[기술 소개서]

이름 : 권오현

번호 : 010.2719.5330

이메일 : ohhyun5442@gmail.com

< 목차 >

5. Framework Function

- Picking
  1) Terrain Picking
  2) Mesh Picking

- Navigation
  1) Navigation Mesh
  2) Slide Vector
  3) 평면의 방정식을 이용한 움직임

- Combine Object

- 충돌 처리
  1) AABB
  2) OBB
  3) 원 충돌

- 최적화
  1) Frustum Culling
  2) LOD (Level of Detail)

- Shader
  1) Alpha Blending
  2) 법선(노말) 맵핑
  3) Particle
  4) UV Animation

6. 참고 문헌

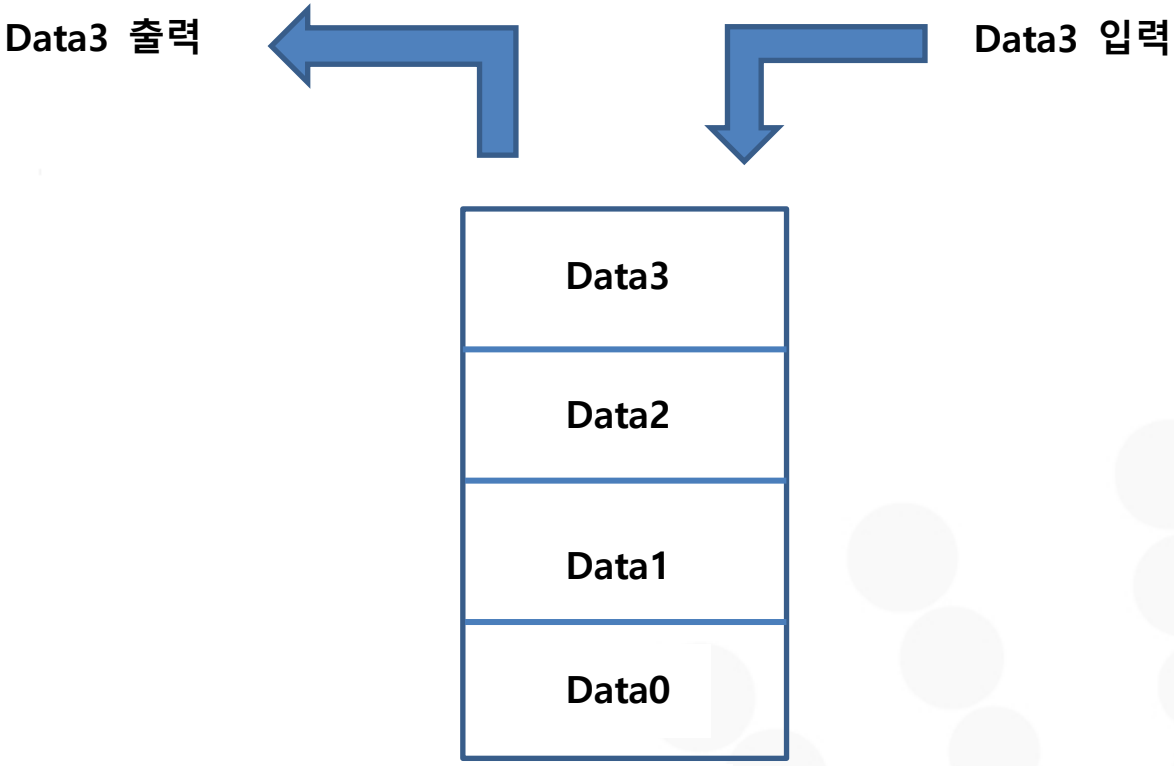
Stack

1. 자료구조

1) 배열을 이용한 스택

- 내용 : 스택의 기본인 후입 선출 (Last In First Out)를 이해하기 위하여 배열을 이용하여 구현하였습니다.

- 이론



< Stack >

- 배열을 이용하여 데이터들을 보관합니다. 처음 입력된 데이터들은 가장 밑에 저장됩니다. 출력되는 데이터는 배열에서 가장 나중에 들어온 데이터를 출력합니다.

- 코드

```
void CStack::Push_Data( )
{
    View_Data( );

    cout << " ========================== " << endl;
    cout << "Data : ";
    cin >> m_iInputData;

    if (false == Full_Data())
    {
        m_pStack.m_iIndex += 1;
        m_pStack.m_iArrStack[m_pStack.m_iIndex] = m_iInputData;

        cout << "Input Data Succed!!" << endl;
    }
    else
    {
        cout << "Input Data Failid!!" << endl;
    }

    system("pause");
}

void CStack::Pop_Data( )
{
    if (true == Empty_Data())
    {
        m_pStack.m_iArrStack[m_pStack.m_iIndex] = 0;
        m_pStack.m_iIndex -= 1;
        View_Data();

        cout << "Delete Data Succed!!" << endl;
    }
    else
    {
        cout << "Delete Data Failid!!" << endl;
    }

    system("pause");
}
```

배열에 값을 채우기 전에 배열이 가득 차 있는지 판별하였습니다.

위에서 배열이 가득 안 찼다고 판별되면 채워 넣습니다.

지우기 전에 배열이 비어 있는지 확인합니다.

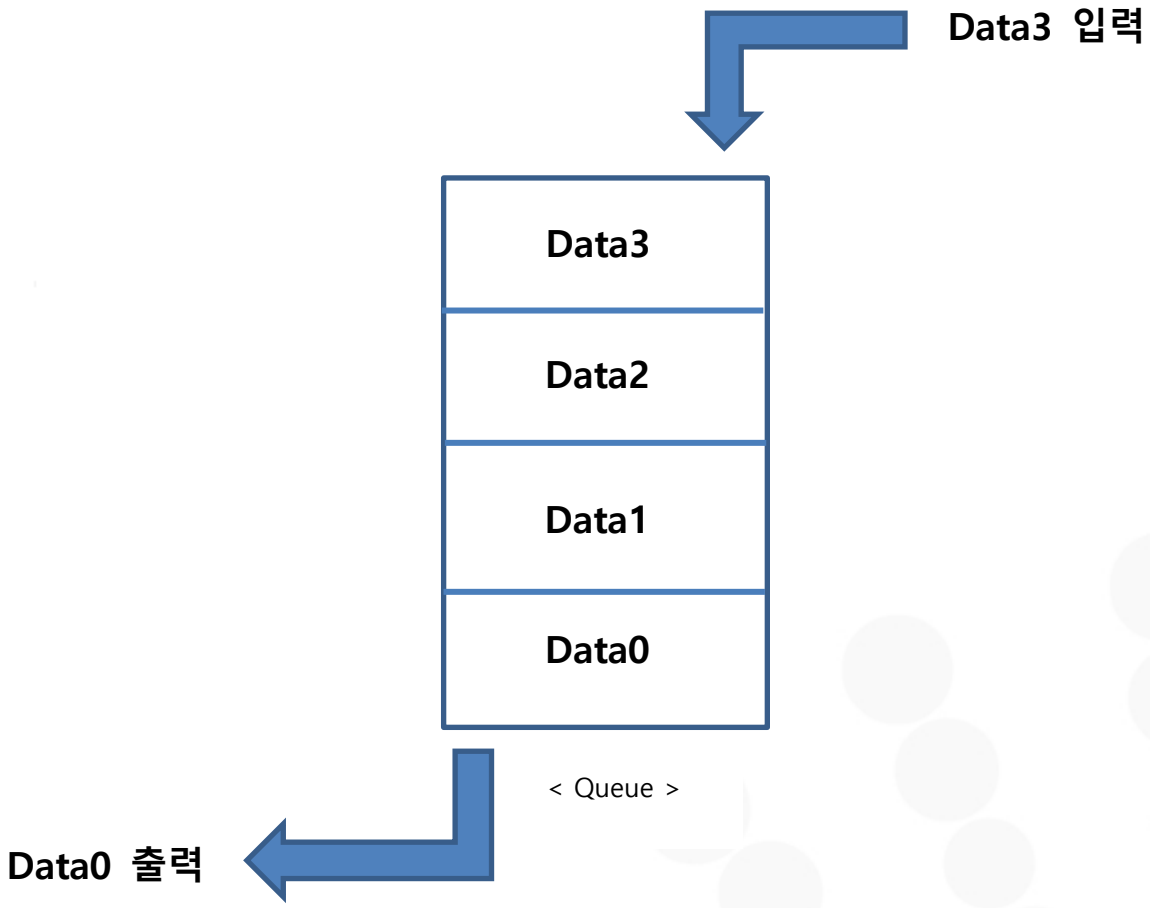
비어있지 않으면 데이터를 초기화 합니다.

- 후입 선출 (Last In First Out)의 기본을 알 수 있었으며 마지막에 들어온 것을 초기화를 하였습니다.

2) 배열을 이용한 큐

- 내용 : 큐의 기본인 선입 선출 (First In First Out)를 이해하기 위하여 배열을 이용하여 구현하였습니다.

- 이론



< Queue >

- 배열을 이용하여 데이터들을 보관합니다. 처음 입력된 데이터들은 가장 밑에 저장됩니다. 그리고 출력되는 데이터는 배열에서 가장 먼저 들어온 데이터를 출력합니다.

- 코드

```
void CQueue::Push_Data()
{
    View_Data();

    cout << " ========================== " << endl;
    cout << "Data : ";
    cin >> m_iInputData;

    if (true == Full_Data())
    {
        cout << "Data is Full!! " << endl;
    }
    else
    {
        m_pQueue.m_iRealIdx += 1;
        m_pQueue.m_iArrQueue[m_pQueue.m_iRealIdx] = m_iInputData;
        cout << "Input Data Succed!!" << endl;
    }

    system("pause");
}
```

배열에 값을 채우기 전에 배열이 가득 차 있는지 판별하였습니다.

위에서 배열이 가득 안 찼다고 판별되면 채워 넣습니다.

```
void CQueue::Pop_Data()
{
    if (true == Empty_Data())
    {
        cout << "Data is Empty!!" << endl;
    }
    else
    {
        m_pQueue.m_iArrQueue[ZERO] = 0;

        for (int i = 0; i < m_pQueue.m_iRealIdx; ++i)
        {
            if( i <= m_pQueue.m_iRealIdx + 1)
                m_pQueue.m_iArrQueue[i] = m_pQueue.m_iArrQueue[i + 1];
        }

        m_pQueue.m_iRealIdx -= 1;

        View_Data();

        cout << "Delete Data Succed!!" << endl;
    }

    system("pause");
}
```

지우기 전에 배열이 비어 있는지 확인합니다.

가장 먼저 들어가 있는 값을 초기화합니다.

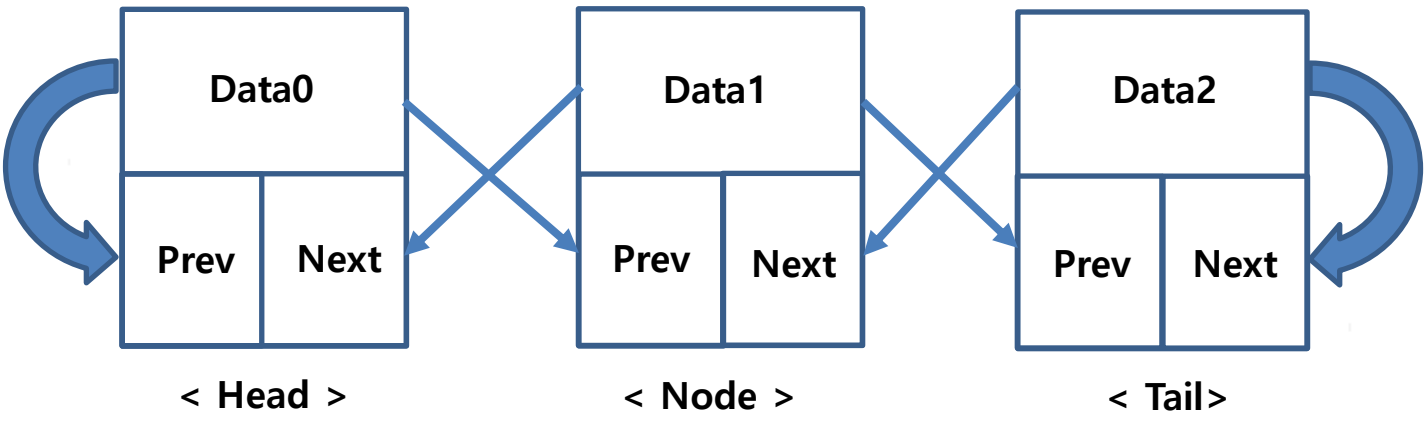
가장 앞의 값이 초기화가 되면 뒤에 있는 값들을 앞으로 한 칸 앞으로 가지고 옵니다.

- 선입 선출 (First In First Out)의 기본을 알 수 있었습니다. 처음 들어온 것을 초기화를 하며 그 뒤에 있는 값들을 한 칸씩 앞으로 가지고 왔습니다.

3) 더블 링크드 리스트

- 내용 : 노드를 앞과 뒤에 존재하는 데이터의 주소를 보관하여 연결을 하였습니다.

- 이론



< Double Linked List >

- 노드가 존재를 하고 있으며 노드는 앞, 뒤의 데이터의 포인터를 보관하며 노드들을 연결 시켰습니다. 연결을 시켜서 단일 링크드 리스트와는 다르게 양방향으로 접근하여 탐색을 할 수 있습니다.

Double Linked List

- 코드

```
void CDoubleList::Insert_Data()
{
    system("cls");

    cout << "Insert Data : ";
    cin >> m_iInputData;

    NODE* pNode = Create_Node(m_iInputData);

    if (m_pTail == m_pHead->Next)
    {
        m_pHead->Next = pNode;

        pNode->Prev = m_pHead;
        pNode->Next = m_pTail;

        m_pTail->Prev = pNode;
    }
    else
    {
        m_pTail->Prev->Next = pNode;

        pNode->Prev = m_pTail->Prev;
        pNode->Next = m_pTail;

        m_pTail->Prev = pNode;
    }

    View_Data();
}
```

노드를 생성합니다.

아무런 노드가 없으면 헤드 노드와 테일 노드에 연결합니다.

연결된 노드가 있으면 꼬리 노드 앞에 생성한 노드를 연결합니다.

```
void CDoubleList::Delete_Data()
{
    View_Data();

    cout << "Delete Data : ";
    cin >> m_iInputData;

    NODE* pSearchNode = m_pHead->Next;

    while ( m_iInputData != pSearchNode->Data && m_pTail != pSearchNode )
    {
        pSearchNode = pSearchNode->Next;
    }

    if (m_pTail == pSearchNode)
        cout << "찾는 데이터가 없습니다." << endl;
    else
    {
        pSearchNode->Prev->Next = pSearchNode->Next;
        pSearchNode->Next->Prev = pSearchNode->Prev;
    }

    View_Data();
}
```

찾으려고 하는 노드의 데이터를 탐색합니다.

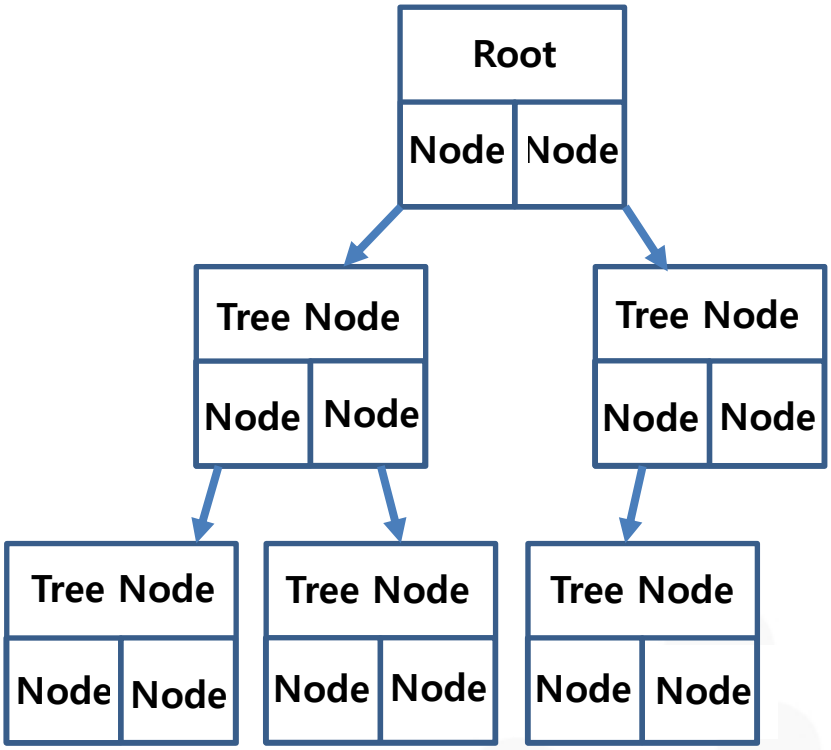
찾는 데이터를 찾으면 찾은 노드를 빼고 앞뒤의 노드를 연결합니다.

- 리스트에서 앞과 뒤쪽을 보관하는 노드를 가지고 있어 서로 연결하면서 삽입, 삭제에 매우 편하게 구현하였습니다.

4) 이진 트리

- 내용 : 이진 트리는 루트부터 자식 노드들이 2개의 노드를 가지고 있어 연결을 하였습니다.

- 이론



< Binary Tree >

- 루트에서부터 각 자식 노드 2개씩 가지고 있어 그 자식들이 또 각자의 자식 노드를 가지고 있습니다. 탐색은 전위 순회, 중위 순회, 후위 순회로 탐색을 할 수 있습니다.

이진 트리

- 코드

```
void CTree::Insert_Tree()
{
    system("cls");
    cout << "Data : ";
    cin >> m_iInputData;

    if ( nullptr == m_pRoot )
    {
        m_pRoot = Create_Tree(m_iInputData);
        return;
    }
    Insert_Node(m_pRoot, m_iInputData);
}
```

루트가 비었으면 채웁니다.

```
void CTree::Insert_Node(TREE* pTree, _int iData)
{
    TREE* pTree = Create_Tree(iData);

    while ( nullptr != pTree )
    {
        pTree->Parent = pTree;

        if (iData < pTree->Data)
            pTree = pTree->Left;
        else
            pTree = pTree->Right;
    }

    if (iData < (pTree->Parent)->Data)
        (pTree->Parent)->Left = pTree;
    else
        (pTree->Parent)->Right = pTree;
}
```

탐색하려고 하는 노드가 비어 있으면 넣으려고 하는 데이터의 정보와 비교하여 왼쪽과 오른쪽에 넣을지 판단하여 넣습니다.

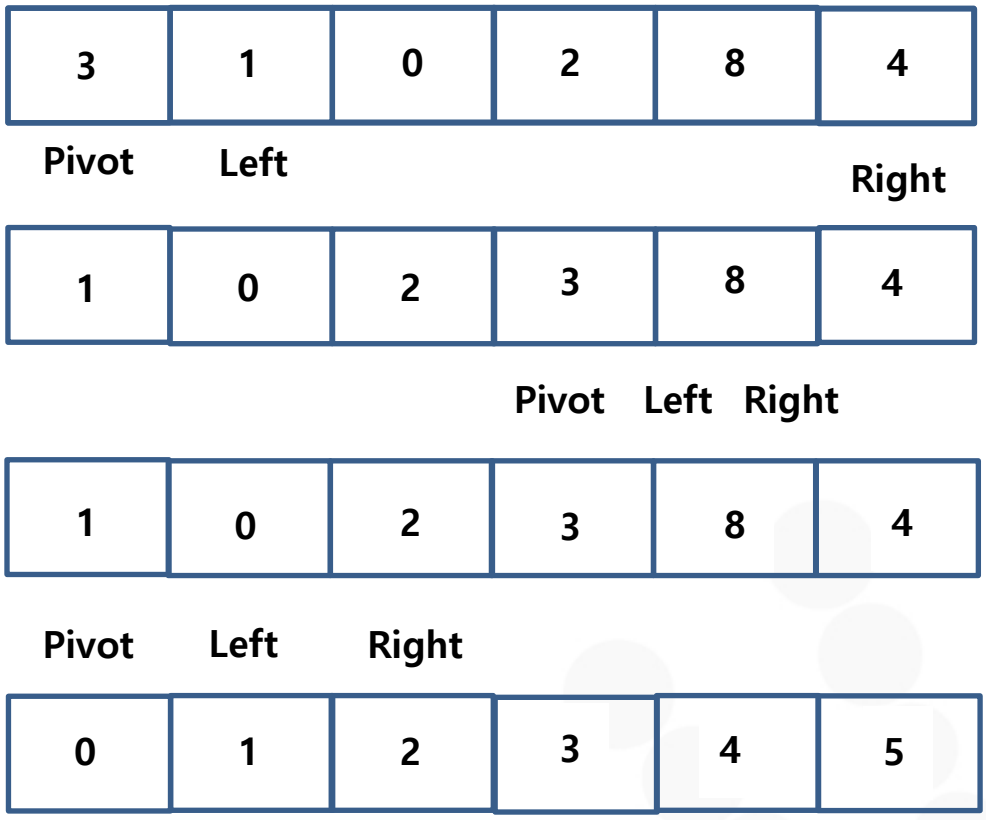
- 데이터는 값이 크면 클수록 오른쪽에 보관이 되도록 하였습니다. 데이터를 보관한 Tree는 부모의 Left와 Right에 보관을 하도록 합니다.

2. 알고리즘

1) 퀵 정렬을 이용한 배열 정리

- 내용 : 정렬 중에서 가장 빠르게 배열을 오름차순이나 내림차순으로 정렬을 하도록 하였습니다.

- 이론



< Quick Sort >

- 정렬 하고 싶은 원소들을 Pivot을 기준하여 오른쪽과 왼쪽으로 분할하여 Pivot보다 작으면 왼쪽 크면 오른쪽으로 바꿔가며 정렬합니다.

퀵 정렬을 이용한 배열정리

- 코드

```
void CQuickSort::QuickSort(int* arr, int left, int right)
{
    int i, j, temp;

    int pivot = arr[left];

    if (left < right)
    {
        i = left + 1 ;
        j = right;

        while (i <= j)
        {
            while (arr[i] < pivot)
                i++;
            while (arr[j] > pivot)
                j--;

            if (i < j)
            {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
            else
                break;
        }

        temp = arr[j];
        arr[j] = arr[left];
        arr[left] = temp;

        QuickSort(arr, left, j - 1);
        QuickSort(arr, j + 1, right);
    }
}
```

오름차순 정렬을 하고 있으며 왼쪽은 기준 값보다 큰 값을 오른쪽은 기준 값보다 작은 값을 찾고 있습니다.

양쪽에서 탐색하는 기준 들이 만나기 전에 작은 값들을 찾으면 값을 바꿔준다.

Pivot의 오른쪽과 왼쪽을 각각 재귀로 탐색하여 정렬시킵니다.

- Pivot이라는 기준점으로 각 원소들을 재귀를 통하여 탐색하며 정렬을 하였습니다. 처음 Pivot값은 가장 왼쪽으로 시작을 기준으로 하였습니다. 비교하는 부호에 따라 오름차순정렬과 내림차순 정렬을 가능하도록 하였습니다.

2) A* 알고리즘

- 내용 : 타일맵에서 가고 싶은 위치를 피킹을 하면 그 위치까지 가장 적은 코스트를 가진 길을 찾아 움직이기 위하여 사용하였습니다.

- 이론

타일을 피킹 하면 현재위치부터 피킹한 위치까지의 최단거리를 구합니다.

최단거리로 그 위치로 갑니다. 만약 가지 못하는 타일의 경우는 검사하지 않습니다.

< AStar >

A* 알고리즘

- 코드

```
while(true)
{
    //탐색 : 위
    iIndex = pFirstNode->iIndex - TILEX * 2;

    // 타일  밖으로 나갈경우
    if ( 0 > iIndex || TILEX * TILEY < iIndex)
        return;

    if(pFirstNode->iIndex >= TILEX * 2  &&  1 == m_SelectY &&
        ListCheck(iIndex) && 0 == (*pVecTile)[iIndex]->byOption ||
        2 == (*pVecTile)[iIndex]->byOption || 3 == (*pVecTile)[iIndex]->byOption)
    {
        pNode = MakeNode(iIndex, pFirstNode, pVecTile, WAY_UP);
        m_OpenList.push_back(pNode);
    }
```

타일의 8방향을 확인을 비슷하게 체크하여 OpenList에 넣습니다.

// 남은 7 방향 체크

```
    //OpenList에 담긴 비용을 정렬한다.
    m_OpenList.sort( Compare );

    //가장 적은 비용의 노드(타일을)
    list<NODE*>::iterator iter = m_OpenList.begin();
    pFirstNode = *iter;

    m_CloseList.push_back(*iter);
    m_OpenList.erase(iter);

    if( m_iEndIndex ==  pFirstNode->iIndex )
    {
        //BestList 구하기
        while(true)
        {
            m_BestList.push_back( pFirstNode->iIndex );
            pFirstNode = pFirstNode->pParent;

            if(pFirstNode->iIndex == m_iStartIndex)
                break;
        }

        //원소를 반전시킨다.
        m_BestList.reverse();
        break;
    }
}
```

코스트에 따라 정렬합니다

OpenList에 담긴 노드가 EndIndex까지 올 때까지 순회하며 CloseList에 넣습니다.

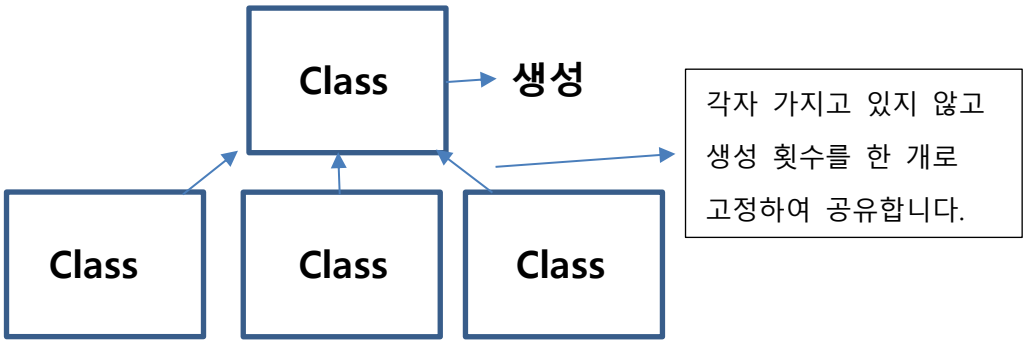
- 플레이어의 위치부터 탐색한 노드의 코스트가 가장 가까운 코스트를 가진 노드를 모아서 최종적으로 BestList에 보관합니다. 이 List에 들어간 노드의 중심 좌표를 따라 움직이도록 하였습니다.

3. 디자인 패턴

1) 싱글톤 (SingleTon)

- 내용 : 클래스 생성의 횟수를 단 한번만 할당 하도록 하기 위하여 사용하였습니다.

- 이론



< Singletone >

- 코드

```
#define DECLARE_SINGLETON(CLASSNAME)                ₩
    NO_COPY(CLASSNAME)                              ₩
    private:                                        ₩
    static CLASSNAME*    m_pInstance;               ₩
    public:                                         ₩
    static CLASSNAME*    GetInstance( void );       ₩
    static unsigned long DestroyInstance( void );
```

```
#define IMPLEMENT_SINGLETON(CLASSNAME)              ₩
    CLASSNAME*  CLASSNAME::m_pInstance = NULL;      ₩
    CLASSNAME*  CLASSNAME::GetInstance( void ) {    ₩
    if(NULL == m_pInstance) {                       ₩
    m_pInstance = new CLASSNAME;                    ₩
    }                                               ₩
    return m_pInstance;                             ₩
    }                                               ₩
    unsigned long CLASSNAME::DestroyInstance( void ) {      ₩
    unsigned long dwRefCnt = 0;                     ₩
    if(NULL != m_pInstance) {                       ₩
    dwRefCnt = m_pInstance->Release( );             ₩
    if(0 == dwRefCnt)                               ₩
    m_pInstance = NULL;                             ₩
    }                                               ₩
    return dwRefCnt;                                ₩
    }
```

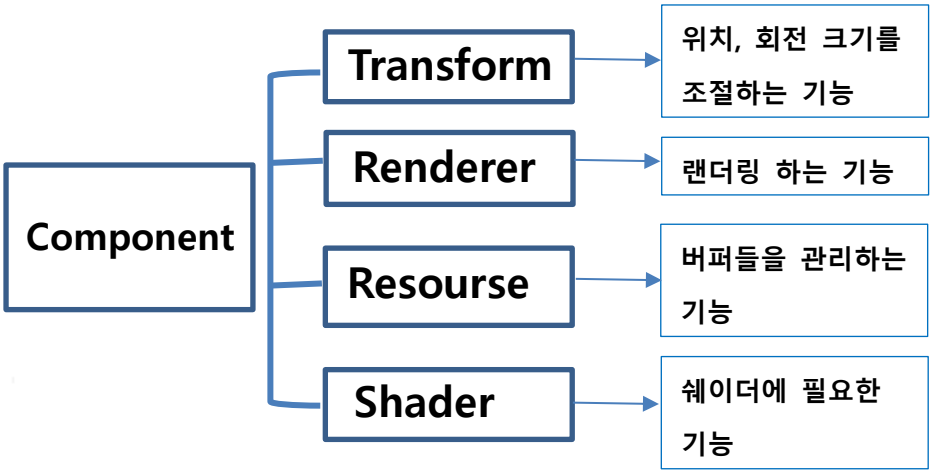
클래스가 생성되어
있는지 판별합니다.
생성되어 있지 않으면
생성하고 아니면
생성하지 않습니다.

- 싱글톤 생성을 Define으로 잡아놓고 생성 횟수를 제한하고 싶은 클래스들을 생성할 때 호출하여 횟수를 제한하도록 합니다.

2) 컴포넌트 패턴 (Component Pattern)

- 내용 : 필요한 공통된 기능들을 Component로 묶어서 사용을 하였습니다.

- 이론



< Component Pattern >

- 코드

```
HRESULT CPlayer::Ready_Component()
{
    CComponent*         pComponent = nullptr;

    // For.Material
    pComponent = m_pMaterialCom = (Engine::CMaterial*)CComponent_Manager::GetInstance()->Clone_Component(e_CurType, L"Component_Material");
    if (nullptr == pComponent)
        return E_FAIL;
    m_mapComponent[Engine::CComponent::TYPE_STATIC].insert(MAPCOMPONENT::value_type(L"Com_Material", pComponent));
    m_pMaterialCom->AddRef();

    // For.Transform
    pComponent = m_pTransformCom = (Engine::CTransform*)CComponent_Manager::GetInstance()->Clone_Component(SCENE_STATIC, L"Component_Transform");
    if (nullptr == pComponent)
        return E_FAIL;
    m_mapComponent[Engine::CComponent::TYPE_STATIC].insert(MAPCOMPONENT::value_type(L"Com_Transform", pComponent));
    m_pTransformCom->AddRef();

    // For.Buffers
    pComponent = m_pBufferCom = (Engine::CDynamic_Mesh*)CComponent_Manager::GetInstance()->Clone_Component(SCENE_STATIC, L"Component_Mesh_Player");
    if (nullptr == pComponent)
        return E_FAIL;
    m_mapComponent[Engine::CComponent::TYPE_STATIC].insert(MAPCOMPONENT::value_type(L"Com_Buffer", pComponent));
    m_pBufferCom->AddRef();

    // For.Renderer
    pComponent = m_pRendererCom = (Engine::CRenderer*)CComponent_Manager::GetInstance()->Clone_Component(SCENE_STATIC, L"Component_Renderer");
    if (nullptr == pComponent)
        return E_FAIL;
    m_mapComponent[Engine::CComponent::TYPE_STATIC].insert(MAPCOMPONENT::value_type(L"Com_Renderer", pComponent));
    m_pRendererCom->AddRef();
```

- 사용하고 싶은 기능들을 Component로 묶어놓은 것을 미리 생성을 해둡니다.

미리 생성한 Component들을 복제하여 map에 보관합니다.

3) 이터레이터 패턴 (Iterator Pattern)

- 내용 : 보관하고 있는 데이터들을 노출하지 않고 반복자를 이용하여 모든 항목을 순회하기위하여 사용하였습니다.

- 이론

< Iterator Pattern >

- 코드

```
_int CLayer::Update_Layer(const _float & fTimeDelta)
{
    _int iExitCode = 0;

    auto& iter = m_ObjectList.begin();
    auto& iter_end = m_ObjectList.end();

    for ( ; iter != iter_end ; )
    {
        iExitCode = (*iter)->Update_GameObject(fTimeDelta);

        if (1 == iExitCode) /*Object Is Dead*/
        {
            Safe_Release((*iter));
            iter = m_ObjectList.erase(iter);
        }
        else if (iExitCode & 0x80000000) /* exit code < 0*/
        {
            return iExitCode;
        }
        else
            ++iter;
    }

    return iExitCode;
}
```

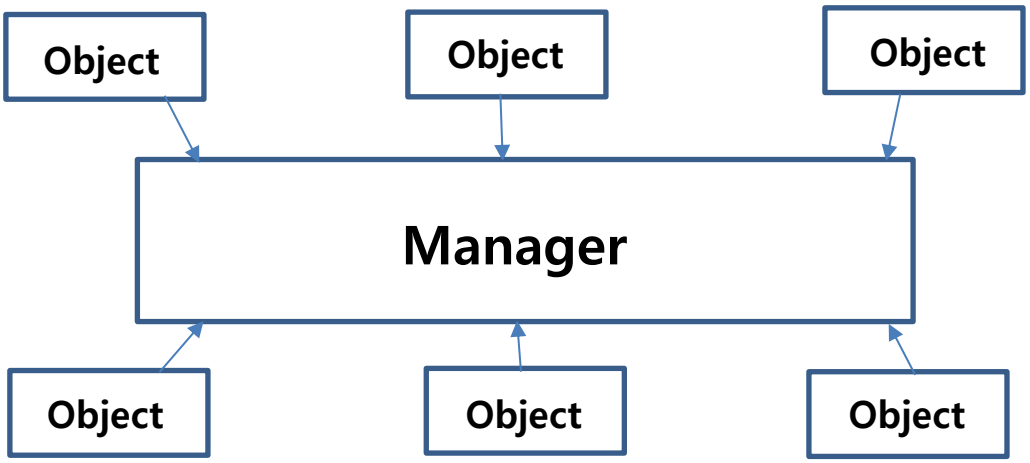
Layer에서 보관하고 있는 Object들의 Update와 Render를 반복자를 통하여 순회하였습니다.

- 집합체로 묶어두고 데이터들을 순회하면서 공통적으로 호출해야 하는 함수를 호출을 하도록 하였습니다.

4) 메디에이터 패턴 (Mediator Pattern)

- 내용 : 여러 객체들이 서로 참조를 해야하는 것을 Manager를 구성하여 한 곳에서 서로 참조를 하도록 하였습니다.

- 이론



< Mediator Pattern >

- 코드

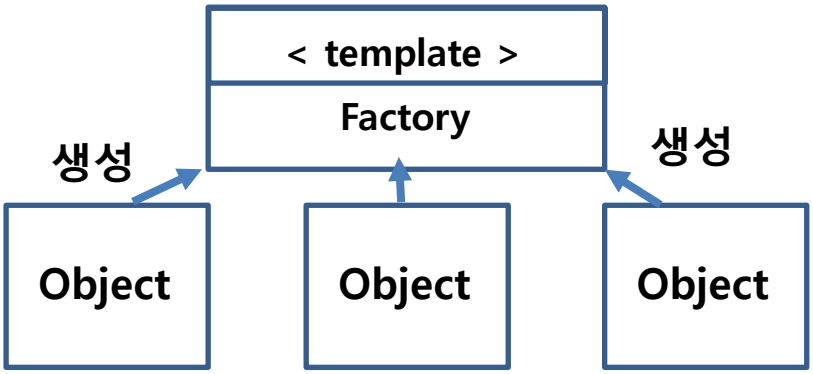
```
void CCollider_Manager::CollisionGroupToGroup(COLLIDERGROUP* pDstColliderGroup, COLLIDERGROUP* pSrcColliderGroup)
{
    COLLIDERDATA* pDstColliderData = nullptr;
    COLLIDERDATA* pSrcColliderData = nullptr;

    COLLISIONDATA CollisionData;

    for (_uint i = 0; i < pDstColliderGroup->iCollisionDataCount; ++i)
    {
        pDstColliderData = &pDstColliderGroup->pColliderData[i];
        for (_uint j = 0; j < pSrcColliderGroup->iCollisionDataCount; ++j)
        {
            ZeroMemory(&CollisionData, sizeof(COLLISIONDATA));
            pSrcColliderData = &pSrcColliderGroup->pColliderData[j];

            // 실제 충돌 검사
            if (true == pDstColliderData->pCollider->Collision_Sphere(pSrcColliderData->pCollider, CollisionData))

                                    ***** 중략 *****

                // 충돌 되었다면 충돌 했다는 메시지를 충돌한 콜라이더들에게 전달
                OnCollisionEvent(pDstColliderData, pSrcColliderData, eCollisionType, CollisionData);

                return;
            }
        }
    }
}
```

- Collider Manager를 구성하여 충돌 시키려고 하는 Collider들을 Manager를 통하여 서로 참조를 하도록 구성하였습니다.

5) 팩토리 패턴 (Factory Pattern)

- 내용 : 객체들을 생성할 때 생성하는 과정을 묶어 캡슐화하여 사용하였습니다.

- 이론



< Factory Pattern >

- 코드

```
template<class T>
class CFactory abstract
{
public:
    // Base Object
    static CGameObject* CreateGameObject( )
    {
        CGameObject* pObject = new T;
        pObject->Initialize( );
        return pObject;
    }

    static CGameObject* CreateGameObject(D3DXVECTOR3 vPos)
    {
        CGameObject* pObject = new T;
        pObject->Initialize( )   ;
        pObject->SetPos(vPos);

        return pObject;
    }
```

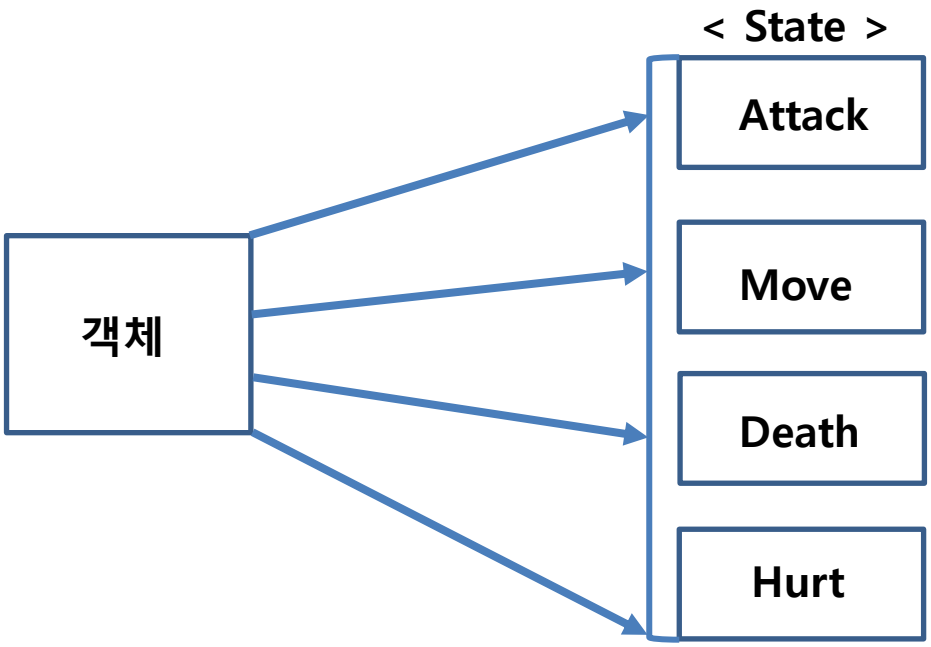
타입이 정해져 있지않은 객체를 생성을 해야 해서 Templete으로 묶어둔 함수로 동적할당하여 생성합니다.

- 기본적인 것만 생성하는 경우나 위치 정보를 넘겨주면서 생성하는 경우마다 나눠 둡니다. 필요한 기능이 있는 함수를 생성하여 객체를 생성하였습니다.

6) 스테이트 패턴 (State Pattern)

- 내용 : 객체들을 FSM(유한 상태 기계) 따라 상태를 변화시켜 행동을 제어하였습니다.

- 이론



< State Pattern >

- 코드

```
if (m_eNewState != m_eCurrState)
{
    switch (m_eNewState)
    {
    case STATE_STAND:
        m_pBufferCom->Set_AnimationSet(22);
        break;
    case STATE_FORWARD:
        m_pBufferCom->Set_AnimationSet(12);
        break;
    case STATE_BACKWORD:
        m_pBufferCom->Set_AnimationSet(13);
        break;
    case STATE_LEFT:
        m_pBufferCom->Set_AnimationSet(11);
        break;
    case STATE_RIGHT:
        m_pBufferCom->Set_AnimationSet(10);
        break;
    case STATE_SPAWN:
        m_fDelayTime = 0.8f;
        m_bMotion = true;
        m_pBufferCom->Set_AnimationSet(9);
        break;
```

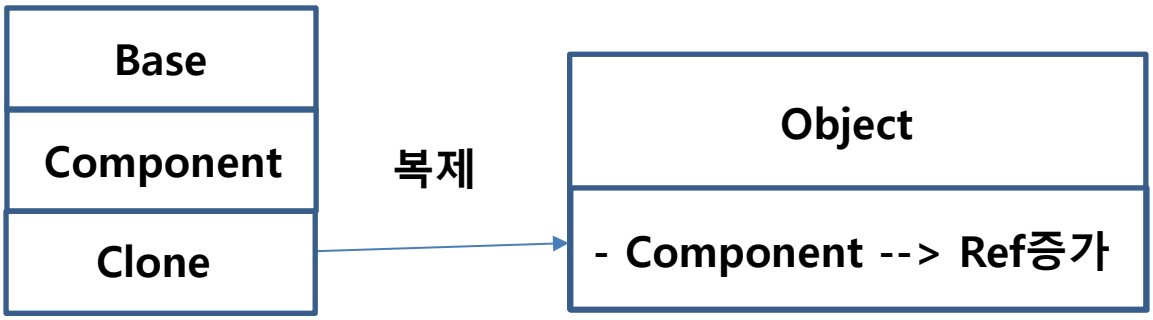
이전 상태와 다음 상태가 다른 경우에 상태를 변화시켰습니다.

- 현재 상태와 다음 상태와 다르면 다음 상태로 셋팅하여 필요한 애니메이션과 움직이는 동안 필요한 값들을 셋팅하였습니다.

7) 프로토타입 패턴 (Prototype Pattern)

- 내용 : 객체들을 미리 생성해 놓은 것을 복제를 하여 사용하였습니다.

- 이론



< Prototype Pattern >

- 코드

```
class DLL_EXPORT CRenderer final : public CComponent
{
public:
    enum RENDER {RENDER_PRIORITY, RENDER_NONALPHA, RENDER_ALPHA, RENDER_UI, RENDER_END};
private:
    explicit CRenderer(LPDIRECT3DDEVICE9 pGraphicDev);
    virtual ~CRenderer( ) = default;

                ***** 중략 *****

private:
    void Clear_RenderGroup(void);
public:
    virtual CComponent* Clone(void) final;
    static CRenderer* Create(LPDIRECT3DDEVICE9 pGraphicDev);
private:
    virtual _ulong Free(void);
};
```

클래스를 생성을 미리 하여 이 함수를 통하여 복제생성 합니다.

- 사용하지 않으면 객체들이나 컴포넌트를 필요할 때 생성을 해야하는 수고를 들여야 합니다. 하지만 이 프로토 타입 패턴을 사용하면 미리 생성하여 Clone을 하기만 하면 그렇게 큰 비용이 들지 않고 생성할 수 있습니다.

프로토타입 패턴

```
class _declspec(dllexport) CBase abstract
{
private:
    inline unsigned long MakeKey(void); //생성할때 사용하는 함수(사용금지)
protected:
    inline explicit CBase(void);
    inline virtual ~CBase(void);
public:
    inline unsigned long GetRefCnt(void) const; // 현재 레퍼런스 카운트 반환
    inline unsigned long GetKey(void) const; // 객체 고유 Key 반환
    inline unsigned long AddRef(void);
    inline unsigned long Release(void);

private:
    unsigned long   m_dwRefCnt = 0;
    unsigned long   m_dwKey = 0;

public:
    virtual unsigned long Free(void) = 0;
```

- Base클래스로 모든 클래스에 상속을 받게 하여 RefCnt를 관리 하였습니다.

```
CComponent * CRenderer::Clone(void)
{
    AddRef();

    return this;
}
```

```
unsigned long CBase::AddRef(void)
{
    return ++(this->m_dwRefCnt);
}
```

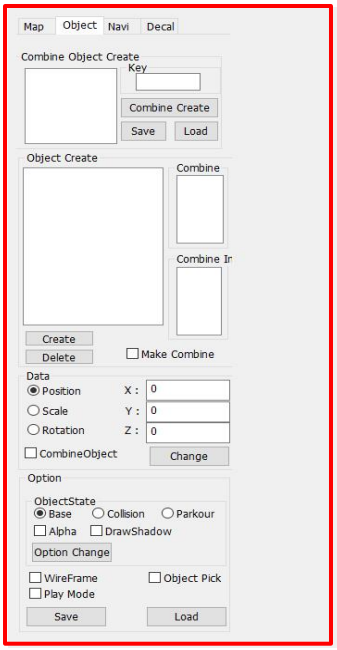
Clone을 할 때 Reference Count를 관리하였습니다.

- 모든 객체들을 Base클래스를 상속 받으며 상속 받은 객체들을 Clone함수를 가지고 있습니다. Clone을 할 때 RefCnt를 관리하여 참조한 개수를 관리 하였습니다. 참조할 때마다 RefCnt를 증가시키고 삭제할 때마다 RefCnt를 감소시켰습니다.

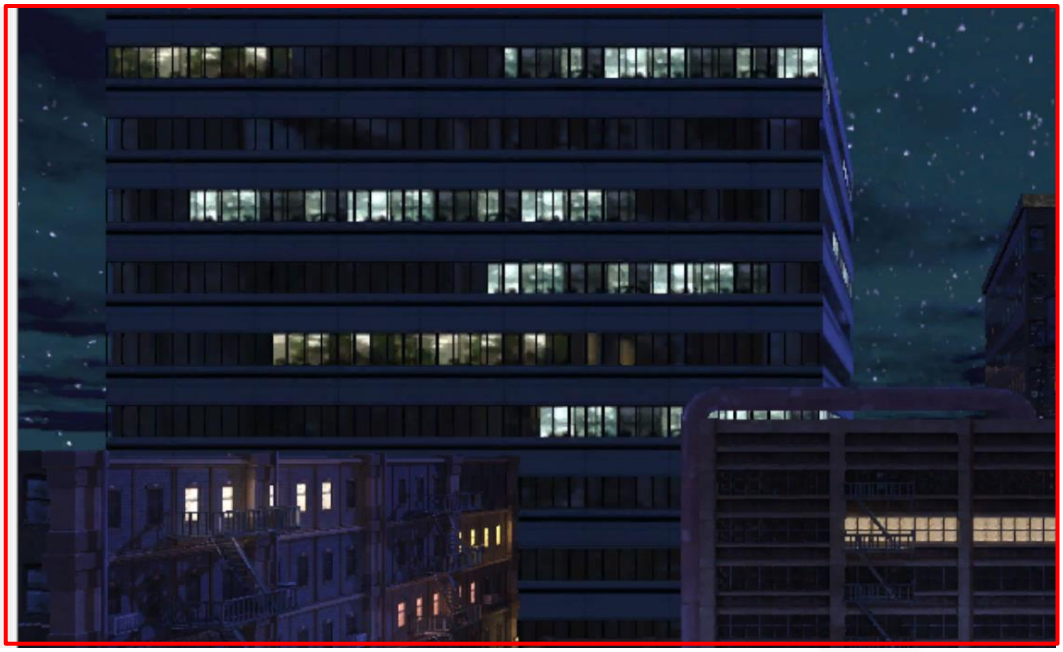
4. MFC

1) Form Control을 화면분할

- 내용 : MainView와 나눠 설정하는 부분과 보는 부분을 나눠 바로 적용할 수 있도록 하였습니다.



< Form View >



< MainView >

- 코드

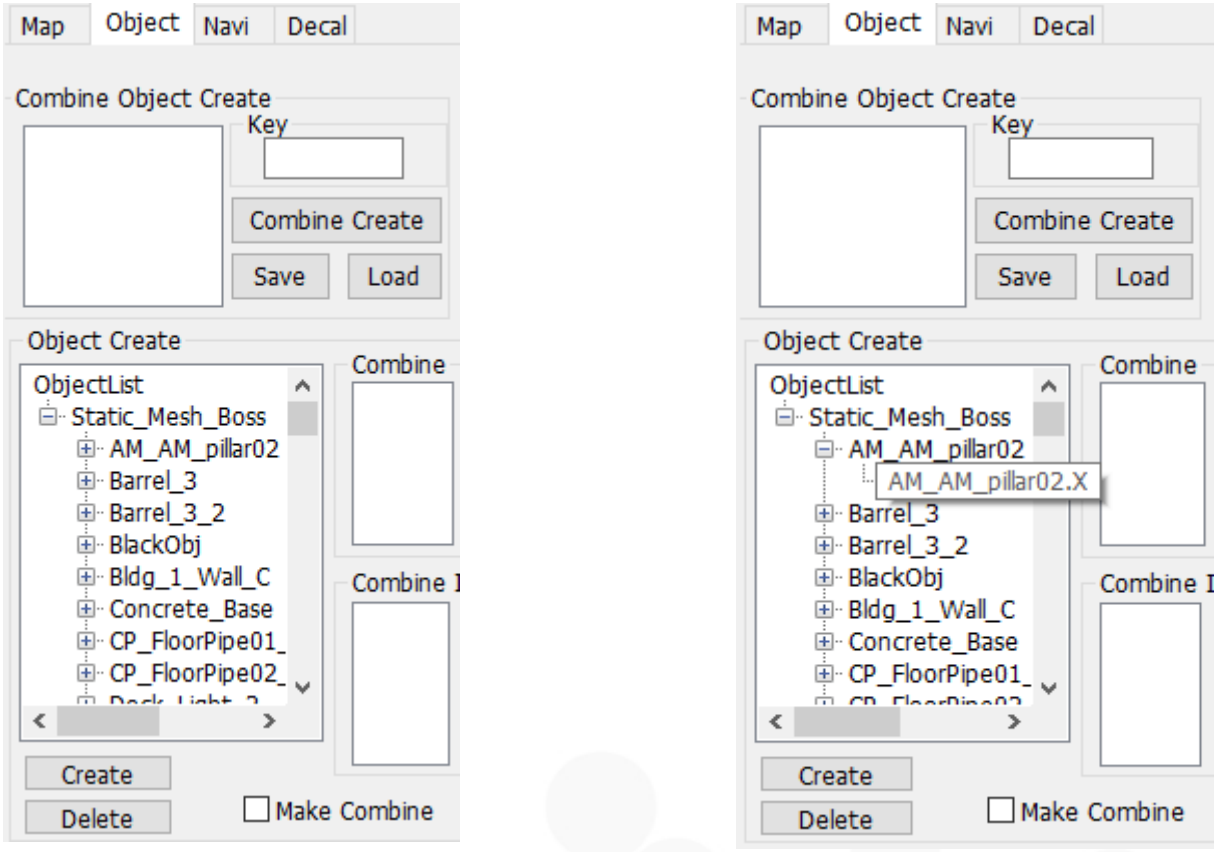
```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)
{
    // TODO: 여기에 특수화된 코드를 추가 및/또는 기본 클래스를 호출합니다.

    CRect rect;
    GetClientRect(&rect);

    if (!m_MainSplitter.CreateStatic(this, 1, 2))
        return FALSE;

    //int iSizeX = rect.right - rect.left;
    int iSizeX = BACKCX;
    int iMenuSize = iSizeX / 5;

    if (!m_MainSplitter.CreateView(0, 0, RUNTIME_CLASS(CMainForm), CSize(iMenuSize, BACKCY), pContext))
        return FALSE;

    if (!m_MainSplitter.CreateView(0, 1, RUNTIME_CLASS(CMapTool_KOHView), CSize(BACKCX, BACKCY), pContext))
        return FALSE;

    m_pMapForm = (CMainForm*)m_MainSplitter.GetPane(0, 0);
    g_pView = m_pMapView = (CMapTool_KOHView*)m_MainSplitter.GetPane(0, 1);


    return TRUE;
}
```

Client를 2개로 나눠 생성합니다.

생성한 곳에 Form과 View를 생성합니다.

2) Tree Control을 이용한 파일 보관

- 내용 : 오브젝트들을 폴더로 파일들을 보관하여 오브젝트를 찾기 쉽게 하였습니다.



< Tree Control >

- Object들을 폴더로 파일들을 관리하여 폴더 별로 로드를 하였습니다. 로드를 하면 ObjectList를 Root로 잡고 파일의 폴더의 이름을 노드로 설정합니다. 각 노드 밑에 오브젝트의 파일 이름을 노드로 잡아 파일들을 찾기 쉽게 만들었습니다..

Tree Control을 이용한 파일 보관

- 코드

```
void CTab2::OnDropFiles(HDROP hDropInfo)
{
    // TODO: 여기에 메시지 처리기 코드를 추가 및/또는 기본값을 호출합니다.

    CDialog::OnDropFiles(hDropInfo);

    UpdateData(TRUE);

    int iFileCount = DragQueryFile(hDropInfo, -1, NULL, 0);

    TCHAR szFullPath[MAX_PATH] = L"";

    for (int i = 0; i < iFileCount; ++i)
    {
        DragQueryFile(hDropInfo, i, szFullPath, MAX_PATH);

        DirectoryInfoExtraction_XFile(szFullPath, m_listImagePath);
    }
```

Drag and Drop을 통하여 넣은 폴더에서 파일명과 폴더명을 받아옵니다.

```
***** 중략 *****

    for (list<IMAGE_PATH*>::iterator iter = m_listImagePath.begin();
        iter != m_listImagePath.end(); ++iter)
    {

***** 중략 *****

        m_pView->Ready_Mesh((*iter)->wstrPath.c_str(), (*iter)->wstrFileName.c_str(), (*iter)->wstrComName.c_str());

        if (curNode.compare((*iter)->wstrObjKey))
        {
            curNode = (*iter)->wstrObjKey;
            tNode[++nodeCount] = m_ObjTreeCtl.InsertItem(((*iter)->wstrObjKey).c_str(), root, TVI_LAST);
        }
        if (curNode2.compare((*iter)->wstrStateKey))
        {
            curNode2 = (*iter)->wstrStateKey;
            tNode2[++nodeCount2] = m_ObjTreeCtl.InsertItem(((*iter)->wstrStateKey).c_str(), tNode[nodeCount], TVI_LAST);
        }
        m_ObjTreeCtl.InsertItem(((*iter)->wstrFileName).c_str(), tNode2[nodeCount2], TVI_LAST);
    }

    Safe_Delete_Array(tNode);
    Safe_Delete_Array(tNode2);

    UpdateData(FALSE);
}
```

Drag and Drop으로 넣은 파일들을 로드합니다.

위에서 구한 정보들을 노드에 추가합니다.

- 첫 번째 노드는 StaticMesh와 DynamicMesh로 나눠집니다. 그 밑으로 파일들을 보관하는 폴더를 노드로 잡았습니다. 그리고 그 노드 안에 파일명을 가지고 있어 파일명을 가지고 Object들을 생성할 수 있게 구현하였습니다.

5. Framework Function

1) Picking

- Terrain Picking

- 내용 : 지형에 오브젝트를 설치하거나 Height조절, Spletting을 하기 위하여 사용하였습니다.

- 코드

```
bool CPicking::Picking_ToBuffer(HWND hWnd, _vec3* vOutPos, Engine::CVIBuffer * pTargetBuffer, const _matrix* pWorldMatrix)
{
    if (nullptr == m_pGraphicDev || nullptr == vOutPos || nullptr == pTargetBuffer)
        return false;

    POINT    ptMouse;

    GetCursorPos(&ptMouse);

    ScreenToClient(hWnd, &ptMouse);

    _vec3           vMousePos(0.f, 0.f, 0.f);

    D3DVIEWPORT9        ViewPort;
    m_pGraphicDev->GetViewport(&ViewPort);
    vMousePos.x = ptMouse.x / (ViewPort.Width * 0.5f) - 1.f;
    vMousePos.y = ptMouse.y / -(ViewPort.Height * 0.5f) + 1.f;
    vMousePos.z = 0.0f;

    _matrix         matProj;
    m_pGraphicDev->GetTransform(D3DTS_PROJECTION, &matProj);
    D3DXMatrixInverse(&matProj, nullptr, &matProj);
    D3DXVec3TransformCoord(&vMousePos, &vMousePos, &matProj);

    _vec3           vRay(0.0f, 0.0f, 0.0f), vPivot(0.0f, 0.0f, 0.0f);
    vRay = vMousePos - vPivot;

    _matrix         matView;
    m_pGraphicDev->GetTransform(D3DTS_VIEW, &matView);
    D3DXMatrixInverse(&matView, nullptr, &matView);
    D3DXVec3TransformCoord(&vPivot, &vPivot, &matView);
    D3DXVec3TransformNormal(&vRay, &vRay, &matView);
```

현재 Screen의 마우스 좌표를 가지고 옵니다.

구한 마우스 좌표를 월드 스페이스까지 변화시켜줍니다.

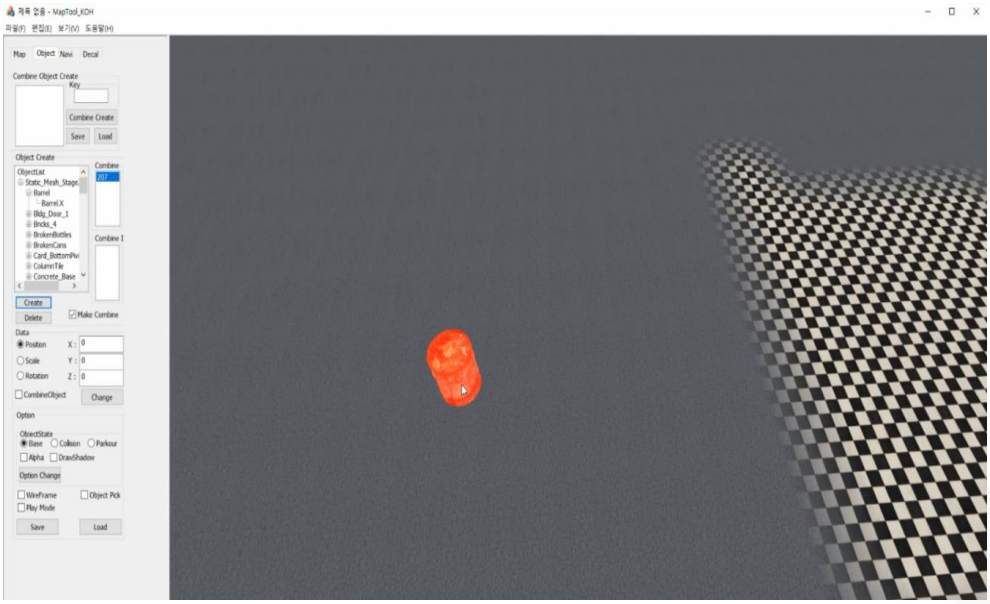
***** 생략 *****

Terrain Picking

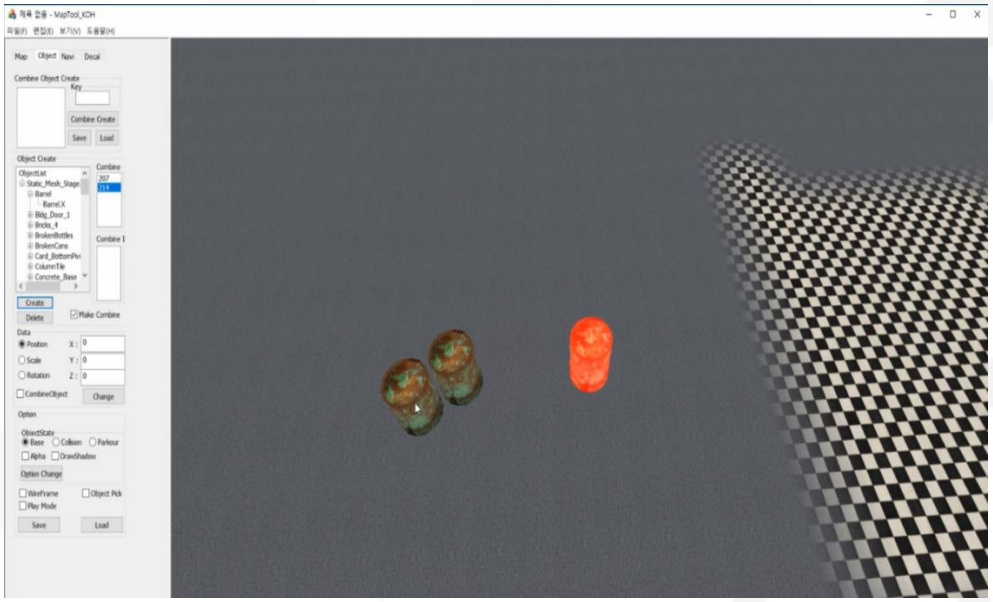
```
// right top
if (TRUE == D3DXIntersectTri(&pVertexPos[iRT], &pVertexPos[iLT], &pVertexPos[iRB], &vPivot, &vRay
    , &fU, &fV, &fDist))
{
    *vOutPos = vPivot + vRay * fDist;
    return true;
}

// left bottomK
if (TRUE == D3DXIntersectTri(&pVertexPos[iLB], &pVertexPos[iRB], &pVertexPos[iLT], &vPivot, &vRay
    , &fU, &fV, &fDist))
{
    *vOutPos = vPivot + vRay * fDist;
    return true;
}
```

D3DXIntersectTri함수를. 이용하여 지형과 Ray Picking을 하여 좌표를 구합니다.

- 클라이언트 내 영상



< 3D Team Project >



< 3D Team Project >

- Ray를 구하여 지형과 충돌되는 좌표를 구하여 현재 마우스의 좌표를 구하였습니다.

- Mesh Picking

- 내용 : 생성한 Object의 위치, 크기. 회전을 수정하기 위하여 사용하였습니다.

- 코드

```
LPD3DXMESH          pMesh = *ppMesh;
pMesh->AddRef( );

BOOL bHit = false;
_float      fU = 0.f, fV = 0.f, fDist = 0.f;
_ulong dwNumFaces = 0;

D3DXIntersect(pMesh, &vPivot, &vRay, &bHit, &dwNumFaces, &fU, &fV, &fDist, nullptr, nullptr);

Safe_Release(pMesh);

if (bHit)
{
    D3DXVec3TransformCoord(&vPivot, &vPivot, pWorldMatrix);
    D3DXVec3TransformNormal(&vRay, &vRay, pWorldMatrix);
    *vOutPos = vPivot + (vRay * fDist);
    *pfDist = fDist;
    return true;
}
return false;
}
```

D3DXIntersect함수를 이용하여 Mesh와 충돌 여부를 확인합니다.

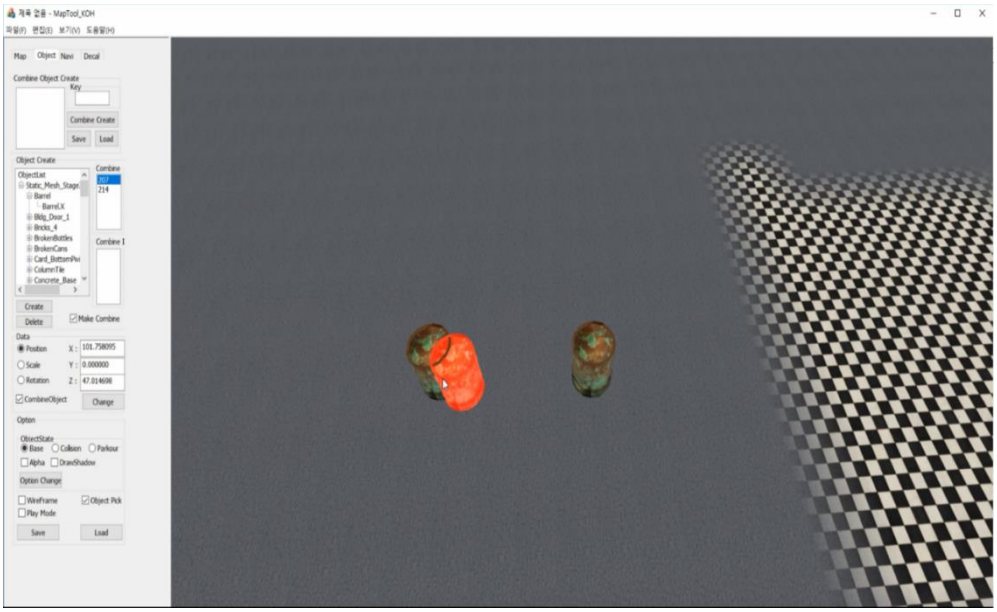
충돌 되었다고 판정하면 Ray가 Mesh와 충돌한 좌표를 구합니다.

```
_uint iObjectSize = pObjectMgr->Get_NumObject(TOOL_STATIC, L"Layer_Object");

for (_uint i = 0; i < iObjectSize; ++i)
{
    CGameObject*    pCreateObject = pObjectMgr->Get_FindObject(TOOL_STATIC, L"Layer_Object", i);
    if (nullptr == pCreateObject)
        continue;

    _float fDist = 0.f;
    _float fFinalDist = 1000.f;
    _vec3  vPickPos = _vec3(0.f, 0.f, 0.f);

    if (true == dynamic_cast<CMapTool_StaticMesh*>(pCreateObject)->Pick_StaticMesh(&fDist, &vPickPos))
    {
        if (fFinalDist > fDist)
        {
            vMousPos = vPickPos;
            fFinalDist = fDist;

            Set_ObjectData(pCreateObject);

        }
    }

}
```

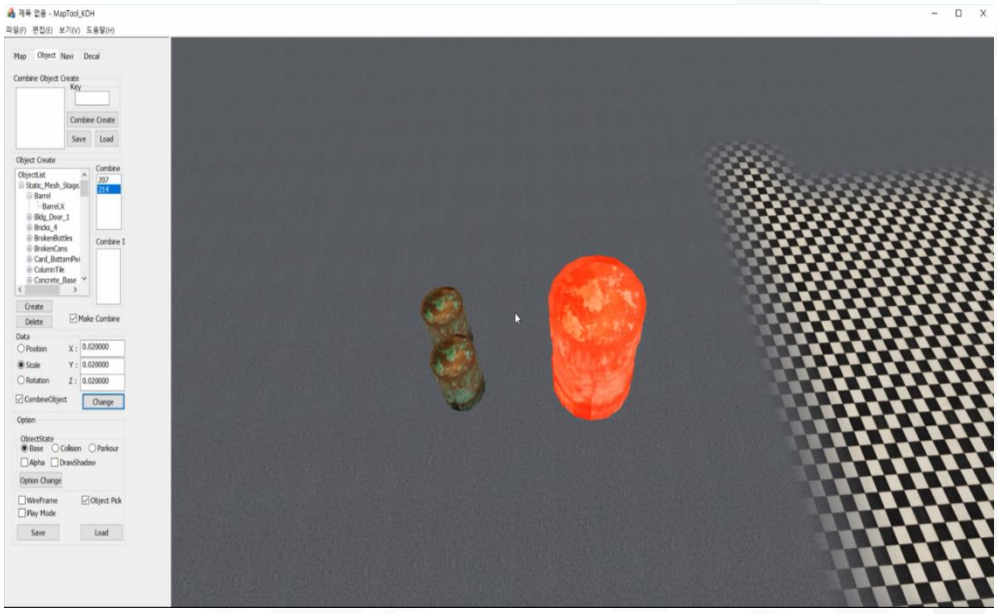
위에서 받아온 거리를 비교하여 가장 짧은 거리를 가진 Mesh를 선택합니다.

Mesh Picking

- 클라이언트 내 영상



< 3D Team Project >



< 3D Team Project >

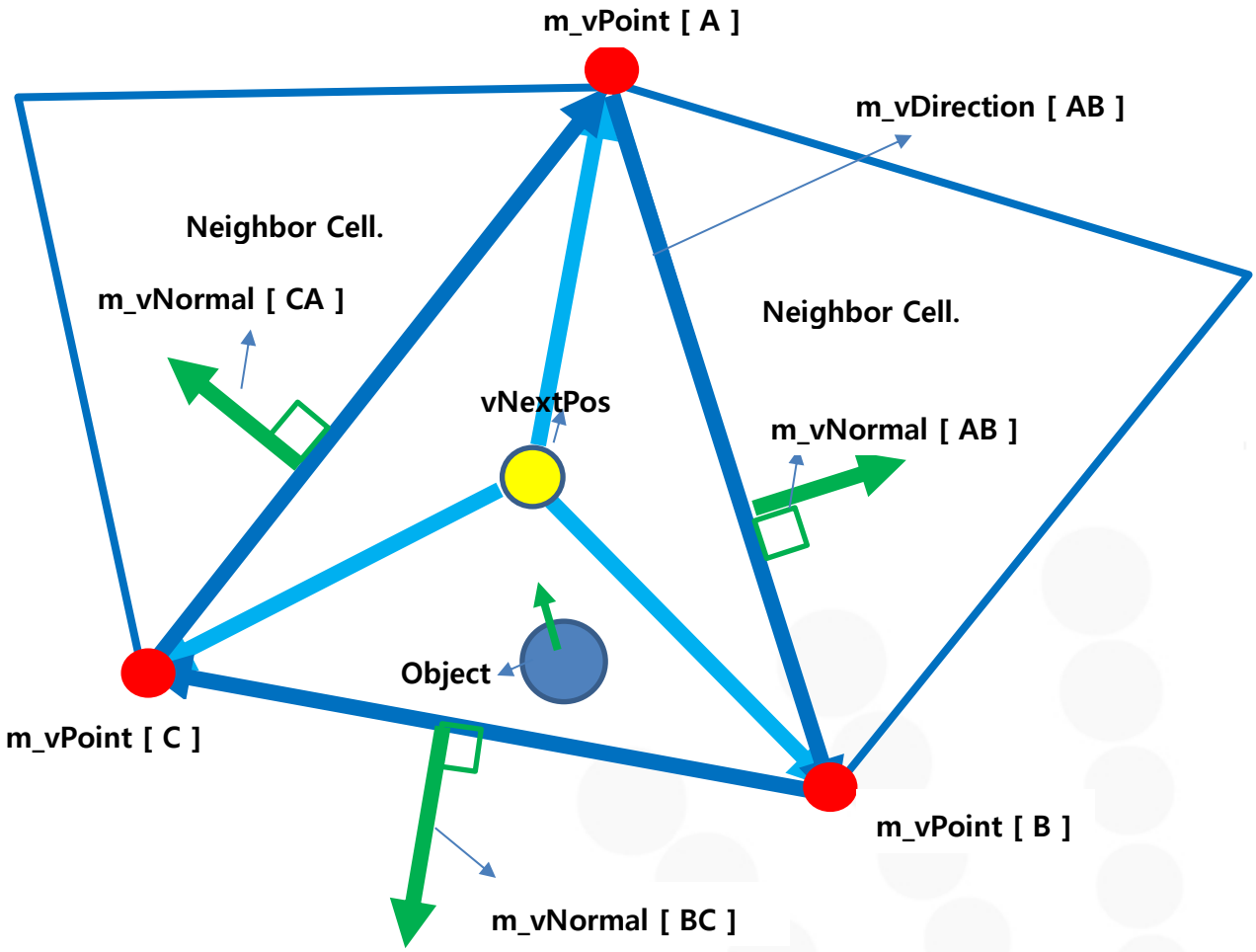
- Ray를 이용하여 오브젝트와 충돌되는 좌표와 거리 구하여 현재 마우스의 좌표를 구하고 가장 짧은 거리의 오브젝트를 구하여 선택하였습니다.

2) Navigation

- Navigation Mesh

- 내용 : 객체들이 움직일 때 갈 수 있는 장소와 못 가는 장소를 분별하고 제어하기 위하여 사용하였습니다.

- 이론



< Navigation **Check** >

- 객체를 Cell에 내부나 이웃에 있는지 체크를 하여 움직임을 제어합니다. 움직이고 난뒤의 좌표를 구하여 Cell들의 m_vPoint들과의 벡터를 구한 뒤 정규화 한 값들을 Cell들이 가지고 있는 m_vNormal 벡터들과 내적을 하여서 Cell안에 있는지 체크를 하였습니다.

Navigation Mesh

- 코드

```
CNavigation::MOVETYPE CCell::Check_Cell(const _vec3 * pNextPos, _uint* pCurrentIdx)
{
    if (nullptr == pCurrentIdx)
        return CNavigation::MOVE_OUT;

    for (size_t i = 0; i < DIR_END; i++)
    {
        _vec3 vDest = *pNextPos;
        vDest.y = 0.f;
        _vec3 vSrc = m_vPoint[i];
        vSrc.y = 0.f;

        _vec3   vSour = vDest - vSrc;

        if (D3DXVec3Dot(D3DXVec3Normalize(&vSour, &vSour), &m_vNormal[i]) > 0)
        {
            if (nullptr == m_pNeightbor_Cell[i])
                return CNavigation::MOVE_OUT;
            else
            {
                if (false == m_pNeightbor_Cell[i]->Get_Option()->bMove)
                {
                    return CNavigation::MOVE_OUT;
                }
                else
                {
                    *pCurrentIdx = m_pNeightbor_Cell[i]->m_iIndex;
                    return CNavigation::MOVE_NEIGHBOR;
                }
            }
        }
    }
    return CNavigation::MOVE_IN;
}
```

변수로 받은 vNextPos를 Cell들의 Point들과의 벡터를 구합니다.

구한 벡터를 Cell들의 m_vNormal 들과 내적하여 내부에 있는지 판단하여 움직임을 제어합니다.

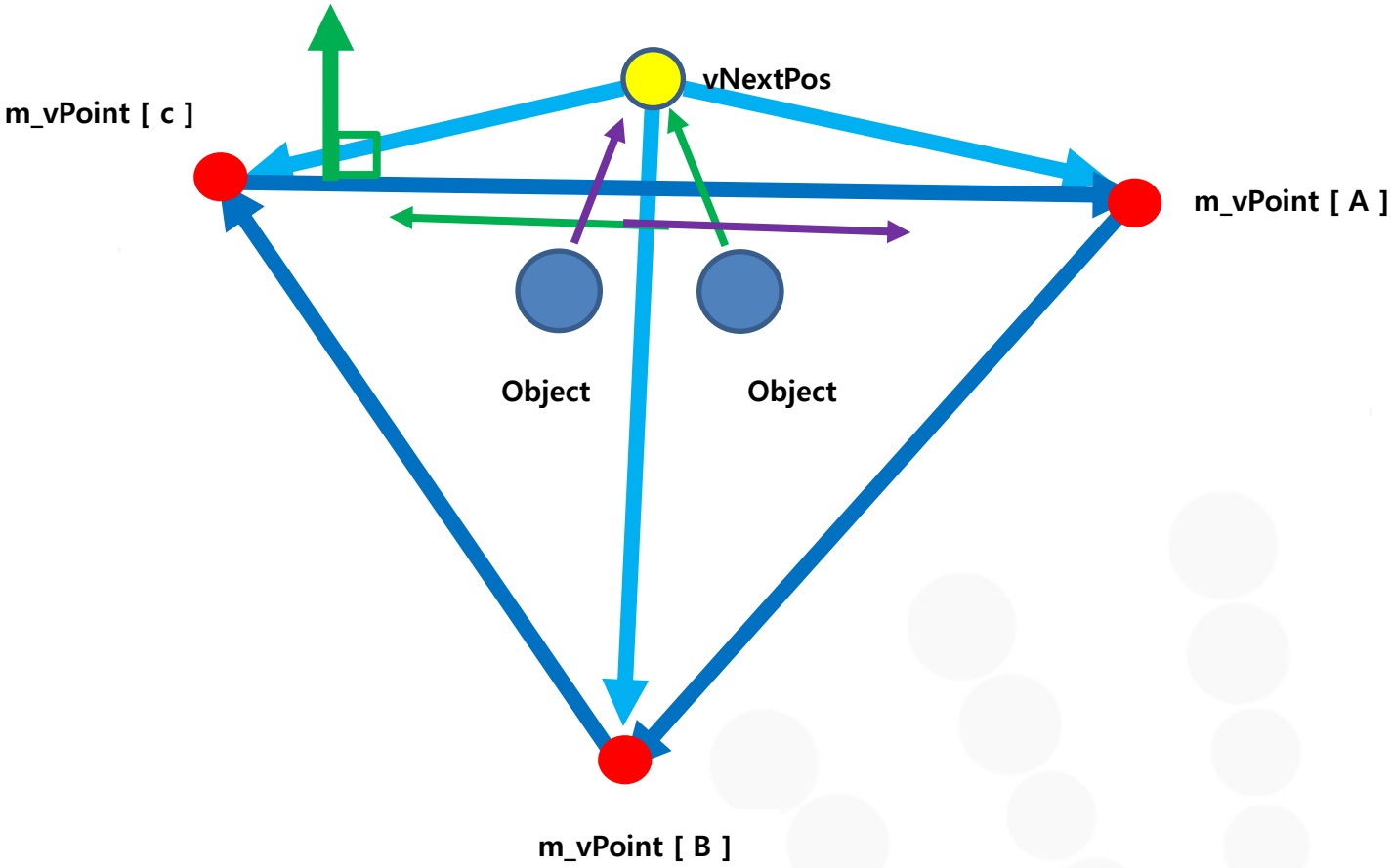
- 클라이언트 내 영상

< 3D Team Project >

- Sliding Vector

- 내용 :. 객체를 네비게이션을 태웠을 때 움직이지 못하는 경우 슬라이딩 벡터를 이용하여 인접한 Cell의 방향 벡터를 태워 움직임을 제어하였습니다.

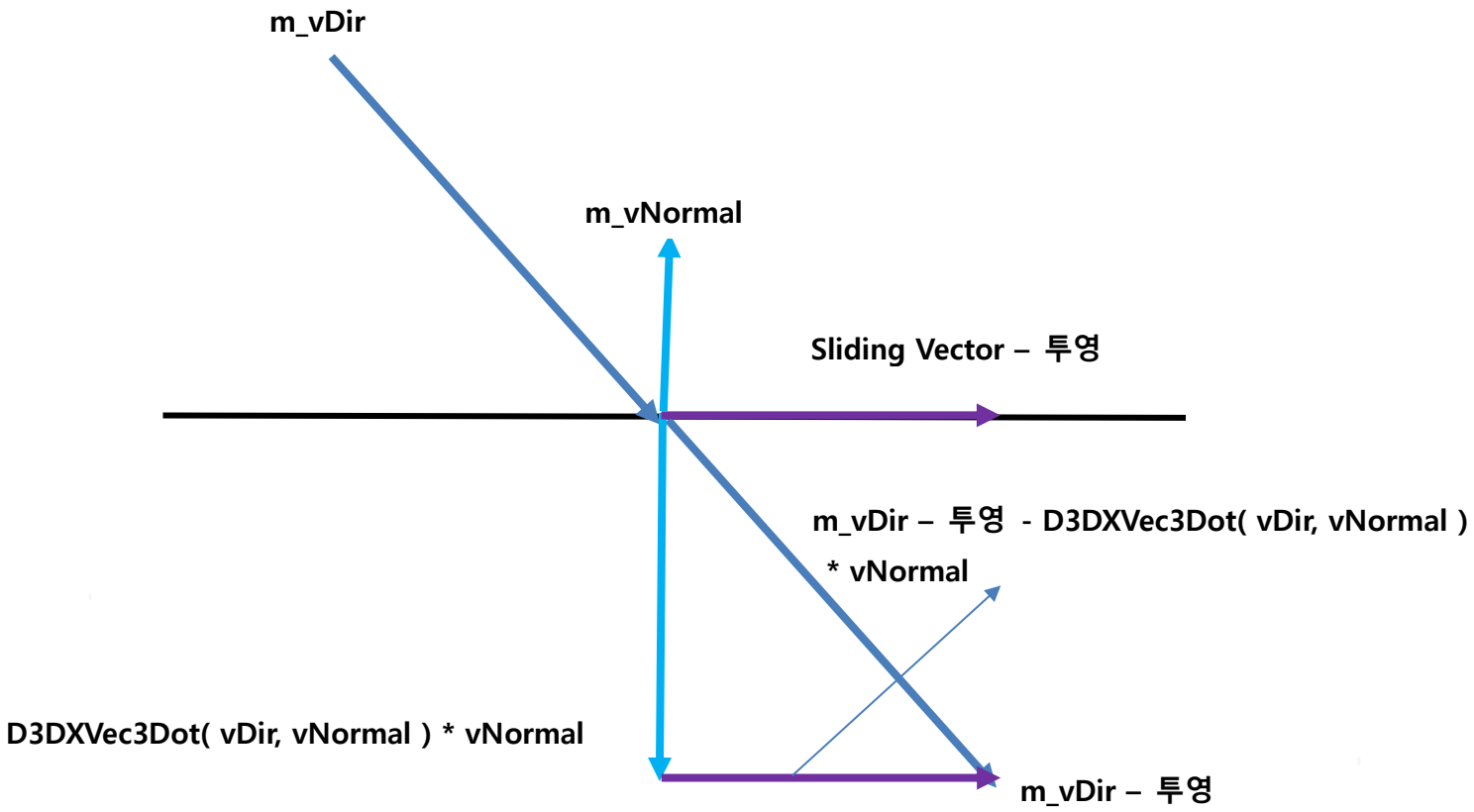
- 이론



- 객체의 움직이고 난 뒤의 좌표가 Cell밖으로 나갔다고 판정이 되면 객체의 방향에 따라 나간 인접 Cell의 선을 타듯이 움직이도록 하였습니다.
슬라이딩 벡터는 투영 한 방향 벡터와 노말 벡터의 내적을 하면 음수 값이 나오고 그 값을 노말 벡터와 곱한 값과 투영된 방향벡터에서 빼서 구하였습니다.

.

Sliding Vector



- 코드

```
void Get_Sliding(_vec3* vNextOut, const _vec3& vDirection, const _float & fSpeed, const _float & fTimeDelta, const _vec3* vNormal)
{
    _vec3   vDir;
    D3DXVec3Normalize(&vDir, &vDirection);

    _vec3 vSliding = vDir - D3DXVec3Dot(&vDir, vNormal) * *vNormal;

    D3DXVec3Normalize(&vSliding, &vSliding);

    *vNextOut = m_vInfo[INFO_POSITION] + vSliding * fSpeed * fTimeDelta;

    m_bUpdateWorld = false;
}
```

슬라이딩 벡터를 구합니다.

- 클라이언트 내 영상

< 3D Team Project >

< 3D Team Project >

- 평면의 방정식을 이용한 네비게이션

- 내용 : 네비게이션 메쉬의 Cell의 Y값을 이용하여 객체의 Y이동을 제어 하였습니다.

- 이론

  - 평면의 방정식은 < Ax + By + Cz + d = 0 > 입니다.
    이 공식에서 객체의 현재 위치에서의 y값을 구하기 위하여
    < y = -(Ax + By + d) / B > 바꾸어 현재 위치하는 y값을 구합니다.

    x, y, z ➔ 현재 객체의 위치

    A,B,C,D ➔ 평면에 위치하는 한 점

  이것을 통하여 객체의 Y값을 바꿔주었습니다.

- 코드

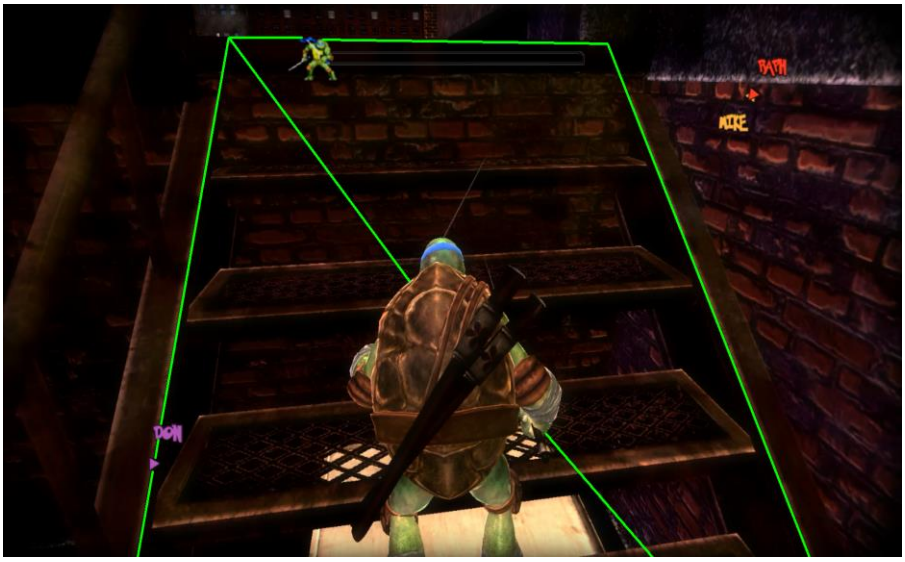
```
void CTransform::Move_OnNavigation(Engine::CNavigation * pNavigation)
{
    if (nullptr == pNavigation)
        return;

    _vec3           vPosition;
    memcpy(&vPosition, &m_vInfo[INFO_POSITION], sizeof(_vec3));


    CCell*      pCell = pNavigation->Get_CurVecCell();

    _vec3 vPoint[3];
    vPoint[0] = *pCell->Get_Point(CCell::POINT_A);
    vPoint[1] = *pCell->Get_Point(CCell::POINT_B);
    vPoint[2] = *pCell->Get_Point(CCell::POINT_C);


    D3DXPLANE tCellPlane;

    D3DXPlaneFromPoints(&tCellPlane, &vPoint[0], &vPoint[1], &vPoint[2]);

    _float fY = (tCellPlane.a * vPosition.x + tCellPlane.c * vPosition.z + tCellPlane.d) / tCellPlane.b;

    m_vInfo[INFO_POSITION].y = -fY;

    m_bUpdateWorld = false;
}
```

평면을 생성합니다.

평면의 방정식인
Ax + By + Cz +d = 0 을
바꿔 y값을 구합니다.

평면의 방정식

- 클라이언트 내 영상



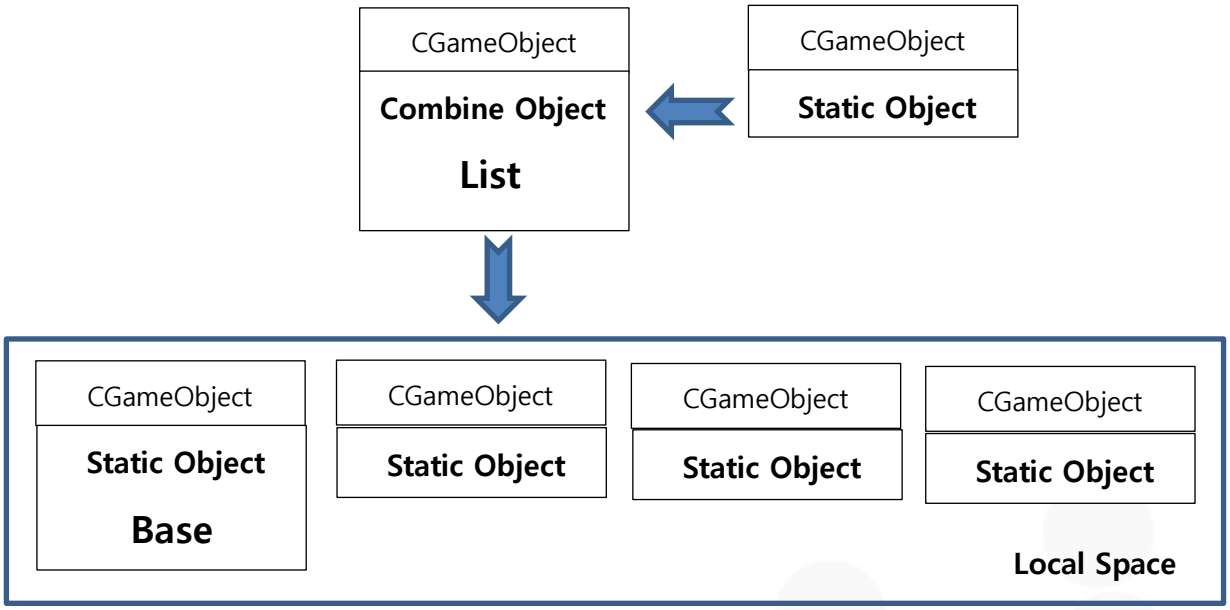
< 3D Team Project >

< 3D Person Project >

3) Combine Object

- 내용 :. Tool 에서 오브젝트를 설치할 때 수정과 효율을 높이기 위하여 사용하였습니다.

- 이론



- 리스트에 첫 번째로 들어간 오브젝트를 베이스 오브젝트로 선택하였습니다. 그 뒤에 들어오는 오브젝트들의 월드 스페이스 행렬에 베이스 오브젝트의 월드 행렬의 역행렬을 곱하여 로컬 스페이스 영역에서 오브젝트들을 묶어 놓았습니다.
  Combine 해둔 오브젝트들을 설치 할 경우 로컬 스페이스로 변화한 변수에 설치한 베이스 오브젝트의 월드 행렬을 곱하여 결합한 것을 설치합니다.

Combine Object

- 코드

```
HRESULT CCombineObject::Add_Object(MAPOBJECT pData, const _vec3* vMousePos)
{
    Engine::CGameObject*    pGameObject = nullptr;

    Set_AllNotPick();

    // For.StaticMesh
    pGameObject = CMapTool_StaticMesh::Create(m_pGraphicDev, pData);
    if (nullptr == pGameObject)
        return E_FAIL;

    dynamic_cast<CMapTool_StaticMesh*>(pGameObject)->Set_ObjectData(CTransform::INFO_POSITION, vMousePos);

    if (m_ListCombineObject.empty())
        Set_FirstInverceMatrix(pGameObject);

    dynamic_cast<CMapTool_StaticMesh*>(pGameObject)->Set_LocalMatrix(&m_matInverseBaseWorld);

    _matrix matWorld = dynamic_cast<CMapTool_StaticMesh*>(pGameObject)->Get_ObjData()->LocalMatrix * m_matInverseBaseWorld;

    dynamic_cast<CMapTool_StaticMesh*>(pGameObject)->Set_WorldMatrix(&matWorld);

    m_ListCombineObject.push_back(pGameObject);

    return NOERROR;
}
```

```
void CCombineObject::Set_FirstInverceMatrix(CGameObject*    pObject)
{
    if (nullptr == m_pBaseTransformCom)
    {
        m_pBaseTransformCom = (CTransform*)(pObject)->Get_Component(L"Com_Transform");
        m_pBaseTransformCom->AddRef();

        dynamic_cast<CMapTool_StaticMesh*>(pObject)->Set_PickObj(true);

        m_matInverseBaseWorld = m_pBaseTransformCom->Get_WorldMatrix_FORCED();
    }
}
```

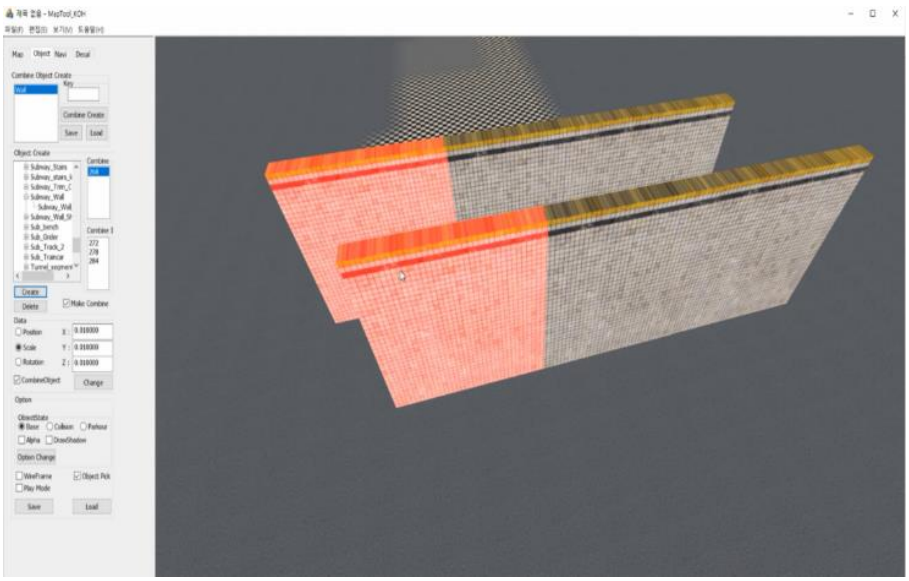
베이스 오브젝트의 월드 행렬만 변수로 보관합니다.

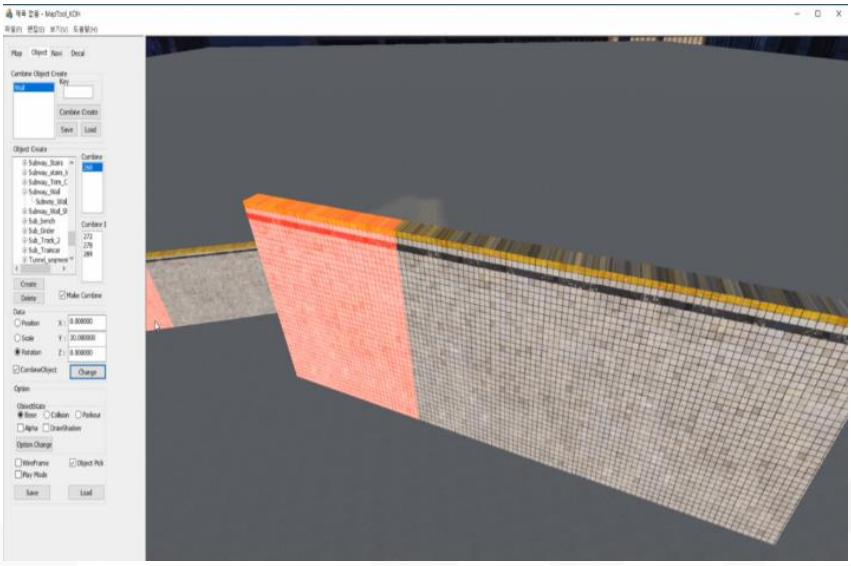
```
void CMapTool_StaticMesh::Set_LocalMatrix(_matrix* pBaseTransform)
{
    _matrix matInverseWorld;

    D3DXMatrixInverse(&matInverseWorld, 0, pBaseTransform);

    m_ObjcetData.LocalMatrix = m_pTransformCom->Get_WorldMatrix_FORCED() * matInverseWorld;
}
```

베이스 오브젝트의 월드 행렬을 받아와 역행렬로 바꾸어 자신의 월드행렬과 곱하여 로컬영역으로 변화한 값을 보관합니다

Combine Object

- 클라이언트 내 영상



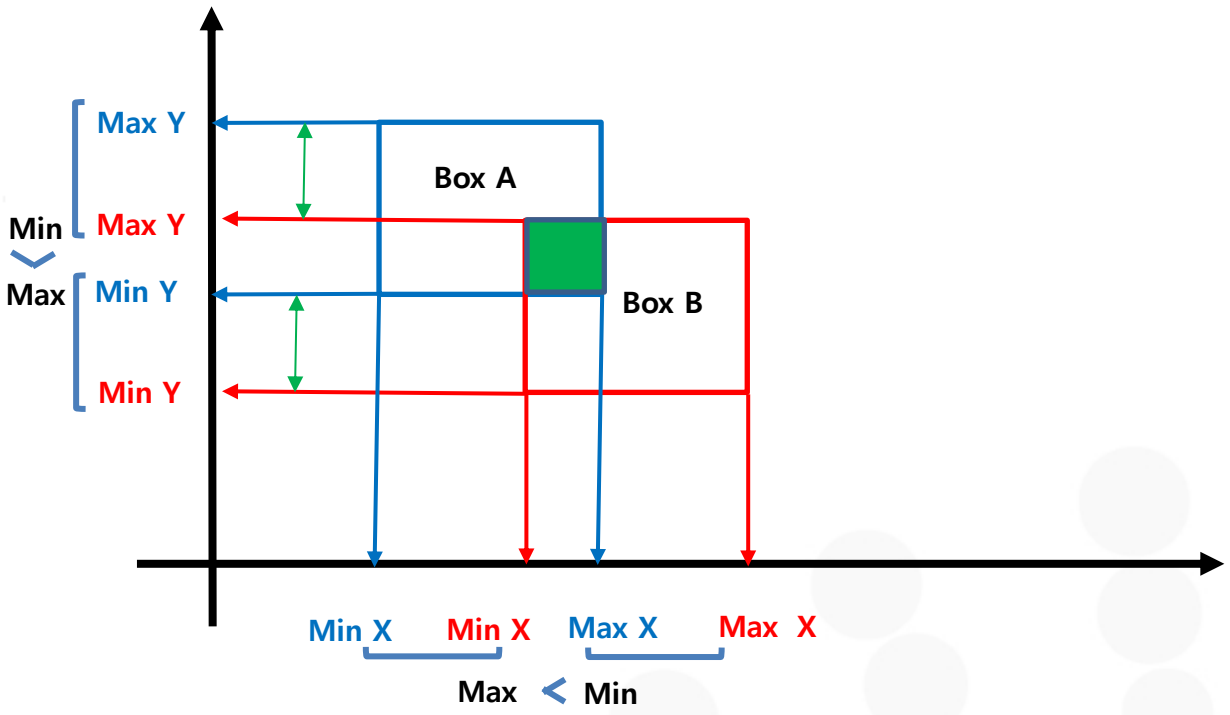


4) 충돌 처리

- AABB

- 내용 : 회전되지 않은 2개의 Bounding Box을 이용하여 충돌을 체크하였습니다.

- 이론



- 비교하려고 하는 2개의 Bounding Box의 X, Y, Z 축의 Min과 Max를 구하여 비교를 합니다.
모든 축이 Max의 값에서 작은 값이 Min에서 큰 값보다 크면 충돌되는 부분이 생성이 되면 충돌이 되었다고 판정하였습니다.
AABB는 크게 사용하지는 않았지만 오브젝트에서 고정되어 있으며 회전하지 않은 오브젝트에 사용하였습니다. 또한 충돌 처리 이전에 상태 체크를 하여 필요한 순간에만 충돌이 되도록 하였습니다.

AABB

- 코드

```
bool CCollider::Collision_AABB(CCollider* pTargetCollider)
{
    // 충돌을 하지 않아도 되는 경우
    if (m_InfoData.eState == COLLIDER_NOACTIVE ||
        pTargetCollider->Get_State() == COLLIDER_NOACTIVE)
        return false;

    // 자신의 Collider의 Min, Max정보.
    _vec3       vSourMin = m_ColliderData.vMinWorld_NotRot;
    _vec3       vSourMax = m_ColliderData.vMaxWorld_NotRot;

    // 충돌할 Collider의 Min, Max정보
    _vec3       vTargetMin = pTargetCollider->Get_Min_WorldNotRot();
    _vec3       vTargetMax = pTargetCollider->Get_Max_WorldNotRot();

    // x 축 선상에서 바라봤을때
    if (max(vSourMin.x, vTargetMin.x) > min(vSourMax.x, vTargetMax.x))
        goto notcoll;

    // y 축 선상에서 바라봤을때
    if (max(vSourMin.y, vTargetMin.y) > min(vSourMax.y, vTargetMax.y))
        goto notcoll;

    // z 축 선상에서 바라봤을때
    if (max(vSourMin.z, vTargetMin.z) > min(vSourMax.z, vTargetMax.z))
        goto notcoll;

    return true;

notcoll:
    return false;
}
```

X, Y, Z축의 Min과 Max를 비교하여서 충돌 처리를 체크하였습니다.

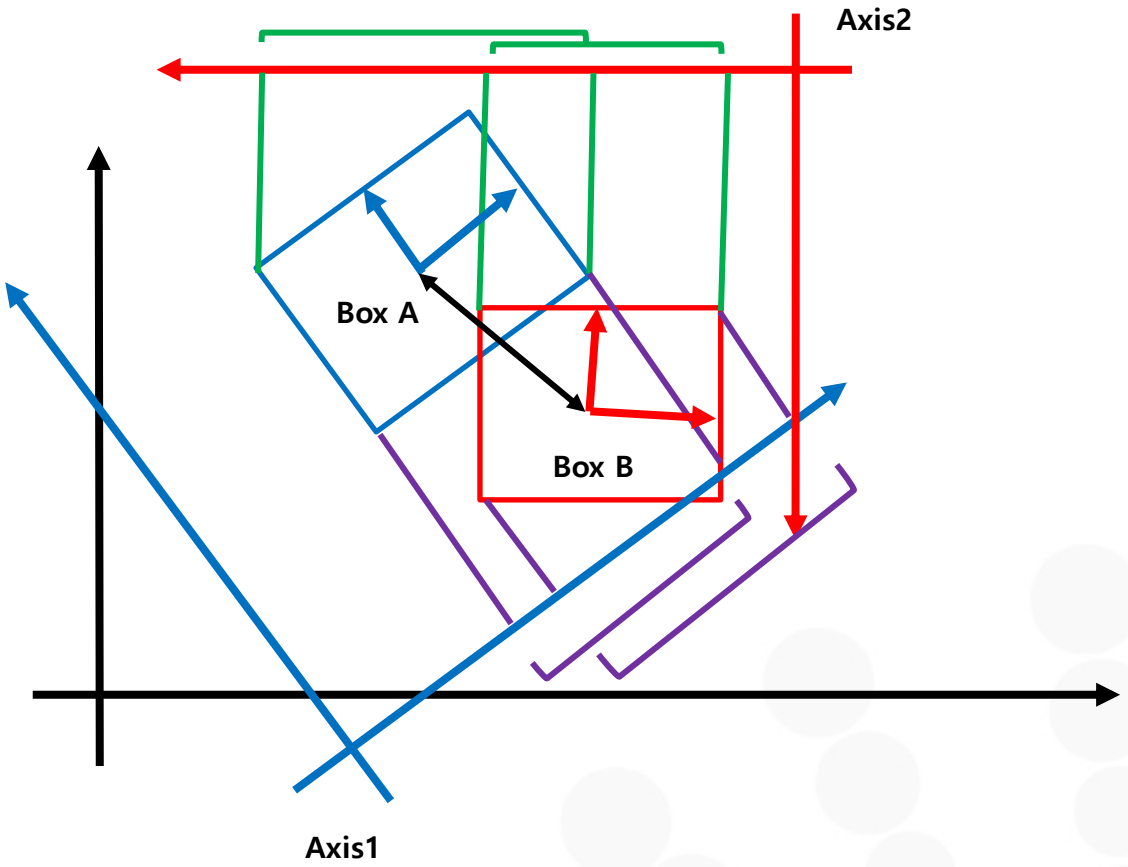
- 클라이언트 내 영상

< 3D Person Project >

- OBB

- 내용 : 회전되어 있는 2개의 Bounding Box를 이용하여 충돌을 체크하였습니다.

- 이론



- 회전 된 두 Bounding Box의 중심 거리를 구하여 새로운 기준 축을 투영합니다. 투영된 축과 두 Bounding Box의 중심에서 거리 벡터와 내적을 하여 나온 값들의 합이 두 Bounding Box의 중심 사이의 거리 벡터를 내적 한 값과 비교합니다. 비교하여 나온 것이 작으면 두 Bounding Box는 충돌을 한 것이고 아니면 충돌을 하지 않았습니다.

OBB

- 코드

```
bool CCollider::Collision_OBB(CCollider * pTargetCollider)
{
    // 충돌을 하지 않아도 되는 경우
    if (m_InfoData.eState == COLLIDER_NOACTIVE ||
        pTargetCollider->Get_State( ) == COLLIDER_NOACTIVE)
        return false;

    // ColliderData를 셋팅한다.
    Modify_ColliderData( );

    COLLIDERDATA        tOBB[2];

    tOBB[0] = m_ColliderData;
    pTargetCollider->Get_ColliderData(&tOBB[1]);

    _float      fDistance[3];

    for (size_t i = 0; i < 2; i++)
    {
        for (size_t j = 0; j < 3; j++)
        {
            fDistance[0] = fabs(D3DXVec3Dot(&tOBB[i].vAlign_Axis[j], &tOBB[0].vProj_Axis[0]))
                + fabs(D3DXVec3Dot(&tOBB[i].vAlign_Axis[j], &tOBB[0].vProj_Axis[1]))
                + fabs(D3DXVec3Dot(&tOBB[i].vAlign_Axis[j], &tOBB[0].vProj_Axis[2]));

            fDistance[1] = fabs(D3DXVec3Dot(&tOBB[i].vAlign_Axis[j], &tOBB[1].vProj_Axis[0]))
                + fabs(D3DXVec3Dot(&tOBB[i].vAlign_Axis[j], &tOBB[1].vProj_Axis[1]))
                + fabs(D3DXVec3Dot(&tOBB[i].vAlign_Axis[j], &tOBB[1].vProj_Axis[2]));

            _vec3       vCenter = tOBB[1].vCenter - tOBB[0].vCenter;
            fDistance[2] = fabs(D3DXVec3Dot(&tOBB[i].vAlign_Axis[j], &vCenter));

            if (fDistance[0] + fDistance[1] < fDistance[2])
            {
                return false;
            }
        }
    }

    return true;
}
```

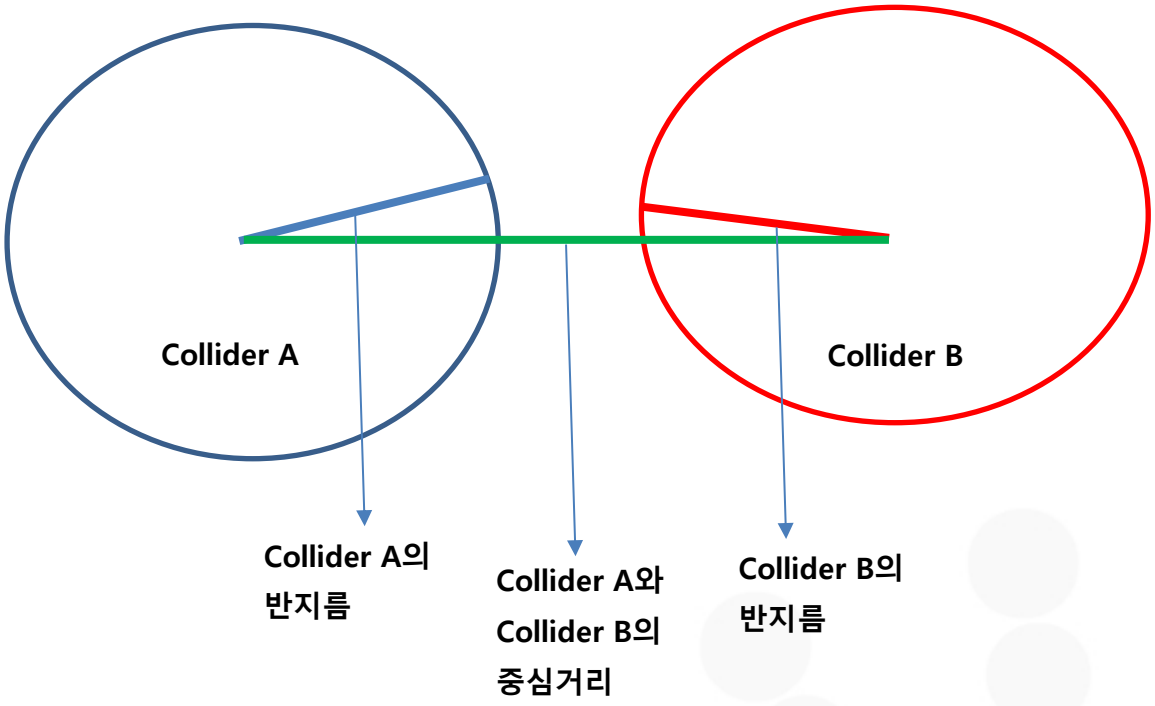
비교를 위한 축과 Center좌표 등 필요한 정보를 셋팅합니다..

중심까지의 거리와 각 축의 투영된 좌표를 비교하여 충돌 여부를 확인합니다..

- 원 충돌

- 내용 :. 2개의 구체를 통하여 프로젝트 내의 무기와 몬스터의 충돌, 충돌하는 오브젝트와 객체의 충돌 등 자주 사용되는 충돌 처리의 경우에 사용 하였습니다.

- 이론



- 원 충돌은 AABB나 OBB와 다르게 충돌 연산이 매우 간단하여 게임 내에서 주 된 충돌 처리를 하였습니다. 연산은 각 원의 중심 거리와 두 원의 중심 거리를 비교하여 충돌 처리를 하였습니다.

원 충돌

- 코드

```
_bool CCollider::Collision_Sphere(CCollider* pTargetCollider, COLLISIONDATA& _CollisionData)
{
    // 충돌검사를 하지 않아도 되는 경우
    if (m_InfoData.eState == COLLIDER_NOACTIVE ||
        pTargetCollider->Get_State() == COLLIDER_NOACTIVE)
        return false;

    COLLIDERDATA ColliderData[2];

    ColliderData[0] = m_ColliderData;
    pTargetCollider->Get_ColliderData(&ColliderData[1]);

    _vec3 vCenter = (ColliderData[0].vCenter - ColliderData[1].vCenter);
    _float fCenter_Distance = D3DXVec3Length(&vCenter);

    _float fRadius = ColliderData[0].fRadius + ColliderData[1].fRadius;

    if (fCenter_Distance > fRadius)
        return false;

    return true;
}
```

각 구체의 반지름과 중심점 사이의 거리를 비교하여 충돌 체크를 하였습니다.

- 클라이언트 내 영상

스킬들을 구 형태로 플레이어와 충돌 처리를 하였습니다.

< 3D Person Project >

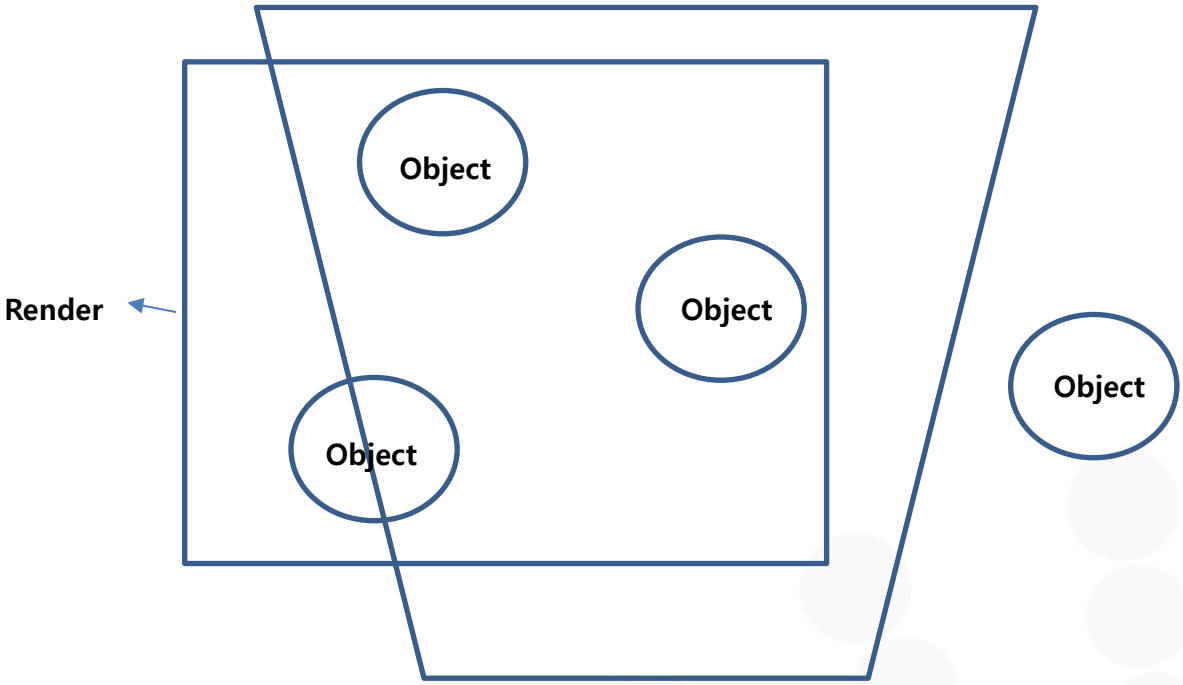
< 3D Person Project >

4) 최적화

- Frustum Culling

- 내용 : 많은 수의 오브젝트에서 Render함수 호출을 최소화하기 위하여 사용하였습니다.

- 이론



< Frustum Culling >

- 절두체를 구성하여 오브젝트가 절두체 내부에 있는지 없는지 내적을 통하여 판별합니다. 또한 절두체의 면에 겹치는 경우에는 일정 값 내부에 있으면 Render를 호출하고 아니면 그리지 않도록 하였습니다.

Frustum Culling

- 코드

```
_bool CFrustum::isIn_Frustum(const _vec3 * pPositionInWorldSpace, _float fRadius)
{
    if (nullptr == m_pGraphicDev)
        return false;

    SetUp_PointsInProjection( );

    _matrix     matView, matProj;

    m_pGraphicDev->GetTransform(D3DTS_PROJECTION, &matProj);
    D3DXMatrixInverse(&matProj, nullptr, &matProj);

    for( size_t i = 0 ; i<8 ; ++i )
        D3DXVec3TransformCoord(&m_vPoint[i], &m_vPoint[i], &matProj);

    m_pGraphicDev->GetTransform(D3DTS_VIEW, &matView);
    D3DXMatrixInverse(&matView, nullptr, &matView);

    for (size_t i = 0; i < 8; i++)
        D3DXVec3TransformCoord(&m_vPoint[i], &m_vPoint[i], &matView);

    D3DXPLANE           Plane[6];

    // +x, -x
    D3DXPlaneFromPoints(&Plane[0], &m_vPoint[1], &m_vPoint[5], &m_vPoint[6]);
    D3DXPlaneFromPoints(&Plane[1], &m_vPoint[4], &m_vPoint[0], &m_vPoint[3]);

    // +y, -y
    D3DXPlaneFromPoints(&Plane[2], &m_vPoint[4], &m_vPoint[5], &m_vPoint[1]);
    D3DXPlaneFromPoints(&Plane[3], &m_vPoint[3], &m_vPoint[2], &m_vPoint[6]);

    // +z, -z
    D3DXPlaneFromPoints(&Plane[4], &m_vPoint[7], &m_vPoint[6], &m_vPoint[5]);
    D3DXPlaneFromPoints(&Plane[5], &m_vPoint[0], &m_vPoint[1], &m_vPoint[2]);

    for (size_t i = 0; i < 6; ++i)
    {
        if (fRadius < D3DXPlaneDotCoord(&Plane[i], pPositionInWorldSpace))
            return false;
    }
    return true;
}

_int CBaseObj::Update_GameObject(const _float & fTimeDelta)
{

        ***** 생략 *****

    if (nullptr != m_pFrustum)
        m_isDraw = m_pFrustum->isIn_Frustum(&vPosition, m_fFrustumSize);

        ***** 생략 *****

    if (nullptr != m_pRendererCom && true == m_isDraw)
        m_pRendererCom->Add_RenderGroup(Engine::CRenderer::RENDER_NONALPHA, this);

    return 0;
}
```

절두체를 구성합니다.

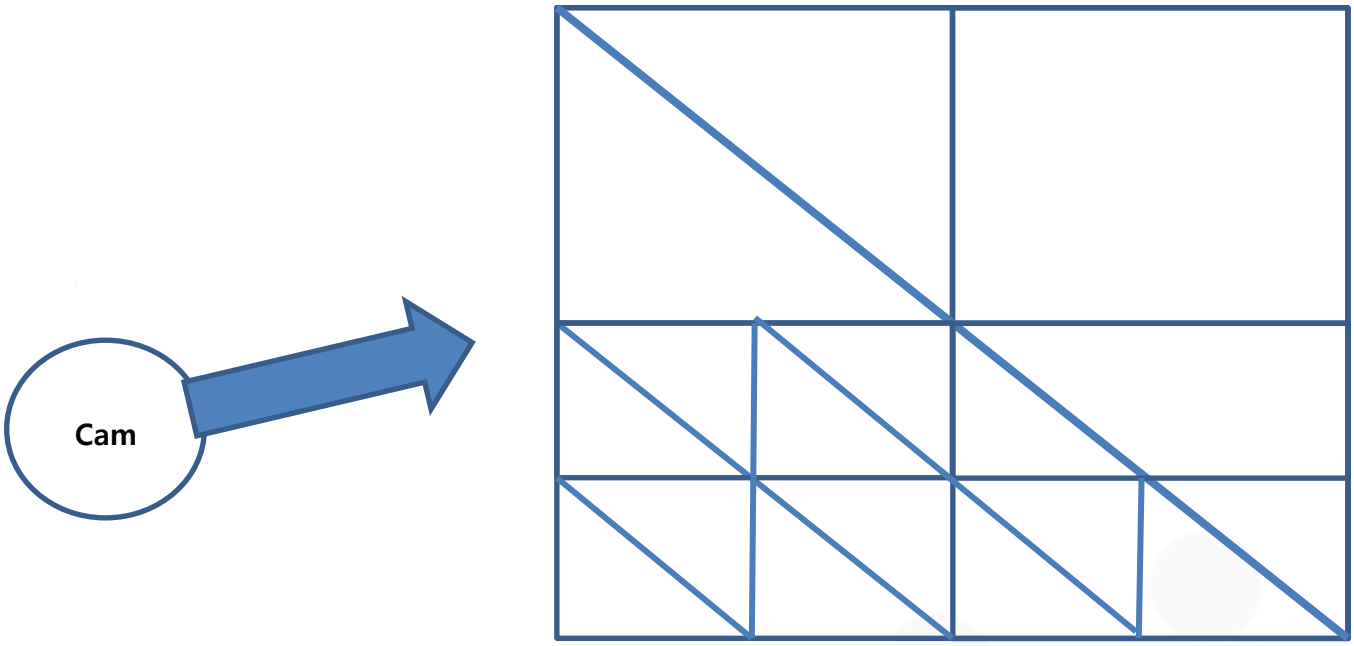
현재 좌표를 절두체와 내적하여 안에 있는지 판별합니다.

LOD (Level of Detail)

- LOD (Level of Detail)

  - 내용 : 카메라와 거리에 따라 지형을 나누어 절두체 컬링을 더 빠르게 하기 위하여 쿼드 트리를 사용하였습니다.

  - 이론



< QuadTree >

- 카메라의 거리에 따라 자세히 그릴지 판별한 뒤에 지형을 여러 단계로 나누어 그립니다. 쿼드 트리로 지형을 표현하여 4개의 자식 노드를 가지고 있습니다. 그리고 카메라의 거리에 따라 그 노드에서 계속 잘게 쪼개며 자세히 그립니다. 하지만 쿼드 트리로 그리다 보면 크랙이 발생하게 됩니다. 크랙을 해결하기 위해 쿼드 트리가 이웃을 가지고 있고 그것이 그려지는지 판별합니다. 이렇게 최종적으로 그려진 쿼드 트리를 사용하여 비교하는 대상을 줄여 빠르게 비교합니다.

LOD (Level of Detail)

- 코드

```
_bool CQuadTree::isRender_LevelofDetail(const _vec3* pVertexPos)
{
    _matrix  matView;
    m_pGraphicDev->GetTransform(D3DTS_VIEW, &matView);
    D3DXMatrixInverse(&matView, nullptr, &matView);

    _vec3 vCamPos;
    memcpy(&vCamPos, &matView.m[3][0], sizeof(_vec3));

    _float fDistance = D3DXVec3Length(&(pVertexPos[m_dwCenter] - vCamPos));

    _float fWidth = D3DXVec3Length(&(pVertexPos[m_dwCorner[CORNER_RT]] - pVertexPos[m_dwCorner[CORNER_LT]]));

    if(fDistance * 0.2f > fWidth)
        return true;

    return false;
}
```

현재 카메라의 위치를 구합니다.

쿼드 트리의 노드의 왼쪽 위와 오른쪽 위의 거리와 카메라와 노드의 중심거리를 비교합니다.

```
void CQuadTree::Make_QuadTree_Child(const _vec3* pVertexPos)
{
    if( 1 >= m_dwCorner[CORNER_RT] - m_dwCorner[CORNER_LT])
        return;

    m_dwCenter = (m_dwCorner[CORNER_LT] + m_dwCorner[CORNER_RB]) >> 1;
    m_fRadius = D3DXVec3Length(&(pVertexPos[m_dwCorner[CORNER_LT]] - pVertexPos[m_dwCenter]));

    for (_uint i = 0; i < CHILD_END; ++i)
    {
        tuple<_ulong, _ulong, _ulong, _ulong>   Corner_Child;

        Corner_Child = Compute_ChildIndex_Corner(CQuadTree::CHILD(i));

        m_pChild[i] = CQuadTree::Create(m_pGraphicDev, get<CORNER_LT>(Corner_Child)
            , get<CORNER_RT>(Corner_Child), get<CORNER_RB>(Corner_Child), get<CORNER_LB>(Corner_Child));

        if(nullptr == m_pChild[i])
            return;

        m_pChild[i]->Make_QuadTree_Child(pVertexPos);
    }
}
```

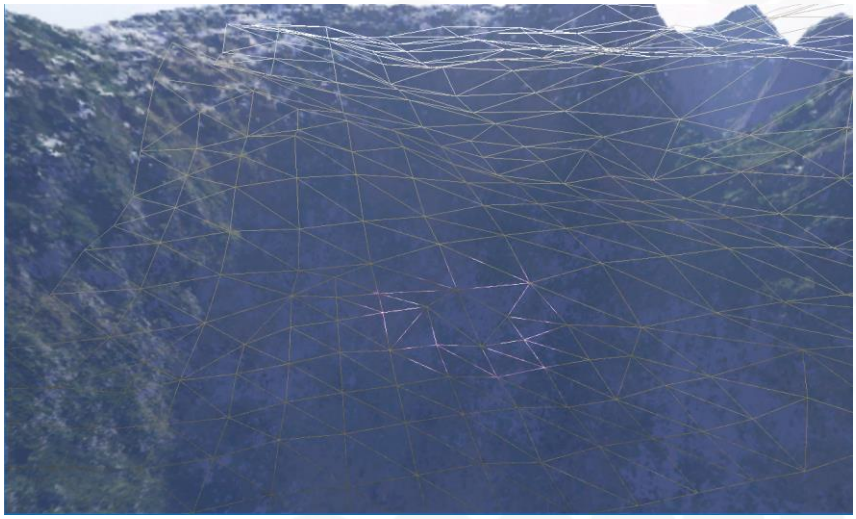
자식 노드를 생성합니다.

***** 생략 *****

LOD (Level of Detail)

```
void CQuadTree::Make_Neighbor(void)
{
    // 가장 작은 쿼드트리가 아닐때
    if(nullptr != m_pChild[CHILD_LT])
    {
        if(nullptr == m_pChild[CHILD_LT]->m_pChild[CHILD_LT])
            return;
        // 자식 이웃들을 셋팅
        m_pChild[CHILD_LT]->m_pNeighbor[NEIGHBOR_RIGHT] = m_pChild[CHILD_RT];
        m_pChild[CHILD_LT]->m_pNeighbor[NEIGHBOR_BOTTOM] = m_pChild[CHILD_LB];

        m_pChild[CHILD_RT]->m_pNeighbor[NEIGHBOR_LEFT] = m_pChild[CHILD_LT];
        m_pChild[CHILD_RT]->m_pNeighbor[NEIGHBOR_BOTTOM] = m_pChild[CHILD_RB];

        m_pChild[CHILD_LB]->m_pNeighbor[NEIGHBOR_TOP] = m_pChild[CHILD_LT];
        m_pChild[CHILD_LB]->m_pNeighbor[NEIGHBOR_RIGHT] = m_pChild[CHILD_RB];

        m_pChild[CHILD_RB]->m_pNeighbor[NEIGHBOR_TOP] = m_pChild[CHILD_RT];
        m_pChild[CHILD_RB]->m_pNeighbor[NEIGHBOR_LEFT] = m_pChild[CHILD_LB];

        if(nullptr != m_pNeighbor[NEIGHBOR_LEFT])
        {
            m_pChild[CHILD_LT]->m_pNeighbor[NEIGHBOR_LEFT] = m_pNeighbor[NEIGHBOR_LEFT]->m_pChild[CHILD_RT];
            m_pChild[CHILD_LB]->m_pNeighbor[NEIGHBOR_LEFT] = m_pNeighbor[NEIGHBOR_LEFT]->m_pChild[CHILD_RB];
        }

        if(nullptr != m_pNeighbor[NEIGHBOR_TOP]) { ... }

        if(nullptr != m_pNeighbor[NEIGHBOR_RIGHT]) { ... }

        if(nullptr != m_pNeighbor[NEIGHBOR_BOTTOM]) { ... }

        for(size_t i = 0; i < CHILD_END; ++i)
        {
            if(nullptr != m_pChild[i])
                m_pChild[i]->Make_Neighbor();
        }
    }
}
```

- 클라이언트 내 영상



< QuadTree >

5) Shader

- Alpha Blend

- 내용 : 오브젝트의 텍스처에서 알파 채널에 값이 존재하는 경우에 투명하게 보이도록 구현하였습니다.

- 코드

```
pass Mapping_DiffuseSpecular_One_SrcAlpha
{
    alphablendenable = true;
    destblend = one;
    srcblend = srcalpha;

    alphatestenable = false;

    CullMode = ccw;

    VertexShader = compile vs_3_0 VS_MAIN_LIGHTING();
    PixelShader = compile ps_3_0 ps_main_Lighting_DS();
}

pass Mapping_DiffuseNormalSpecular_One_SrcAlpha
{
    alphablendenable = true;
    destblend = one;
    srcblend = srcalpha;

    alphatestenable = false;

    CullMode = ccw;

    VertexShader = compile vs_3_0 VS_MAIN_TBN_LIGHTING();
    PixelShader = compile ps_3_0 ps_main_Lighting_DNS();
}
```

텍스처의 색과 뒤의 배경의 색을 더하여 그렸습니다.

- 클라이언트 내 영상

Alpha Blend를 하지 않았을 때

Alpha Blend를 적용하였을 때

< 3D Team Project >

- 법선(노말) 맵핑

- 내용 : 폴리곤의 굴곡을 표현을 하는 것처럼 보이게 하기 위하여 사용하였습니다. 탄젠트 공간에서 바이 노말을 구하여 노말을 하였습니다.

- 코드

```
VS_OUT_TBN VS_MAIN_TBN(VS_IN_TBN In)
{
    VS_OUT_TBN Out = (VS_OUT_TBN)0;

    matrix matWV, matWVP;

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

    float3 T = mul(In.vTangent, (float3x3)g_matWorld);
    float3 B = mul(In.vBinormal, (float3x3)g_matWorld);
    float3 N = mul(In.vNormal, (float3x3)g_matWorld);

    Out.vPosition = mul(vector(In.vPosition, 1.f), matWVP);
    Out.vTexUV = In.vTexUV;
    Out.vViewPos = mul(vector(In.vPosition, 1.f), matWV);
    Out.vWorldPos = mul(vector(In.vPosition, 1.f), g_matWorld);
    Out.T = normalize(T);
    Out.B = normalize(B);
    Out.N = normalize(N);

    return Out;
}

PS_OUT_SPECULAR ps_main_DN(PS_IN_TBN In)
{
    PS_OUT_SPECULAR Out = (PS_OUT_SPECULAR)0;

    float3 vNormal = tex2D(NormalSampler, In.vTexUV).xyz * 2.f - 1.f;
    float3x3 matTBN = float3x3(float3(In.T), float3(In.B), float3(In.N));
    matTBN = transpose(matTBN);
    vNormal = mul(matTBN, vNormal) * 0.5f + 0.5f;

    Out.vColor = tex2D(BaseSampler, In.vTexUV);
    Out.vNormal = float4(vNormal, 0.f);
    Out.vDepth = vector(In.vViewPos.z / g_fCamFar, 0.f, 0.f, 0.0f);
    Out.vSpecular = g_vSpecularDefaultColor;

    return Out;
}
```

Tangent, Normal, Binormal을 월드 스페이스로 변화합니다.

탄젠트 스페이스에서 월드 스페이스로 변화 합니다.

법선(노말) 맵핑

- 클라이언트 내 영상

< 3D Team Project >

< 3D Team Project >

- 파티클 (Particle)

- 내용 : 동적 버퍼에 이미지를 입혀 눈을 구현하였습니다. 파티클은 일정 범위를 나가면 다시 사용하기 위하여 위로 가지고 왔습니다. 눈은 랜덤한 장소에서 생성되어 한쪽 방향으로 천천히 내려가도록 하였습니다.

- 코드

```
HRESULT CPaticle::Ready_Paticle(const _tchar* pFilePath)
{
    HRESULT Hr = 0;

    Hr = m_pGraphicDev->CreateVertexBuffer(dwSize * sizeof(PARTICLE),
        D3DUSAGE_DYNAMIC | D3DUSAGE_POINTS | D3DUSAGE_WRITEONLY,
        D3DFVF_XYZ | D3DFVF_DIFFUSE, D3DPOOL_DEFAULT, &m_pVertex, 0);

    if (FAILED(Hr))
    {
        MSG_BOX("Created Paticle Vertex Buffer Failed!!");
        return E_FAIL;
    }

    Hr = D3DXCreateTextureFromFile(m_pGraphicDev, pFilePath, &m_pTexture);

    if (FAILED(Hr))
    {
        MSG_BOX("Created Paticle Texture Failed!!");
        return E_FAIL;
    }

    return NOERROR;
}
```

동적 버퍼를 생성 하였습니다.

```
void CSnowPaticle::Paticle_Update(const _float& fTimeDelta)
{
    list<PATICLEATTRIBUTE>::iterator iter = m_PaticleList.begin();
    list<PATICLEATTRIBUTE>::iterator iter_end = m_PaticleList.end();

    for ( ; iter != iter_end ; ++iter)
    {
        iter->vPositon += iter->vVelocity * fTimeDelta;
        iter->fAge += fTimeDelta;

        // 포인트의 경계를 벗어나는가?
        if (iter->fAge >= iter->fLifeTime)
        {
            // 경계를 벗어난 파티클을 재활용한다.
            Reset_Paticle(&(*iter));
        }
    }
}
```

파티클이 정해둔 바운딩 박스를 나가면 다시 셋팅되도록 하였습니다.

파티클

```
void CSnowPaticle::Reset_Paticle(PATICLEATTRIBUTE * attribute)
{
    attribute->fLifeTime = static_cast<float>( rand( ) % 6 + 5);
    attribute->fAge = 0.f;

    attribute->bisAlive = true;

    // 눈송이의 위치를 정하기 위하여 임의의 좌표를 얻는다
    GetRandomVector(&attribute->vPositon, &m_boundingbox.vMin, &m_boundingbox.vMax);

    // 높이 y는 항상 경계 상자의 최상단이 된다.
    attribute->vPositon.y = m_boundingbox.vMax.y;

    _float fXRd = _float(rand( )%5 );

    // 눈송이는 아래쪽으로 떨어지며 약간 왼쪽을 향한다.
    attribute->vVelocity.x = GetRandomFloat(0.0f, 1.0f ) * -fXRd;
    attribute->vVelocity.y = (GetRandomFloat(0.0f, 1.0f ) * -6.0f ) - 3.f;
    attribute->vVelocity.z = 0.0f;

    // 흰색의 눈송이
    attribute->dwColor = D3DXCOLOR(1.f, 1.f, 1.f, 1.f );
}
```

파티클이 다시 셋팅 되면 위치와 방향을 다시 셋팅 하도록 하였습니다.

- 클라이언트 내 영상

< 3D Person Project >

< 3D Person Project >

- UV Animation

- 사용 목적 : 특정 서브셋에 존재하는 텍스쳐의 정점의 시간에 따라 UV값을 변화하여 움직이는 것처럼 구현하였습니다.

- 코드

```
VS_OUT VS_MAIN_UVANIMATION(VS_IN In)
{
    VS_OUT          Out = (VS_OUT)0;

    matrix          matWV, matWVP;

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

    vector          vNormal_World = mul(vector(In.vNormal, 0.f), g_matWorld);

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

    Out.vWorldNormal = normalize(vNormal_World);

    Out.vTexUV = In.vTexUV + g_vOffsetUV;

    Out.vViewPos = mul( vector( In.vPosition, 1.f ), matWV );

    Out.vWorldPos = mul( vector( In.vPosition, 1.f ), g_matWorld );

    return Out;
}
```

텍스처의 UV좌표를 필요한 부분을 fTimeDelta만큼씩 움직이도록 하였습니다.

- 클라이언트 내 영상

< 3D Team Project >

< 3D Team Project >

6. 참고 문헌

< 서적 >

* 열혈 강의 C언어 프로그램

- 저자 : 윤성우

* 열혈 강의 자료구조

- 저자 : 윤성우

* DirectX 9을 이용한 3D GAME 프로그래밍 입문

- 저자 : Frank D. Luna

< 참고 사이트 >

* Slide Vector

http://toymaker.tistory.com/entry미끄러짐-벡터-Sliding-Vector

* 법선(노말) 맵핑

http://dolphin.ivyro.net/file/shader9.0/tutorial04.html

* UV 애니메이션

http://blog.naver.com/PostView.nhn?blogId=lhj57095709&logNo=220735328894

감사합니다.

이름 : 권오현

번호 : 010.2719.5330

이메일 : ohhyun5442@gmail.com