

게임엔진

제9강 카메라

한국산업기술대학교 이대현



학습 안내

■ 학습 목표

- 오우거 엔진의 입력 처리 방식을 종합적으로 이해한다.
- 오우거 엔진의 카메라의 활용 방법 및 뷰포트의 활용 방법을 이해하고 다양한 방식으로 게임 화면을 구성해본다.

■ 학습 내용

- 오우거 엔진의 입력 처리 방식.
- 키입력 및 마우스입력 리스너 인터페이스.
- 카메라의 이동 및 회전.
- 시야 절두체 컬링의 이해.
- 뷰포트를 이용한 PIP의 구현.

오우거 엔진의 입력 처리 방식

■ 무버퍼 입력(unbuffered input) 방식

- 일종의 폴링(polling).
- 프레임 리스너에서 매번 입력 장치의 상태를 캡처한 후, 그 결과에 따른 처리.

■ 버퍼 입력(buffered input) 방식

- 일종의 인터럽트 방식.
 - 이벤트의 발생 및 이에 따른 처리.
- 키 리스너, 마우스 리스너 등의 인터페이스를 사용하여 처리.
 - 키가 눌리면 `KeyListener::keyPressed` 이벤트 발생.
 - 키가 떼어지면 `KeyListener::keyReleased` 이벤트 발생.
 - 마우스가 움직이면 `MouseListener::mouseMoved` 이벤트 발생.

KeyListener 인터페이스

■ 키보드의 입력을 처리하는 인터페이스 클래스

- 키가 눌렸다 떼어질 때마다, 두개의 이벤트: *keyPressed*, *keyReleased* 가 발생.
- 눌린 키의 정보는 *KeyEvent* 구조체에 담겨있음.

```
class _OISExport KeyListener
{
public:
    virtual ~KeyListener() {}
    virtual bool keyPressed( const KeyEvent &eve ) = 0;
    virtual bool keyReleased( const KeyEvent &evt ) = 0;
};
```

- 처리가 끝나면 반드시 *true* 를 *return* 해 주어야 함.
- *false*를 리턴하게 되면, 입력 버퍼에 남아있는 내용을 제거함.


MouseListener 인터페이스


■ 마우스의 클릭 입력을 처리하는 인터페이스 클래스

```
class _OIExport MouseListener
{
public:
    virtual ~MouseListener() {}
    virtual bool mouseMoved( const MouseEvent &evt ) = 0;

    virtual bool mousePressed( const MouseEvent &evt, MouseButtonID id ) = 0;
    virtual bool mouseReleased( const MouseEvent &evt, MouseButtonID id ) = 0;
};
```

• 마우스 버튼ID는 *MB_Left, MB_Right, MB_Middle*



- 
- *evt.state.X.rel* 에는 마우스 x 위치의 상대좌표(원래 위치와의 차이)값.
 - *evt.state.X.abs* 에는 마우스 x 위치 절대 좌표값(렌더링 윈도우 기준).
 - *evt.state.y.rel, evt.state.y.abs*
 - *evt.state.z.rel, evt.state.z.abs* → 마우스 휠 정보

키 리스너 및 마우스 리스너 설정

• *KeyListener, MouseListener* 인터페이스 사용

```
class InputController : public FrameListener,
                        public OIS::KeyListener,
                        public OIS::MouseListener
{
public:
    InputController(Root* root, OIS::Keyboard *keyboard, OIS::Mouse *mouse) :
        mRoot(root), mKeyboard(keyboard), mMouse(mouse)
    {
        ... 중략 ...
        keyboard->setEventCallback(this);
        mouse->setEventCallback(this);
    }
    ... 중략 ...
```

• 마우스 및 키입력 콜백 함수 등록

```
bool mouseMoved( const OIS::MouseEvent &evt );
bool mousePressed( const OIS::MouseEvent &evt, OIS::MouseButtonID id );
bool mouseReleased( const OIS::MouseEvent &evt, OIS::MouseButtonID id );

bool keyPressed( const OIS::KeyEvent &evt );
bool keyReleased( const OIS::KeyEvent &evt );
```

} 마우스
입력 처리 구현

} 키 입력 처리 구현

```
... 후략 ...
};
```

오우거 엔진의 카메라

■ Camera 클래스

- 씬을 담아내어, 뷰포트에 연결시켜주는 클래스
- 주요 멤버 함수
 - setPosition(), lookAt(), yaw(), roll(), pitch()
- 어떤 장면을 여러 각도에서 보고 싶으면?
 - “카메라 홀더” 라고 불리는 씬노드들을 여러 개 만들고, 필요에 따라 씬노드에 카메라를 담으면 된다.

실습



CameraControl

WASD 키와 마우스를 이용한 카메라 조작

카메라 조작 시나리오

■ 키보드 입력

- 'W' 키: 카메라의 상향 이동
- 'S' 키: 카메라의 하향 이동
- 'A' 키: 카메라의 왼쪽 이동
- 'D' 키: 카메라의 오른쪽 이동

■ 마우스 입력

- 마우스 좌우이동: 카메라의 좌우 회전(Yaw) - 패닝(Panning)
- 마우스 상하이동: 카메라의 상하 회전(Pitch) - 피칭(Pitching)
- 마우스 휠이동: 카메라의 전진 후진 - 주밍(Zooming)

class InputController



```
class InputController : public FrameListener,
                        public OIS::KeyListener,
                        public OIS::MouseListener
{
public:
    InputController(Root* root, OIS::Keyboard *keyboard, OIS::Mouse *mouse) :
        mRoot(root), mKeyboard(keyboard), mMouse(mouse)
    {
        mCamera = mRoot->getSceneManager("main")->getCamera("main");
        mCameraMoveVector = Ogre::Vector3::ZERO;

        mContinue = true;

        keyboard->setEventCallback(this);
        mouse->setEventCallback(this);
    }

    bool frameStarted(const FrameEvent &evt)
    {
        mKeyboard->capture();
        mMouse->capture();

        mCamera->moveRelative(mCameraMoveVector);

        return mContinue;
    }
}
```

class InputController



```
bool keyPressed( const OIS::KeyEvent &evt )
{
    switch(evt.key)
    {
        case OIS::KC_W: mCameraMoveVector.y += 1; break;
        case OIS::KC_S: mCameraMoveVector.y -= 1; break;
        case OIS::KC_A: mCameraMoveVector.x -= 1; break;
        case OIS::KC_D: mCameraMoveVector.x += 1; break;
        case OIS::KC_ESCAPE: mContinue = false; break;
    }

    return true;
}

bool mouseMoved( const OIS::MouseEvent &evt )
{
    if (evt.state.buttonDown(OIS::MB_Right))
    {
        mCamera->yaw(Degree(-evt.state.X.rel));
        mCamera->pitch(Degree(-evt.state.Y.rel));
    }

    mCamera->moveRelative(Ogre::Vector3(0, 0, -evt.state.Z.rel * 0.1f));

    return true;
}
```



```
void go(void)
{
    ... 중략 ...

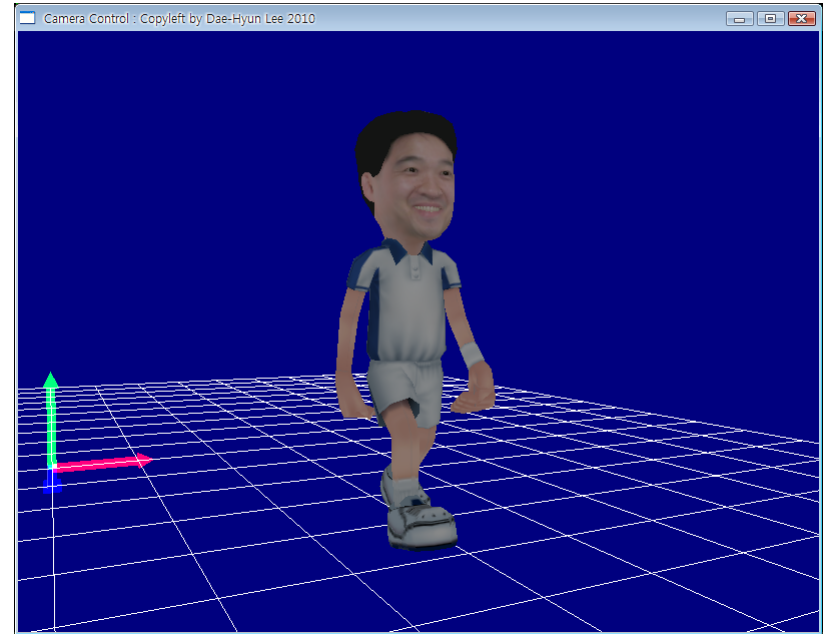
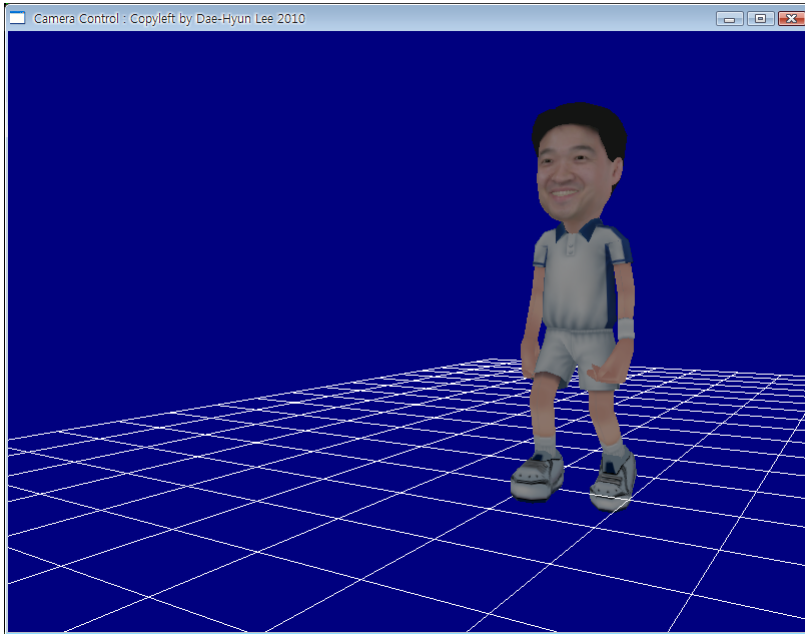
    mKeyboard = static_cast<OIS::Keyboard*>(mInputManager->createInputObject(OIS::OISKeyboard, true));
    mMouse = static_cast<OIS::Mouse*>( mInputManager->createInputObject(OIS::OISMouse, true));

    InputController* inputController = new InputController(mRoot, mKeyboard, mMouse);
    mRoot->addFrameListener(inputController);

    ... 중략 ...
}
```

실행 결과

- WASD 키를 이용한 카메라의 상하좌우 이동
- 마우스(오른쪽버튼누른상태)를 이용한 패닝 및 피칭
- 마우스 휠을 이용한 카메라 Zoom-in / Zoom-out



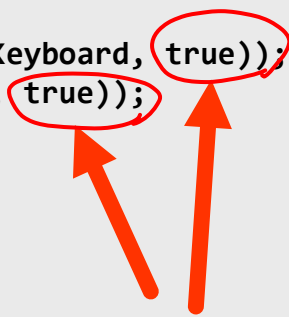
버퍼 입력(Buffered Input) 설정

```
void go(void)
{
    ... 중략 ...

    mKeyboard = static_cast<OIS::Keyboard*>(mInputManager->createInputObject(OIS::OISKeyboard, true));
    mMouse = static_cast<OIS::Mouse*>( mInputManager->createInputObject(OIS::OISMouse, true));

    InputController* inputController = new InputController(mRoot, mKeyboard, mMouse);
    mRoot->addFrameListener(inputController);

    ... 중략 ...
}
```



- 버퍼 입력 설정

콜백 함수 지정

```
InputController(Root* root, OIS::Keyboard *keyboard, OIS::Mouse *mouse) :  
    mRoot(root), mKeyboard(keyboard), mMouse(mouse)  
{  
    mCamera = mRoot->getSceneManager("main")->getCamera("main");  
  
    mCameraMoveVector = Ogre::Vector3::ZERO;  
  
    mContinue = true;  
    keyboard->setEventCallback(this);  
    mouse->setEventCallback(this);  
}
```

→ WASD 키에 의해 조정되는 카메라 이동 vector

→ ESC 이 눌리면 InputController 프레임워크가 false를 return하도록 함.

→ 키입력이 capture된 경우, 이 객체의 관련함수를 호출함.

```
bool frameStarted(const FrameEvent &evt)
```

```
{
```

```
    mKeyboard->capture();
```

```
    mMouse->capture();
```

→ 매 프레임마다 입력을 capture 함

```
    mCamera->moveRelative(mCameraMoveVector);
```

```
    return mContinue;
```

```
}
```

~~~~~  
렌더링 루프의 반복 여부

↘ 저장된 이동벡터만큼 카메라 위치를 조정함.  
벡터의 삭제값은 keyPressed() /  
keyReleased() 에서 저장됨.



# WASD 키를 이용한 카메라 이동 벡터 설정

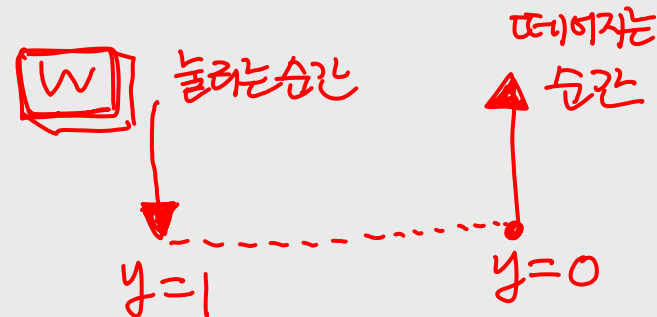
```
bool keyPressed( const OIS::KeyEvent &evt )
{
    switch(evt.key)
    {
        case OIS::KC_W: mCameraMoveVector.y += 1; break;
        case OIS::KC_S: mCameraMoveVector.y -= 1; break;
        case OIS::KC_A: mCameraMoveVector.x -= 1; break;
        case OIS::KC_D: mCameraMoveVector.x += 1; break;
        case OIS::KC_ESCAPE: mContinue = false; break;
    }
    return true;
}
```

• WASD 키에 따라, y축 또는 x축으로 이동값 설정.

ESC 누르면 렌더링 loop 반복을 중단함.

```
bool keyReleased( const OIS::KeyEvent &evt )
{
    switch(evt.key)
    {
        case OIS::KC_W: mCameraMoveVector.y -= 1; break;
        case OIS::KC_S: mCameraMoveVector.y += 1; break;
        case OIS::KC_A: mCameraMoveVector.x += 1; break;
        case OIS::KC_D: mCameraMoveVector.x -= 1; break;
        case OIS::KC_ESCAPE: mContinue = false; break;
    }

    return true;
}
```



떼어지는 순간 이동 벡터의 값이 설정

# 마우스 입력에 따른 카메라 조작

```
bool mouseMoved( const OIS::MouseEvent &evt )  
{
```

- 오른쪽 마우스가 눌렸을 때, 마우스  $x, y$  좌표의 상대적 이동량을 이용하여, 카메라의 회전을 처리함.

```
    if (evt.state.buttonDown(OIS::MB_Right))  
    {  
        mCamera->yaw(Degree(-evt.state.X.rel));  
        mCamera->pitch(Degree(-evt.state.Y.rel));  
    }
```

```
    mCamera->moveRelative(Ogre::Vector3(0, 0, -evt.state.Z.rel * 0.1f));
```

~~~~~  
카메라의 좌표축을
기점으로 한 이동

```
    return true;
```

```
}
```

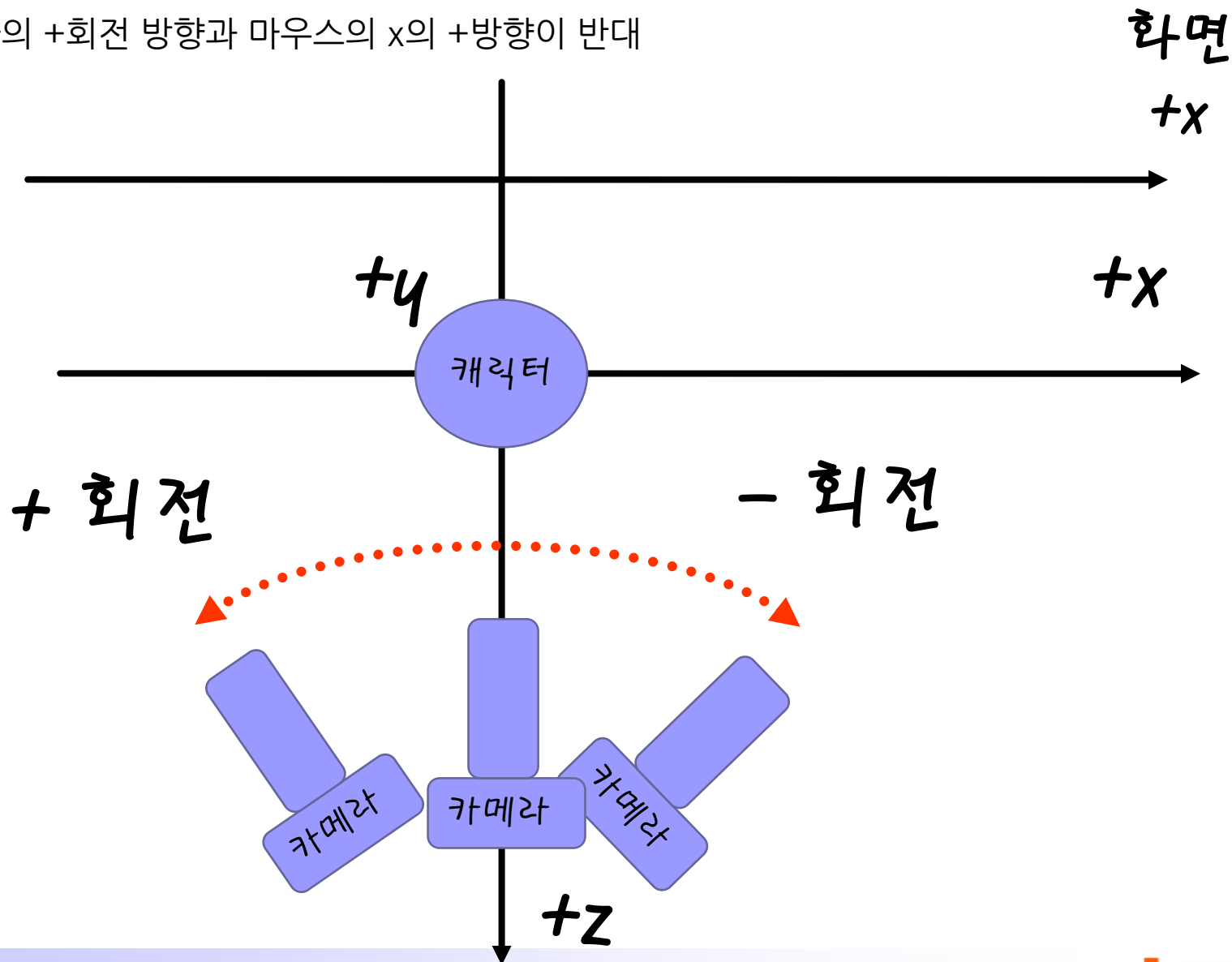
- 휠 상태에 따른 카메라의 Zooming

± 2 방향으로만
↓ 카메라를 이동시킴.

마우스 이동에 따른 카메라의 회전: yaw

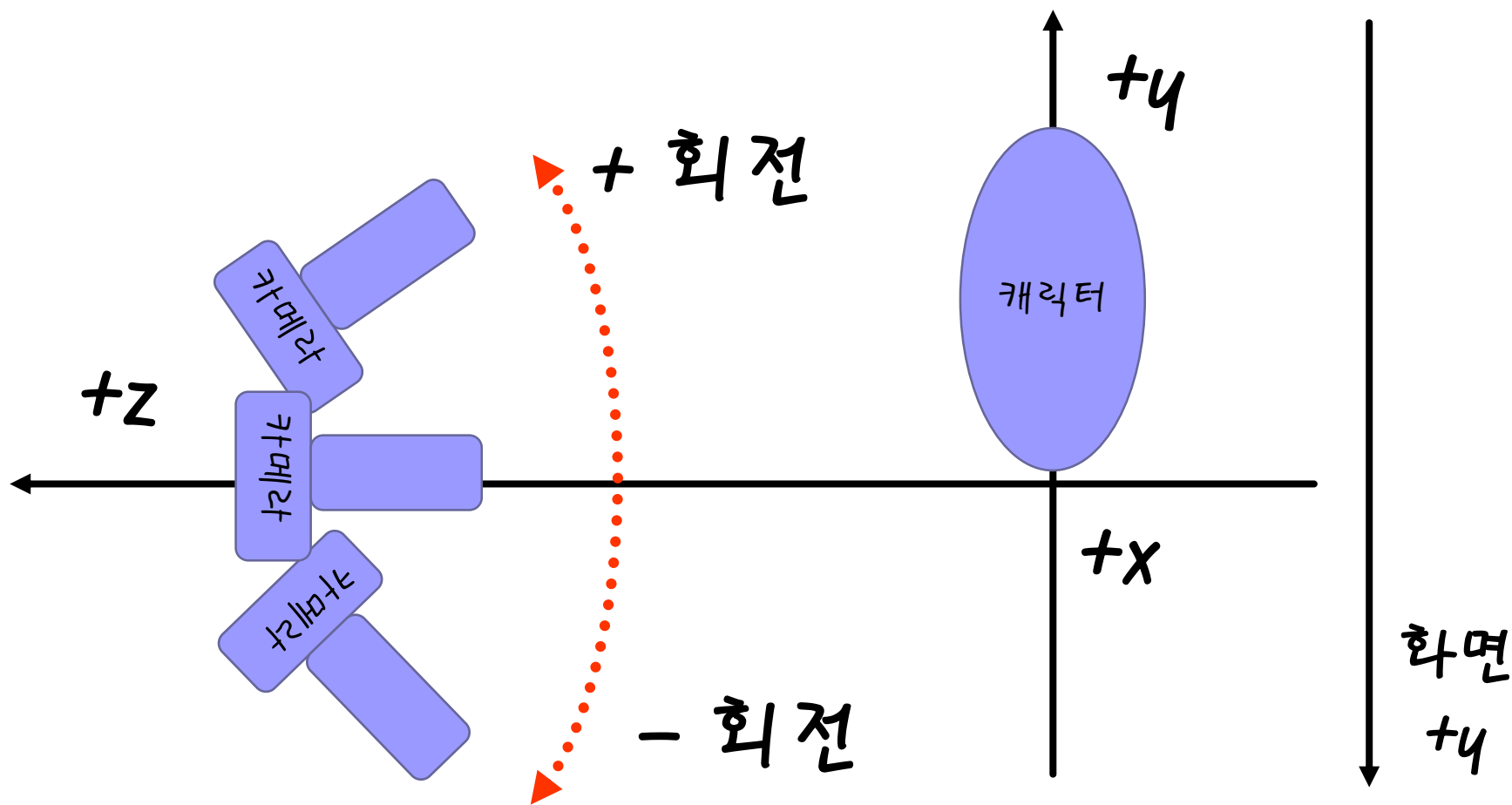
- XZ 평면에서 Y축을 중심으로 하는 회전.

- 카메라의 +회전 방향과 마우스의 x의 +방향이 반대

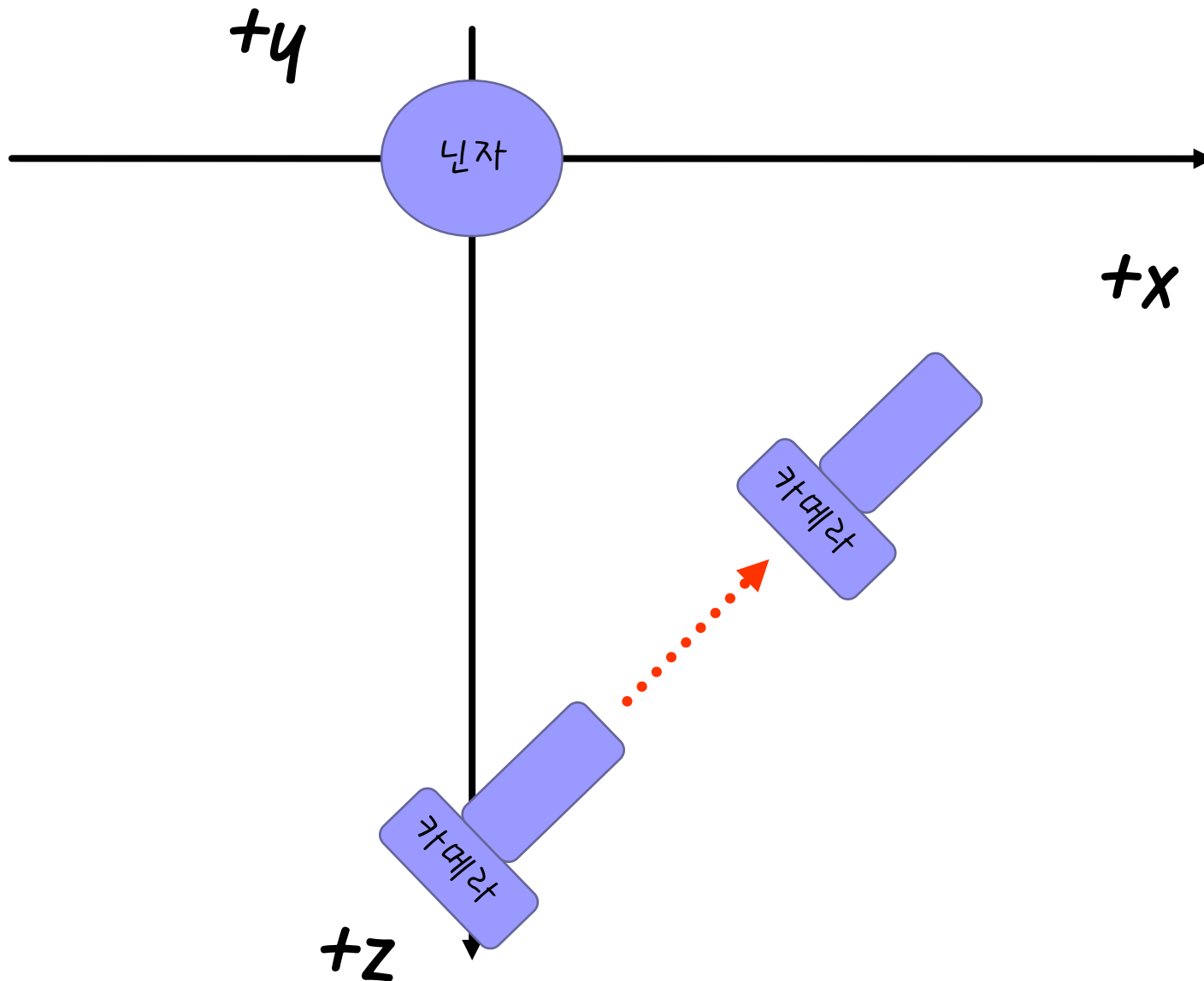


마우스 이동에 따른 카메라의 회전: pitch

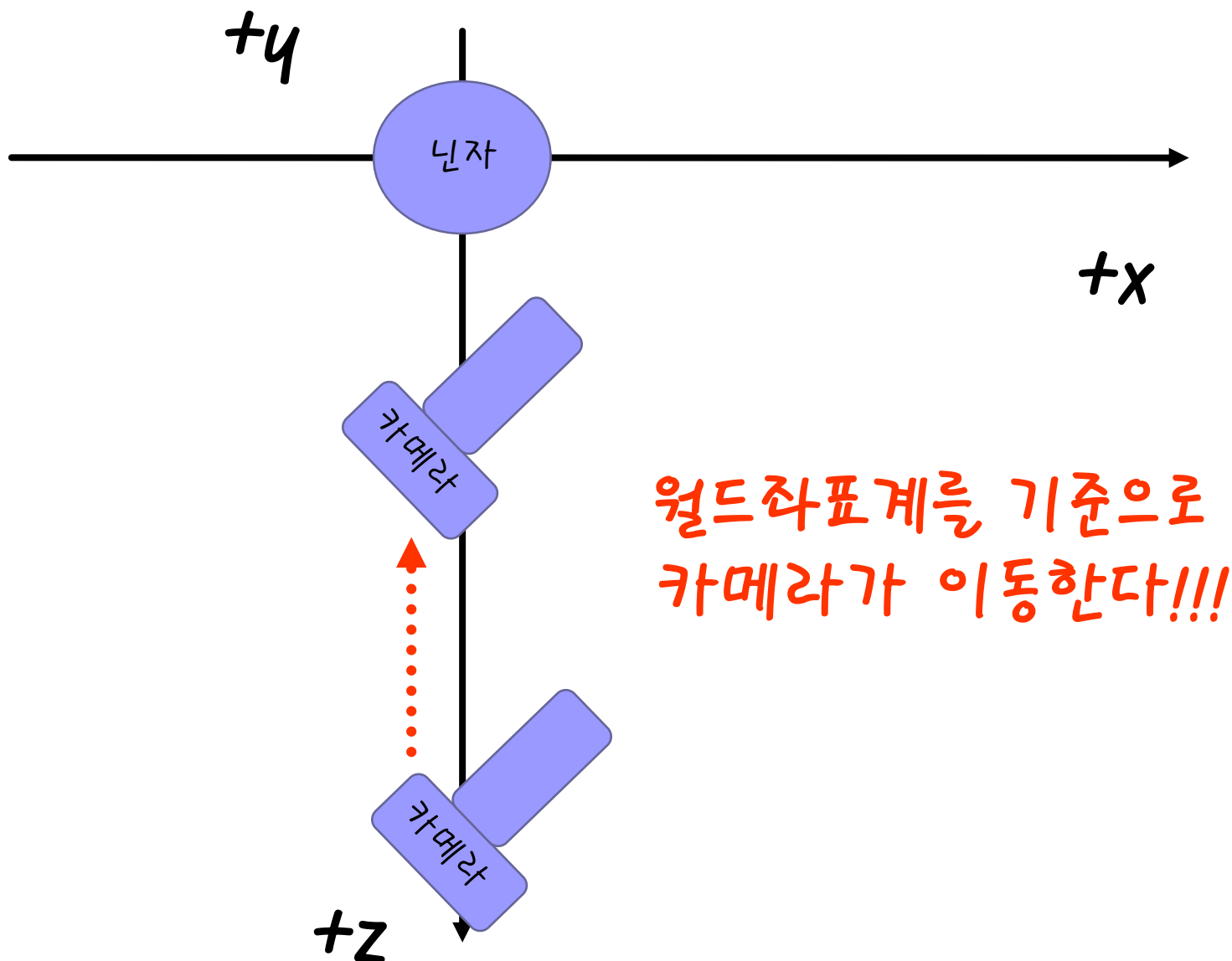
- YZ 평면에서 X축을 중심으로 하는 회전.
 - 카메라의 + 회전 방향과 마우스의 y의 +방향이 반대



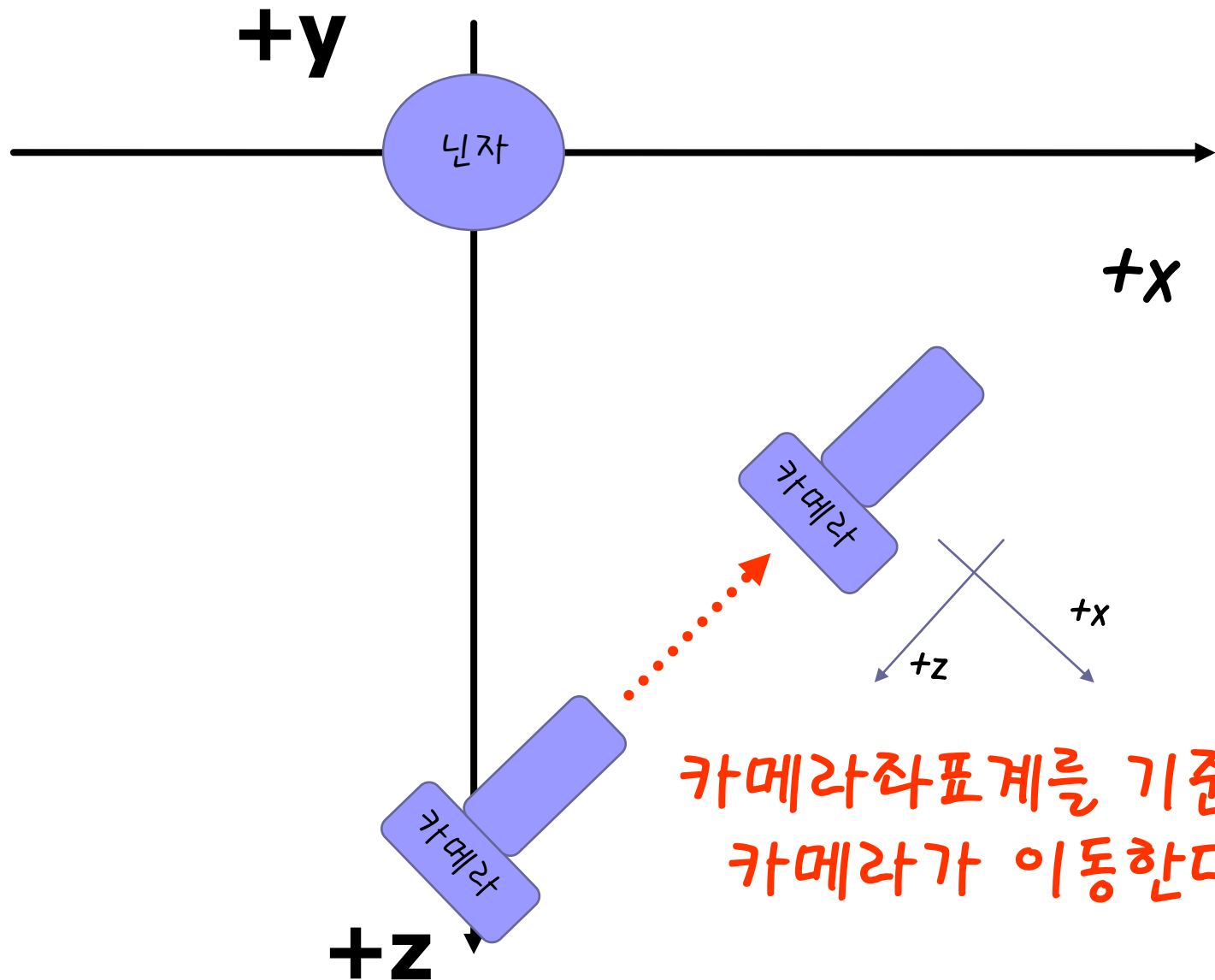
Zoom-in 할 때, 실제로 요구되는 카메라의 동작



move()함수의 경우, 카메라의 이동



moveRelative() 함수의 동작



카메라좌표계를 기준으로
카메라가 이동한다!!!

실습

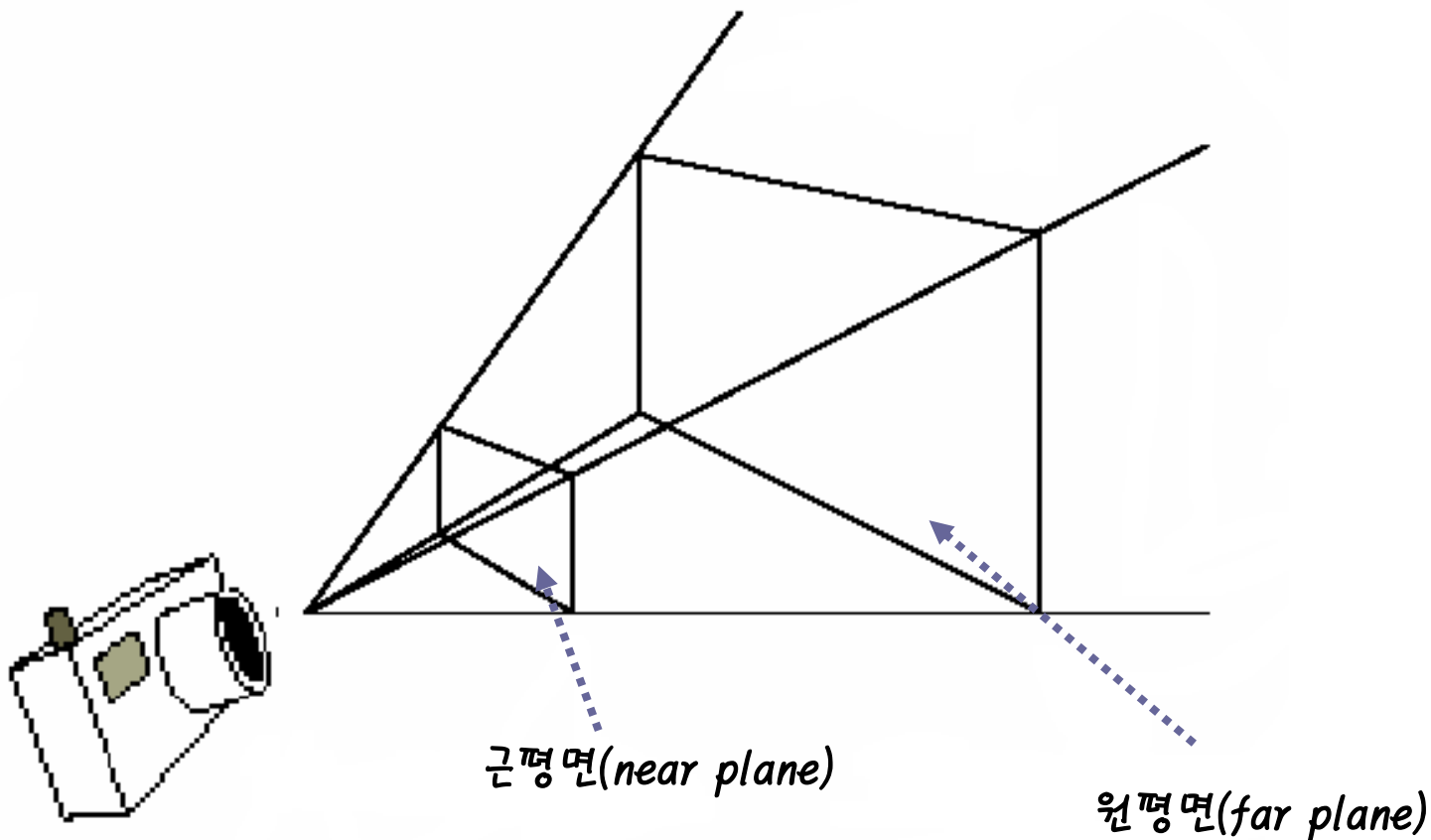


Frustrum
시야 절두체 설정

시야 절두체(Viewing Frustum)

■ 3차원의 공간상에서 카메라에 의해 보일 수 있는 것들을 담는 범위

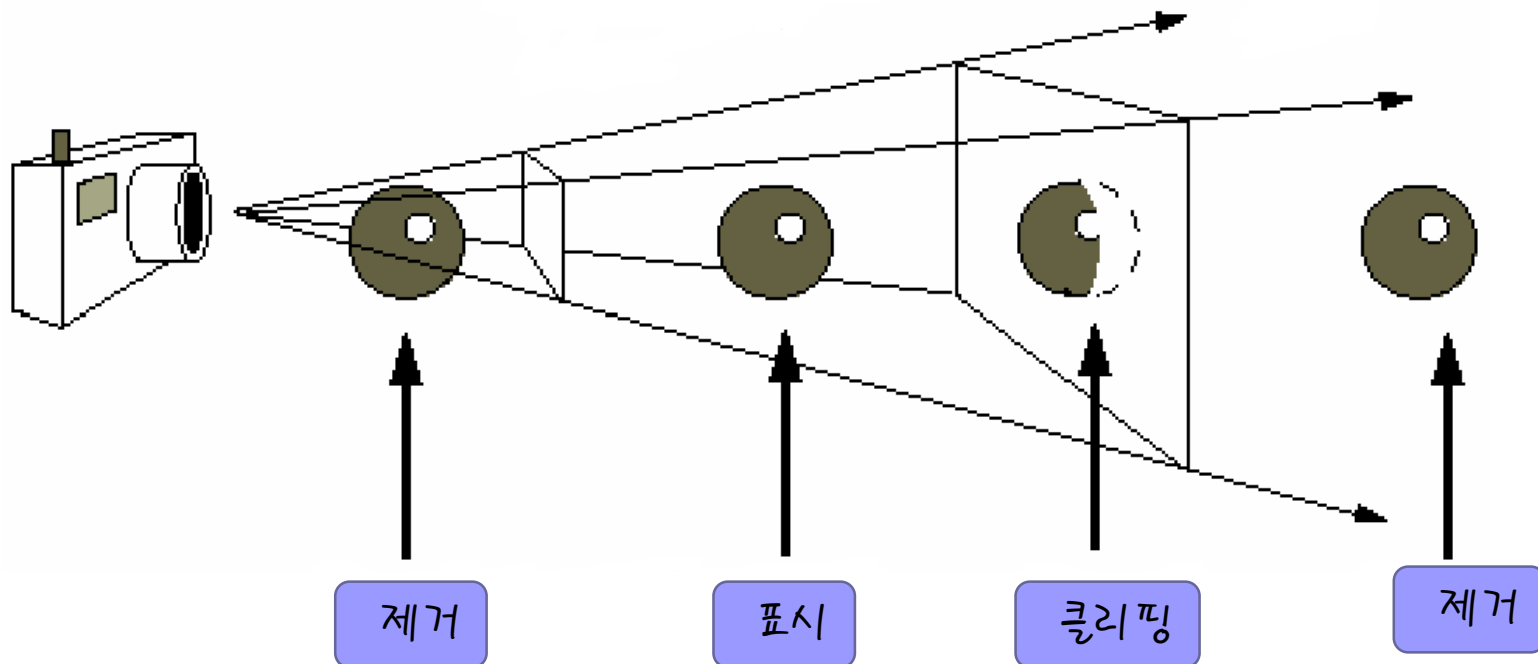
- 근클리핑거리(near clipping distance): 카메라와 근평면까지의 거리
- 원클리핑거리(far clipping distance): 카메라와 원평면까지의 거리



절두체 컬링(Frustum Culling)

■ 컬링 방법

- 카메라로부터 아주 멀리 있는 물체는? → 눈에 안보인다 → 표시할 필요가 없다. → 렌더링하지 않는다. → 계산량이 줄어든다.
- 카메라로부터 아주 가까이 있는 물체는? 그 물체를 투과한다. → 표시할 필요가 없다. → 렌더링하지 않는다. → 계산량이 줄어든다.

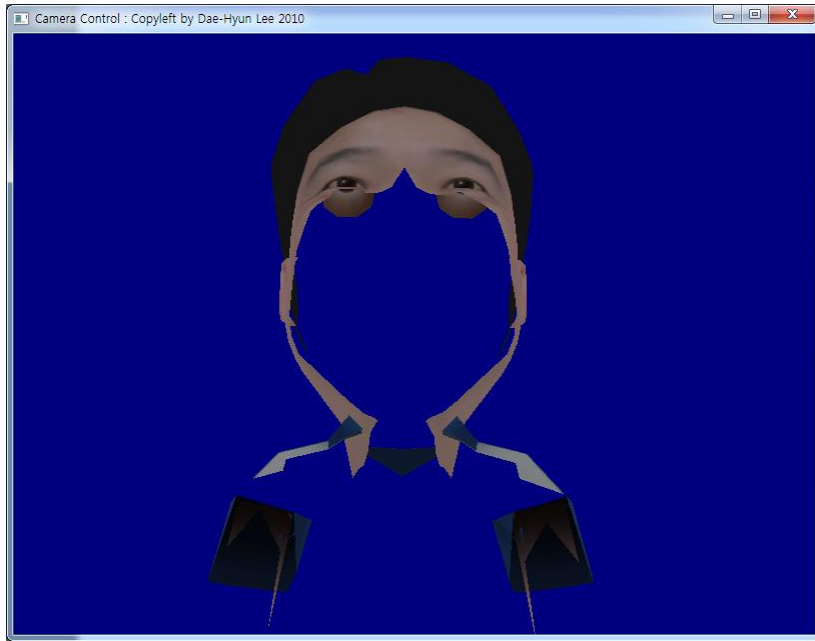




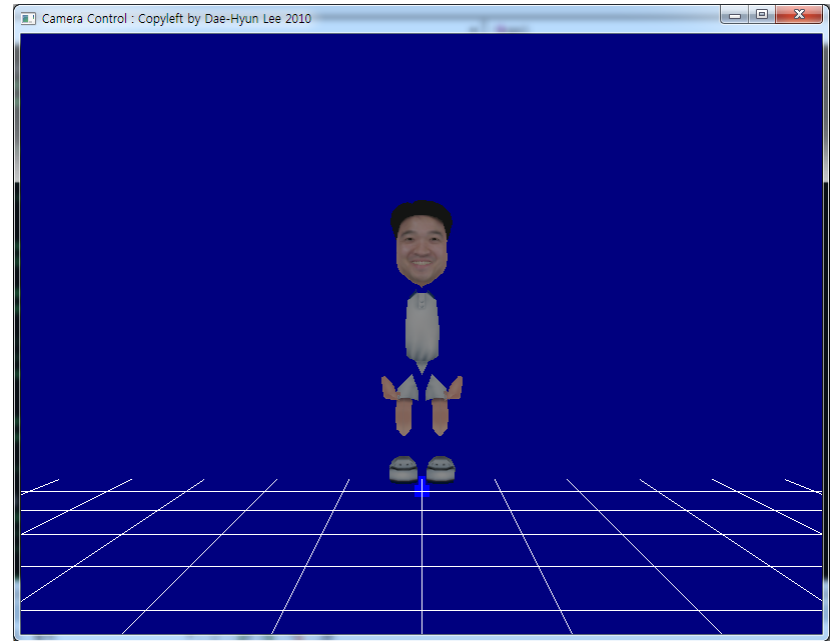
```
void go(void)
{
    ... 중략 ...
    mCamera->setPosition(0.0f, 100.0f, 500.0f);
    mCamera->lookAt(0.0f, 100.0f, 0.0f);
    mCamera->setNearClipDistance(100);
    mCamera->setFarClipDistance(500);

    ... 중략 ...
}
```

카메라 줌인 줌아웃에 따른 클리핑 효과



Near clipping 거리에 따른 효과



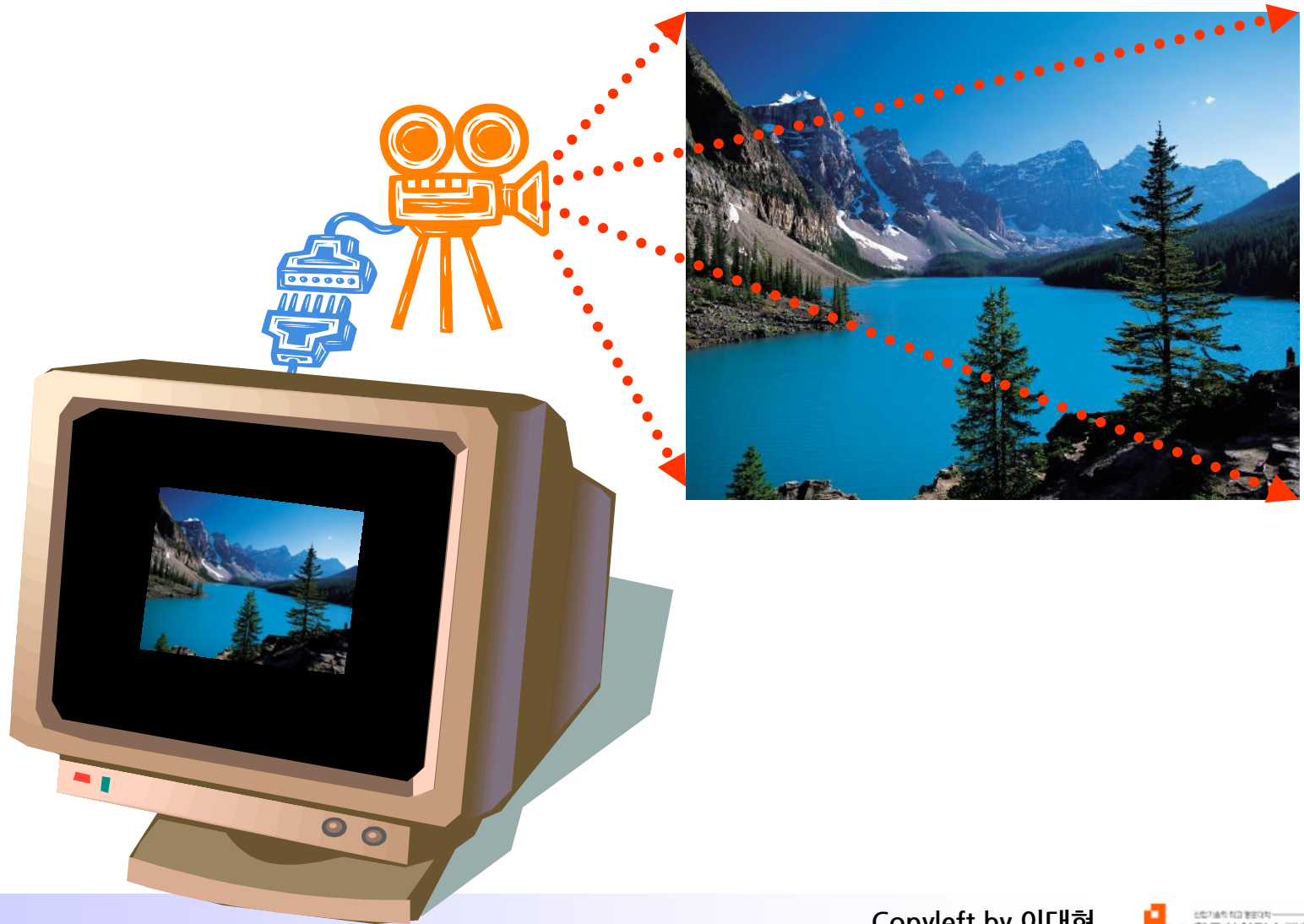
Far clipping 거리에 따른 효과

실습



PictureInPicture
PIP의 구현

- 카메라로 잡은 장면을 실제로 보여주는 모니터(렌더 윈도우) 상의 2차원 평면



■ Viewport* Ogre::RenderTarget::addViewport (Camera * cam, int ZOrder = 0, float left = 0.0f, float top = 0.0f, float width = 1.0f, float height = 1.0f)

- cam: 카메라 객체
- ZOrder: 뷰포트의 계층 순서(0이 맨 아래)
- left, top: 뷰포트의 왼쪽 위의 좌표(0-1사이의 값으로써 렌더 윈도우상에서 비율값으로 표시).
- width, height: 뷰포트의 너비와 높이(0-1사이의 값으로써 렌더 윈도우 상에서 비율값으로 표시).



```
void go(void)
{
    ... 중략 ...

    mCamera = mSceneMgr->createCamera("main");
    mCamera->setPosition(0.0f, 100.0f, 500.0f);
    mCamera->lookAt(0.0f, 100.0f, 0.0f);

    mViewport = mWindow->addViewport(mCamera, 0);
    mViewport->setBackgroundColour(ColourValue(0.0f,0.0f,0.5f));

    mCamera->setAspectRatio(Real(mViewport->getActualWidth())
                           /Real(mViewport->getActualHeight()));

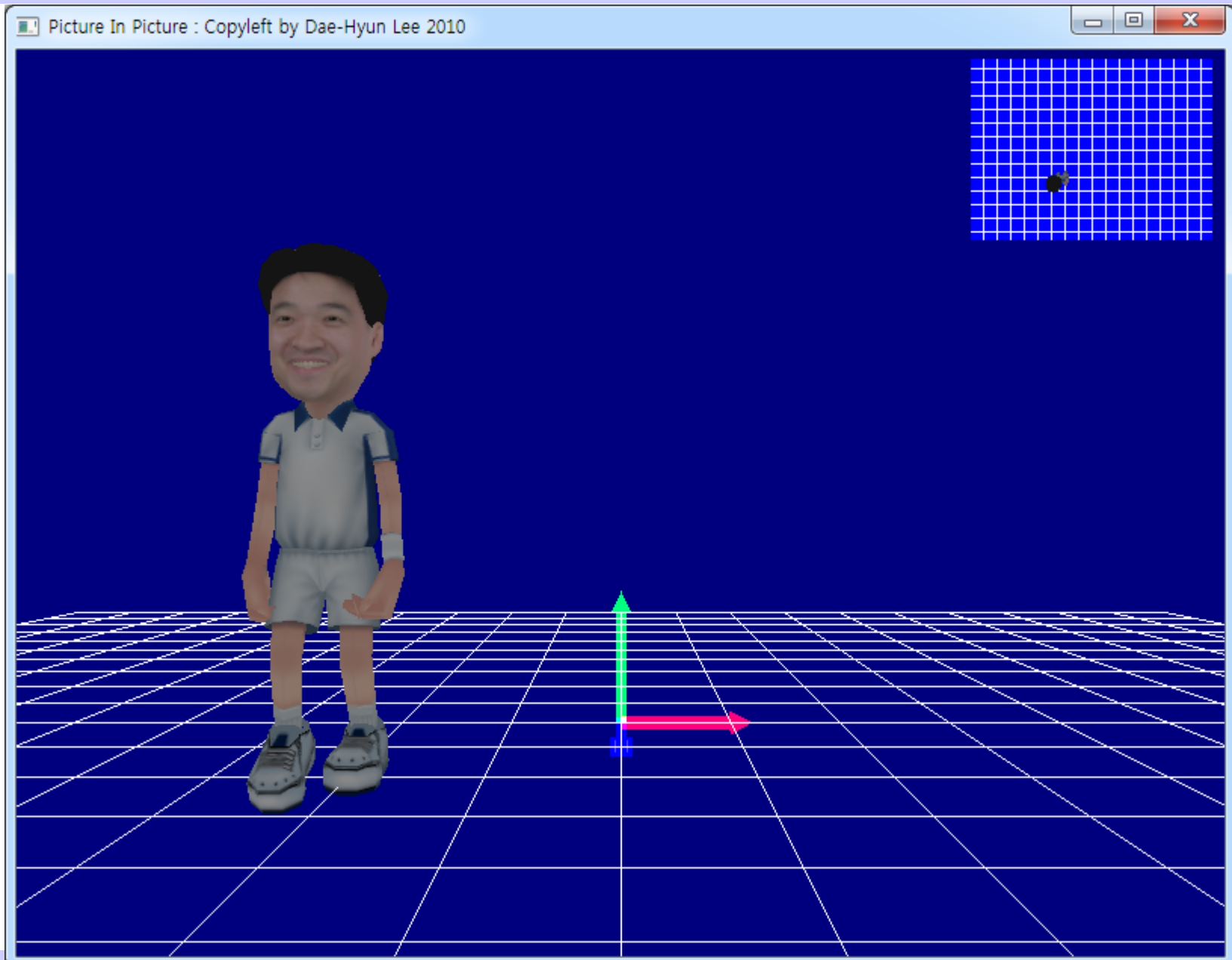
    mMapCamera = mSceneMgr->createCamera("MapCamera");
    mMapCamera->setPosition(0.0f, 800.0f, 1.0f);
    mMapCamera->lookAt(0.0f, 0.0f, 0.0f);

    mMapViewport = mWindow->addViewport(mMapCamera, 1, 0.79, 0.01, 0.2, 0.2);
    mMapViewport->setBackgroundColour(ColourValue(0.0f,0.0f,1.0f));

    mMapCamera->setAspectRatio(Real(mMapViewport->getActualWidth())
                              /Real(mMapViewport->getActualHeight()));

    ... 중략 ...
}
```


실행 화면: PIP 구현 (Picture-In-Picture)



MapCamera의 설정

```
void go(void)
```

```
{
```

```
    mMapCamera = mSceneMgr->createCamera("MapCamera");
```

```
    mMapCamera->setPosition(0.0f, 800.0f, 1.0f);
```

```
    mMapCamera->lookAt(0.0f, 0.0f, 0.0f);
```

```
    mMapViewport = mWindow->
```

```
        addViewport(mMapCamera, 1, 0.79, 0.01, 0.2, 0.2);
```

메인뷰포트(0)
위에 놓음
Screen
W=1, H=1로
한 바퀴

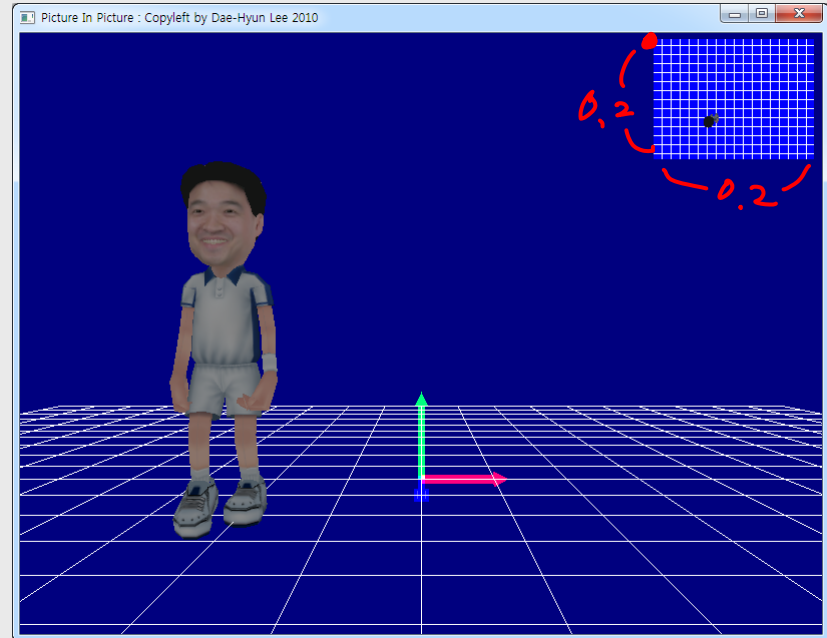
```
    mMapViewport->setBackgroundColour(ColourValue(0.0f,0.0f,1.0f));
```

```
    mMapCamera->setAspectRatio(Real(mMapViewport->getActualWidth())  
                               /Real(mMapViewport->getActualHeight()));
```

```
}
```

카메라가 +y 방향(하늘)으로
높은 위치에서 원점을 내려다보
도록 설정

(0.79, 0.01)



약 1.33 정도임

학습 정리

■ 오우거 엔진의 입력 처리 방식

- 무버퍼 입력 방식 - 폴링.
- 버퍼 입력 방식 - 인터럽트.

■ 키 입력 및 마우스 입력 처리 리스너

- 입력을 capture()하면 OIS는 콜백함수의 keyPressed(), keyReleased(), mouseMoved() 등의 함수를 호출함.

■ 카메라

- 3D 씬을 담아서 뷰포트에 연결함.
- 주요 멤버 함수 - setPosition(), lookAt(), yaw(), roll(), pitch()
- move() 와 moveRelative() 기능 차이.

■ 절두체 컬링

- 시야 내에 있는 것만 렌더링함으로써, 엔진 실행 속도 최적화.

■ 뷰포트

- 카메라를 통해서 담아 낸 장면을 표시하는 사각 영역.
- addViewport(), setBackgroundColour(), setAspectRatio()