장하고, 주기적인 패턴이 나타나는 것을 방지했습니다.

월드 좌표를 기반으로 고유한 난수 방향 벡터를 생성하여, 같은 파편은 항상 동일한 방향으로 이동하도록 일관성을 유지합니다.

```
[maxvertexcount(3)]
void GS_MAIN_PARTICLE(triangle GS_IN_DESTRUCT input[3], inout TriangleStream<GS_OUT_DESTRUCT> triStream)
{
    float3 vMoveDir = HashDirection(input[0].vWorldPos.xyz);
    float fRandomPerlinNoiseSeedNum = frac(sin(dot(input[0].vWorldPos.xyz,
        float3(12.9898, 78.233, 45.164))) * 43758.5453);
    float fMoveScale = lerp(0.8f, 1.5f, fRandomPerlinNoiseSeedNum) * 2.5f;

    float3 vMoveValue = vMoveDir * (_fExplosionPower * fMoveScale);

    float fFallingTime = _fFallingTime;
    float fGravityPower = -9.8f;

    float fInitialPopVelocity = 0.45f;
    float fFallingValue = (fInitialPopVelocity * fFallingTime) +
        (0.4f) * (fGravityPower * fFallingTime * fFallingTime);
    vMoveValue.y += fFallingValue;

    float fAngle = fFallingTime * lerp(3.0f, 6.0f, fRandomPerlinNoiseSeedNum);
    float fSinAngle = sin(fAngle);
    float fCosAngle = cos(fAngle);

    float3x3 matRotation = float3x3(
        fCosAngle, -fSinAngle, 0,
        fSinAngle, fCosAngle, 0,
        0, 0, 1
    );
}
```

별도의 난수 스칼라 값을 사용하여 이동 거리, 속도, 회전 속도를 다양화합니다.

이를 방향 벡터에 곱하여 파편마다 서로 다른 세기와 회전이 적용되도록 구현합니다.

# THYMESIA → GS기반 연출효과 → 돌 파편 복제 연출



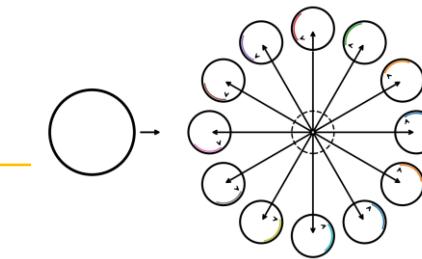
```
#define PARTS_COUNT 12
[maxvertexcount(3 * PARTS_COUNT)]
void GS_MAIN_MAGNITUDE(triangle GS_IN_DESTRUCT input[3],
[inout TriangleStream<GS_OUT_DESTRUCT> triStream])
{
    for (int iPartIndex = 0; iPartIndex < PARTS_COUNT; ++iPartIndex)
    {
        float fAngle = (2.0f * 3.141592f) * (iPartIndex / (float) PARTS_COUNT);
        float3 vMoveDir = normalize(float3(cos(fAngle), 1.0f, sin(fAngle)));
        float3 vMoveValue = vMoveDir * (g_fExplosionPower * 1.5f);
        float fFallingTime = g_fFallingTime;
        float fGravityPower = -9.8f;
        float fInitialPopVelocity = 1.2f;
        float fFallingValue = (fInitialPopVelocity * fFallingTime) +
        (0.4f * fGravityPower * fFallingTime * fFallingTime);
        vMoveValue.y += fFallingValue;
        float3 vRotationAxis = normalize(float3(cos(fAngle), 0.5f, sin(fAngle)));
        float fRotationSpeed = 30.0f;
        float fRotationAngle = g_fFallingTime * fRotationSpeed;
        float fCosAngle = cos(fRotationAngle);
        float fSinAngle = sin(fRotationAngle);

        for (int i = 0; i < 3; ++i)
        {
            GS_OUT_DESTRUCT Out = (GS_OUT_DESTRUCT) 0;
            Out.vTexCoord = input[i].vTexCoord;
            Out.vNormal = input[i].vNormal;
            Out.vTangent = input[i].vTangent;
            Out.vBinormal = input[i].vBinormal;
            float3 vLocalOffset = input[i].vWorldPos.xyz - g_ModelPosition.xyz;
            float3 vRotated = mul(vLocalOffset, matRotation);
            float3 vFinalPos = g_ModelPosition.xyz + vRotated + vMoveValue;
            Out.vWorldPos = float4(vFinalPos, 1);
            Out.vPosition = mul(Out.vWorldPos, mul(g_ViewMatrix, g_ProjMatrix));
            triStream.Append(Out);
        }
        triStream.RestartStrip();
    }
}
```

한 개의 돌을 여러 파편으로 복제해 원형으로 사방으로 퍼지도록 배치합니다.

각 파편은 퍼지는 힘·초기 상승·회전 속도가 서로 다르게 설정됩니다.

각 파편은 모델 중심을 기준으로 회전한 뒤, 상승 후, 중력의 영향을 받는 모습으로 나타나게 됩니다.



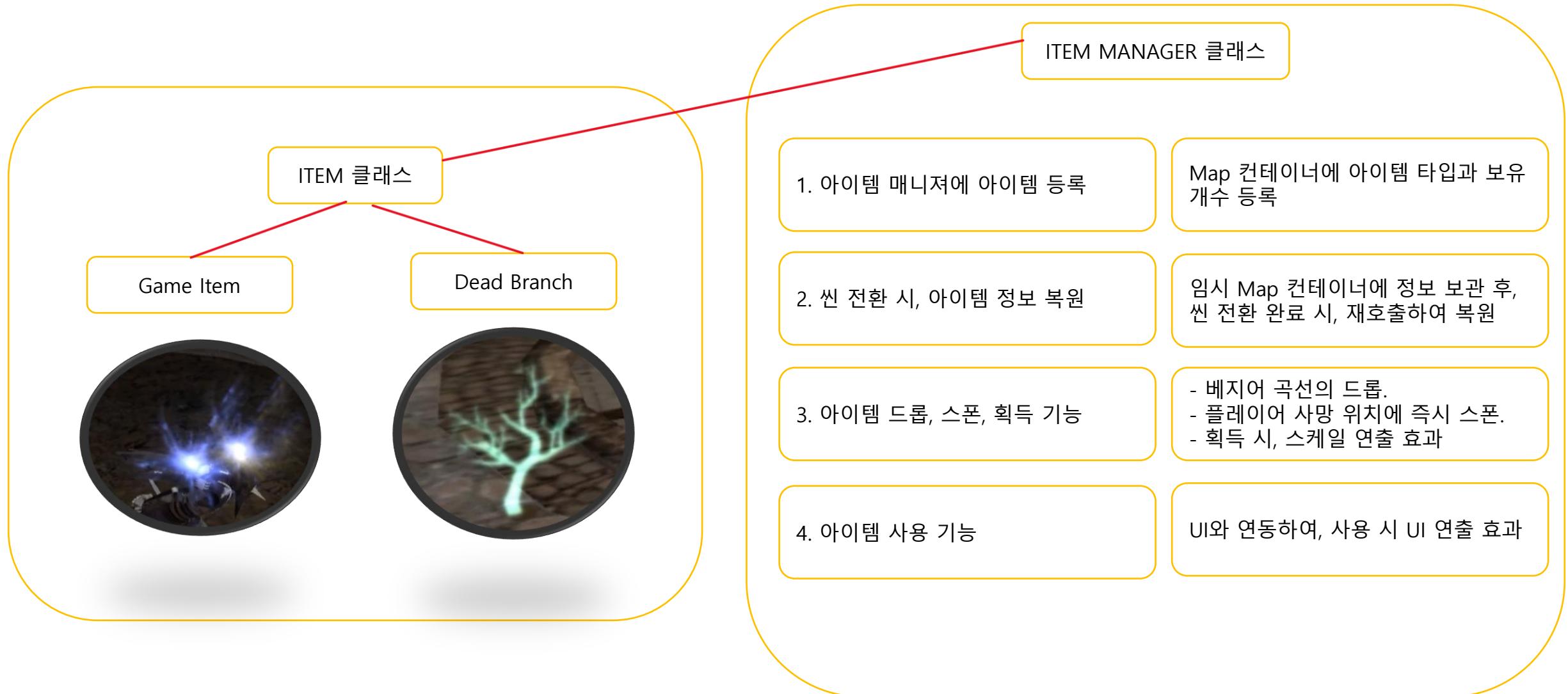
파편 하나(삼각형 하나)를 출력할 때마다 스트립을 끊어 서로 연결되지 않게 처리합니다.

Part 3

## 아이템 시스템



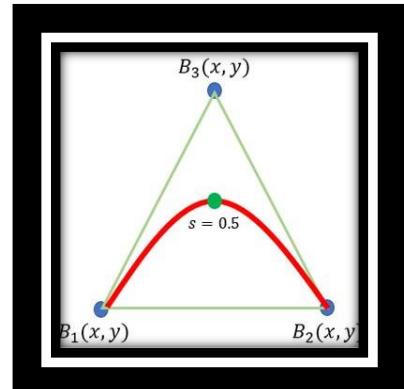
# THYMEΣIA → 아이템 시스템 → 아이템 매니저 구조 설계



# THYMESIA → 아이템 시스템 → 베지어 곡선 드롭



베지어 곡선 공식  
 $B(t) = (1-t)^2 S + 2(1-t)tC + t^2 E \quad (0 \leq t \leq 1)$



```

float4 CItem::Bezier(_float4 _vStartPos, _float4 _vCurvePos, _float4 _vEndPos, _float _fTimeDelta)
{
    _float4 vResult;

    XMStoreFloat4(&vResult, ((1 - _fTimeDelta) * (1 - _fTimeDelta) * XMLoadFloat4(&_vStartPos)
        + 2 * (1 - _fTimeDelta) * _fTimeDelta * XMLoadFloat4(&_vCurvePos)
        + _fTimeDelta * _fTimeDelta * XMLoadFloat4(&_vEndPos)));

    return vResult;
}

void CItem::Set_BezierPosition(const _float4& _vStartPos, CGameObject* _pGameObject)
{
    m_bActivate = true;

    m_vInitialPos = _vStartPos;
    m_pTransformCom->Set_State(CTransform::STATE_POSITION, XMLoadFloat4(&_vStartPos));

    m_fElapsed = 0.0f;

    _vector vTargetPos = _pGameObject->Get_Transform()->Get_State(CTransform::STATE_POSITION);
    const float fRadius = 0.8f;
    float fRandomX = (rand() % 100 / 100.f - 0.5f) * 2.0f * fRadius;
    float fRandomZ = (rand() % 100 / 100.f - 0.5f) * 2.0f * fRadius;
    vTargetPos = XMVectorSetX(vTargetPos, XMVectorGetX(vTargetPos) + fRandomX);
    vTargetPos = XMVectorSetZ(vTargetPos, XMVectorGetZ(vTargetPos) + fRandomZ);

    _vector vTargetDir = XMVector3Normalize(vTargetPos - XMLoadFloat4(&_vStartPos));
    m_pTransformCom->LookAt(XMLoadFloat4(&_vStartPos) + vTargetDir);

    vector vMiddlePoint = (XMLoadFloat4(&_vStartPos) + vTargetPos) * 0.5f;
    vMiddlePoint = XMVectorSetY(vMiddlePoint, XMVectorGetY(vMiddlePoint) + 2.0f);

    vTargetPos = XMVectorSetY(vTargetPos, XMVectorGetY(vTargetPos) + 0.5f);

    XMStoreFloat4(&m_vEndPos, vTargetPos);
    XMStoreFloat4(&m_vCurvePos, vMiddlePoint);
}

```

몬스터가 처치되면 그 순간의 몬스터 위치를 시작점 S,

주변의 랜덤 오프셋 좌표를 도착점 E로 잡고, S와 E의 중점에 높이(+2.0)를 준비어점 C를 둡니다.

이후 2차 베지어 곡선 공식으로 매 프레임 위치를 갱신해, 아이템이 자연스러운 포물선을 그리며 드롭되도록 구현했습니다.

# THYMESIA → 아이템 시스템 → 아이템 렌더링 연출



```
PS_OUT Out = (PS_OUT) 0;  
  
float fSpeedOffsetX = [ 0.0f ];  
float fSpeedOffsetY = [ 0.5f ];  
float fRatioNoiseTexture = [ 1.0f ];  
  
float2 vTexCoord = In.vTexCoord;  
  
float noise = g_NoiseTexture.Sample(LinearSampler, vTexCoord * fRatioNoiseTexture  
+ float2(fSpeedOffsetX, g_Time * fSpeedOffsetY)).r;  
  
float2 vDistortion = (noise - 0.5f) * float2(0.1f, 0.2f);  
  
vTexCoord += vDistortion;  
  
float3 vRedColor = lerp(float3(1.0, 0.3, 0.0), float3(1.0, 0.8, 0.0), float3(1.0, 0.0, 0.0));  
float3 vGreenColor = lerp(float3(0.8, 1.0, 0.7), float3(0.5, 1.0, 0.6), float3(0.3, 1.0, 0.4));  
float3 vBlueColor = lerp(float3(0.5, 0.4, 1.0), float3(0.2, 0.6, 1.0), float3(0.4, 0.3, 1.0));  
float3 vYellowColor = lerp(float3(1.0, 1.0, 0.4), float3(1.0, 0.9, 0.2), float3(1.0, 0.8, 0.0));  
float3 vWhiteColor = lerp(float3(0.8f, 0.8f, 0.8f), float3(0.9, 0.9, 0.9), float3(1.0, 0.8, 0.9));  
float fDistortionStrength = 1.2f;  
  
float4 FlareColor = g_Texture.Sample(LinearSampler_Clamp, vTexCoord);
```

```
if (_m_bEnLarging)  
[  
    m_fEnLargingTime += _fTimeDelta;  
  
    _fFloat fEnlargeDuration = 0.1f;  
    _fFloat fRatio = min(m_fEnLargingTime / fEnlargeDuration, 1.0f);  
    _fFloat fScale = Compute_LerpItemScale(1.3f, 2.5f, fRatio);  
    m_vCurrentScale = _float3(fScale, fScale, fScale);  
    m_fAlphaValue.w = 5.0f;  
  
    if (fRatio >= 1.0f)  
    [  
        m_bEnLarging = false;  
        m_bEnLargingDone = true;  
        m_fAcquireEffectTime = 0.f;  
    ]  
]  
  
else if (m_bEnLargingDone)  
[  
    m_fAcquireEffectTime += _fTimeDelta;  
  
    _fFloat fShrinkDuration = 0.35f;  
    _fFloat fRatio = min(m_fAcquireEffectTime / fShrinkDuration, 1.0f);  
    _fFloat fScale = Compute_LerpItemScale(2.5f, 0.0f, fRatio);  
    m_vCurrentScale = _float3(fScale, fScale, fScale);  
  
    m_fAlphaValue.w = Compute_LerpItemScale(1.0f, 0.0f, fRatio);  
  
    if (fRatio >= 1.0f)  
    [  
        m_bFinishAcquireEffect = true;  
        m_bAcquired = true;  
        m_pSameInstance->Acquire_Item(m_eItemType, m_iItemIndexNumber);  
    ]  
]
```

빌보드된 아이템 텍스처에 시간 기반 노이즈 스크롤을 합성해 UV를 미세하게 흔듭니다.

아이템 종류별 색상을 적용해 빛 번짐(플레이어)처럼 보이도록 처리하여 가시성을 높였습니다.

획득 시 짧은 '팽창 → 수축' 애니메이션을 적용합니다.

잠깐 크게 키운 뒤 스케일과 알파를 부드럽게 줄여 자연스럽게 사라지게 하며, 애니메이션이 끝나면 아이템 매니저에 획득 사실을 보고해 인벤토리에 반영합니다.

# THYMESIA → 아이템 시스템 → 아이템 정보 복원

Map컨테이너를 순회해, 임시 컨테이너에 보관하고, 각 포인터 목록을 비우고, Map컨테이너를 초기화 합니다.

```
void CItemMgr::Clear_ItemInfo()
{
    m_mapTakenItems.clear();

    for (auto& Pair : m_mapItems)
    {
        ITEM_TYPE eItemType = Pair.first;
        _uint iItemCount = Pair.second.first;

        m_mapTakenItems[eItemType].first = iItemCount;

        Pair.second.second.clear();
        Pair.second.first = 0;
    }

    m_mapItems.clear();
}
```

씬 전환 직전에 위의 함수를 호출해, 크래시 오류를 방지합니다.

```
if (_bNextLevelOpen)
{
    m_pGameInstance->Clear_ItemInfo();
    m_pGameInstance->Open_Level(LEVEL_LOADING,
        CLevel_Loading::Create(m_pDevice, m_pContext,
            static_cast<LEVELID>(m_iNextLevel), 2, false));
}
```



씬 전환 시, 이전 씬 오브젝트를 전부 Destroy 하기 때문에, 참조를 정리하지 않게 된다면, 댱글링 포인터가 남아 크래시를 유발하는 문제가 있어 해당 과정으로 해결했습니다.

새 씬 로딩 중, 임시 컨테이너에 보관되어 있던 아이템의 정보를 읽어와, 새로 생성된 아이템들과 재바인딩 하여, 복원합니다.

```
if (_bTaken)
{
    auto iter = m_mapTakenItems.find(_eItemType);
    if (iter == m_mapTakenItems.end())
        return S_OK;

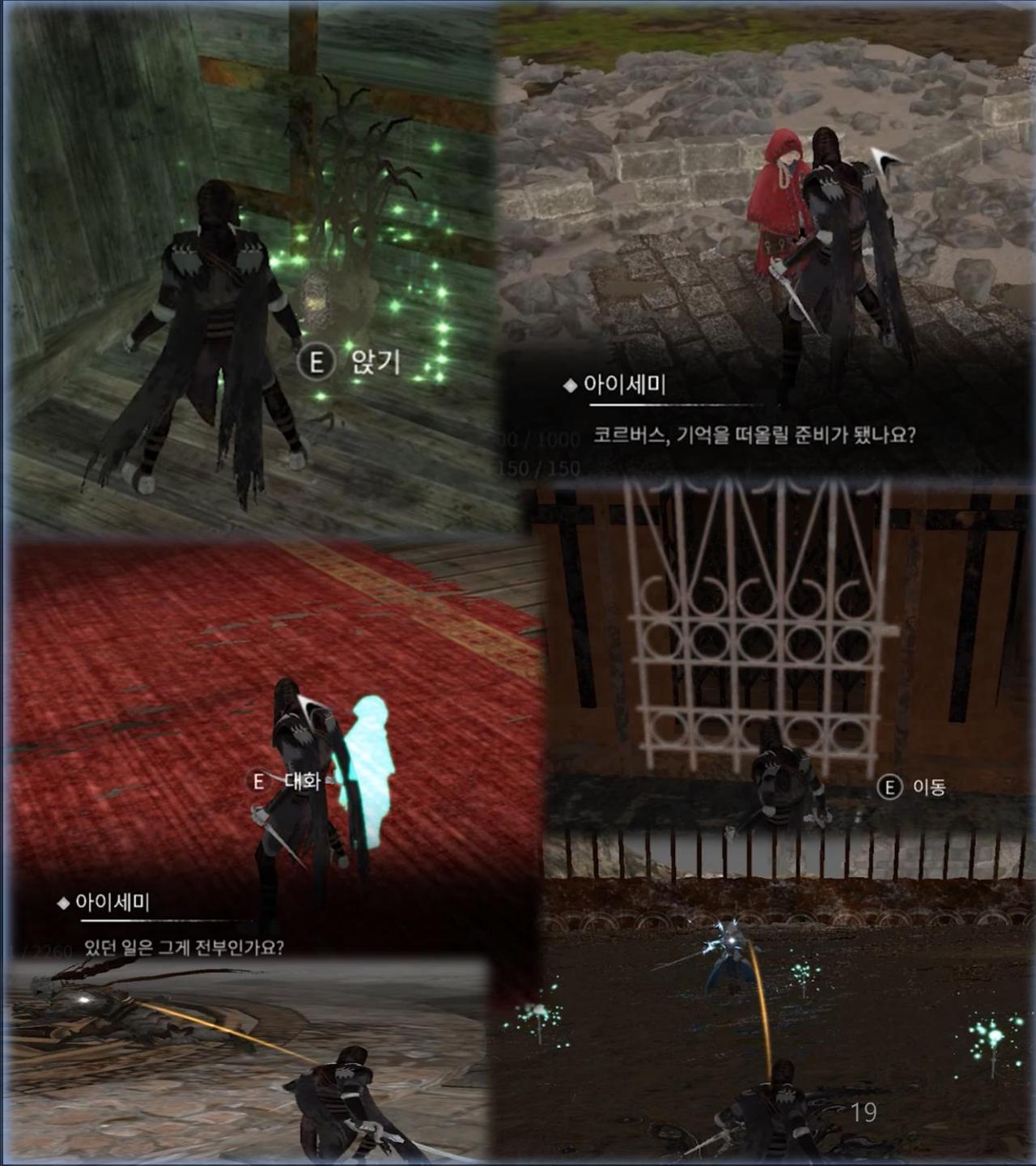
    ITEM_TYPE eItemType = iter->first;
    _uint iTakenItemCount = iter->second.first;

    if (_pGameObject)
    [
        m_mapItems[_eItemType].second.push_back(_pGameObject);
        _pGameObject->Set_ItemCount(iTakenItemCount);
    ]

    m_mapItems[_eItemType].first = iTakenItemCount;
}
```

Part 4

## 상호작용 시스템



# THYMESIA → 상호작용 시스템 → 림라이트 연출



```

return E_SUCCESS;
if (FAILED(m_pShaderCom->Bind_RawValue("g_fObjectAlpha",
&m_fAlphaValue, sizeof(_float))))
return E_FAIL;

m_pShaderCom->Begin(11);

m_pModelCom->Render(i);

```

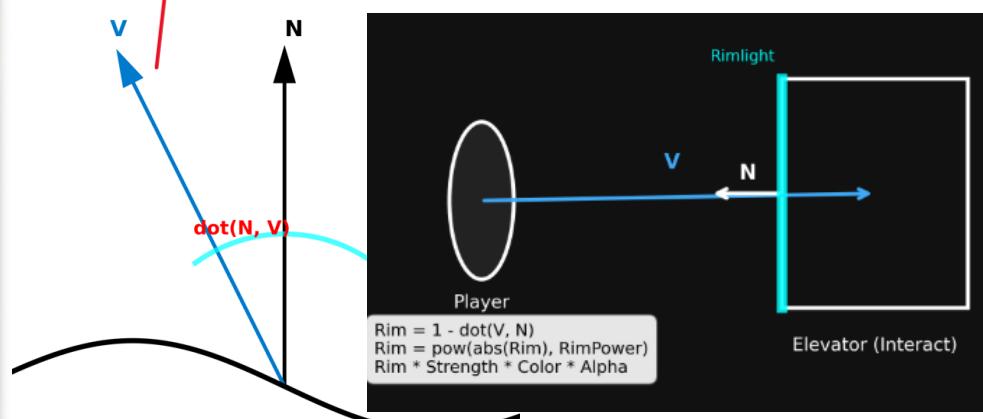
**PS\_OUT\_RIMLIGHT PS\_MAIN\_OBJECT\_RIMLIGHT(PS\_IN In)**

```

{
    PS_OUT_RIMLIGHT Out = (PS_OUT_RIMLIGHT) 0;
    float3 vDirection = normalize(g_vCamPosition.xyz - In.vWorldPos.xyz);
    float fRim = (1 - dot(vDirection, In.vNormal.xyz));
    fRim = pow(abs(fRim), g_RimPower);

    float4 vRimLight = fRim * fRimStrength * g_RimColor * g_fObjectAlpha;
    Out.vColor += vRimLight;
}
return Out;

```



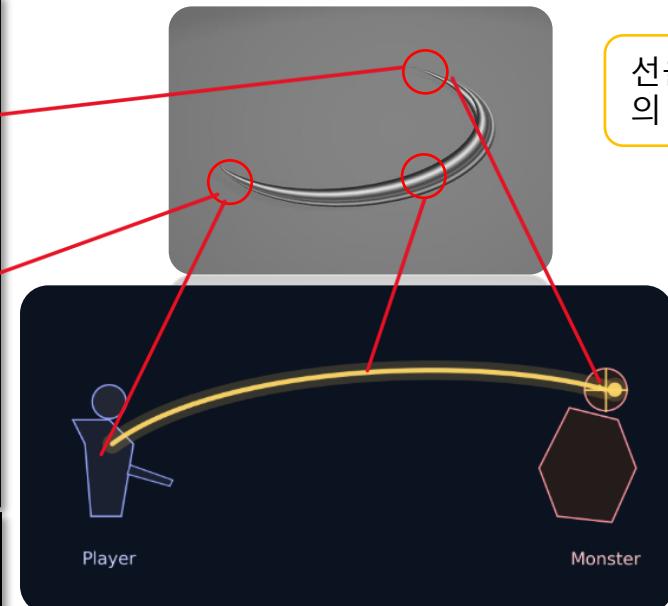
카메라 위치( $g_vCamPosition$ )와 해당 오브젝트의 월드 위치( $In.vWorldPos$ )를 빼서, 카메라에서 픽셀로 향하는 뷰 벡터  $V$ 를 계산합니다.

오브젝트 표면의 법선 벡터  $N$ ( $In.vNormal$ )은 각 픽셀이 어느 방향을 향하고 있는지를 나타냅니다.

이 두 벡터를 내적하여 각도를 구하고,  $1 - \text{dot}(V, N)$  연산을 통해 카메라에 비스듬히 보이는 영역일수록 값이 커지게 합니다.

$\text{pow}$  함수를 사용해 감도를 조절하고, 강도·색상·알파 값을 곱해 윤곽선을 강조하는 림라이트 효과를 구현합니다.

# THYMESIA → 상호작용 시스템 → 몬스터 Lock-On



선을 표현하기 위해, 비슷한 모양의 메쉬 리소스를 사용했습니다.

메쉬는 세 지점을 기준으로 배치합니다.

플레이어 위치(시작점)와 락온된 몬스터 위치(끝점)를 잡고, 그 중점을 메쉬의 위치로 사용합니다.

그 다음 RIGHT/UP/LOOK 축으로 방향을 정렬하고, X축만 두 점 사이 거리만큼 스케일해 한 줄기 락온 라인을 만듭니다.

```
_f float4x4 fMatrix = dynamic_cast<CCContainerObject*>(m_pPlayer->Get_TargetObjectPtr())
    ->Find_Part0bject(TEXT("Part_Locked_On"))->Get_CombineWorldMatrix();

_f float4 vEndPosition = _f float4(fMatrix._41, fMatrix._42, fMatrix._43, fMatrix._44);
_vector vEndPos = XMLoadFloat4(&vEndPosition);

vEndPos = XMVectorSetY(vEndPos, XMVectorGetY(vEndPos));
_vector vDir = vEndPos - XMLoadFloat4(&m_vStartPosition);
_f float fLength = XMVectorGetX(XMVector3Length(vDir));
vDir = XMVector3Normalize(vDir);
_vector vCenter = (XMLoadFloat4(&m_vStartPosition) + vEndPos) * 0.5f;

_vector vRight = vDir;
_vector vUp = XMVectorSet(0.f, 1.f, 0.f, 0.f);
_vector vLook = XMVector3Normalize(XMVector3Cross(vUp, vRight));
vUp = XMVector3Normalize(XMVector3Cross(vRight, vLook));

m_pTransformCom->Set_State(CTransform::STATE_RIGHT, vRight);
m_pTransformCom->Set_State(CTransform::STATE_UP, vUp);
m_pTransformCom->Set_State(CTransform::STATE_LOOK, vLook);
m_pTransformCom->Set_State(CTransform::STATE_POSITION, vCenter);

m_pTransformCom->Scaling(_float3(0.01f * fLength, 0.01f, 0.01f));
```

```
float2 UV = In.vTexCoord;
UV.y += g_Time * 0.7f;

vector vLineTex = g_LinePointTexture.Sample(LinearSampler, UV);
```

메시에 라인 텍스처를 입히고 UV 좌표의 y값을 시간에 따라 이동시켜, 라인이 목표 쪽으로 날아가는 효과를 구현했습니다.

# THYMESIA → 상호작용 시스템 → 상호작용 UI 버튼

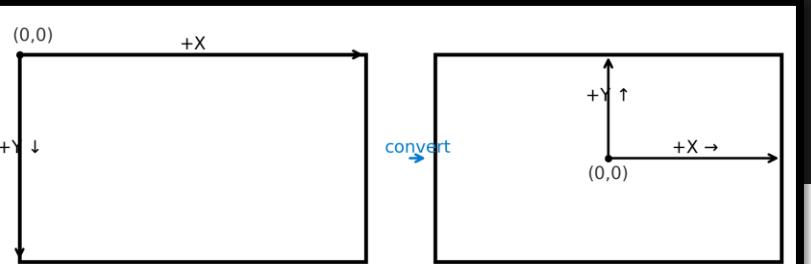


```
_vector vWorldPos = XMLoadFloat4(&m_vWorldPosition);
_float4x4 matView = m_pGameInstance->Get_Transform_Float4x4(CPipeLine::D3DTS_VIEW);
_vector vViewPos = XMVector3TransformCoord(vWorldPos, XMLoadFloat4x4(&matView));

_float4x4 matProj = m_pGameInstance->Get_Transform_Float4x4(CPipeLine::D3DTS_PROJ);
_vector vProjPos = XMVector3TransformCoord(vViewPos, XMLoadFloat4x4(&matProj));
//-----
// (-1 ~ 1) -> (0 ~ 1)
_float2 vScreenPos;
vScreenPos.x = (XMVectorGetX(vProjPos) + 1.0f) * 0.5f;
vScreenPos.y = (1.0f - XMVectorGetY(vProjPos)) * 0.5f;

_uint2 vViewportSize = m_pGameInstance->Get_ViewportSize();
vScreenPos.x *= vViewportSize.x;
vScreenPos.y *= vViewportSize.y;
//-----

vScreenPos.x -= vViewportSize.x * 0.5f;
vScreenPos.y = vViewportSize.y * 0.5f - vScreenPos.y;
```



```
_float3 fMyPos = m_pTransformCom->Get_State_UISub(UTransform::STATE_POSITION);
_float2 TextSize = m_pGameInstance->Get_TextSize(TEXT("Font_NotoSansKR18"), m_strText.c_str());
_float2 vTextPosition;
vTextPosition.x = (fMyPos.x - TextSize.x / 2) - 3.0f;
vTextPosition.y = (fMyPos.y - TextSize.y / 2) - 1.0f;

_float2 vSideTextPosition;
vSideTextPosition.x = fMyPos.x + TextSize.x * 1.5f;
vSideTextPosition.y = vTextPosition.y;

_float4 vTextColor = { 1.f, 1.f, 1.f, m_fAlphaValue };

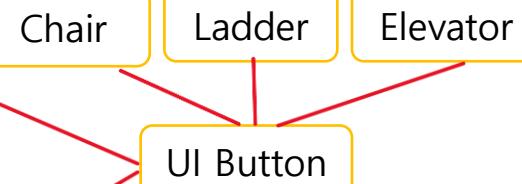
m_pGameInstance->Render_World(TEXT("Font_NotoSansKR18"), m_strText.c_str(),
    [ vTextPosition.x, vTextPosition.y ], vTextColor, 0.0f, [ 0.0f, 0.0f ], 1.0f);

m_pGameInstance->Render_World(TEXT("Font_NotoSansKR18"), m_strDescriptionText.c_str(),
    [ vSideTextPosition.x, vSideTextPosition.y ], vTextColor, 0.0f, [ 0.0f, 0.0f ], 1.0f);

m_pButton->Set_WorldPosition(vChairPosition);
m_pButton->Set_ButtonText(TEXT("E"), TEXT("앉기"));
m_pButton->Activate_Button(true);
m_bInteractOn = true;
```

UI 버튼은 다음의 좌표계 변환을 거쳐 각 오브젝트에 맞는 위치에 나타나게 됩니다.

1. 월드 좌표 -> 뷰 좌표
2. 뷰 좌표 -> NDC 좌표 (투영 + w 나누기)
3. NDC 좌표 -> 스크린 좌표 (0 ~ 1 범위)
4. 스크린 좌표 -> 픽셀 좌표 (뷰포트 해상도 곱셈)
5. 픽셀 좌표 -> HUD 좌표 (화면 중앙 원점 기준)



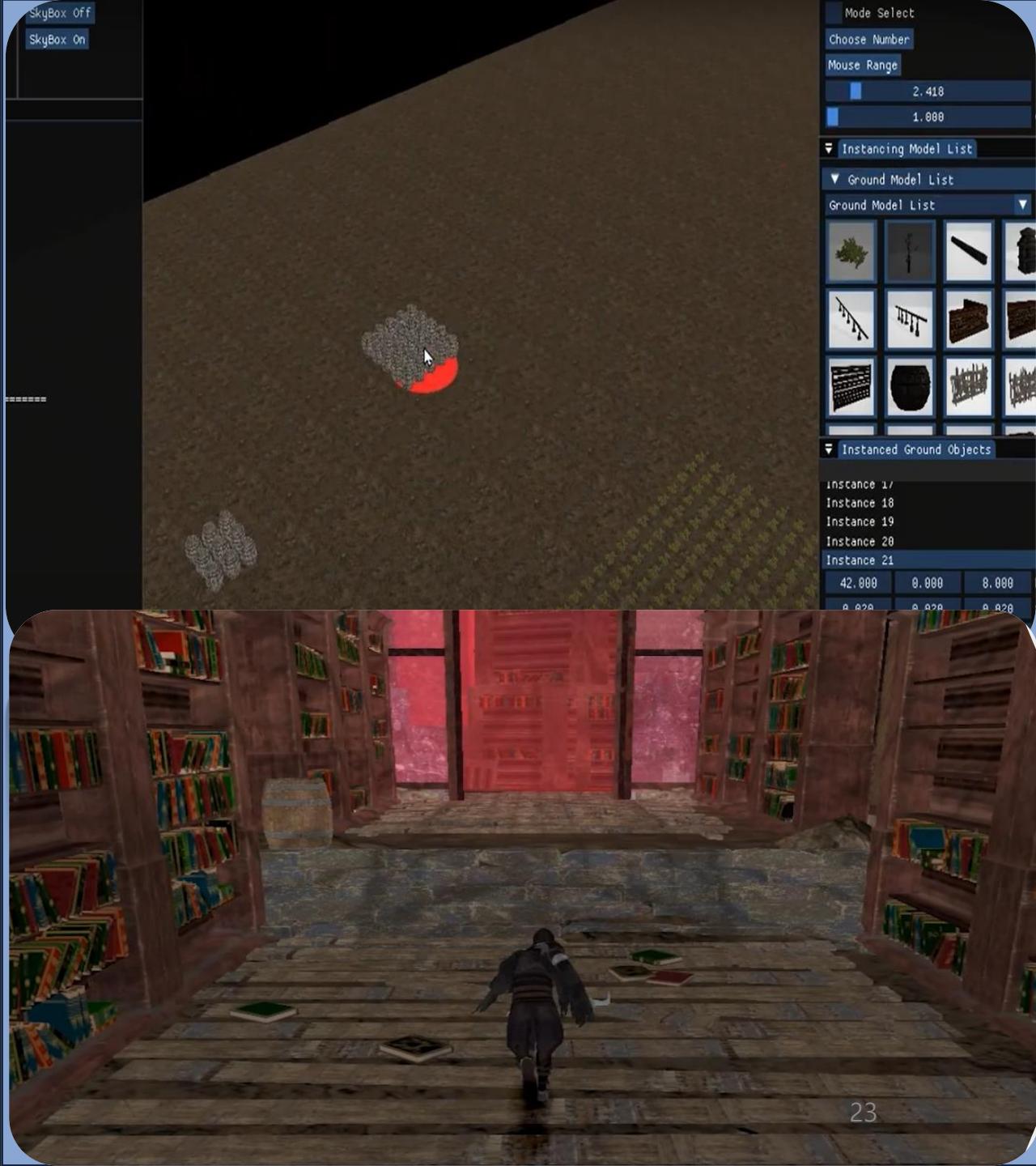
상호작용 가능한 모든 오브젝트 클래스는 UI 버튼 클래스를 참조합니다.

UI 버튼 클래스는 전달받은 오브젝트의 월드 위치를 화면 좌표로 변환하여 버튼 위치를 설정합니다.

각 오브젝트 클래스는 UI 버튼에 자신의 목적에 맞는 텍스트를 지정하여 출력합니다.

Part 5

## 메쉬 인스턴싱



# THYMESIA → 메쉬 인스턴싱 → 메쉬 인스턴싱 구조 설계

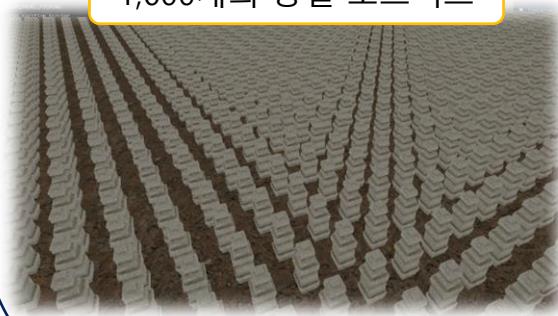


처음 게임 맵을 구성할 때, 동일한 모델의 메쉬를 수많이 렌더링 하여, 프레임 드랍이 발생했습니다.  
따라서 인스턴싱을 적용하여 프레임을 향상 시켰습니다.  
그림과 같은 구조로 인스턴싱을 설계, 구현했습니다.

Environment 클래스

Ground Object 클래스

1,600개의 동일 오브젝트



Model 클래스

Mesh 클래스

정점, 인스턴스 버퍼 바인딩

인스턴스 드로우 호출

Shader

인스턴싱용 행렬로 월드 행렬 재구성

# THYMESIA → 메쉬 인스턴싱 → 인스턴싱 흐름 소개 (1)

인스턴싱 전용 오브젝트 Ground Object 클래스  
Initialize 함수

```
XMFLOAT4 Quaternion = XMLoadFloat4(&m_vecInstanceRotation[i]);
XMMATRIX matRotation = XMMatrixRotationQuaternion(Quaternion);

XMFLOAT3 fTerrainPos = pDesc->vecInstancePosition[i];
XMMATRIX matScale = XMMatrixScaling(
    XMVectorGetX(XMLoadFloat3(&pDesc->vecInstanceScale[i])),
    XMVectorGetY(XMLoadFloat3(&pDesc->vecInstanceScale[i])),
    XMVectorGetZ(XMLoadFloat3(&pDesc->vecInstanceScale[i]))
);

XMMATRIX matPosition = XMMatrixTranslation(fTerrainPos.x,
    fTerrainPos.y, fTerrainPos.z);
XMMATRIX matWorld = matScale * matRotation * matPosition;
XMFLOAT4X4 tempMatrix;
XMStoreFloat4x4(&tempMatrix, matWorld);

instance.InstanceMatrix[0] =
    XMFLOAT4(tempMatrix._11, tempMatrix._12, tempMatrix._13, tempMatrix._14);

instance.InstanceMatrix[1] =
    XMFLOAT4(tempMatrix._21, tempMatrix._22, tempMatrix._23, tempMatrix._24);

instance.InstanceMatrix[2] =
    XMFLOAT4(tempMatrix._31, tempMatrix._32, tempMatrix._33, tempMatrix._34);

instance.InstanceMatrix[3] =
    XMFLOAT4(tempMatrix._41, tempMatrix._42, tempMatrix._43, tempMatrix._44);

m_vecInstanceData.push_back(instance);
if (FAILED(m_pModelCom->Create_InstanceBuffer(m_iNumInstance,
    m_vecInstanceData.data())))
{
    return E_FAIL;
}
```

각 인스턴스의 스케일, 회전, 위치 정보를 기반으로 월드 행렬을 생성합니다.

행렬은 이후 GPU에서 사용할 수 있도록 float4 배열로 변환됩니다.

월드행렬을 4개의 float4(행 단위)로 분해하여 'VTX\_MODEL\_INSTANCE' 구조체에 저장합니다.

이 구조체가 GPU 인스턴스 버퍼로 넘겨질 데이터입니다.

```
struct ENGINE_DLL VTX_MODEL_INSTANCE
{
    XMFLOAT4 InstanceMatrix[4];
    const static unsigned int iNumElements = 8;
    const static D3D11_INPUT_ELEMENT_DESC Elements[iNumElements];
};

const D3D11_INPUT_ELEMENT_DESC VTX_MODEL_INSTANCE::Elements[] = [
    { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { "NORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 12, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { "TEXCOORD", 0, DXGI_FORMAT_R32G32_FLOAT, 0, 24, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { "TANGENT", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 32, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { "INSTANCE_MATRIX", 0, DXGI_FORMAT_R32G32B32A32_FLOAT, 1, 0, D3D11_INPUT_PER_INSTANCE_DATA, 1 },
    { "INSTANCE_MATRIX", 1, DXGI_FORMAT_R32G32B32A32_FLOAT, 1, 16, D3D11_INPUT_PER_INSTANCE_DATA, 1 },
    { "INSTANCE_MATRIX", 2, DXGI_FORMAT_R32G32B32A32_FLOAT, 1, 32, D3D11_INPUT_PER_INSTANCE_DATA, 1 },
    { "INSTANCE_MATRIX", 3, DXGI_FORMAT_R32G32B32A32_FLOAT, 1, 48, D3D11_INPUT_PER_INSTANCE_DATA, 1 }
];
```

구성된 인스턴스 데이터를 벡터에 저장한 후, GPU에 인스턴스 버퍼를 생성합니다.

오브젝트 정보 입력

월드 행렬 생성

행렬을 구조체에 저장

인스턴스 데이터  
push\_back

GPU 버퍼 생성 호출

# THYMESIA → 메쉬 인스턴싱 → 인스턴싱 흐름 소개 (2)

GPU 버퍼 생성 호출이 완료 후 렌더링을 시작합니다.

```
m_pShaderCom->Begin(m_iPassIndex);
m_pModelCom->Update_InstanceBuffer(iVisibleCount,
    m_vecVisibleInstances.data());
m_pModelCom->Render_Instance(i, iVisibleCount);

HRESULT CModel::Update_InstanceBuffer(_uint _iNumInstances,
    const VTX_MODEL_INSTANCE* _TagInstanceData)
{
    if (nullptr == m_pInstanceBuffer || nullptr == _TagInstanceData)
        return E_FAIL;

    if (_iNumInstances == 0)
        return E_FAIL;

    D3D11_MAPPED_SUBRESOURCE tagSubResource = {};
    HRESULT hr = m_pContext->Map(m_pInstanceBuffer, 0,
        D3D11_MAP_WRITE_DISCARD, 0, &tagSubResource);

    if (FAILED(hr) || nullptr == tagSubResource.pData)
        return E_FAIL;

    memcpy(tagSubResource.pData, _TagInstanceData,
        sizeof(VTX_MODEL_INSTANCE) * _iNumInstances);

    m_pContext->Unmap(m_pInstanceBuffer, 0);

    return S_OK;
}

HRESULT CModel::Render_Instance(_uint _iMeshIndex, _uint _iNumInstanceNumber)
{
    if (!m_pInstanceBuffer)
        return E_FAIL;

    m_Meshes[_iMeshIndex]->Bind_InputAssembler_Instance(m_pInstanceBuffer);
    m_Meshes[_iMeshIndex]->Render_Instance(m_pInstanceBuffer, _iNumInstanceNumber);

    return S_OK;
}
```

컬링 조건을 통과한 인스턴스를 저장한 컨테이너.

CPU에서 계산된 월드 행렬 데이터를 GPU 인스턴스 버퍼에 전달합니다.

```
HRESULT CMesh::Bind_InputAssembler_Instance(ID3D11Buffer* pInstanceBuffer)
{
    ID3D11Buffer* pVertexBuffer[] =
    {
        m_pVB,
        pInstanceBuffer
    };

    uint iVertexStrides[] =
    {
        m_iVertexStride,
        sizeof(VTX_MODEL_INSTANCE)
    };
    uint iOffsets[] =
    {
        0,
        0
    };

    m_pContext->IASetVertexBuffers(0, 2, pVertexBuffer, iVertexStrides, iOffsets);
    m_pContext->IASetIndexBuffer(m_pIB, m_eIndexFormat, 0);
    m_pContext->IASetPrimitiveTopology(m_ePrimitiveTopology);

    return S_OK;
}

HRESULT CMesh::Render_Instance(ID3D11Buffer* pInstanceBuffer,
    _uint _iNumInstance)
{
    if (nullptr == m_pContext)
        return E_FAIL;

    m_pContext->DrawIndexedInstanced(m_iNumIndices,
        _iNumInstance, 0, 0, 0);

    return S_OK;
}
```

GPU로 데이터 전송

정점, 인스턴스 버퍼 동시 바인딩

드로우콜 호출

정점 버퍼와 인스턴스 버퍼를 동시에 바인딩하여 Input Assembler에 연결합니다.

인스턴스 개수만큼 GPU가 한 번에 렌더링하도록 Draw Call을 수행합니다.

# THYMESIA → 메쉬 인스턴싱 → 인스턴싱 절두체 컬링



```
bool CFrustum::isIn_AABB_Box(const XMFLOAT3& _fMin, const XMFLOAT3& _fMax)
{
    float fOffset = 0.1f;
    XMFLOAT3 fAdjustMin = XMFLOAT3(_fMin.x - fOffset, _fMin.y - fOffset, _fMin.z - fOffset);
    XMFLOAT3 fAdjustMax = XMFLOAT3(_fMax.x + fOffset, _fMax.y + fOffset, _fMax.z + fOffset);

    XMFLOAT3 fPoints[8] =
    {
        XMFLOAT3(fAdjustMin.x, fAdjustMax.y, fAdjustMin.z),
        XMFLOAT3(fAdjustMax.x, fAdjustMax.y, fAdjustMin.z),
        XMFLOAT3(fAdjustMax.x, fAdjustMin.y, fAdjustMin.z),
        XMFLOAT3(fAdjustMin.x, fAdjustMin.y, fAdjustMin.z),

        XMFLOAT3(fAdjustMin.x, fAdjustMax.y, fAdjustMax.z),
        XMFLOAT3(fAdjustMax.x, fAdjustMax.y, fAdjustMax.z),
        XMFLOAT3(fAdjustMax.x, fAdjustMin.y, fAdjustMax.z),
        XMFLOAT3(fAdjustMin.x, fAdjustMin.y, fAdjustMax.z)
    };

    for (_uint i = 0; i < 8; ++i)
    {
        XMVECTOR vPlane = XMLoadFloat4(&m_vWorld_Planes[i]);
        _uint iCulledPointCount = 0;

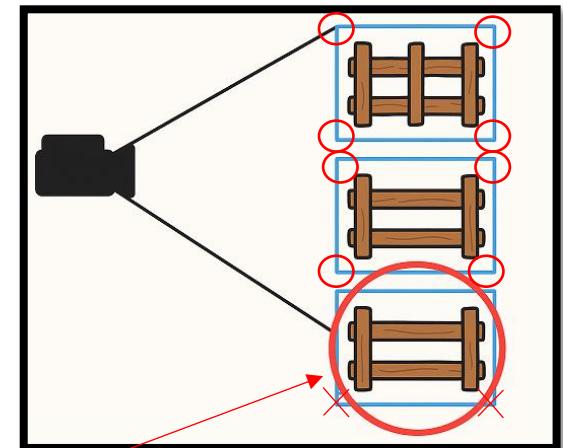
        for (_uint j = 0; j < 8; ++j)
        {
            XMVECTOR vPoint = XMLoadFloat3(&fPoints[j]);
            if (XMVectorGetX(XMPlaneDotCoord(vPlane, vPoint)) > 0.0f)
            {
                iCulledPointCount++;
            }
        }

        if (iCulledPointCount == 8)
            return false;
    }
    return true;
}

void CGroundObject::Update_InstanceBuffer_ForCulling()
{
    if (m_iNumInstance == 0)
        return;

    m_vecVisibleInstances.clear();
    m_vecVisibleInstances.resize(0);

    for (_uint i = 0; i < m_iNumInstance; ++i)
    {
        if (m_pGameInstance->isAABB_InFrustum(m_vecAABBMin[i], m_vecAABBMax[i]))
        {
            m_vecVisibleInstances.push_back(m_vecInstanceData[i]);
        }
    }
}
```



카메라 시야를 기준으로, 인스턴스 오브젝트들의 AABB 박스 8개의 꼭짓점을 검사합니다.

카메라 절두체의 6개 평면과 비교해, 꼭짓점들이 모두 바깥에 있다면, 해당 인스턴스 모델은 렌더링을 제외합니다.

보이는 인스턴스들은, 따로 저장하여, GPU에 전송합니다.

이전 슬라이드에서 언급한 GPU 전송 후, 버퍼 동시 바인딩 전에 지금과 같은 컬링을 수행 후 렌더링을 진행하게 됩니다.

# MULTI\_SHOOTING 개인 프로젝트

MULTI  
SHOOTING



방 생성하기  
방 참가하기

MULTI  
SHOOTING



방 생성하기  
방 참가하기



제작 기간

2025.06 ~ 2025.07

개발 환경

Unity Engine C#

개발 인원

1인

핵심 구현

- [Photon PUN2 기반 멀티플레이어 동기화] (방 생성/참가, RPC 통신)
- [TMP 기반 실시간 채팅 시스템]
- [플레이어, 몬스터 AI]
- [슈팅, 스킬, 탄막 패턴]
- [체력 바UI, 스테이지 진행 UI, 알람UI]
- [Shader 연출]

영상 링크

<https://youtu.be/Ch-EU6h1Ru4>

# ROBOQUEST 개인 프로젝트



제작 기간

2024.11 ~ 2025.01

개발 환경

Directx11 / C++ / ImGui

개발 인원

1인

- [FPS 조작 시스템]

- [플레이어 컨트롤]

- [몬스터 AI]

- [보스 AI]

- [UI 연출]

- [이펙트]

핵심 구현

영상 링크

<https://youtu.be/q8lf1wbsMpg>

# ROBOQUEST 개인 프로젝트 → 상세 구현 내용

## 핵심 구현

[FPS 조작 시스템]

[플레이어 컨트롤]

[몬스터 AI]

[보스 AI]

[UI 연출]

[이펙트]

### FPS 조작 시스템

- [RayCast 충돌 기반 슈팅]
- [무기 스왑 시스템]
- [스나이퍼 저격 줌 효과]

### 몬스터 AI

- [Navigation 영역 이동]
- [감지, 추적, 공격]
- [사망 삭제 처리]
- [대형 몬스터 패턴 로직]  
(돌진, 저격 대포 발사)

### 보스 AI

- [보스 등장 연출]
- [4가지 공격 패턴]  
(1. 기본 미사일 발사)  
(2. 베지어 곡선 기반 대포 미사일 발사)  
(3. 화염 지대 필드 생성)  
(4. 방패 생성, 플레이어 공격 차단)

### UI 연출

- [체력 UI, 유닛 Hp연동]
- [직교 투영 체력 UI & 빌보드 적용 체력 UI]
- [무기 탄약 UI 연동]
- [상호작용 폰트 UI]

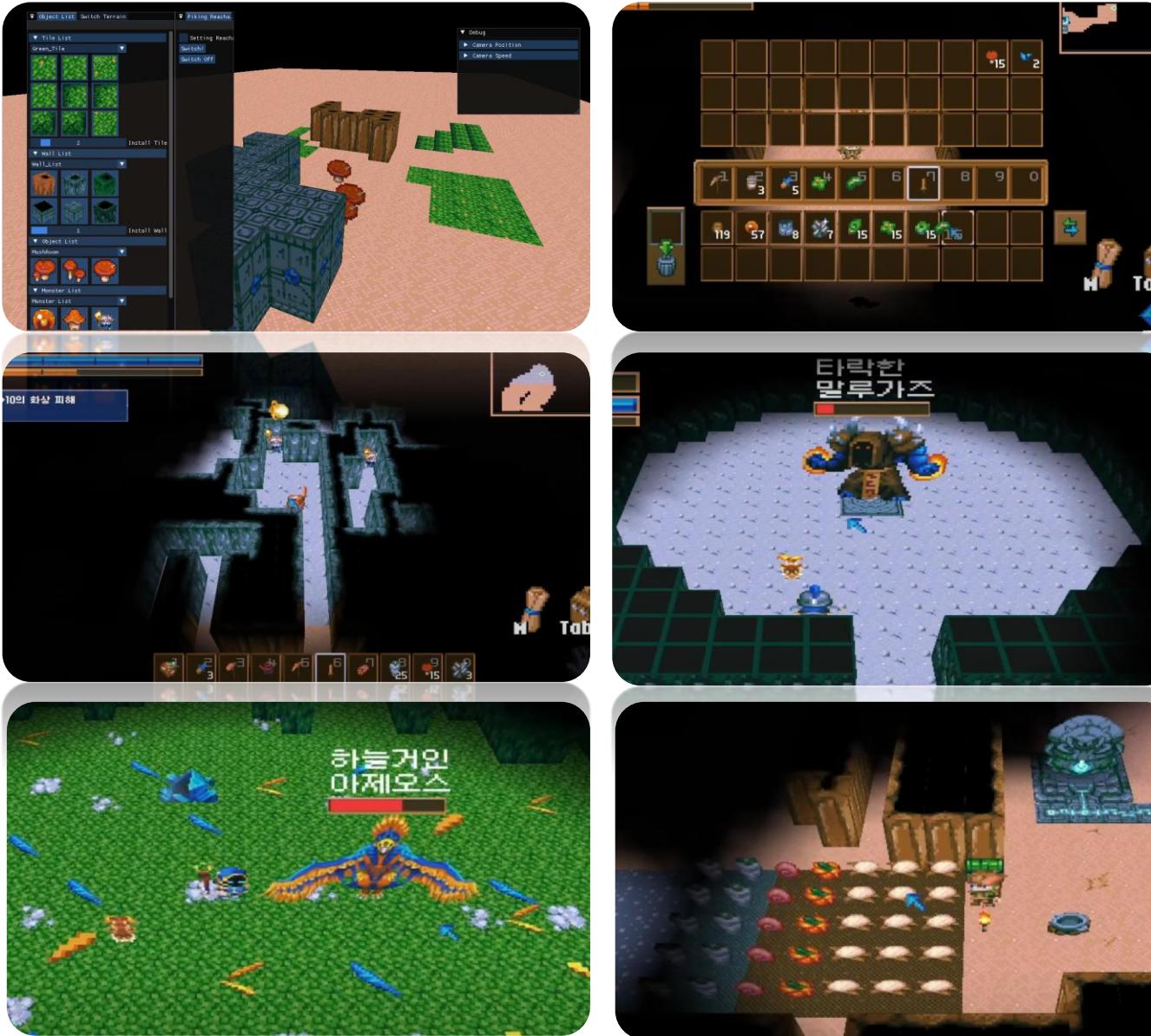
### 플레이어 컨트롤

- [WASD 움직임]
- [대쉬, 점프, 슈팅, 장전]

### 이펙트

- [총알 트레일 이펙트]
- [대포 미사일 충돌 지점 경고 표시 이펙트]
- [대포 미사일 꼬리 연기 이펙트]
- [몬스터 피격 시, 흰색 점멸 쉐이더 이펙트]
- [몬스터 사망 시, 디졸브 쉐이더 이펙트]
- [벽에 슈팅 시, 탄흔 이펙트]
- [대쉬 상태 화면 효과 이펙트]
- [플레이어 피격 시, 화면 효과 이펙트]
- [플레이어 슈팅 시, "BANG" 활자 이펙트]

# COREKEEPER 팀 프로젝트



제작 기간

2024.09 ~ 2024.11

개발 인원

4인

개발 환경

Directx9 / C++ / ImGui

담당 파트  
&  
구현 내용

- [맵 툴]
- [인 게임 맵, 미니맵 구현]
- [텔레포트 시스템]

영상 링크

<https://youtu.be/p2AZdOhzHo4>

# SNOW BROTHERS 2 개인 프로젝트



제작 기간

2024.06 ~ 2024.07

개발 인원

1인

개발 환경

Windows API , C++

핵심 구현

- [플레이어 로직]
- [몬스터 & 보스 AI]
- [충돌 로직]
- [기본 UI]
- [아이템]
- [씬 전환]

영상 링크

[https://youtu.be/7BwQi\\_yRieA](https://youtu.be/7BwQi_yRieA)