기술소개서

이장표

E-Mail : jpl1221@naver.com

핸드폰 : 010.2723.6769

URL : https://youtu.be/ZK7z8gRGid0

Contents

Bless

jpl1221@naver.com

010.2723.6769

1. 게임소개

- 게임소개

- 게임 이름 : Bless
- 게임 장르 : RPG
- 구현 언어 : C++
- 개발 환경 : Visual Studio 2015,

  DirectX9 SDK Jun
- URL : https://youtu.be/ZK7z8gRGid0
- Tool URL : https://youtu.be/bKl8WhowvFE
- 게임 특징 : 2개의 캐릭터를 실시간으로

  플레이 할 수 있습니다.

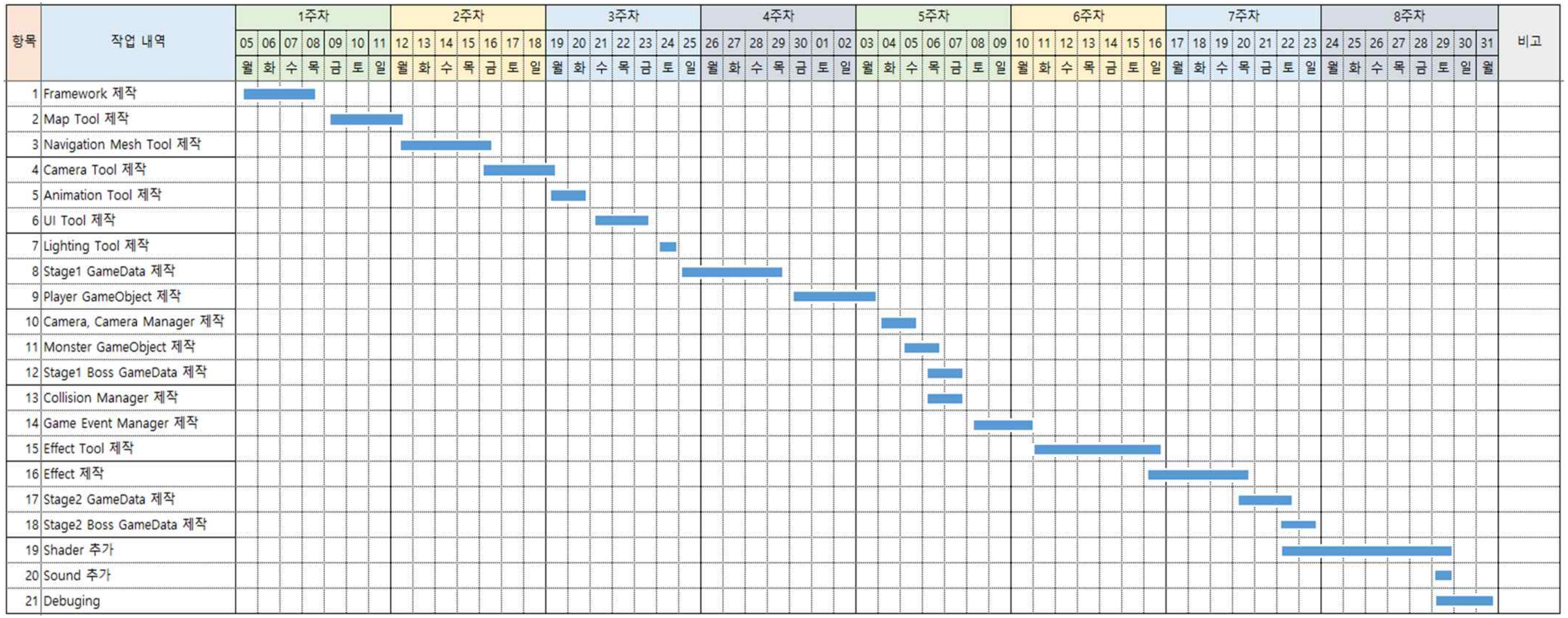
jpl1221@naver.com
010.2723.6769

2. 프로젝트 일정

Bless Doom Super Mario Crazy Arcade Publis

- 제작기간
  - 2018.11.3 ~ 2018.12.31

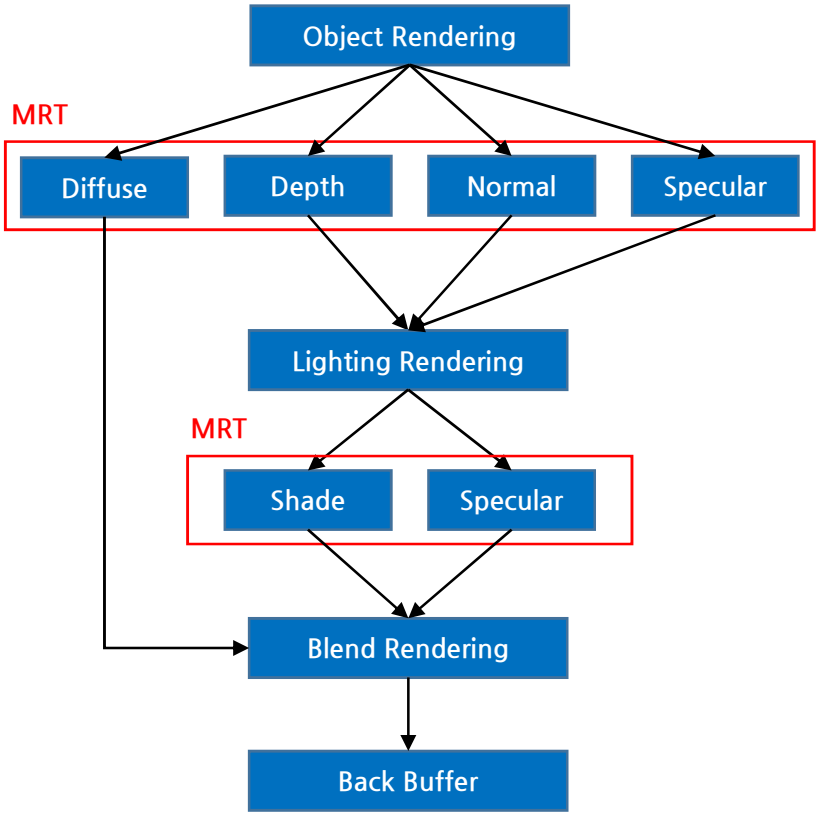
- 프로젝트 진행 일정



jpl1221@naver.com
010.2723.6769

2. 기술소개 Shader

- Deferred Rendering, Multi Rendering Target



- Deferred Rendering
  - 랜더링 결과를 중간 저장 버퍼에 쓰는 쉐이딩 알고리즘입니다.
  - 조명으로부터의 기하요소의 중복을 제거하기 위해서 사용합니다.
  - 조명은 실제 적용되는 픽셀들만 계산합니다.

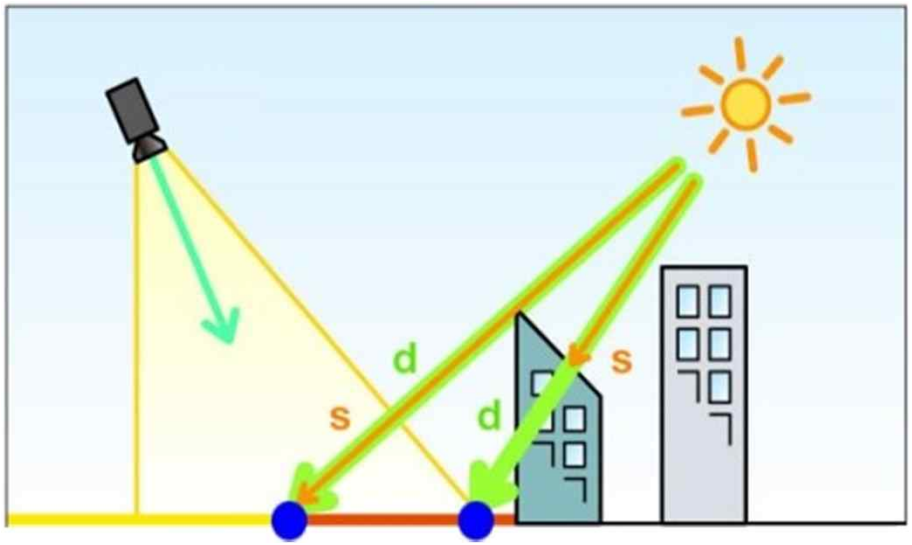
- Multi Render Target(MRT)
  - 여러 개의 경로를 가지는 랜더링 방법, 1개의 화면을 만들기 위해 여러 개의 서피스에 랜더링하는 기술입니다.
  - 조명연산을 위해 여러 가지 데이터(Diffuse, Depth, Normal, Specular)를 저장하였습니다.

jpl1221@naver.com
010.2723.6769

https://youtu.be/ZK7z8gRGid0

2. 기술소개 Shader

- Depth Shadow
  - Depth Shadow Theory



D : 랜더링 하려는 픽셀의 광원까지의 거리

S : 깊이 맵의 값, 광원으로부터 빛이 차폐되기 까지의 거리

- 조명 공간을 정의하여, 조명으로부터의 깊이 값을 텍스처에 저장합니다.
- 랜더링 시 현재 깊이 값과 조명으로부터의 깊이 값을 비교하여,
  그림자 생성 판단합니다.

- Vertex Shader

```
//Shaow Vertex Shader
VS_OUT_SHADOW VS_MAIN_SHADOW(VS_IN_SHADOW In)
{
    VS_OUT_SHADOW       Out = (VS_OUT_SHADOW)0;

    matrix      matWV, matWVP;

    matWV = mul(g_matWorld, g_matView);
    matWVP = mul(matWV, g_matProj);

    Out.vPosition = mul(vector(In.vPosition, 1.f), matWVP);
    Out.vProjPos = Out.vPosition;

    return Out;
}
```

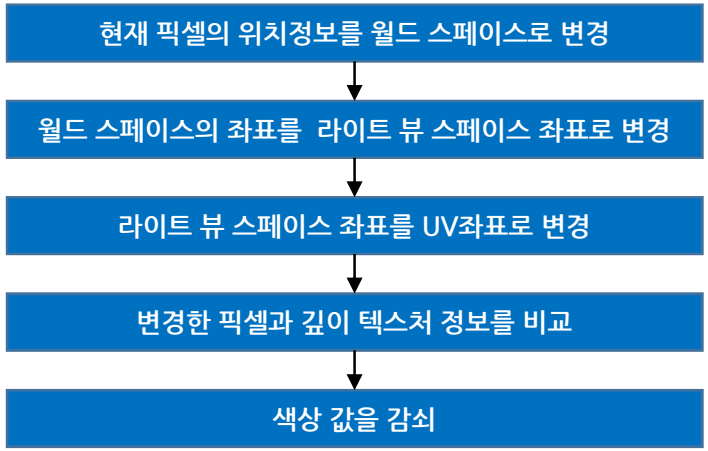
- Pixel Shader

```
//Shadow Pixel Shader
PS_OUT_SHADOW PS_MAIN_SHADOW(PS_IN_SHADOW In)
{
    PS_OUT_SHADOW       Out = (PS_OUT_SHADOW)0;

    //Save Depth Info for Shader Target
    Out.vShadow = vector(In.vProjPos.z / In.vProjPos.w,
                         In.vProjPos.w / g_fFarPlane,
                         0.f,
                         0.f);

    return Out;
}
```

2. 기술소개 Shader

- Depth Shadow
  - Apply Image

  - Compare Depth Info(Flow Chart)



  - Compare Depth Info(Pixel Shader)

```
//Current Piexl is Changed to WorldSpace
float  vViewZ = vDepthInfo.y * g_fFarPlane;
vector vProjPos;
vProjPos.x = (In.vTexUV.x * 2.f - 1.f) * vViewZ;
vProjPos.y = (In.vTexUV.y * -2.f + 1.f) * vViewZ;
vProjPos.z = vDepthInfo.x * vViewZ;
vProjPos.w = vViewZ;

vector  vViewPos;
vViewPos = mul(vProjPos, g_matProjInv);

vector  vWorldPos;
vWorldPos = mul(vViewPos, g_matViewInv);
// WorldSpace change to LightSpace
vector  vLightViewPos;
vLightViewPos = mul(vWorldPos, g_matLightView);
vector  vLightProjPos;
vLightProjPos = mul(vLightViewPos, g_matProj);

float fDepth = vLightProjPos.z / vLightProjPos.w;
float2  UV = vLightProjPos.xy / vLightProjPos.w;

// LightSpace to  UV Space
UV.y = -UV.y;    UV = UV * 0.5f + 0.5f;

//Get ShadowDepth Data
vector  vShadowDepth = tex2D(ShadowSampler, UV);

//Compare Depth
if (fDepth > vShadowDepth.x)
    Out.vColor *= 0.5f;
```

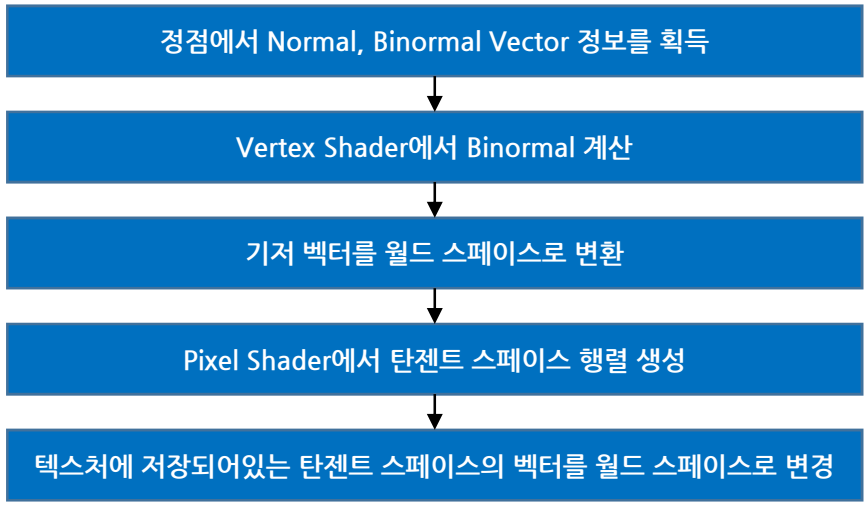
jpl1221@naver.com
010.2723.6769

https://youtu.be/ZK7z8gRGid0

2. 기술소개 Shader

- Normal Mapping
  - Apply Image

  - Flow Chart



  - Vertex Shader

```
//Calculate Binormal Vector
float3 vBinormal = cross(In.vNormal, In.vTangent);

//Change Basis Vector to WorldSpace
Out.vNormal = normalize(mul(vector(In.vNormal, 0.f), g_matWorld));
Out.vBinormal = normalize(mul(vector(vBinormal, 0.f), g_matWorld));
Out.vTangent = normalize(mul(vector(In.vTangent, 0.f), g_matWorld));
```

  - Pixel Shader

```
//Get Normal Vector of TangentSpace
vector vTangetNormal = tex2D(NormalSampler, In.vTexUV);
//Current Range 0 ~ 1
//Change Range TargetNormal to -1 ~ 1
vTangetNormal = normalize(vTangetNormal * 2 - 1);

//Make TangentMatrix
float4x4 matTangent = float4x4(In.vTangent,
                               In.vBinormal,
                               In.vNormal,
                               float4(0.f, 0.f, 0.f, 1.f));
//Chagne TangeSpcae Normal Vector to WorldSpace
vector vWorldNormal = mul(vTangetNormal, matTangent);

Out.vNormal = vector(vWorldNormal.xyz * 0.5f + 0.5f, 0.f);
```

jpl1221@naver.com
010.2723.6769

https://youtu.be/ZK7z8gRGid0

2. 기술소개 Shader

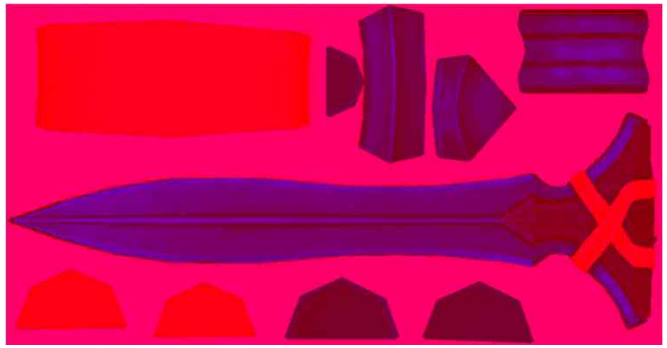
- Specular Mapping
  - Forward Pixel Shader

```
//Specular Mapping Pixel Shader
PS_OUT_NORMAL_SPECULAR PS_MAIN_NORMAL_SPECULAR(PS_IN_NORMAL_SPECULAR In)
{
    //Get Specular Data
    vector vSpecularMap = tex2D(SpecularSampler, In.vTexUV);
    //Save Specular Data
    Out.vSpecular = vector(vSpecularMap.r,
                           vSpecularMap.r,
                           vSpecularMap.r,
                           vSpecularMap.b);

    return Out;
}
```

  - Deferred Pixel Shader

```
//Speculatr Compute
vector  vReflect = reflect(vLightDir, vWorldNormal);
vector  vLook = vWorldPos - g_vCamPosition;
vector  vSpecular = pow(max(dot(normalize(vLook) * -1.f,
                           normalize(vReflect)), 0.f), g_fPow);
Out.vSpecular = g_vMtrlSpecular * vSpecular * fAtt
                           *  tex2D(SpecularSampler, In.vTexUV);
```

- Specular Texture



- 부분적인 Specular 조명연산을 위해 Specular Texture로부터 데이터를 얻어와 전달하였습니다.
- Texture의 R 부분을 Diffuse 컬러의 새기로 설정했으며, B부분을 Specular 연산의 알파세기로 사용하였습니다.

jpl1221@naver.com
010.2723.6769

https://youtu.be/ZK7z8gRGid0

2. 기술소개 Shader

- Rim Light
  - Apply Image

적용 전

적용 후

적용 범위 변경

  - Shader Code

```
//Calculate RimLight Color
float RimLightColor = smoothstep(1.f - g_fLimRightWidth, 1.f,
                    1.f - max(0,dot(vCameraDir, normalize(vWorldNormal))));
 //Add LimLight Color
 Out.vDiffuse = tex2D(DiffuseSampler, In.vTexUV) + RimLightColor;
```

- 피사체의 위나 측면 모서리를 따라서 화면의 빛의 테(Rim)를 보여주는 기술입니다.
- 카메라의 방향 벡터와 오브젝트의 법선 벡터의 각도에 따라서 외곽선을 판별합니다.
- 외곽선의 흰색을 더해 빛으로 표현합니다.

jpl1221@naver.com
010.2723.6769

https://youtu.be/ZK7z8gRGid0

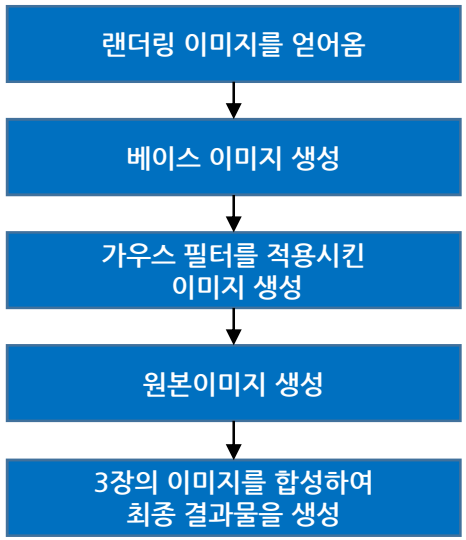
2. 기술소개 Shader

- Glow Effect
  - Apply Image

적용 전

적용 후

- Flow Chart



- Blur Effect의 한 종류인 Glow Effect입니다.
- 가우스 분포함수로부터 얻은 수치 값을 상수 값으로 사용하여 필터로 적용시켰습니다.

jpl1221@naver.com
010.2723.6769

https://youtu.be/ZK7z8gRGid0

2. 기술소개 Shader

- Glow Effect
  - Gaussian Constants

```
// Gaussian Constants
float g_BlurWeights[13] =
{
    0.002216,    0.008764,    0.026995,    0.064759,    0.120985,    0.176033,
    0.199471,
    0.176033,    0.120985,    0.064759,    0.026995,    0.008764,    0.002216,
};
```

  - Pixel Offset

```
//Pixel Offset
float2 g_fOffset = { 1/800.f, 1/600.f };
```

  - Pixel Index

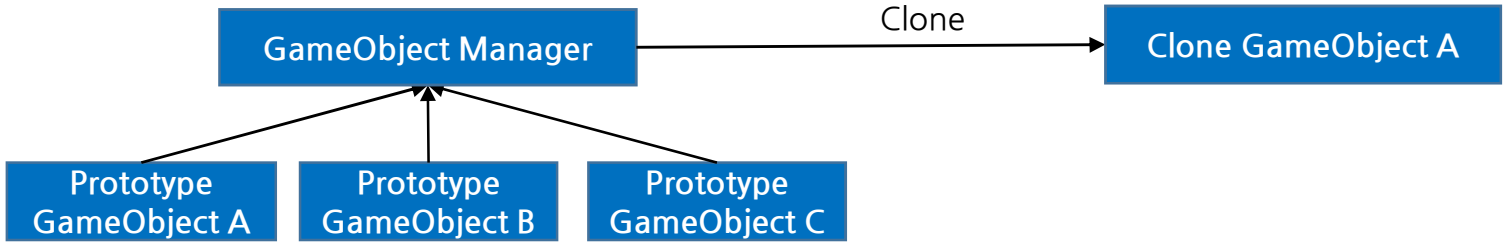
```
//IndexX
float2 g_fPixelIndexX[13] =
{
    { -6, 0 },{ -5, 0 },{ -4, 0 },{ -3, 0 },{ -2, 0 },{ -1, 0 },
    { 0, 0 },
    { 1, 0 },{ 2, 0 },{ 3, 0 },{ 4, 0 },{ 5, 0 },{ 6, 0 },
};
```

  - Pixel Shader

```
//Base Color
//Decrease TextureColor
float4 BaseColor = tex2D(BlendedSampler, In.vTexUV);
BaseColor = pow(BaseColor, 32);

//Blur Color
float4 BlurColor = g_vBaseColor;
for (int index = 0; index < 13; ++index)
{
    BlurColor += tex2D(BlendedSampler, In.vTexUV +
                (g_fPixelIndexX[index] * g_fOffset)) * g_BlurWeights[index];
    BlurColor += tex2D(BlendedSampler, In.vTexUV +
                (g_fPixelIndexY[index] * g_fOffset)) * g_BlurWeights[index];
}
BlurColor *= g_fDivideConst;
saturate(BlurColor);

//Src Color
float4 SrcColor = tex2D(BlendedSampler, In.vTexUV);

//Combine Color
Out.vColor = BaseColor + BlurColor + SrcColor;

return  Out;
```

2. 기술소개 Design Pattern

- Prototype Pattern
  - Diagram



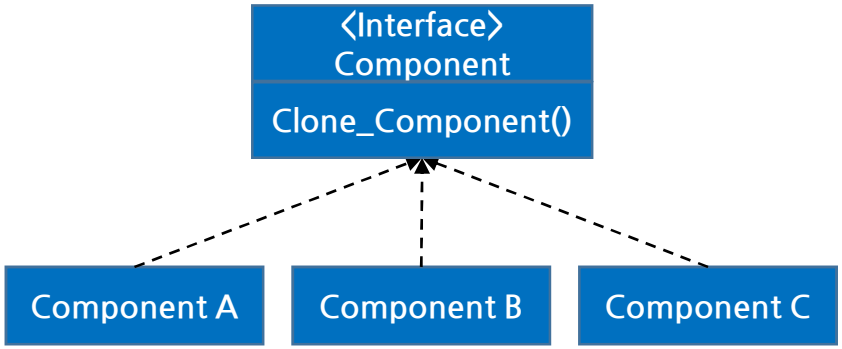

```
//Clone GameObject
CGameObject * CEffectGroup::Clone_GameObject()
{
    CEffectGroup*   pInstance = new CEffectGroup(*this);

    if (FAILED(pInstance->Ready_GameObject()))
    {
        _MSG_BOX(L"CEffectGroup Clone Failed");
        Safe_Release(pInstance);
    }
    return pInstance;
}
```

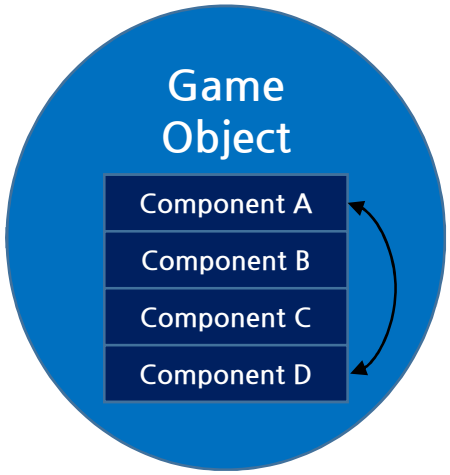
```
//Copy Constructor
CEffectGroup::CEffectGroup(const CEffectGroup & rhs)
    :CGameObject(rhs)
    , m_fTimeAcc(0.f)
    , m_bDeadObject(false)
    , m_iMeshEffectNum(rhs.m_iMeshEffectNum)
    , m_iRectEffectNum(rhs.m_iRectEffectNum)
    , m_iParticleEffectNum(rhs.m_iParticleEffectNum)
    , m_EffectrGroup_Data(rhs.m_EffectrGroup_Data)
    , m_iEffecType(0)
    , m_Damage(rhs.m_Damage)
{
    D3DXMatrixIdentity(&m_matParentMatrix);
    m_vecMeshData.assign(rhs.m_vecMeshData.begin(),
                         rhs.m_vecMeshData.end());
    m_vecRectData.assign(rhs.m_vecRectData.begin(),
                         rhs.m_vecRectData.end());
    m_vecParticleData.assign(rhs.m_vecParticleData.begin(),
                         rhs.m_vecParticleData.end());
}
```

- Prototype 패턴은 원본 객체를 Clone() 맴버 함수를 통해 복제하는 방식의 디자인 패턴입니다.
- 파일 입출력을 통해 미리 데이터를 설정한 후 복제하여 사용해, 파일 입출력 횟수를 줄일 수 있었습니다.

2. 기술소개 Design Pattern

- Component Pattern
  - Diagram





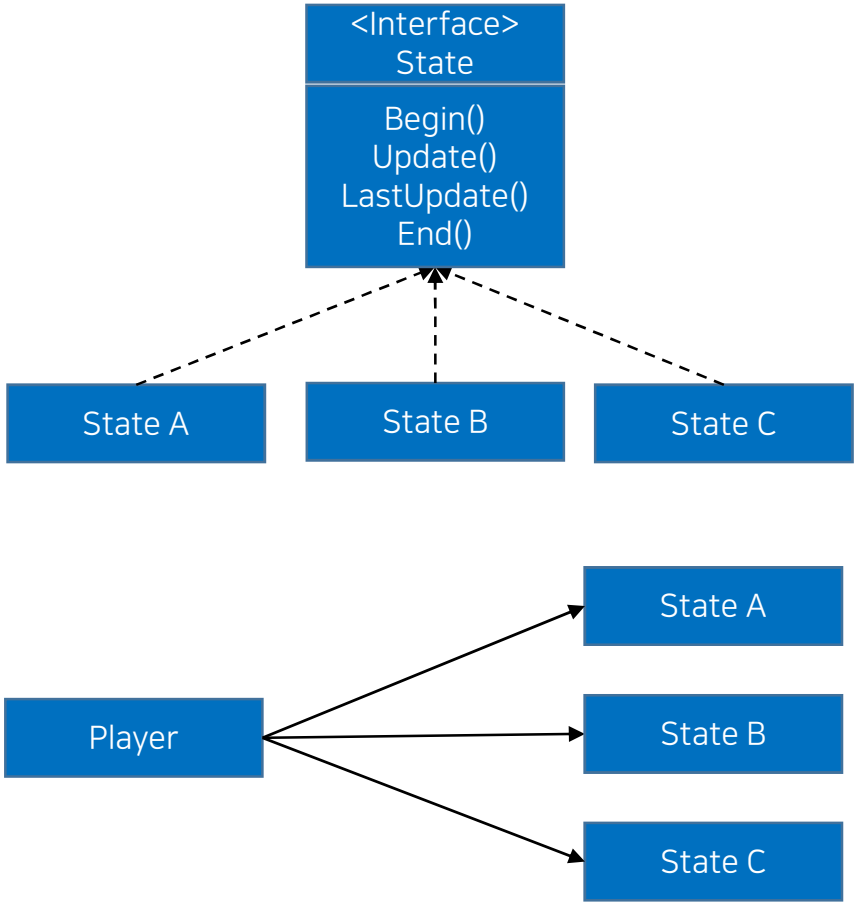
  - Code

```
////////////////////////////////////////////////////////////////////////////
//Static Component
m_pRendererCom = dynamic_cast<CRenderer*>(pManagement->
                Clone_Component(SCENE_STATIC, L"Component_Renderer"));
if (FAILED(Add_Component(L"Com_Renderer", m_pRendererCom)))
    return E_FAIL;

m_pTransformCom = dynamic_cast<CTransform*>(pManagement->
                Clone_Component(SCENE_STATIC, L"Component_Transform"));
if (FAILED(Add_Component(L"Com_Transform", m_pTransformCom)))
    return E_FAIL;

m_pCalculatorCom = dynamic_cast<CCalculator*>(pManagement->
                Clone_Component(SCENE_STATIC, L"Component_Calculator"));
if (FAILED(Add_Component(L"Com_Calculator", m_pCalculatorCom)))
    return E_FAIL;

m_pOptimizationCom = dynamic_cast<COptimization*>(pManagement->
                Clone_Component(SCENE_STATIC, L"Component_Optimization"));
if (FAILED(Add_Component(L"Com_Optimization", m_pOptimizationCom)))
    return E_FAIL;
```

  - 재사용 가능한 모듈(컴포넌트) 기반으로 프로그래밍 패턴입니다.
  - 다수의 컴포넌트 클래스들 포함할 수 있습니다.
  - 컴포넌트 간의 데이터 통신이 가능합니다.

2. 기술소개 Design Pattern

- State Pattern
  - Diagram



  - State Pointer and State Container

```
//State
CState*                     m_pCurrentState = nullptr;
map<const _tchar*, CState*> m_mapState;
```

  - Ready State

```
pState = CLups_DownAndUp::Create(this);
m_mapState.emplace(L"DownAndUp", pState);

Change_State(L"General");
```

  - Change State

```
_bool CGameObject_Dynamic::Compare_State(const _tchar * pStateTag)
{
    if (pStateTag == nullptr)
        return false;

    auto& iter = find_if(m_mapState.begin(), m_mapState.end(), CFinder_Tag(pStateTag));
    if (iter == m_mapState.end())
        return false;

    if (iter->second == m_pCurrentState)
        return true;

    return false;
}
```

2. 기술소개 Camera

- Spring Camera
  - SpringDamp Formula

$$\mathbf{F}_1 = -\{k_s(L-r) + k_d[(\mathbf{v}_1 - \mathbf{v}_2)\cdot\mathbf{L}]/L\}\mathbf{L}/L$$

    - Ks : 스프링 상수
    - L : 스프링의 늘어나거나 줄어든 길이
    - r : 스프링의 원래 길이
    - Kd : 감쇠상수(Damping Constant)
    - L : 스프링의 변위
    - V1 - V2 : 연결 물체의 상대 속도

  - Calculate Eye & At Position

```
m_vPrevTarget_Pos = m_vTarget_Pos;
m_vTarget_Pos = *CPlayer_Manager::GetInstance()->Get_Camera_Eye_Position();
m_Camera_Desc.vAt = *CPlayer_Manager::GetInstance()->Get_Camera_At_Position();
m_Camera_Desc.vEye = m_pCalculatorCom->SpringDamp(&m_Camera_Desc.vEye, &m_vTarget_Pos,
                                                  &m_vPrevTarget_Pos,
                                                  fTimeDelta,
                                                  2.5f, //Spring Const
                                                  0.25f, //Damp Const
                                                  1.f); //Spring Length
```

  - SpringDamp Function Code

```
_vec3 vDisplacement;            // Displacement(변위)
_vec3 vVelocity;                // Velocity (속도)
_float fForceMagnitude;         // Force Magnitude(힘의 크기)

vDisplacement = *pCurrentPos - *pTargetPos;
vVelocity = (*pPrevTargetPos - *pTargetPos) * fTimeDelta;
//Use the SpringDamp Formula
fForceMagnitude = fSpringConst * (fSpringLen - D3DXVec3Length(&vDisplacement))
                + fDmapConst * (D3DXVec3Dot(&vDisplacement, &vVelocity)
                / D3DXVec3Length(&vDisplacement));

D3DXVec3Normalize(&vDisplacement, &vDisplacement);
vDisplacement *= fForceMagnitude * fTimeDelta;

_vec3 ResultPos = *pCurrentPos + vDisplacement;
return ResultPos;
```

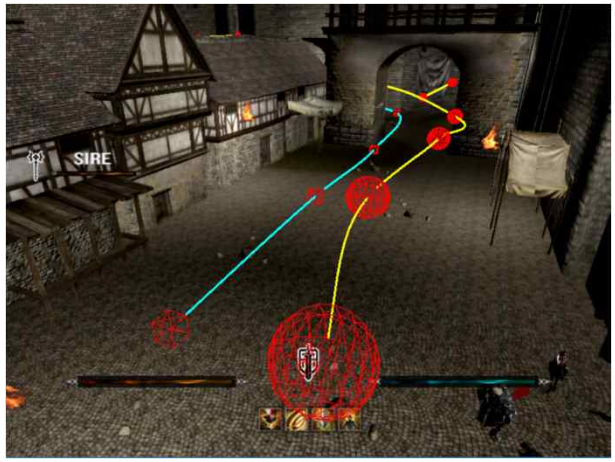
- Static Camera에 Eye를 Spring Damp 공식을 이용해 자연스럽게 움직임을 표현하였습니다.
- 플레이어 간의 시점 변화와 캐릭터의 스킬 사용 시 위치변환 등 게임진행에 필요한 카메라 움직임에 적용시켰습니다.

jpl1221@naver.com
010.2723.6769

https://youtu.be/ZK7z8gRGid0

2. 기술소개 Camera

- CutScene Camera
  - Apply Image



- Tool에서 미리 저장한 카메라의 위치 데이터를 받아옵니다.
- STL Vector Container에 데이터를 저장합니다.
- Cut Scene 카메라 재생 시 D3DXVec3CatmullRom 함수를 통해 보간 하여, 자연스러운 카메라 움직임을 구현하였습니다.

  - Code

```
// SectionSize
_int iSectionSize = (_int)m_vecCamera_Eye.size() - 1;

// Spline Interpolation Time
m_fTime += fTimeDelta * m_fSpeed;

// Go Next Section
if (m_fTime > 1.f)
{
    m_fTime = 0.f;
    m_iSection++;

    //SectionInfo Update
    m_Camera_Desc.vEye = *m_vecCamera_Eye[m_iSection]->Get_Position();
    m_Camera_Desc.vAt = *m_vecCamera_At[m_iSection]->Get_Position();
    m_Camera_Desc.vUp = _vec3(0.f, 1.f, 0.f);

    //if CutScene is End
    if (m_iSection >= iSectionSize)
    {
        m_iSection = 0;
        m_bIsSceneEnd = true;
    }
}
// Eye Interpolation
D3DXVec3CatmullRom(&m_Camera_Desc.vEye ,
    m_vecCamera_Eye[m_iSection - 1 < 0 ? 0 : m_iSection - 1]->Get_Position(),
    m_vecCamera_Eye[m_iSection + 0]->Get_Position(),
    m_vecCamera_Eye[m_iSection + 1 > iSectionSize ?
                    iSectionSize : m_iSection + 1]->Get_Position(),
    m_vecCamera_Eye[m_iSection + 2 > iSectionSize ? iSectionSize :
                    m_iSection + 2]->Get_Position(), m_fTime);
```

jpl1221@naver.com
010.2723.6769

https://youtu.be/ZK7z8gRGid0

2. 기술소개 Navigation Mesh & Sliding Vector

- Navigation Mesh
  - Apply Image

- 이동과 길 찾기를 위해서 3D공간을 2D적으로 표현방법입니다.
- 연속된 삼각형으로 지형에 설치하여 캐릭터의 이동 구역을 한정하기 위해서 사용하였습니다.

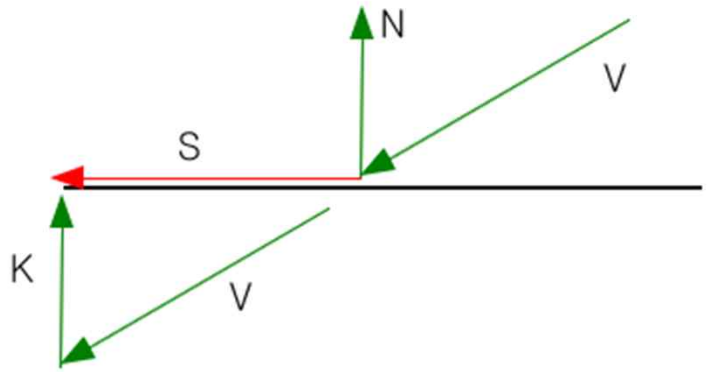
- Cell
  - 위치 좌표 3개를 연결하여 1개의 삼각형을 Cell이라 정의하였습니다.
  - Cell은 3개의 위치정보와 3개의 선분, 3개의 이웃 셀의 정보를 가지고 있습니다.
- Line
  - Cell에서 정의된 정점을 기반으로 2개의 위치정보, 2개의 위치정보를 기반으로 만든 선분의 법선 벡터를 가지고 있습니다.
  - 캐릭터의 위치와 선분을 비교해서 Cell의 안과 밖을 판단합니다.
- Navigation Mesh
  - Vector 컨테이너로 Cell을 관리합니다.
  - Cell들의 인접정보를 생성합니다.
  - 네비게이션 메쉬의 파일 입출력을 관리합니다.

jpl1221@naver.com
010.2723.6769

https://youtu.be/ZK7z8gRGid0

2. 기술소개 Navigation Mesh & Sliding Vector

- Slide Vector
  - Image

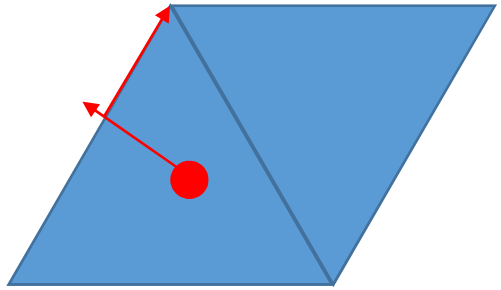


  - Formula

$$S = V + K = V + (-V \cdot N) N = V - (V \cdot N) N$$

    - 캐릭터가 네비게이션 메쉬에서 이동 시 이웃정보가 없는 Cell 일 때 자연스러운 이동을 위해 슬라이딩 벡터를 구해 미끄러지는 움직임을 구현하였습니다.

  - Image



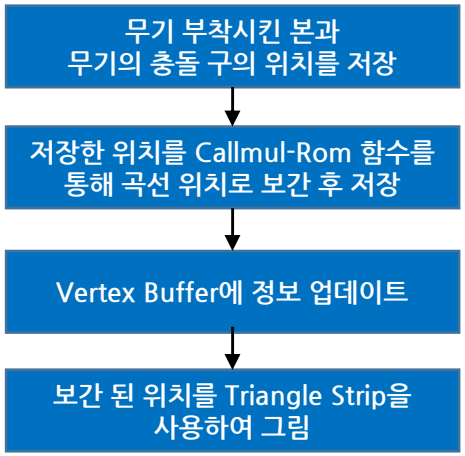
  - Code

```
//Check Character Position,
//if Character Position is Out of Navigation Mesh
if(CNavigation_Line::COMPARE_OUT == m_pLine[i]->Compare(pEndPos))
{
    //Neighbor is empty
    if (m_pNeighbor[i] == nullptr)
    {
        //Get Line Normal Vector
        _vec3 vNormal = m_pLine[i]->Get_Normal();

        //DotProduct Normal and Character Direction
        _float fDotResult = D3DXVec3Dot(&vNormal, pDir);

        //Calculate Projection Vector
        _vec3 vProjection = vNormal * fDotResult;

        *pSlideDir = *pDir - vProjection;

        return CCell::COMPARE_SLIDE;
    }
```

2. 기술소개 Sword Trail

- Sword Trail
  - Apply Image

  - Flow Chart



  - Code

```
HRESULT CTrail::Begin_Trail(CGameObject_Weapon* pTrailWeapon)
{
    //Update Activate
    m_bIsActivate = true;

    //Set Weapon Pointer
    m_pTrail_Weapon = pTrailWeapon;

    //Vertex Buffer Initialize
    VTXTEX*     pVtxTex = NULL;
    m_pVB->Lock(0, 0, (void**)&pVtxTex, 0);
    for (_int i = 0; i < m_iMaxTrail; i += 2)
    {
        pVtxTex[i].vPos = *m_pTrail_Weapon->Get_NearPosition();
        pVtxTex[i].vNormal = g_vUp;
        pVtxTex[i].vTex = _vec2((_float)i / m_iTrailIndex, 0.f);

        pVtxTex[i + 1].vPos = *m_pTrail_Weapon->Get_FarPosition();
        pVtxTex[i + 1].vNormal = g_vUp;
        pVtxTex[i + 1].vTex = _vec2((_float)i / m_iTrailIndex, 1.f);
    }

    m_pVB->Unlock();
    return S_OK;
}
```

2. 기술소개 Sword Trail

- Sword Trail
  - Code

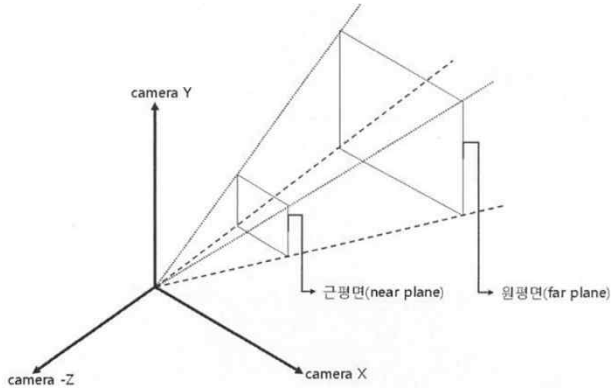
```
if (m_iTrailIndex > m_iMaxTrail)
    return 0;

//Save Trail Position
TRAILPOS    trailPos;
trailPos.vNearPosition = *m_pTrail_Weapon->Get_NearPosition();
trailPos.vFarPosition = *m_pTrail_Weapon->Get_FarPosition();

m_parrTrailPos[m_iTrailIndex++] = trailPos;
```

```
for (_int i = 0; i < m_iTrailIndex; ++i)
{
    for (_int j = 0; j < INTERPOLATION_COUNT; ++j)
    {
        TRAILPOS interpolation_TrailPos;

        //Interpolation NearPosition
        D3DXVec3CatmullRom(&interpolation_TrailPos.vNearPosition,
            &m_parrTrailPos[i - 1 <= 0 ? 0 : i - 1].vNearPosition,
            &m_parrTrailPos[i + 0].vNearPosition,
            &m_parrTrailPos[i + 1 > m_iTrailIndex - 1 ?
                            m_iTrailIndex - 1 : i + 1].vNearPosition,
            &m_parrTrailPos[i + 2 > m_iTrailIndex - 1 ?
                            m_iTrailIndex - 1 : i + 2].vNearPosition,
                            j / INTERPOLATION_RADIO);

        //Interpolation FarPosition
        D3DXVec3CatmullRom(&interpolation_TrailPos.vFarPosition,
            &m_parrTrailPos[i - 1 <= 0 ? 0 : i - 1].vFarPosition,
            &m_parrTrailPos[i + 0].vFarPosition,
            &m_parrTrailPos[i + 1 > m_iTrailIndex - 1 ?
                            m_iTrailIndex - 1 : i + 1].vFarPosition,
            &m_parrTrailPos[i + 2 > m_iTrailIndex - 1 ?
                            m_iTrailIndex - 1 : i + 2].vFarPosition,
                            j / INTERPOLATION_RADIO);

        m_parrInterpolation_TrailPos[iIndex++] = interpolation_TrailPos;
    }
}
```

```
//Vertex Buffer Update
m_pVB->Lock(0, 0, (void**)&pVtxTex, 0);
for (_int i = 0; i < iIndex; i += 2)
{
    pVtxTex[i].vPos = m_parrInterpolation_TrailPos[i].vNearPosition;
    pVtxTex[i].vNormal = g_vUp;
    pVtxTex[i].vTex = _vec2((_float)i / iIndex, 0.f);

    pVtxTex[i + 1].vPos = m_parrInterpolation_TrailPos[i].vFarPosition;
    pVtxTex[i + 1].vNormal = g_vUp;
    pVtxTex[i + 1].vTex = _vec2((_float)i / iIndex, 1.f);
}
m_pVB->Unlock();
```

2. 기술소개 Frustum Culling

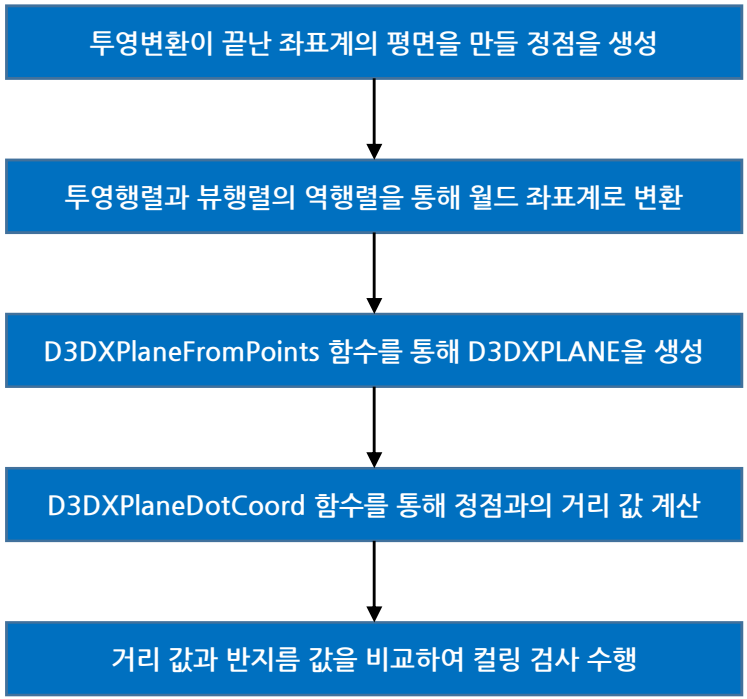
- Frustum Culling
  - Frustum



결과값	의미
$ax+by+cz+d=0$	점이 평면 위에 있다.
$ax+by+cz+d>0$	점이 평면 앞에 있다.
$ax+by+cz+d<0$	점이 평면 뒤에 있다.

  - 평면의 방정식을 사용하여 점과의 관계를 통해 판단합니다.
  - 6개의 평면을 경계 구와 비교하였습니다.

- Flow Chart



2. 기술소개 Frustum Culling

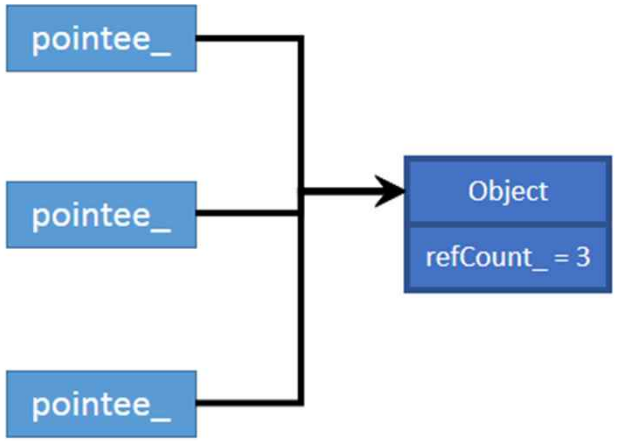
- Frustum Culling
  - Code

```
//Ready Frustum Point
Ready_Frustum();
//Go to View Space
_matrix         matProj;
m_pDevice->GetTransform(D3DTS_PROJECTION, &matProj);
D3DXMatrixInverse(&matProj, NULL, &matProj);

for (_int i = 0; i < END_PONT; ++i)
    D3DXVec3TransformCoord(&m_vPoint[i], &m_vPoint[i], &matProj);
//Go to World Space
_matrix         matView;
m_pDevice->GetTransform(D3DTS_VIEW, &matView);
D3DXMatrixInverse(&matView, NULL, &matView);

for (_int i = 0; i < END_PONT; ++i)
    D3DXVec3TransformCoord(&m_vPoint[i], &m_vPoint[i], &matView);
//Make Plane form Point in World Space
D3DXPlaneFromPoints(&m_Plane[RIGHT_PLANE],
                    &m_vPoint[NTR], &m_vPoint[FTR], &m_vPoint[FBR]);
D3DXPlaneFromPoints(&m_Plane[LEFT_PLANE],
                    &m_vPoint[FTL], &m_vPoint[NTL], &m_vPoint[NBL]);
D3DXPlaneFromPoints(&m_Plane[TOP_PLANE],
                    &m_vPoint[FTL], &m_vPoint[FTR], &m_vPoint[NTR]);
D3DXPlaneFromPoints(&m_Plane[BOTTOM_PLANE],
                    &m_vPoint[NBL], &m_vPoint[NBR], &m_vPoint[FBR]);
D3DXPlaneFromPoints(&m_Plane[FAR_PLANE],
                    &m_vPoint[FBL], &m_vPoint[FBR], &m_vPoint[FTR]);
D3DXPlaneFromPoints(&m_Plane[NEAR_PLANE],
                    &m_vPoint[NTL], &m_vPoint[NTR], &m_vPoint[NBR]);
//Check in Frustum
return IsIn_Frustum(pPosInWorld, fRadius);
```

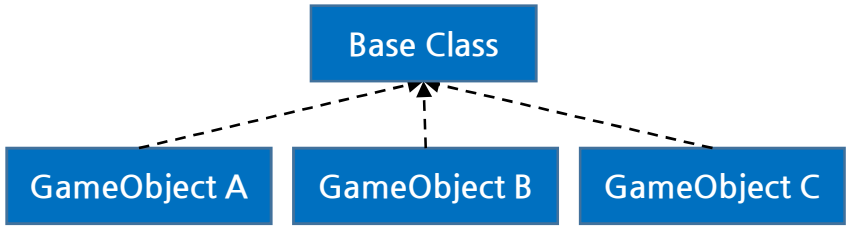
```
HRESULT Engine::CFrustum::Ready_Frustum(void)
{
    //Ready Frustum Point in Projection Space
    m_vPoint[NTL] = _vec3(-1.f, 1.f, 0.f);
    m_vPoint[NTR] = _vec3(1.f, 1.f, 0.f);
    m_vPoint[NBR] = _vec3(1.f, -1.f, 0.f);
    m_vPoint[NBL] = _vec3(-1.f, -1.f, 0.f);

    m_vPoint[FTL] = _vec3(-1.f, 1.f, 1.f);
    m_vPoint[FTR] = _vec3(1.f, 1.f, 1.f);
    m_vPoint[FBR] = _vec3(1.f, -1.f, 1.f);
    m_vPoint[FBL] = _vec3(-1.f, -1.f, 1.f);

    return S_OK;
}

_bool CFrustum::IsIn_Frustum(const _vec3* pPosition, const _float& fRadius)
{
    _float  fDistance = 0.f;

    for (_int i = 0; i < END_PLANE; ++i)
    {
        fDistance = D3DXPlaneDotCoord(&m_Plane[i], pPosition);

        if (fDistance > fRadius)
            return false;
    }

    return true;
}
```

2. 기술소개 Reference Counting

- Reference Counting
  - Image



  - Diagram



  - Code

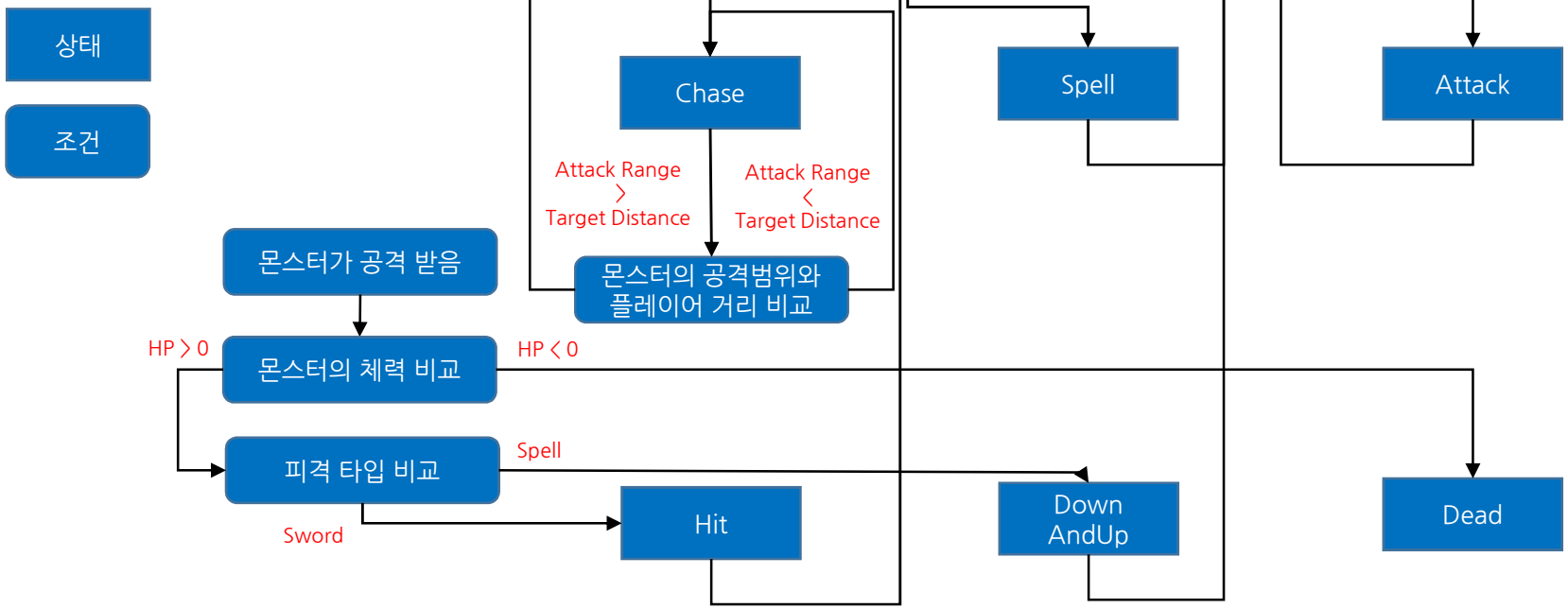
```
_ulong CBase::AddRef()
{
    //Add RefCount
    return ++m_dwRefCnt;
}
```

```
_ulong CBase::Release()
{
    //Subtract RefCount
    //if RefCount is Zero, Delete Object
    if (0 == m_dwRefCnt)
    {
        Free();

        delete this;

        return 0;
    }

    else
        return m_dwRefCnt--;

}
```

  - 참조 카운팅은 스마트 포인터에서 사용되는 소유권 관리 정책입니다.
  - 참조 카운팅 기법에서는 동일한 객체를 가리키는 포인터의 개수를 추적합니다.
  - 삭제 시 카운트가 0이 될 때 객체를 소멸시킵니다.
  - 모든 게임 오브젝트는 Base 클래스에서 파생되며, Base 클래스에서 참조 카운팅을 관리합니다.

2. 기술소개 AI

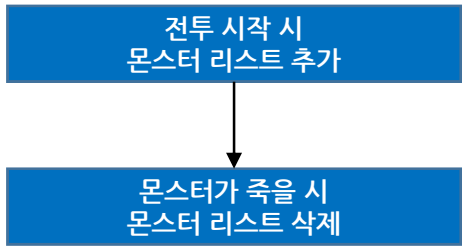
- Monster AI
  - 유한 상태 기계 머신(FSM)을 통해 몬스터의 AI를 표현하였습니다.
  - 몬스터의 상태변화는 State 패턴을 사용하였습니다.
- Introductory Remarks



2. 기술소개 Manager

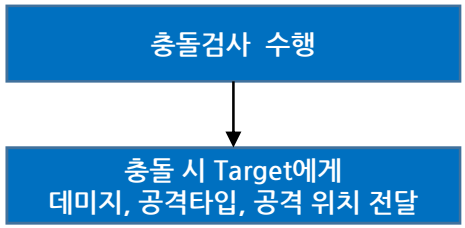
- Collision Manager
  - 충돌 검사할 몬스터와 플레이어의 정보를 관리합니다.
  - 충돌검사를 수행하고, 분기에 따라 처리합니다.
  - Hit 충돌은 대상에게 피해효과를 줍니다.
  - Push 충돌은 몬스터의 겹침 현상을 해소하기 위해 움직인 방향으로 밀어냅니다.
  - 몬스터 충돌 리스트는 검사를 최소화를 위해 유동적으로 삽입과 삭제를 수행합니다.

- Collision List Add And Delete



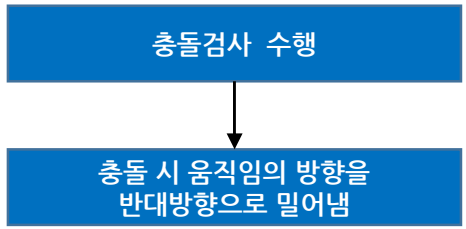

```
//Add Monster to Collision_Manager
CCollision_Manager::GetInstance()->Add_MonsterList(this);

//Delete Monster to Collision_Manager
CCollision_Manager::GetInstance()->Delete_MonsterList(this);
```

- Collision Hit




```
//Check Collision
if(pPlayer->Check_Collision(pCollider))
{
    pPlayer->Damaged(iDamag, iEffect_Type, nullptr);
}
```

- Collision Push




```
//Get Position
_vec3 vPos = *pOtherObject->m_pTransformCom->Get_StateInfo(CTransform::STATE_POSITION);
D3DXVec3Normalize(&vDir, &vDir);

//Set Position
pOtherObject->m_pTransformCom->Set_StateInfo(CTransform::STATE_POSITION,
                                             &_vec3(vPos + vDir * fDistnace * 0.1f));
```

2. 기술소개 Manager

• Event Manager

- 게임에서 발생하는 이벤트를 관리하는 매니저입니다.
- 플레이어의 네비게이션 메쉬의 위치를 비교하여, 미리 생성해 놓은 이벤트를 발생시킵니다.
- 이벤트로는 Cut Scene 재생, 전투 시작과 종료, 스테이지 전환을 담당합니다.

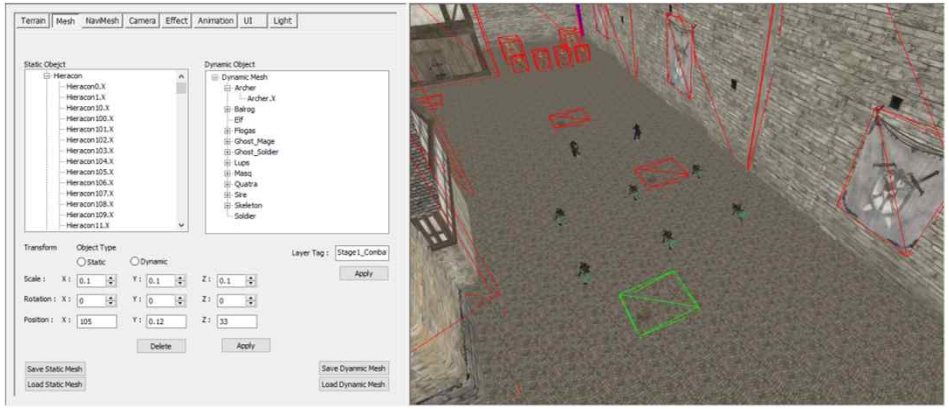
• Flow Chart

• Code

```
//Update Event
if (m_pCurrent_GameEvent != nullptr)
{
    if(m_pCurrent_GameEvent->Update_Event() == EVENT_END)
    {
        //if Event is End, Delete GameEvent in List
        auto iter = find_if(m_GameEventList.begin(), m_GameEventList.end(),
                            [&](CGameEvent *pEvent){
            if (pEvent == m_pCurrent_GameEvent)
                return true;
            return false;
        });
        m_GameEventList.erase(iter);
        Safe_Release(m_pCurrent_GameEvent);
    }
}

//Check Event
if (CPlayer_Manager::GetInstance()->Get_CurrentPlayer()
                ->Get_NaviOnIndex() == STAGE1_CUTSCENE)
{
    m_pCurrent_GameEvent = Find_Event(L"Opening");
    if (m_pCurrent_GameEvent == nullptr ||
        m_pCurrent_GameEvent->Get_IsUsed())
        return;
    m_pCurrent_GameEvent->Start_Event();
    return;
}
```

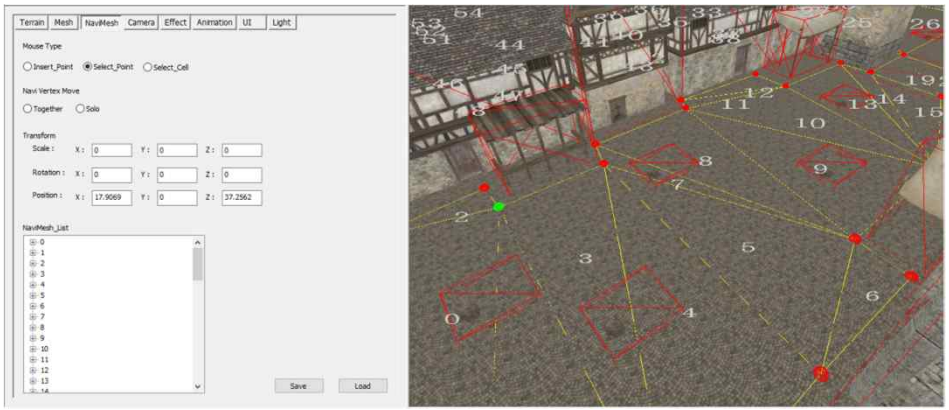
2. 기술소개 Tool

- Map Tool



- Static Mesh와 Dynamic Mesh 배치해 맵을 구성하는 툴입니다.
- 픽킹을 통해 오브젝트 위치, 크기, 회전을 지정할 수 있습니다.
- Dynamic Mesh는 Layer를 구분하여 저장할 수 있습니다.

- Navigation Mesh Tool



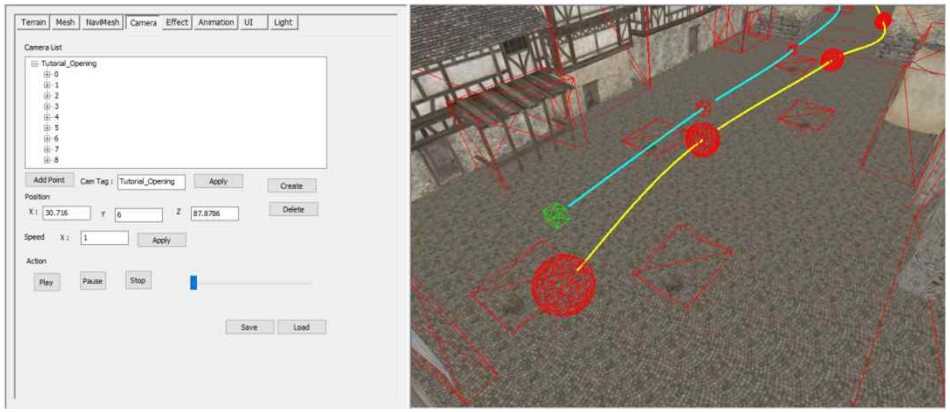
- 캐릭터의 움직임을 제어하는 네비게이션 메쉬 툴입니다.
- 지형 픽킹과 메쉬 픽킹을 통해 위치를 지정할 수 있습니다.
- 메쉬를 구성한 후 정점들에 충돌체를 씌워 이동 및 삭제 가능합니다.

jpl1221@naver.com
010.2723.6769

https://youtu.be/bKl8WhowvFE

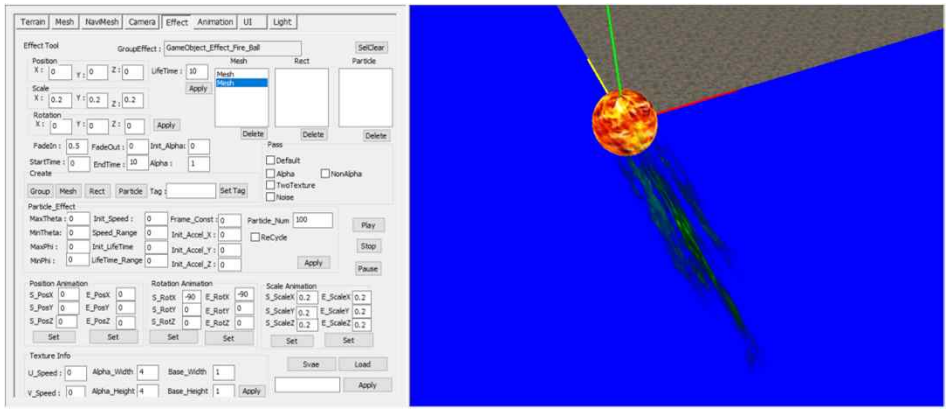
2. 기술소개 Tool

- Camera Tool



- CutScene 카메라의 이동경로를 만들 수 있는 카메라 툴입니다.
- 지형 픽킹을 통해 카메라의 Eye위치와 At위치를 지정할 수 있습니다.
- 메쉬를 구성한 후 정점들에 충돌체를 씌워 이동 및 삭제 가능합니다.
- 미리 보기를 할 수 있습니다.

- Effect Tool



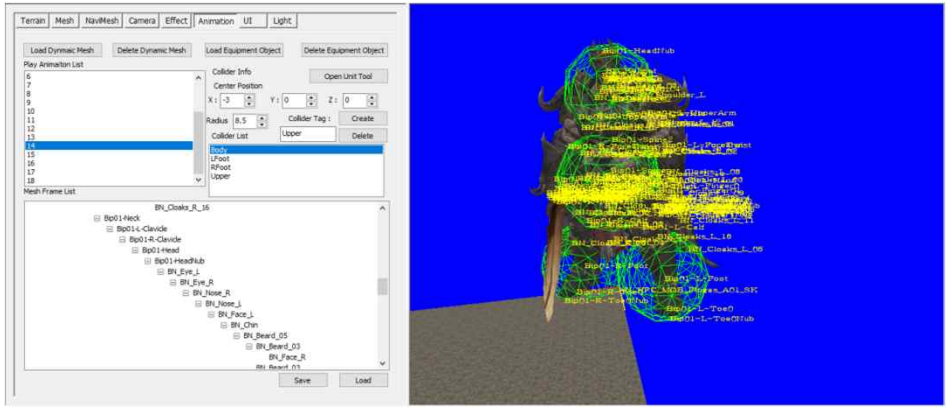
- Static Mesh와 Rect Buffer, Particle로 이펙트를 제작할 수 있습니다.
- Mesh에 쉐이더로 텍스쳐를 입혀 원하는 재질을 설정할 수 있습니다.
- 위치 보간, 회전 보간, 스케일 보간을 통해 애니메이션을 만들 수 있습니다.
- UV교란과 Texture 애니메이션을 적용 할 수 있습니다.

jpl1221@naver.com
010.2723.6769

https://youtu.be/bKl8WhowvFE

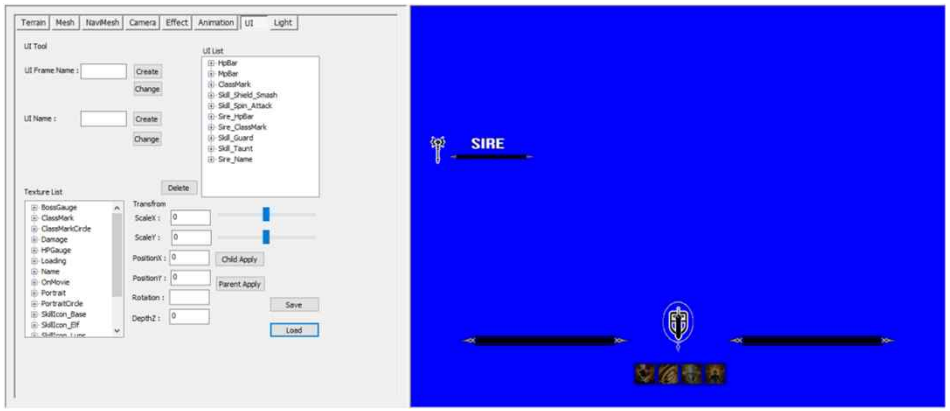
2. 기술소개 Tool

- Animation & Collider Tool



- Dynamic Mesh의 애니메이션을 확인 할 수 있습니다.
- Dynamic Mesh에 본에 충돌체를 넣을 수 있습니다.
- 본의 이름을 출력하여 대략적인 위치를 확인할 수 있습니다.

- UI Tool



- 직교투영을 통해 UI를 제작 할 수 있습니다.
- UIFrame라는 그룹을 지정하여 그룹단위로 움직일 수 있습니다.
- UI에 텍스처를 정하고, 위치, 크기, 깊이 값을 지정할 수 있습니다.
- UI를 여러 장을 겹쳐 UV 움직임으로 스킬 재사용대기시간과 체력, 마나 사용량을 표시하였습니다.

jpl1221@naver.com
010.2723.6769

https://youtu.be/bKl8WhowvFE

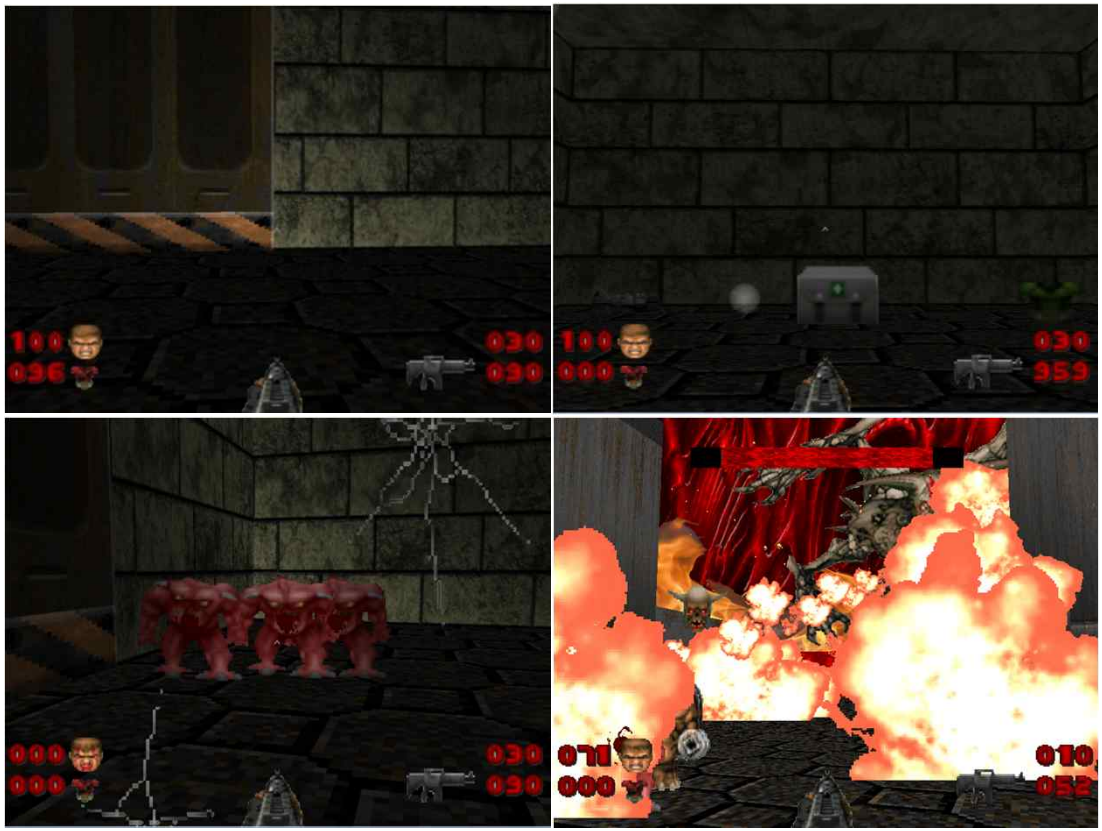
Doom

1. 게임소개
2. 기술소개
    1. Software Rendering
        1. Make World Matrix
        2. Make View Matrix
        3. Make Projection Matrix
    2. Collision
        1. Sphere Collision
        2. Cube and Sphere Collision
        3. Ray Sphere Collision
    3. BillBoard

jpl1221@naver.com
010.2723.6769

1. 게임소개

- 게임소개



- 게임 이름 : Doom
- 게임 장르 : FPS
- 구현 언어 : C++
- 개발 환경 : Visual Studio 2015,

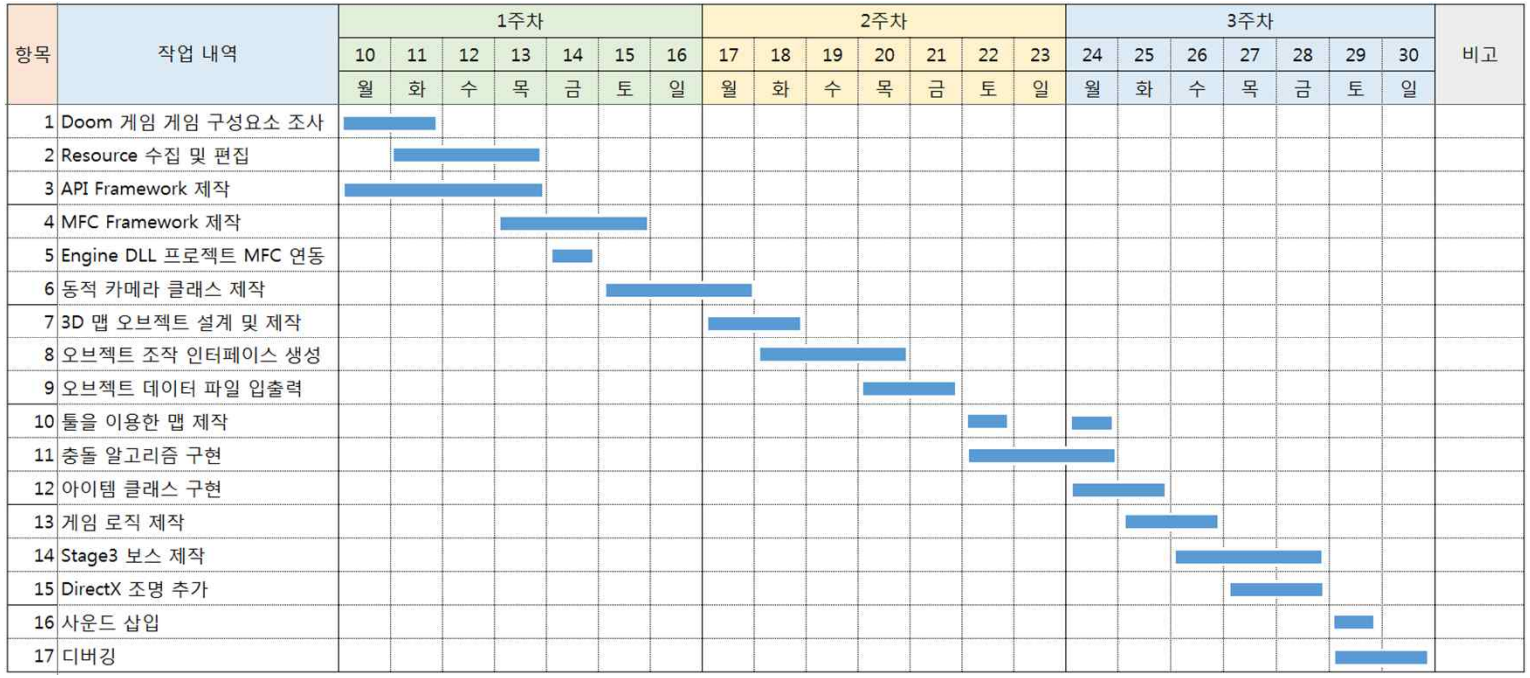
  DirectX9 SDK Jun
- 개발인원 : 3명
- URL : https://youtu.be/BBmXJTYEJt8
- 게임 특징 : 1인칭 슈팅게임 둠을

  모작하였습니다.

jpl1221@naver.com
010.2723.6769

2. 프로젝트 일정

- 제작기간
  - 2018.8.17 ~ 2018.9.1
- 맡은 역할
  - 팀장, 프레임워크, 맵 툴, Stage3 보스 제작
- 프로젝트 진행 일정



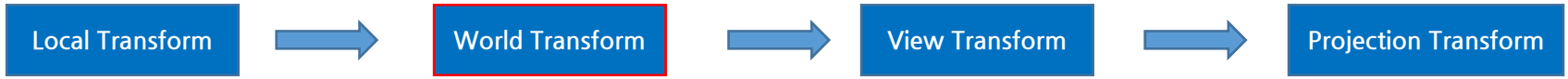
항목	작업 내역	1주차							2주차							3주차							비고
		10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	
		월	화	수	목	금	토	일	월	화	수	목	금	토	일	월	화	수	목	금	토	일	
1	Doom 게임 게임 구성요소 조사																						
2	Resource 수집 및 편집																						
3	API Framework 제작																						
4	MFC Framework 제작																						
5	Engine DLL 프로젝트 MFC 연동																						
6	동적 카메라 클래스 제작																						
7	3D 맵 오브젝트 설계 및 제작																						
8	오브젝트 조작 인터페이스 생성																						
9	오브젝트 데이터 파일 입출력																						
10	툴을 이용한 맵 제작																						
11	충돌 알고리즘 구현																						
12	아이템 클래스 구현																						
13	게임 로직 제작																						
14	Stage3 보스 제작																						
15	DirectX 조명 추가																						
16	사운드 삽입																						
17	디버깅																						

jpl1221@naver.com
010.2723.6769

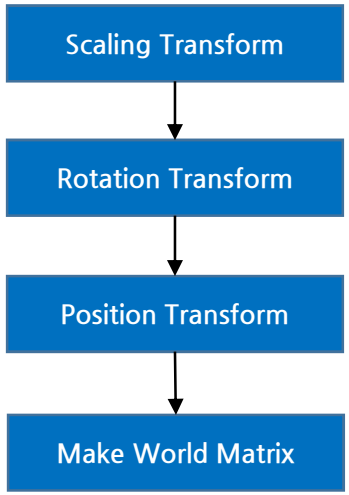
https://youtu.be/BBmXJTYEJt8

2. 기술소개 Software Rendering

- World Matrix
  - Software Rendering Flow Chart



  - Flow Chart



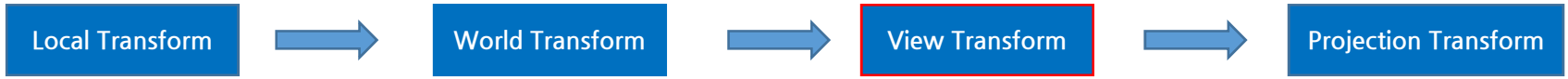
  - 랜더링 파이프라인에서 월드 변환 행렬을 직접 구현해보았습니다.

  - Code

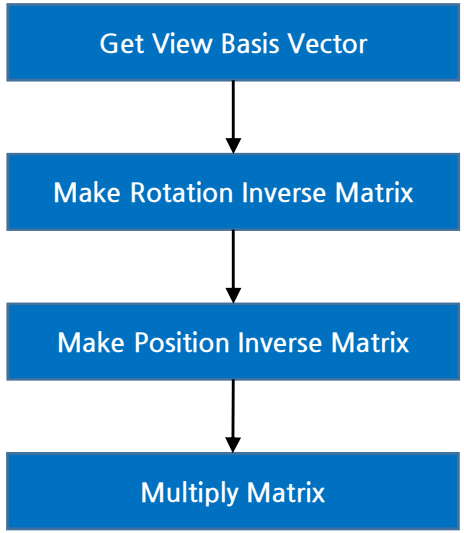
```
//Initialize Matrix
D3DXMatrixIdentity(pMatrix);

D3DXVECTOR3      vRight = D3DXVECTOR3(1.f, 0.f, 0.f);
D3DXVECTOR3      vUp = D3DXVECTOR3(0.f, 1.f, 0.f);
D3DXVECTOR3      vLook = D3DXVECTOR3(0.f, 0.f, 1.f);
D3DXVECTOR3      vPos = D3DXVECTOR3(0.f, 0.f, 0.f);


//Set Scaling
vRight *= pScale->x;
vUp *= pScale->y;
vLook *= pScale->z;

//Multiply Basis Vector and Rotation Matrix
CMathMgr::RotationX(&vRight, &vRight, pAngle[ANGLE_X]);
CMathMgr::RotationX(&vUp, &vUp, pAngle[ANGLE_X]);
CMathMgr::RotationX(&vLook, &vLook, pAngle[ANGLE_X]);

//Position Translation
vPos = *pPosition;

MakeTransformMatrix(pMatrix, &vRight, &vUp, &vLook, &vPos);
```

2. 기술소개 Software Rendering

- View Matrix
  - Software Rendering Flow Chart



  - Flow Chart



  - 랜더링 파이프라인에서 뷰 변환 행렬을 직접 구현해보았습니다.

  - Code

```
//Look Vector
D3DXVECTOR3     vLook = *pAt - *pEye;
D3DXVec3Normalize(&vLook, &vLook);

//Right Vector
D3DXVECTOR3     vRight;
D3DXVec3Cross(&vRight, pUp, &vLook);
D3DXVec3Normalize(&vRight, &vRight);

//Up Vector
D3DXVECTOR3     vUp = *pUp;
D3DXVec3Cross(&vUp, &vLook, &vRight);
D3DXVec3Normalize(&vUp, &vUp);

//Make Rotation Inverse
memcpy(pMatrix->m[0], vRight, sizeof(D3DXVECTOR3));
memcpy(pMatrix->m[1], vUp, sizeof(D3DXVECTOR3));
memcpy(pMatrix->m[2], vLook, sizeof(D3DXVECTOR3));
D3DXMatrixTranspose(pMatrix, pMatrix);

//Make Position Inverse And Combine Rotation Matrix
D3DXVECTOR3     vPos = D3DXVECTOR3(-D3DXVec3Dot(&vRight, pEye),
                                   -D3DXVec3Dot(&vUp, pEye),
                                   -D3DXVec3Dot(&vLook, pEye));
memcpy(pMatrix->m[3], vPos, sizeof(D3DXVECTOR3));
```

jpl1221@naver.com
010.2723.6769

https://youtu.be/BBmXJTYEJt8

2. 기술소개 Software Rendering

- Projection Matrix
  - Software Rendering Flow Chart

Local Transform → World Transform → View Transform → Projection Transform

  - Projection Matrix

$$\mathbf{P}=\begin{bmatrix} \frac{1}{r\tan(\alpha/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\alpha/2)} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 1 \\ 0 & 0 & \frac{-nf}{f-n} & 0 \end{bmatrix}$$

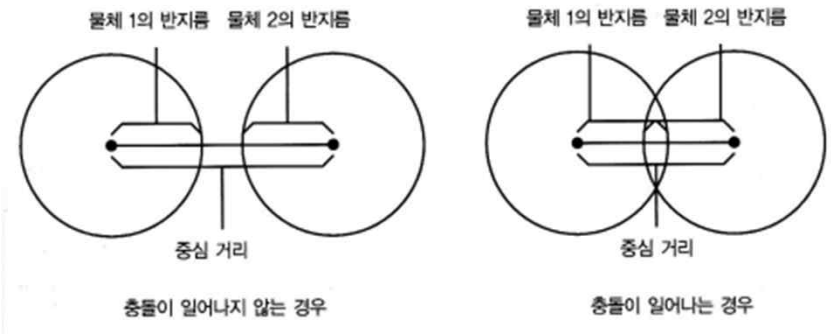
    - r : Aspect(해상도 종횡비)
    - α : Fovy(시야각)
    - f : Far Plane
    - n : Near Plane

  - Code

```
void Engine::CPipeline::MakeProjectionMatrix(D3DXMATRIX * pMatrix
                                             , float fFovy
                                             , float fAspect
                                             , float fZn
                                             , float fZf)
{
    D3DXMatrixIdentity(pMatrix);

    (*pMatrix)(0, 0) = 1.f / tanf(fFovy * 0.5f) / fAspect;
    (*pMatrix)(1, 1) = 1.f / tanf(fFovy * 0.5f);
    (*pMatrix)(2, 2) = fZf / (fZf - fZn);
    (*pMatrix)(3, 2) = fZn * fZf / (fZf - fZn);
    (*pMatrix)(2, 3) = 1.f;
    (*pMatrix)(3, 3) = 0.f;
}
```

2. 기술소개 Collision

- Sphere Collision
  - Image



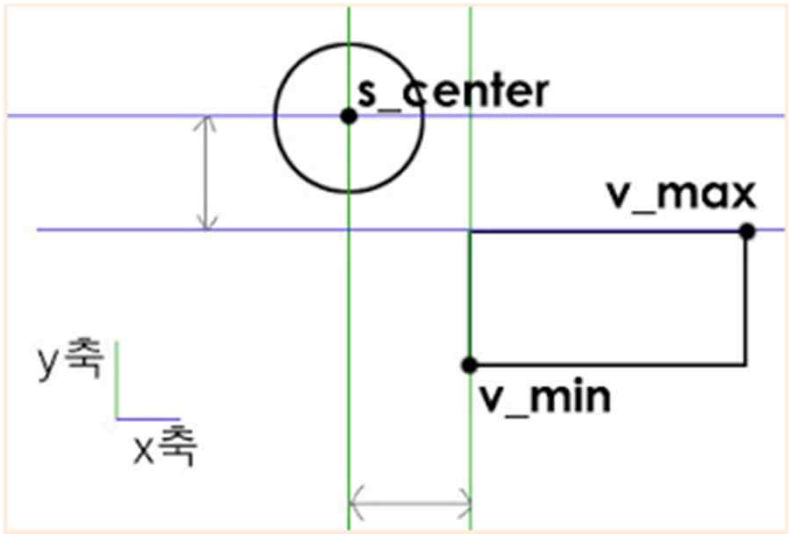
- 구와 구 충돌 검사입니다.
- 각 구의 위치 값의 거리가 두 반지름의 합보다 작으면 충돌 판단합니다.
- 플레이어, 아이템, 몬스터, 총알 등 기본적인 충돌알고리즘으로 사용하였습니다..

  - Code

```
bool Engine::CObjectCollisionMgr::SphereCollision(const CCollider* pSrcCollider, const CCollider* pDestCollider)
{
    float fDistance;
    D3DXVECTOR3 vDifference= pDestCollider->m_vCenter - pSrcCollider->m_vCenter;

    fDistance = D3DXVec3Length(&vDifference);

    if (fDistance <= (pSrcCollider->m_fRadius + pDestCollider->m_fRadius))
        return true;

    return false;
}
```

2. 기술소개 Collision

- Cube And Sphere Collision
  - Image



  - 구와 박스의 충돌 검사입니다.
  - 박스의 최소값과 구의 위치 값을 비교, 박스의 최소값과 구의 위치 거리와 반지름을 비교하여 충돌을 검사합니다.
  - 맵의 구조물과 캐릭터 충돌에 사용하였습니다.
  - Code

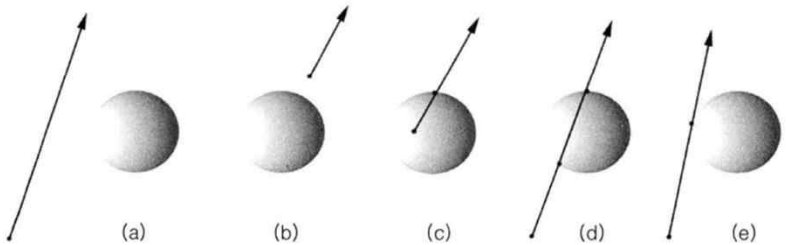
```
//X Axis
if (pSrcCollider->m_vCenter.x < pDestCollider->m_vMin.x &&
    pDestCollider->m_vMin.x - pSrcCollider->m_vCenter.x > pSrcCollider->m_fRadius)
    return false;

if (pSrcCollider->m_vCenter.x > pDestCollider->m_vMax.x &&
    pSrcCollider->m_vCenter.x - pDestCollider->m_vMax.x > pSrcCollider->m_fRadius)
    return false;
//Y Axis
if (pSrcCollider->m_vCenter.y < pDestCollider->m_vMin.y &&
    pDestCollider->m_vMin.y - pSrcCollider->m_vCenter.y > pSrcCollider->m_fRadius)
    return false;

if (pSrcCollider->m_vCenter.y > pDestCollider->m_vMax.y &&
    pSrcCollider->m_vCenter.y - pDestCollider->m_vMax.y > pSrcCollider->m_fRadius)
    return false;

//Z Axis
if (pSrcCollider->m_vCenter.z < pDestCollider->m_vMin.z &&
    pDestCollider->m_vMin.z - pSrcCollider->m_vCenter.z > pSrcCollider->m_fRadius)
    return false;

if (pSrcCollider->m_vCenter.z > pDestCollider->m_vMax.z &&
    pSrcCollider->m_vCenter.z - pDestCollider->m_vMax.z > pSrcCollider->m_fRadius)
    return false;

// Collision
return true;
```

jpl1221@naver.com
010.2723.6769

https://youtu.be/BBmXJTYEJt8

2. 기술소개 Collision

- Ray and Sphere Collision
  - Image



〔그림 15.4〕 a) 광선이 구체를 빗나갔다. b) 광선이 구체의 중심을 교차한다. t_0과 t_1이 모두 음수이다. c) 광선이 구체 내부에 있다. 두 해 중 하나는 양수이고 하나는 음수이다. 하나의 양수 해가 단일 교차점을 제공한다. d) 광선이 구체를 교차한다. t_0과 t_1이 모두 양수이다. e) 광선이 구체와 접촉해 있다. 이 경우에 해는 양수이며 $t_0 = t_1$이다.

  - Formula

$$At^2 + Bt + C = 0$$

$$A = \mathbf{u} \cdot \mathbf{u}, \quad B = 2(\mathbf{u} \cdot (\mathbf{p}_0 - \mathbf{c})), \quad C = (\mathbf{p}_0 - \mathbf{c}) \cdot (\mathbf{p}_0 - \mathbf{c}) - r^2$$

- 광선과 구를 월드 스페이스 공간으로 변환 후 구 방정식에 대입하여, 2차 방정식을 구합니다.
- 2차 방정식에 해를 구해 충돌을 판단합니다.
- 맵툴에서 오브젝트 픽킹을 검사하기 위해 사용하였습니다.

  - Code

```
D3DXVECTOR3 vMouse = GetMousePos();
m_pRay->InputMousePos(vMouse);

D3DXVECTOR3 vCenterToRay = *(m_pRay->GetPivotPos()) -
                           pSphereCollision->m_vCenter;

// Linear term
float fOneDgree = 2.0f * D3DXVec3Dot(m_pRay->GetRayDir(), &vCenterToRay);

// Constant term
float fConstDgree = D3DXVec3Dot(&vCenterToRay, &vCenterToRay) -
                    (pSphereCollision->m_fRadius * pSphereCollision->m_fRadius);

// Discriminant
float fDiscriminant = (fOneDgree * fOneDgree) - (4.0f * fConstDgree);

if (fDiscriminant <0.f)
    return false;

fDiscriminant = sqrtf(fDiscriminant);

float fFirstSolution = (-fOneDgree + fDiscriminant) / 2.0f;
float fSecondSolution = (-fOneDgree - fDiscriminant) / 2.0f;

// if Solution is bigger then 0, true
if (fFirstSolution >= 0.0f || fSecondSolution >= 0.0f)
    return true;

return false;
```

2. 기술소개 BillBoard

- BillBoard
  - Image

- 빌보드는 카메가 어느 방향에서 바라보아도 항상 카메라의 정면을 향하고 있습니다.
- Y축 고정 빌보드를 사용하였습니다.
- 몬스터와 아이템, 이펙트에 적용하였습니다.
- 카메라 변환행렬의 성분 중 Y축 회전행렬 부분만 역 변환하여 빌보드 행렬을 구했습니다.

- Code

```
//Get ViewMatrix
_matrix matView;
m_pDevice->GetTransform(D3DTS_VIEW, &matView);
//Save Scale value
_vec3   vScale;
vScale.x = Get_ScaleInfo(CTransform::STATE_RIGHT);
vScale.y = Get_ScaleInfo(CTransform::STATE_UP);
vScale.z = Get_ScaleInfo(CTransform::STATE_LOOK);
//Set Scaling
Set_ScaleInfo(&_vec3(1.f, 1.f, 1.f));
//Get ViewMatrix Inverse
_matrix matInvView;
D3DXMatrixInverse(&matInvView, nullptr, &matView);
//Get YAxis Rotation Value
_matrix matBillBoard;
D3DXMatrixIdentity(&matBillBoard);
matBillBoard._11 = matInvView._11;
matBillBoard._13 = matInvView._13;
matBillBoard._31 = matInvView._31;
matBillBoard._33 = matInvView._33;
_matrix matWorld = matBillBoard * m_matWorld;
//Set Original Scale Value;
*(_vec3*)&matWorld.m[0][0] *= vScale.x;
*(_vec3*)&matWorld.m[1][0] *= vScale.y;
*(_vec3*)&matWorld.m[2][0] *= vScale.z;

*(_vec3*)&m_matWorld.m[0][0] *= vScale.x;
*(_vec3*)&m_matWorld.m[1][0] *= vScale.y;
*(_vec3*)&m_matWorld.m[2][0] *= vScale.z;
```

Super Mario

1. 게임소개
2. 기술소개
    1. Design Pattern
        1. Observer
        2. Bridge
    2. Line Collision

jpl1221@naver.com
010.2723.6769

1. 게임소개

- 게임 스크린샷

- 게임 이름 : Super Mario
- 게임 장르 : 횡 스크롤 액션 아케이드
- 구현 언어 : C++
- 개발 환경 : Visual Studio 2015,

  DirectX9 SDK Jun
- **제작기간 : 2주**
- URL : https://youtu.be/TcP36BdnP2k
- Tool URL : https://youtu.be/0PfPM2OcYXk
- 게임 특징 : 잠입 게임을 모방하였습니다.

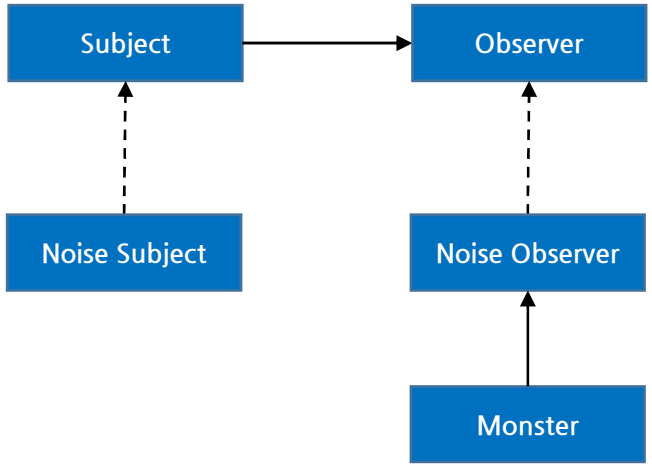
  소음시스템, 경비 시스템 넣어 잠입요소를

  추가하였습니다.

jpl1221@naver.com
010.2723.6769

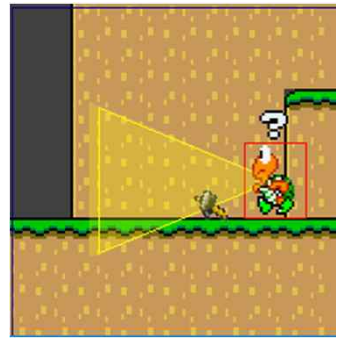
2. 기술소개 Design Pattern

- Observer Pattern
  - Diagram



- 한 객체의 상태가 변경되면 그 객체에 의존하는 다른 객체들에게 연락이 가는 방식의 디자인 패턴입니다.
- Subject에서 데이터가 추가되면 Observer에게 데이터를 전달하는 Push 형태의 옵저버 패턴을 사용하였습니다,
- 소음이 발생하면 몬스터에게 통지하여 충돌 검사를 수행하였습니다.

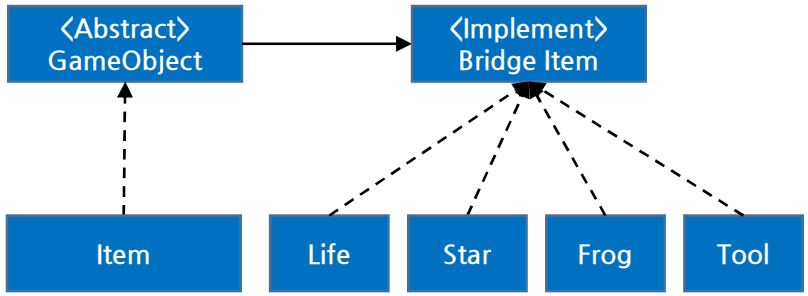
  - Code



```
void CNoiseSubject::AddData(CNoise * pNoise)
{
    m_NoiseList.push_back(pNoise);
    Notify();
}

void CNoiseSubject::RemoveData()
{
    m_NoiseList.pop_front();
    Notify();
}
//Update
void CNoiseObserver::Update()
{
    //Get Noise List
    const list<CNoise*>* list = CNoiseSubject::GetInstance()->GetNoiseList();

    for (auto iter = list->begin(); iter != list->end(); ++iter)
    {
        RECT    rt;
        //Collision Check
        if (IntersectRect(&rt, (*iter)->GetRect(), &(*m_pMonster->GetRect())))
        {
            m_pNoise = *iter;
            m_bIsNoiseRange = true;
            return;
        }
    }

    m_pNoise = nullptr;
    m_bIsNoiseRange = false;
}
```

2. 기술소개 Design Pattern

- Bridge Pattern
  - Diagram



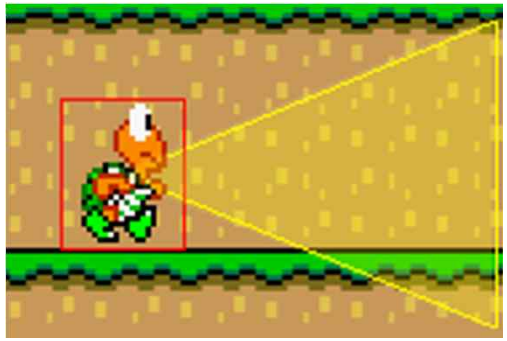
- 인터페이스 클래스와 구현 부 클래스의 상속관계를 독립적으로 정의합니다.
- 이들을 다리처럼 연결해주는 패턴이 브릿지 패턴입니다.
- 인터페이스 클래스와 구현클래스가 별도로 상속함으로, 서로 독립적인 확장이 가능합니다.

  - Code

```
int CItem::Update(void)
{
    int     iResult = NO_EVENT;
    if(m_pBridge)
        iResult = m_pBridge->Update(m_tInfo);

    if(iResult == DEAD_OBJ)
    {
        delete this;
        return DEAD_OBJ;
    }

    return 0;
}
int CItemBridge::Update(void)
{
    if (m_bIsDead)
        return DEAD_OBJ;

    //Collision Update
    CObj::UpdateRect();

    //Make WorldMatrix
    const D3DXVECTOR3*  vScroll = CScrollMgr::GetInstance()->GetScroll();
    D3DXMATRIX matTrans, matScale;
    D3DXMatrixScaling(&matScale, m_fImgDir, 1.f, 0.f);
    D3DXMatrixTranslation(&matTrans
        , m_tInfo.vPos.x - vScroll->x
        , m_tInfo.vPos.y - vScroll->y
        , 0.f);
    m_tInfo.matWorld = matScale * matTrans;

    return NO_EVENT;
}
```

2. 기술소개 Line Collision

- Line Collision
  - Apply Image



- 3개의 정점을 기준으로 3개의 라인을 생성합니다.
- 3개의 선분의 법선 벡터와 플레이어와 각 정점까지의 벡터를 생성합니다.
- 선분의 법선 벡터와 플레이어와 정점까지의 벡터를 내적 연산을 통해,
  삼각형 안에 있는지 검사합니다.

- Code

```
D3DXVECTOR3      vDirection[MAX_POINT] = {};
/*
        o(2)
                o(1)
        o(3)
*/
vDirection[POINT_ONE] = D3DXVECTOR3(vPoint[POINT_TWO] -
                        vPoint[POINT_ONE]);
vDirection[POINT_ONE].z = 0.f;
vDirection[POINT_TWO] = D3DXVECTOR3(vPoint[POINT_THREE] -
                        vPoint[POINT_TWO]);
vDirection[POINT_TWO].z = 0.f;
vDirection[POINT_THREE] = D3DXVECTOR3(vPoint[POINT_ONE] -
                        vPoint[POINT_THREE]);
vDirection[POINT_THREE].z = 0.f;
// Calculate Normal Vector
D3DXVECTOR3      vNormal[MAX_POINT] = {
    D3DXVECTOR3(vDirection[POINT_ONE].y, -vDirection[POINT_ONE].x, 0.f),
    D3DXVECTOR3(vDirection[POINT_TWO].y, -vDirection[POINT_TWO].x, 0.f),
    D3DXVECTOR3(vDirection[POINT_THREE].y, -vDirection[POINT_THREE].x, 0.f),
};
//Conduct Dot Product if Player Position is in the Triangle return true
const D3DXVECTOR3* vScroll = CScrollMgr::GetInstance()->GetScroll();
for (int i = 0; i < MAX_POINT; ++i)
{
    D3DXVECTOR3      vPlayerPos = pPlayer->GetInfo()->vPos - *vScroll;
    D3DXVECTOR3      vDestDir = vPlayerPos -
                                D3DXVECTOR3(vPoint[i].x, vPoint[i].y,0.f);

    float fDotResult = D3DXVec3Dot(&vNormal[i], &vDestDir);

    if (fDotResult < 0.f)
        return false;
}
return true;
```

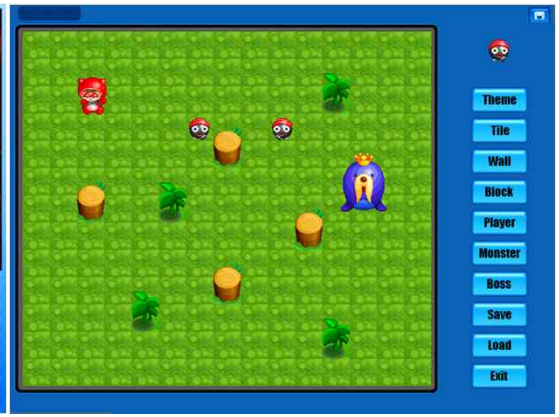
Crazy Arcade

1. 게임소개
2. 기술소개
    1. Design Pattern
        1. Abstract Factory
        2. Singleton

jpl1221@naver.com
010.2723.6769

1. 게임소개

- 게임소개

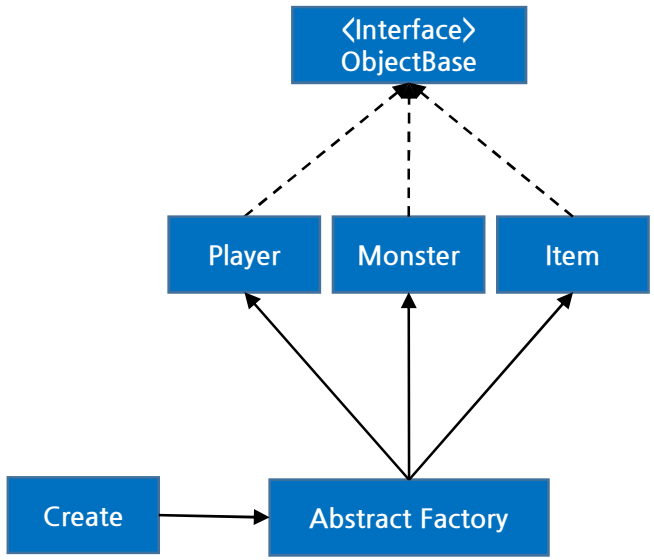


- 게임 이름 : Crazy Arcade
- 게임 장르 : 액션 아케이드
- 구현 언어 : C++
- 개발 환경 : Visual Studio 2015
- 제작 기간 : 2주
- URL : https://youtu.be/FSIEyWOqg5Q
- 게임 특징 : 크레이지 아케이드를 모작하였습니다.

jpl1221@naver.com
010.2723.6769

2. 기술소개 Design Pattern

- Abstract Factory Pattern
  - Diagram



  - Code

```
template <typename T>
class CAbstractFactory
{
public:
    static CObjectBase* CreateObject()
    {
        CObjectBase* pObject = new T;
        pObject->Initialize();

        return pObject;
    }
    static CObjectBase* CreateObject(float x, float y)
    {
        CObjectBase* pObject = new T;
        pObject->SetPos(x+VIEWX, y+VIEWY);
        pObject->Initialize();

        return pObject;
    }
};
```

  - 객체의 생성을 클라이언트가 직접 하지 않고, 간접적으로 수행함으로써 클라이언트가 객체의 생성이나 구성 또는 표현 방식을 독립적으로 만들 수 있습니다.

2. 기술소개 Design Pattern

- Singleton Pattern
  - Code

```
//Singleton Pattern
public:
    static CObjectManager* GetInstance()
    {
        if (m_pInstance == nullptr)
            m_pInstance = new CObjectManager;
        return m_pInstance;
    }
    void DestroyInstance()
    {
        if (m_pInstance)
        {
            delete m_pInstance;
            m_pInstance = nullptr;
        }
    }
private:
    CObjectManager();
    CObjectManager(const CObjectManager&) {};
    CObjectManager& operator = (const CObjectManager&) {};
    ~CObjectManager();
private:
    static CObjectManager* m_pInstance;
```

```
CSceneManager::GetInstance()->Update();
CSoundManager::GetInstance()->Update();

CSceneManager::GetInstance()->DestroyInstance();
CKeyManager::GetInstance()->DestroyInstance();
CBitmapManager::GetInstance()->DestroyInstance();
CObjectManager::GetInstance()->DestroyInstance();
CProbabilityManager::GetInstance()->DestroyInstance();
CSoundManager::GetInstance()->DestroyInstance();
```

- 싱글톤 패턴은 해당 클래스의 객체의 개수를 제한 할 수 있습니다.
- 게임상에서 개수 제한이 필요한 각 매니저들을 구현할 때 싱글톤을 사용했습니다.

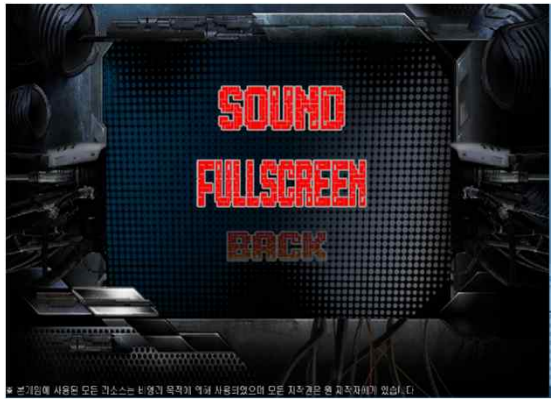
Publis

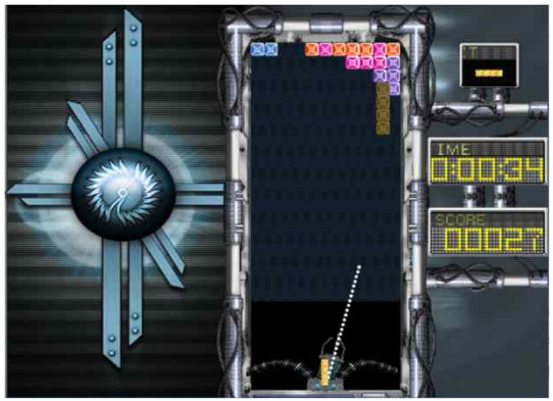
1. 게임소개
2. 콘텐츠 소개

jpl1221@naver.com
010.2723.6769

1. 게임소개

- 게임소개



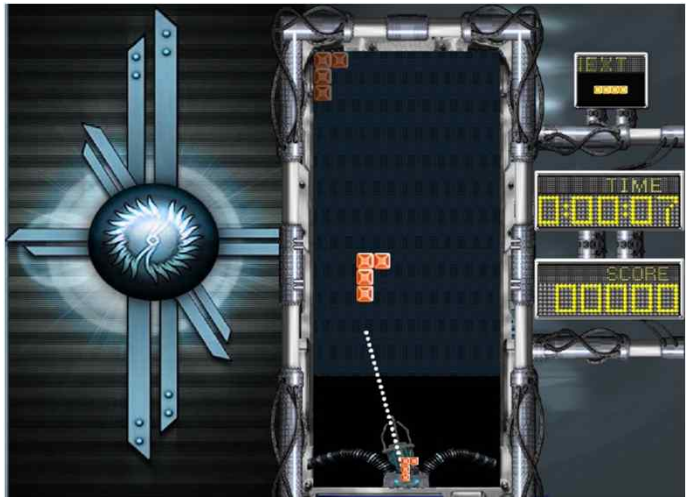


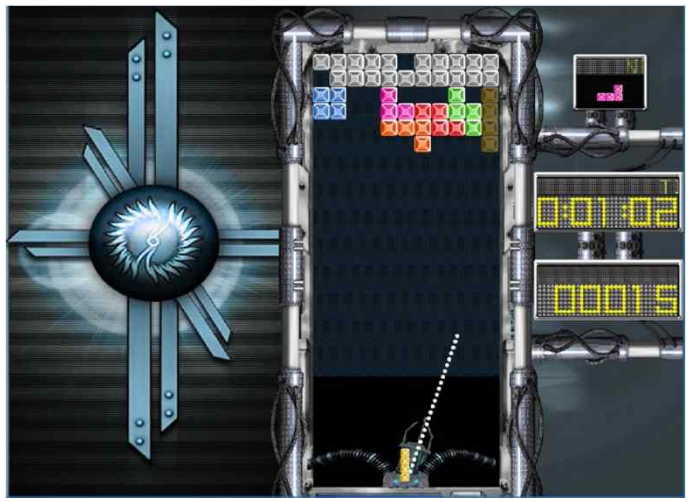
- 게임 이름 : Publis
- 게임 장르 : 퍼즐
- 구현 언어 : C++
- 개발 환경 : Visual Studio 2015,

  DirectX9 SDK Jun
- 제작기간 : 1달
- URL : https://youtu.be/gOTHJezS-tM
- 게임 특징 : 테트리스와 보글보글을 합쳐 개발하였습니다.

1. 게임소개

- 게임소개





- 퍼즐 보글보글의 블록을 보내는 방식과, 테트리스의 게임 방식을 혼합한 퍼즐 게임입니다.
- 방향키로 발사대의 각도를 제어할 수 있습니다.
- 가이드라인과 도착예정지를 확인할 수 있습니다.
- 일정시간이 지나면 방해 블록이 쌓이며, 일정 블록이 쌓이면 게임오버가 됩니다.
- 다음 블록 확인과, 시간, 점수를 확인할 수 있습니다.

jpl1221@naver.com
010.2723.6769

https://youtu.be/gOTHJezS-tM

감사합니다

이장표

E-Mail : jpl1221@naver.com

핸드폰 : 010.2723.6769

URL : https://youtu.be/ZK7z8gRGid0