게임엔진

제7강 3D 변환

학습 안내

■ 학습 목표

- □ 3D 오브젝트를 3D 공간에서 다양한 방식으로 변형(transform)하는 방법을 이해한다.
- □ 사원수의 개념을 이해하고, 오우거 엔진의 사원수 관련 함수 실습을 통해서 공간 상에서 3D 오브젝트를 자유 자재로 회전할 수 있는 능력을 기른다.

■ 학습 내용

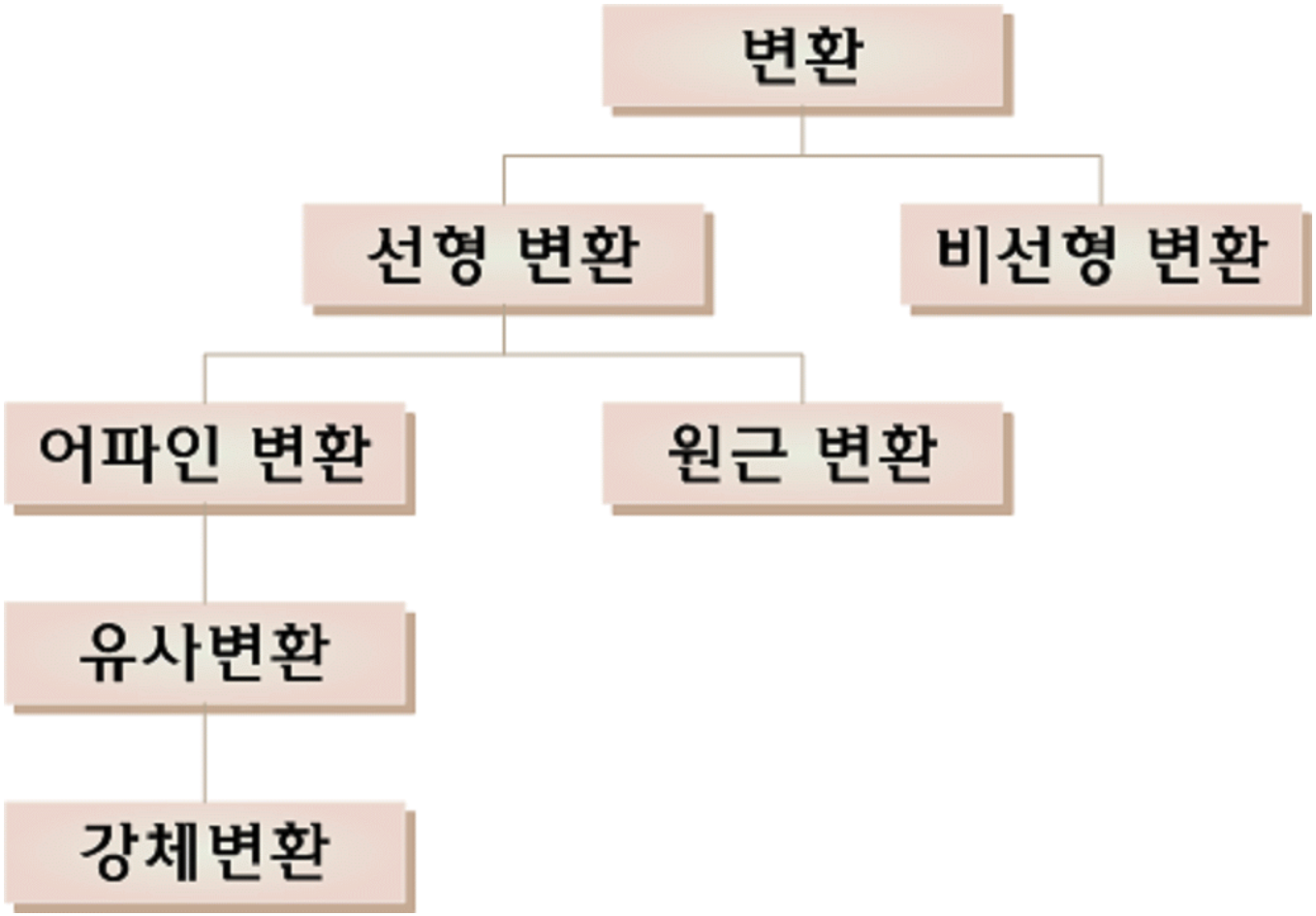
- □ 3D 오브젝트의 이동
- □ 3D 오브젝트의 확대 및 축소(Scaling)
- □ 3D 오브젝트의 회전(Rotation)
- □ 변환 공간(Transform Space)
- □ Ogre3D 엔진을 3D 변환 실습
- □ 오일러 회전의 개념과 특성의 이해.
- □ 사원수의 개념.
- □ 사원수를 이용한 회전 실습.
- □ Slerp 구면 보간.
- □ 캐릭터의 부드러운 회전 실습.

기하 변환(Geometry Transformation)

- 물체 변환 또는 좌표계 변환의 기본
- 이동, 회전, 크기조절 등
- 행렬을 이용한 계산이 주로 활용(선형 변환)

$$(x, y, z) \rightarrow (x', y', z')$$

변환의 분류



■ 강체변환(Rigid Body Transformation)

- □ 이동변환, 회전변환
- □ 물체 자체의 모습은 불변

■ 유사변환(Similarity Transformation)

- □ 강체변환 + 균등 크기조절 변환, 반사변환
- □ 물체면 사이의 각이 유지됨.
- □ 물체내부 정점간의 거리가 일정한 비율로 유지됨

■ 어파인변환(Affine Transformation)

- □ 유사변환 + 차등 크기조절 변환, 전단변환
- □ 물체의 타입이 유지
  - 직선은 직선으로, 다각형은 다각형으로, 곡면은 곡면으로
  - 평행선이 보존
  - 변환행렬의 마지막 행이 항상 (0, 0, 0, 1)

■ 원근변환(Perspective Transformation)

- □ 평행선이 만남.
- □ 직선이 직선으로 유지
- □ 변환행렬의 마지막 행이 (0, 0, 0, 1) 아님.

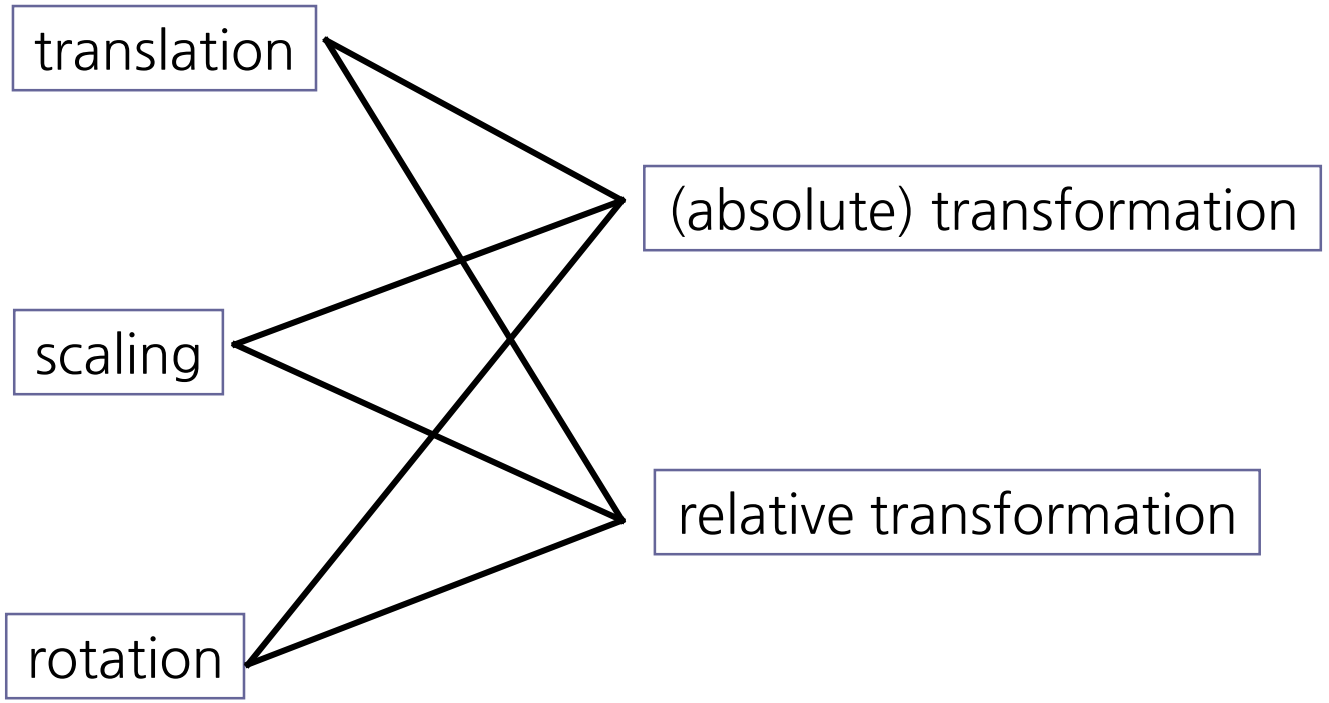
■ 선형변환(Linear Transformation)

- □ 어파인 변환 + 원근 변환
- □ 선형 조합(Linear Combination)으로 표시되는 변환
- □ x' = ax + by + cz에서 x'는 x, y, z 라는 변수를 각각 상수 배 한 것을 더한 것이다.

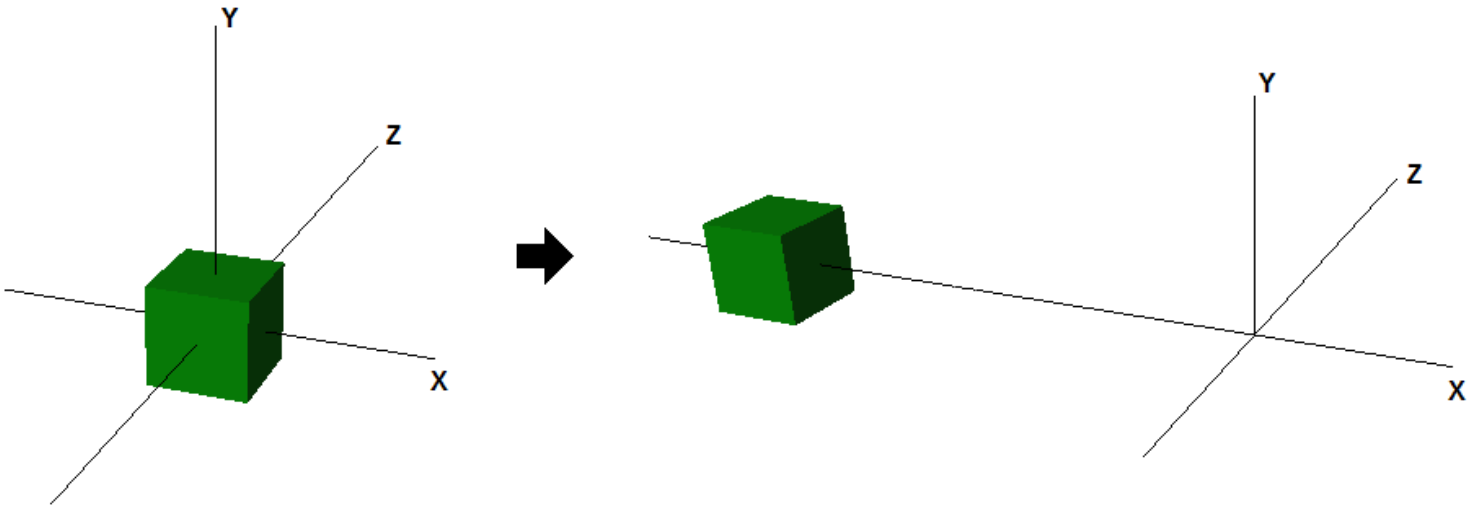
Absolute VS. Relative Transformation

- **절대 변환**
  - 최종값을 지정
  - 원점을 기준으로 하면, 원점과의 차이값

- **상대변환**
  - 차이값을 지정



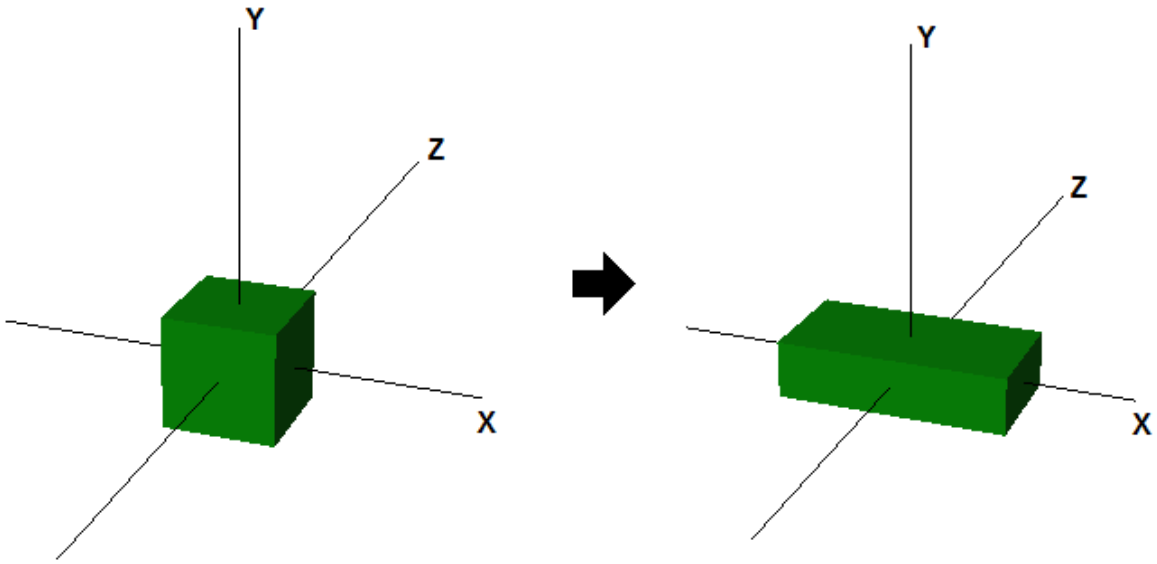
이동(translation)



- **void setPosition (const Vector3 &pos)**
  - 씬노드의 위치를 재설정.
  - 씬노드의 부모노드의 위치를 기준으로 하는 상대값임.

- **void translate (const Vector3 &d, TransformSpace relativeTo=TS_PARENT)**
  - 씬노드를 지정된 벡터만큼 이동한다.
  - 디폴트로 부모노드의 transform space를 기준으로 하여 이동한다.

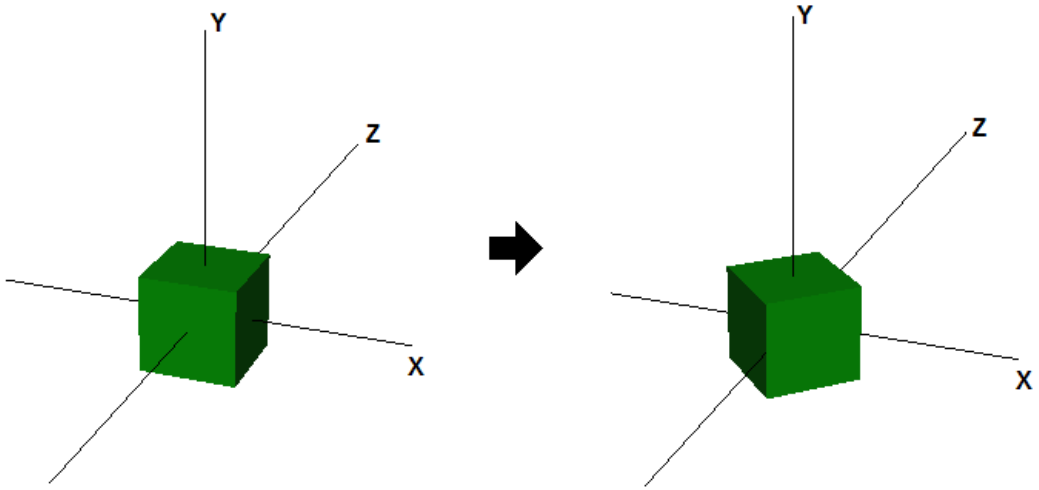
크기조정(scale)



- **void setScale (const Vector3 & scale), void setScale (Real x, Real y, Real z)**
  - 씬노드에 속해있는 엔터티들의 크기 확장 및 축소.
  - 씬노드의 절대적 스케일 값을 설정.

- **void scale (Real x, Real y, Real z), void scale (const Vector3 & scale)**
  - 씬노드에 속해있는 엔터티들의 크기 확장 및 축소.
  - 이미 크기변환된 노드에 적용하면 앞서 했었던 크기변환의 위에 적용됨. 상대적 개념.
  - scale(Vector3(2,2,2))를 두 번 적용하면, setScale(Vector3(4,4,4))와 같음.

회전(Rotation)

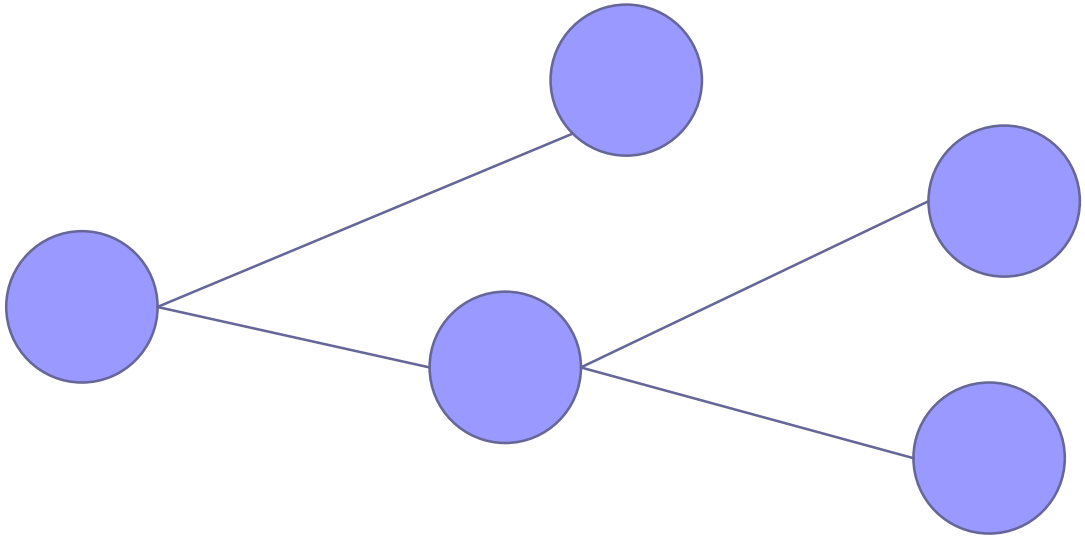


- void setOrientation (const Quaternion & q)
  - 씬노드의 회전값을 설정.
  - 회전없는 상태(회전 원점)을 기준으로 한, 회전값의 설정

- void rotate (const Quaternion & q, TransformSpace relativeTo = TS_LOCAL)
- void rotate(const Vector3 & axis,const Radian & angle,TransformSpace relativeTo = TS_LOCAL)
  - 씬노드를 회전시킴.

- void resetOrientation(void)
  - 씬노드를 회전없는 상태(회전 원점)으로 되돌림.

변환의 상속



- 부모씬노드를 변환하면, 자식씬노드 역시 따라서 변환됨. - Default
- 이동 변환은 항상 변환이 상속됨.
- 스케일과 회전은 상속 여부를 결정할 수 있음.
- void setInheritScale (bool inherit)
  - 부모씬노드의 스케일변환을 상속받을지 여부를 자식씬노드가 결정함.
- void setInheritOrientation(bool inherit)
  - 부모씬노드의 회전변환을 상속받을지 여부를 자식씬노드가 결정함.

변환 공간

enum Ogre::Node::TransformSpace

Enumeration denoting the spaces which a transform can be relative to.

Enumerator:

TS_LOCAL Transform is relative to the local space. ← default

TS_PARENT Transform is relative to the space of the parent node.

TS_WORLD Transform is relative to world space.

실습

Scale
엔터티 크기 변환

main.cpp


```
bool frameStarted(const FrameEvent &evt)
{
  if (mKeyboard->isKeyDown(OIS::KC_1))
    mProfessorNode->setScale(1.0f, 1.0f, 1.0f);
  else if (mKeyboard->isKeyDown(OIS::KC_2))
    mProfessorNode->setScale(2.0f, 1.0f, 1.0f);
  else if (mKeyboard->isKeyDown(OIS::KC_3))
    mProfessorNode->setScale(3.0f, 1.0f, 1.0f);
  else if (mKeyboard->isKeyDown(OIS::KC_E))
    mProfessorNode->scale(1.01f, 1.0f, 1.0f);
  else if (mKeyboard->isKeyDown(OIS::KC_I))
    mNinjaNode->setInheritScale(!mNinjaNode->getInheritScale());

  return true;
}
```

실행 화면


Scale & Rotate : Copyleft by Dae-Hyun Lee 2010
• 키이 입력 1,2,3,E,I 에 따라 다양한 스케일 실험.

```
if (mKeyboard->isKeyDown(OIS::KC_1))
  mProfessorNode->setScale(1.0f, 1.0f, 1.0f);
else if (mKeyboard->isKeyDown(OIS::KC_2))
  mProfessorNode->setScale(2.0f, 1.0f, 1.0f);
else if (mKeyboard->isKeyDown(OIS::KC_3))
  mProfessorNode->setScale(3.0f, 1.0f, 1.0f);

else if (mKeyboard->isKeyDown(OIS::KC_E))
  mProfessorNode->scale(1.1f, 1.0f, 1.0f);

else if (mKeyboard->isKeyDown(OIS::KC_I))
  mNinjaNode->setInheritScale(!mNinjaNode->getInheritScale());
```

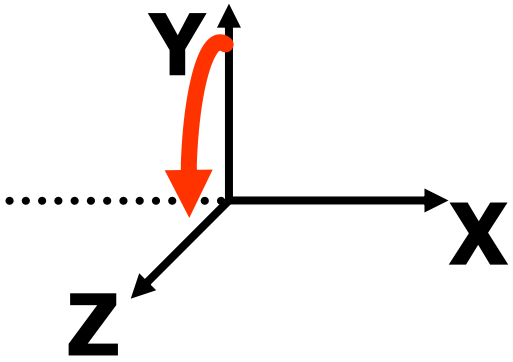
엔터티의 초기 크기를 기준으로 한 크기의 설정

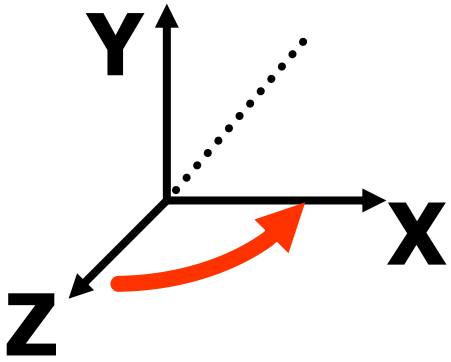
엔터티의 현재 크기를 기준으로 한 크기 조절

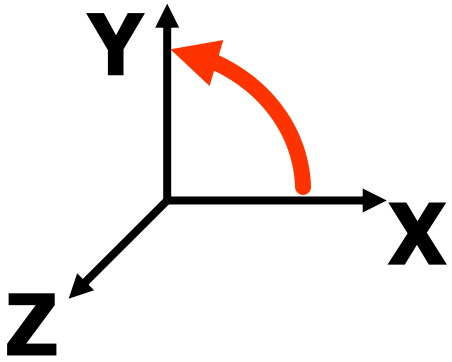
Professor node 와 Ninja Node 의 scale 영향 상속을 ON/OFF

오일러(Euler) 회전

- **오일러 각도(Euler Angles) 원리를 이용한 회전**
  - 오일러 각도: 3차원 공간에서 물체가 취할 수 있는 방향을 나타내는데 사용되는 세개의 각도값의 조합
  - 18세기 수학자 오일러의 착안점: 3차원 직교 좌표계의 좌표축인 x,y,z축에 대한 회전을 적당히 조합하면 임의의 방향을 나타낼 수 있다.
  - Pitch, yaw, roll의 조합을 통해 회전을 함.

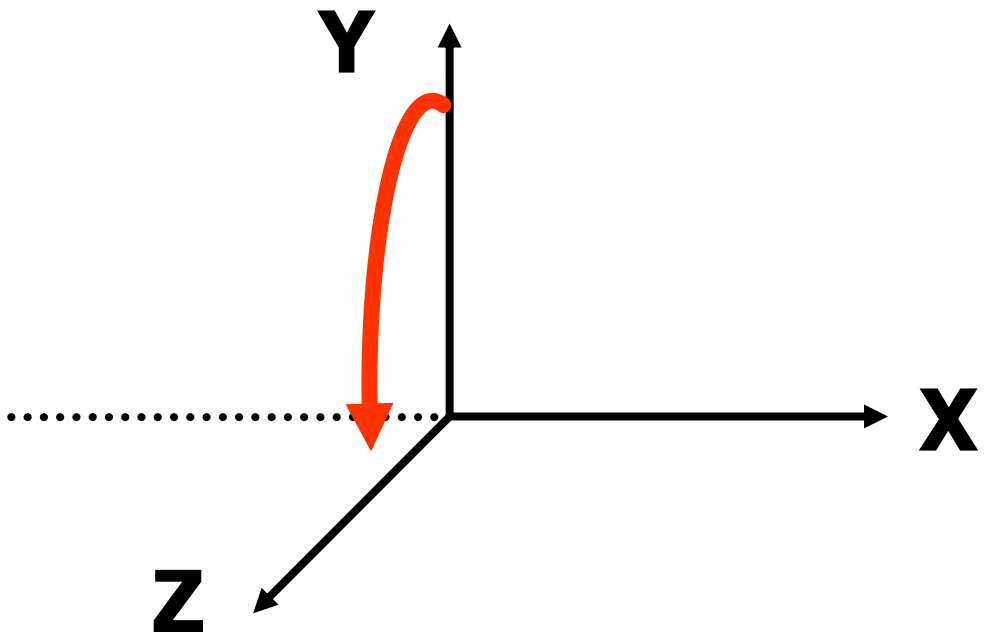






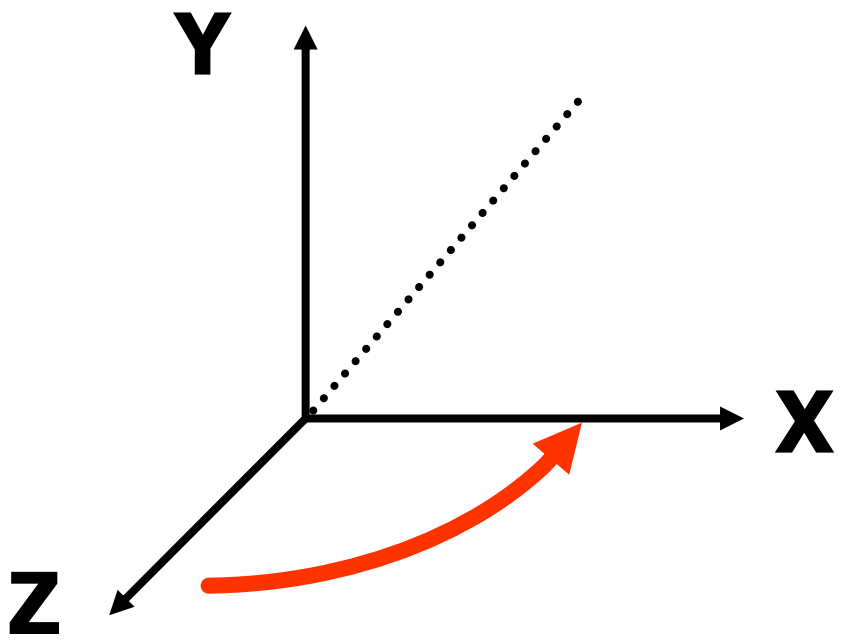
SceneNode의 회전(Rotation)

- void pitch (const Radian &angle, TransformSpace relativeTo=TS_LOCAL)
  - x 축을 회전축으로 하여 주어진 각도만큼 회전.



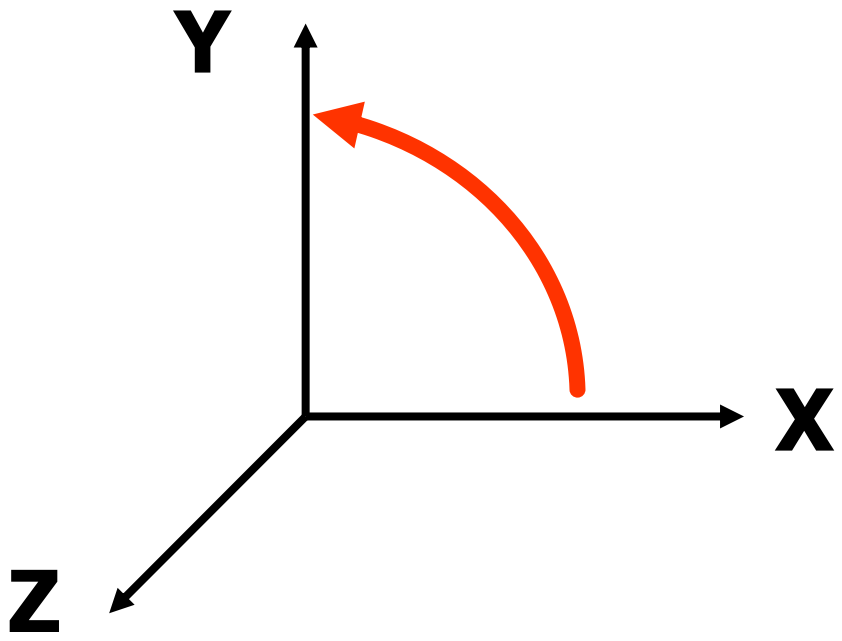
SceneNode의 회전(Rotation)

- void yaw (const Radian &angle, TransformSpace relativeTo=TS_LOCAL)
  - y 축을 회전축으로 하여 주어진 각도만큼 회전.



SceneNode의 회전(Rotation)

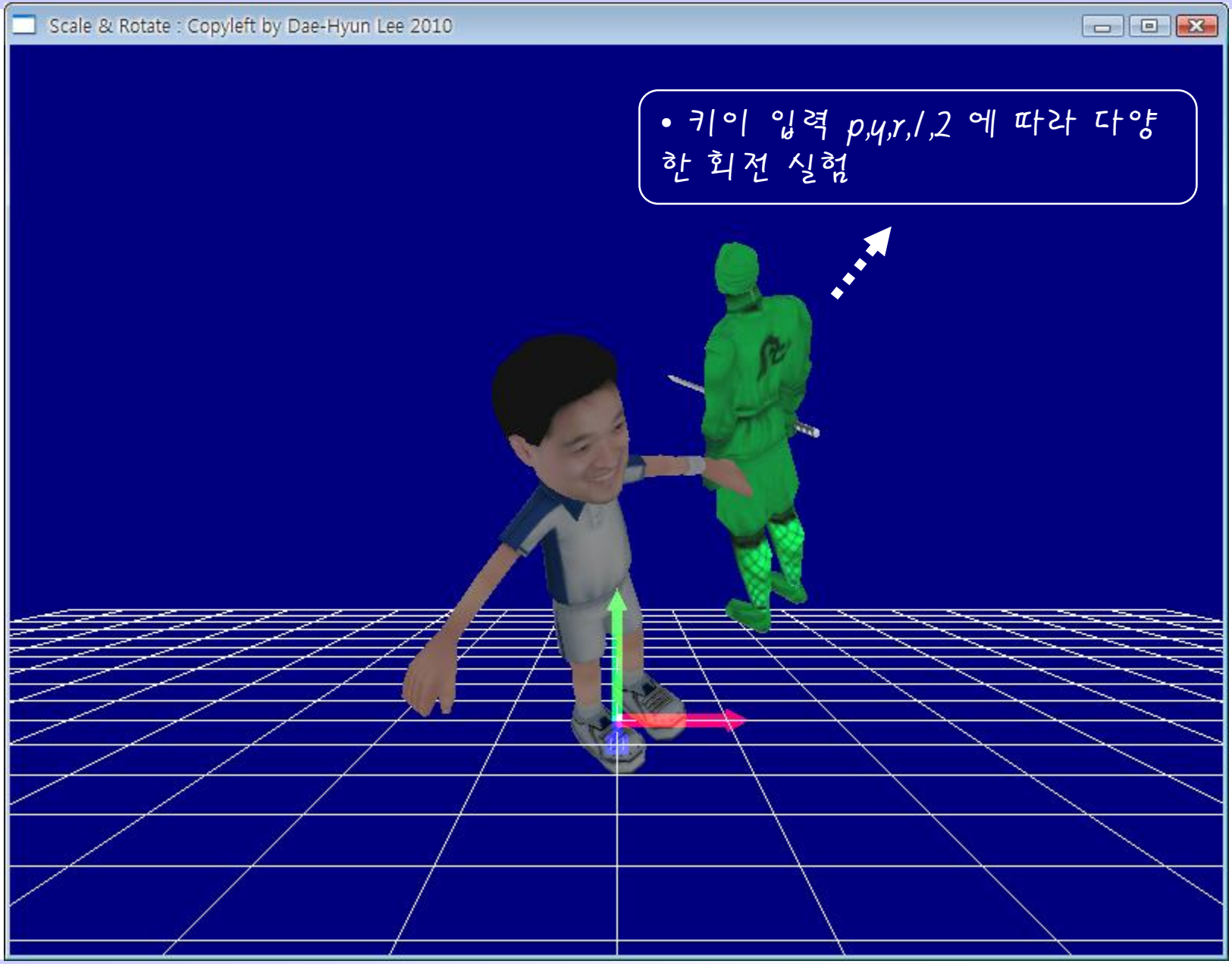
- void roll (const Radian &angle, TransformSpace relativeTo=TS_LOCAL)
  - z 축을 회전축으로 하여 주어진 각도만큼 회전.



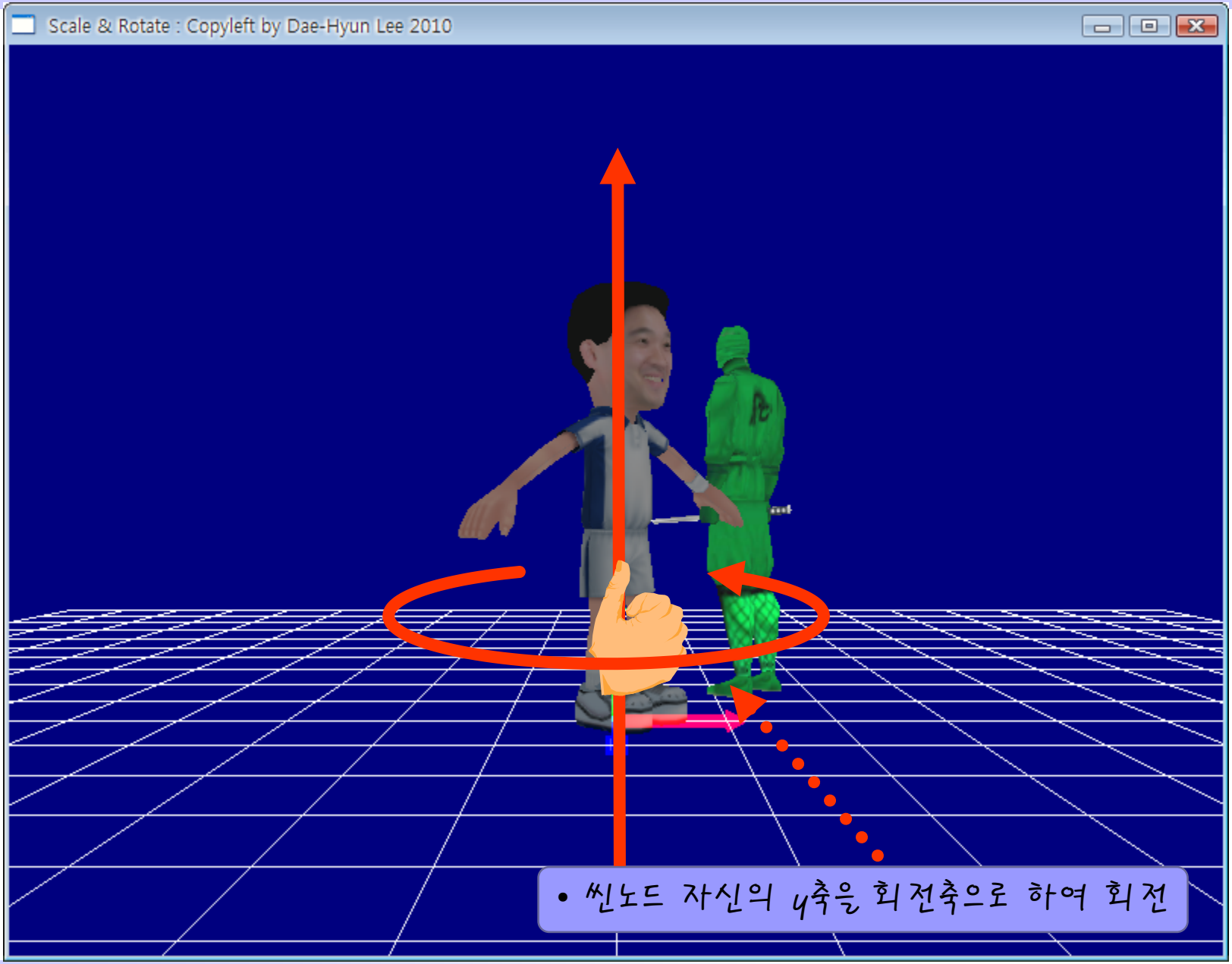
main.cpp


```
bool frameStarted(const FrameEvent &evt)
{
  static SceneNode *curNode = mProfessorNode;

  if (mKeyboard->isKeyDown(OIS::KC_P))
    curNode->pitch(Degree(1.0f));
  else if (mKeyboard->isKeyDown(OIS::KC_Y))
    curNode->yaw(Degree(1.0f));
  else if (mKeyboard->isKeyDown(OIS::KC_R))
    curNode->roll(Degree(1.0f));
  else if (mKeyboard->isKeyDown(OIS::KC_1))
    curNode = mProfessorNode;
  else if (mKeyboard->isKeyDown(OIS::KC_2))
    curNode = mNinjaNode;

  return true;
}
```

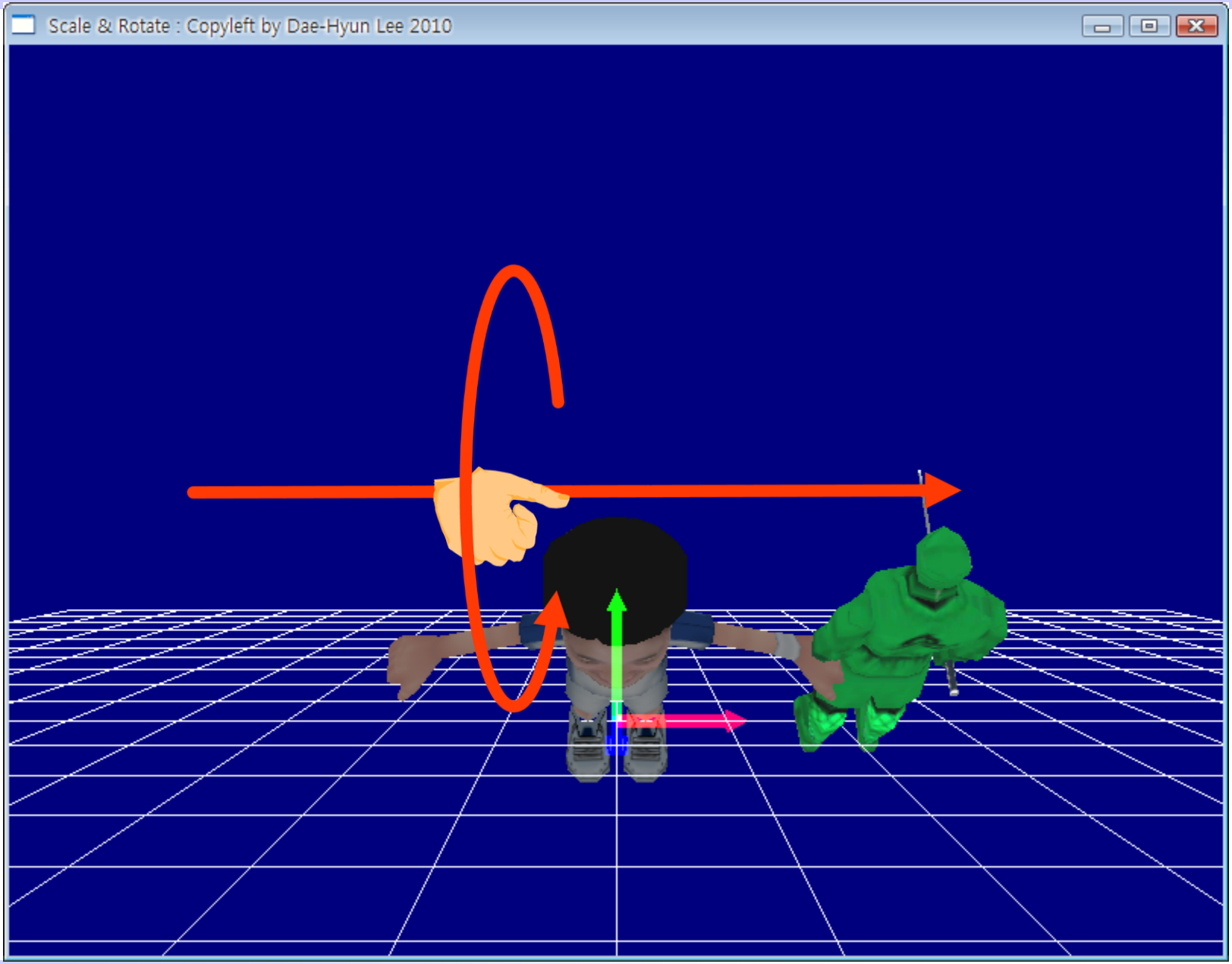
실행 결과


Scale & Rotate : Copyleft by Dae-Hyun Lee 2010
• 키이 입력 p,y,r,1,2 에 따라 다양한 회전 실험

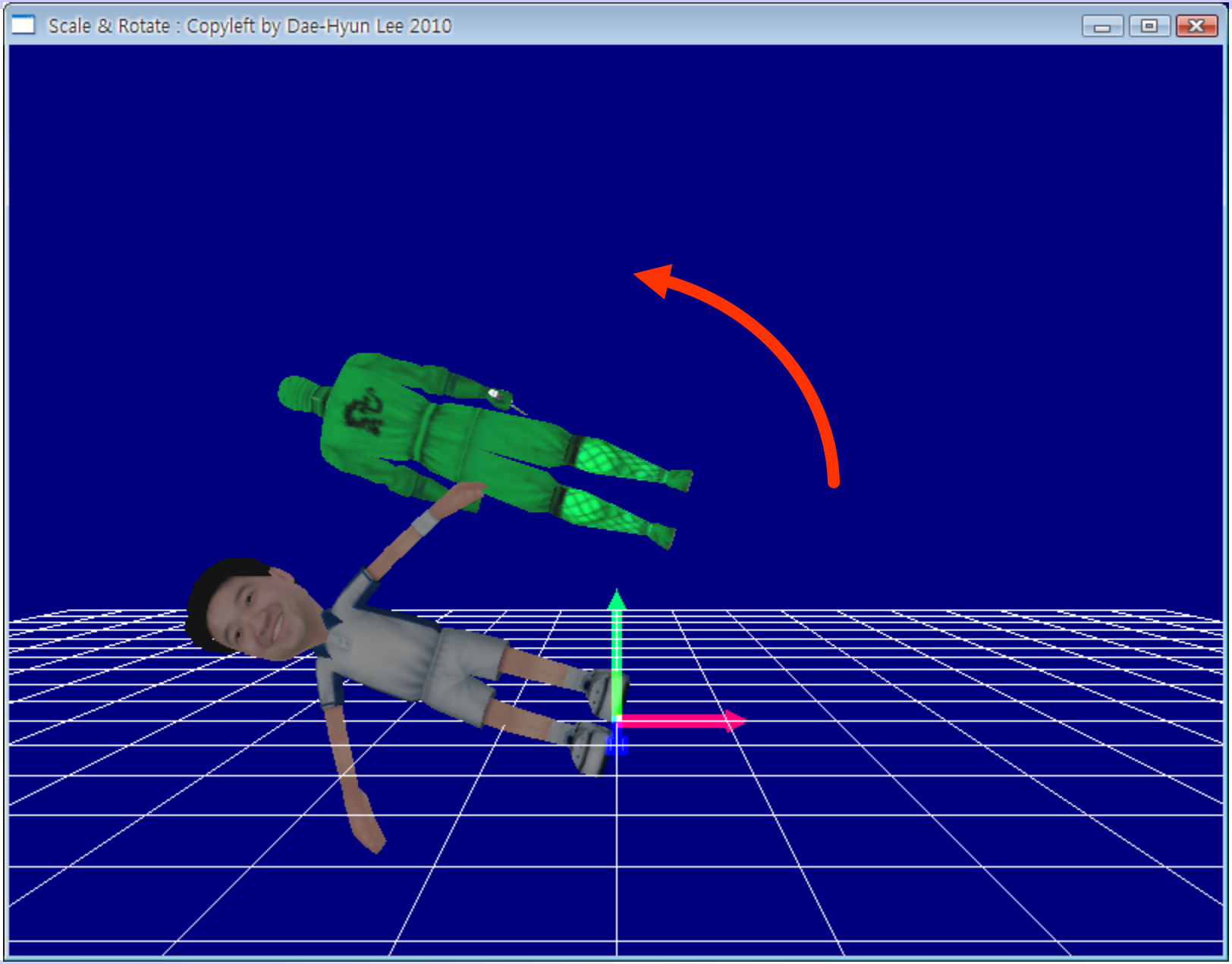
yaw


Scale & Rotate : Copyleft by Dae-Hyun Lee 2010
• 씬노드 자신의 y축을 회전축으로 하여 회전

pitch



roll



```
static SceneNode *curNode = mProfessorNode;

if (mKeyboard->isKeyDown(OIS::KC_P))
  curNode->pitch(Degree(1.0f));
else if (mKeyboard->isKeyDown(OIS::KC_Y))
  curNode->yaw(Degree(1.0f));
else if (mKeyboard->isKeyDown(OIS::KC_R))
  curNode->roll(Degree(1.0f));
else if (mKeyboard->isKeyDown(OIS::KC_1))
  curNode = mProfessorNode;
else if (mKeyboard->isKeyDown(OIS::KC_2))
  curNode = mNinjaNode;
```

* Degree(각도)
  → 오우거 엔진 함수
  각도 → 라디안값 변환

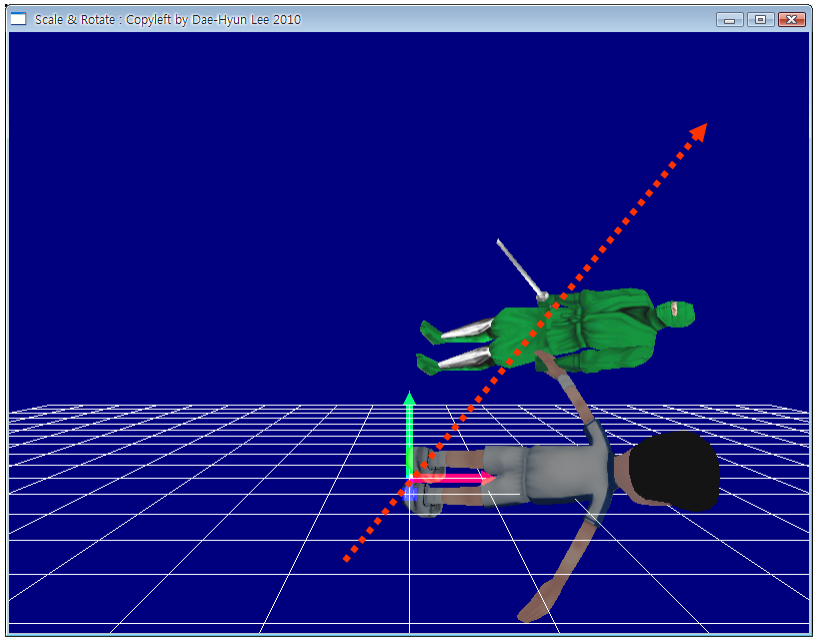
- pitch / yaw / roll 모두 현재 자신의 축을 기준으로 회전함

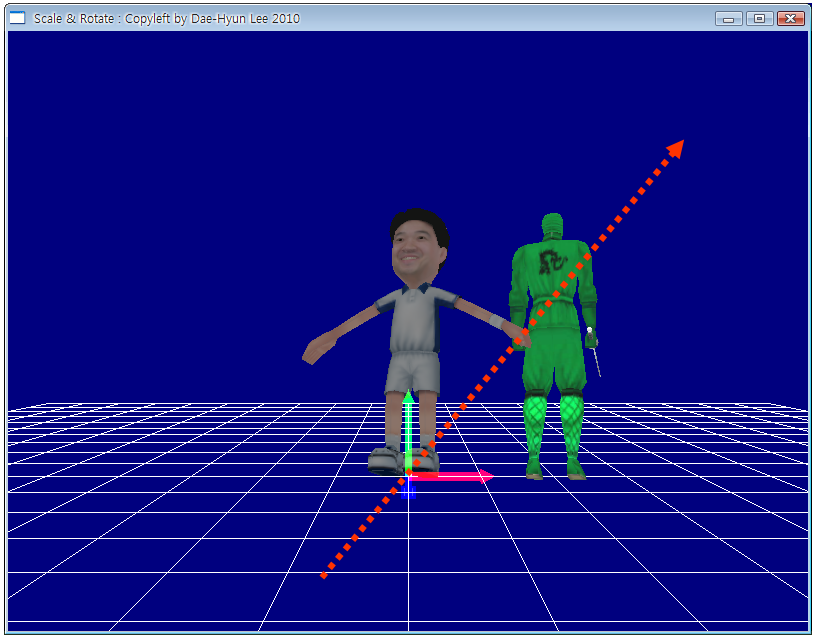
Rotate Ramdom Axis
회전변환(임의축)

main.cpp

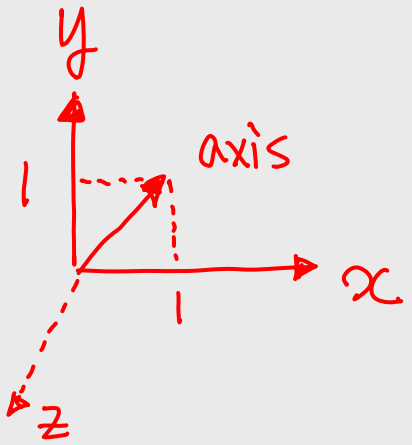

```
bool frameStarted(const FrameEvent &evt)
{

  Vector3 axis(1.0f, 1.0f, 0.0f);
  axis.normalise();
  mProfessorNode->rotate(axis, Degree(1.0f));

  return true;
}
```

실행 결과 : x = y 축을 회전축으로 회전





```
Vector3 axis(1.0f, 1.0f, 0.0f);

axis.normalise();

mProfessorNode->rotate(axis, Degree(1.0f));
```



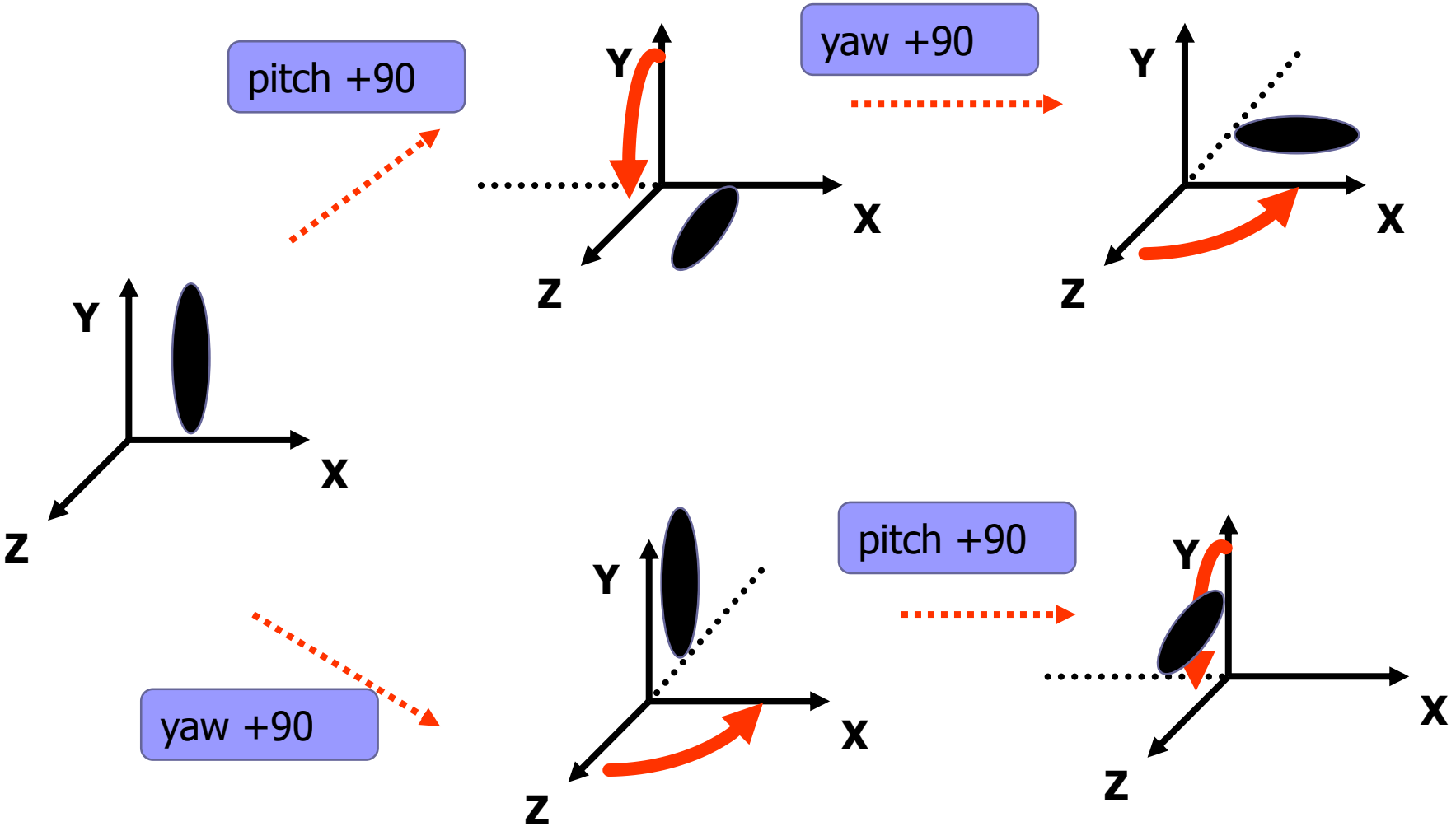
axis.normalise(); → axis 벡터의 크기를 1로 변경. 방향은 유지

✓축 vector

✓반드시 크기가 1 이어야 함

오일러 회전의 문제점

- Pitch, yaw, roll의 순서에 따라 다른 결과가 얻어진다.



■ 오일러 회전의 문제점

□ Gimbal Lock

- 기준 회전축이 사라지는 현상. 여기에 빠지게 되면, 회전값을 결정할 수가 없게 된다.
- Ex) 로보트 팔을 제어하거나 할때 어떤 축의 회전이 하나마나 한 경우가 생김.
- Ex) 북극점과 남극점에서는 경도의 의미가 없슴. 나침반이 의미가 없어진다. 내가 북극점에 있는지 남극점이 있는지 알 수가 없다.

□ 임의의 방향을 설정할때 세축의 회전을 조합해야 하는데 이게 얼른 직관적으로 되질 않음. 따라서, 현재의 특정 방향에서 다른 방향으로 바꾸려고 할때 각 회전 값을 계산하는 것이 효율적이지 않음.

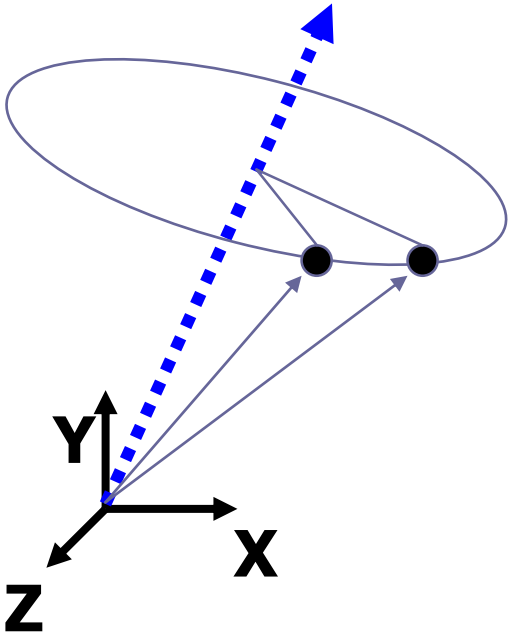
사원수(Quaternion) 회전

■ 사원수

- 한 개의 실수 성분과 세개의 허수 성분으로 이루어진 개념의 수.

$$q = w + xi + yj + zk$$

- 삼차원 공간에서의 회전을 쉽게 나타낼 수 있고, 여러 조합의 회전 연산을 쉽게 할 수 있는 특성을 지님.



사원수의 생성자 함수 (1)

- Quaternion (Real fW=1.0, Real fX=0.0, Real fY=0.0, Real fZ=0.0)
  - 직접 4개의 값을 이용하여 생성.
  - 몇가지 사원수 값의 설명:

w	x	y	z	Description
1	0	0	0	Identity quaternion, no rotation
0	1	0	0	180' turn around X axis
0	0	1	0	180' turn around Y axis
0	0	0	1	180' turn around Z axis
sqrt(0.5)	sqrt(0.5)	0	0	90' rotation around X axis
sqrt(0.5)	0	sqrt(0.5)	0	90' rotation around Y axis
sqrt(0.5)	0	0	sqrt(0.5)	90' rotation around Z axis
sqrt(0.5)	-sqrt(0.5)	0	0	-90' rotation around X axis
sqrt(0.5)	0	-sqrt(0.5)	0	-90' rotation around Y axis
sqrt(0.5)	0	0	-sqrt(0.5)	-90' rotation around Z axis

사원수의 생성자 함수 (2)

- Quaternion (const Quaternion &rkQ)
  - 다른 사원수와 동일한 값을 사용.

- Quaternion (const Matrix3 &rot)
  - 회전 행렬을 이용하여 생생.

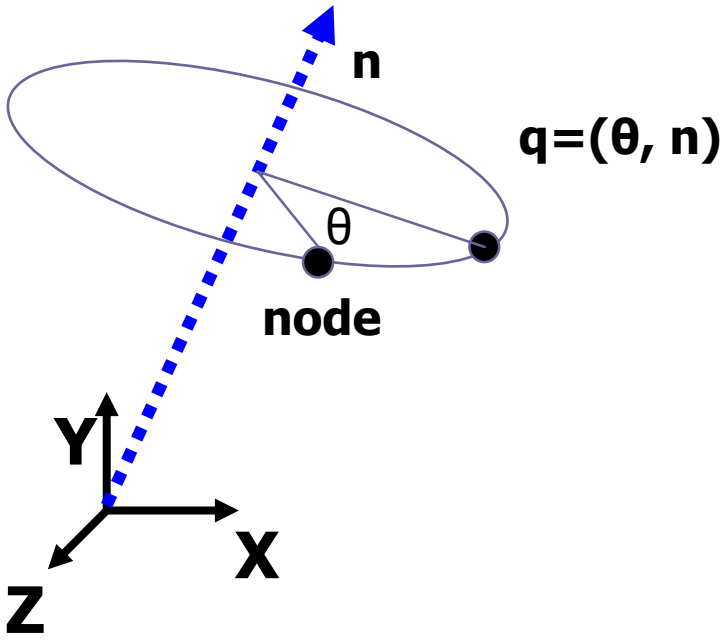
- Quaternion (const Radian &rfAngle, const Vector3 &rkAxis)
  - rfAngle: 회전각. 라디안값.
  - rfAxis: 회전축을 나타내는 벡터. 벡터의 크기값에 따라, 크기 변환도 함께 일어난다.

사원수를 이용한 노드의 회전 함수

■ void Ogre::Node::rotate (const Quaternion & q, TransformSpace relativeTo = TS_LOCAL)

□ 노드를 사원수 q 를 적용하여 회전한다.

```
node->rotate(q);
```



실습

Quaternion
사원수를 이용한 오일러 회전

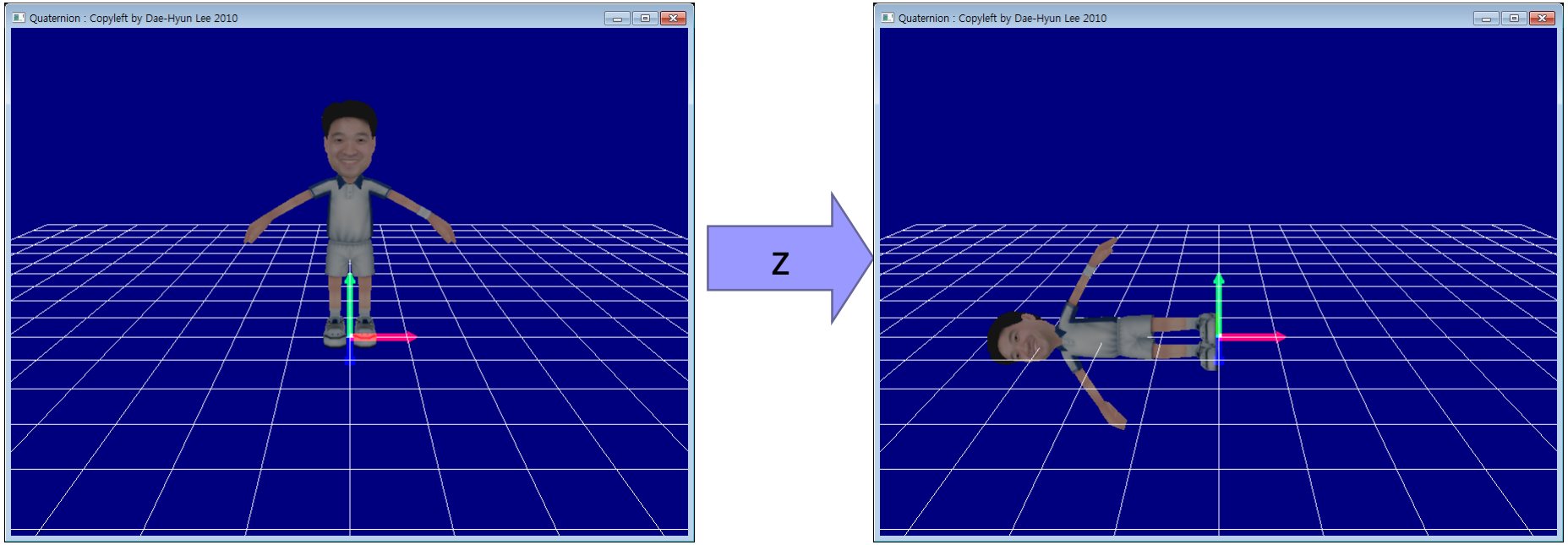
bool InputController::keyPressed(const OIS::KeyEvent &evt)

```
switch(evt.key)
{
case OIS::KC_1:
  {
    mProfessorNode->setOrientation(Ogre::Quaternion::IDENTITY);
    Quaternion z(Degree(90), Vector3::UNIT_Z);
    mProfessorNode->rotate(z);
  }
  break;
case OIS::KC_2:
  {
    mProfessorNode->setOrientation(Ogre::Quaternion::IDENTITY);
    Quaternion z(Degree(90), Vector3::UNIT_Z);
    Quaternion x(Degree(90), Vector3::UNIT_X);
    Quaternion p = x * z;
    mProfessorNode->rotate(p);
  }
  break;
case OIS::KC_3:
  {
    mProfessorNode->setOrientation(Ogre::Quaternion::IDENTITY);
    Quaternion z(Degree(90), Vector3::UNIT_Z);
    Quaternion x(Degree(90), Vector3::UNIT_X);
    Quaternion p = z * x;
    mProfessorNode->rotate(p);
  }
  break;
case OIS::KC_4:
  {
    mProfessorNode->setOrientation(Ogre::Quaternion::IDENTITY);
    Quaternion q(Degree(180), Vector3(1, 1, 0));
    mProfessorNode->rotate(q);
  }
  break;
case OIS::KC_R:
  mProfessorNode->setOrientation(Ogre::Quaternion::IDENTITY);
  break;

case OIS::KC_ESCAPE: mContinue = false; break;
}

return true;
```

실행 결과

```
mProfessorNode->setOrientation(Ogre::Quaternion::IDENTITY);
Quaternion z(Degree(90), Vector3::UNIT_Z);
mProfessorNode->rotate(z);
```

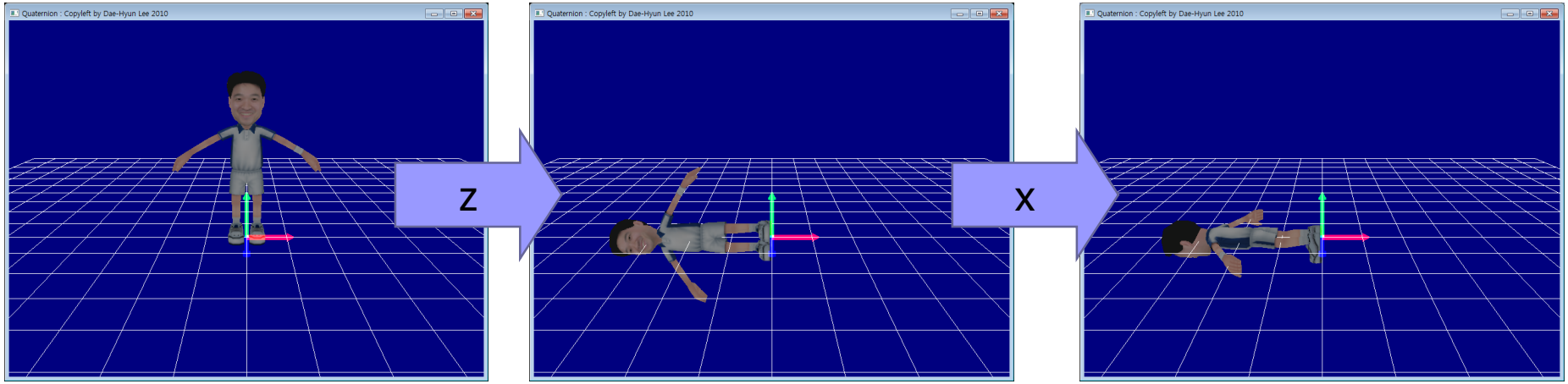
- +z 축을 기준으로 +90도 회전.



```
mProfessorNode->setOrientation(Ogre::Quaternion::IDENTITY);
Quaternion z(Degree(90), Vector3::UNIT_Z);
Quaternion x(Degree(90), Vector3::UNIT_X);
Quaternion p = x * z;
```

- z축 중심 회전을 먼저하고, 그리고 x축 중심 회전을 수행함.

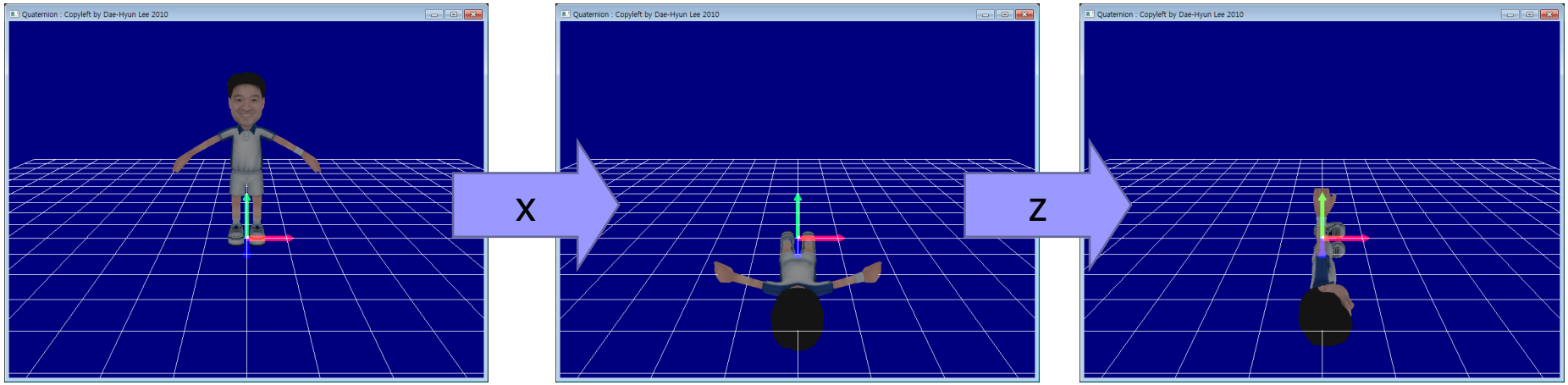
```
mProfessorNode->rotate(p);
```



```
mProfessorNode->setOrientation(Ogre::Quaternion::IDENTITY);
Quaternion z(Degree(90), Vector3::UNIT_Z);
Quaternion x(Degree(90), Vector3::UNIT_X);
Quaternion p = z * x;
```

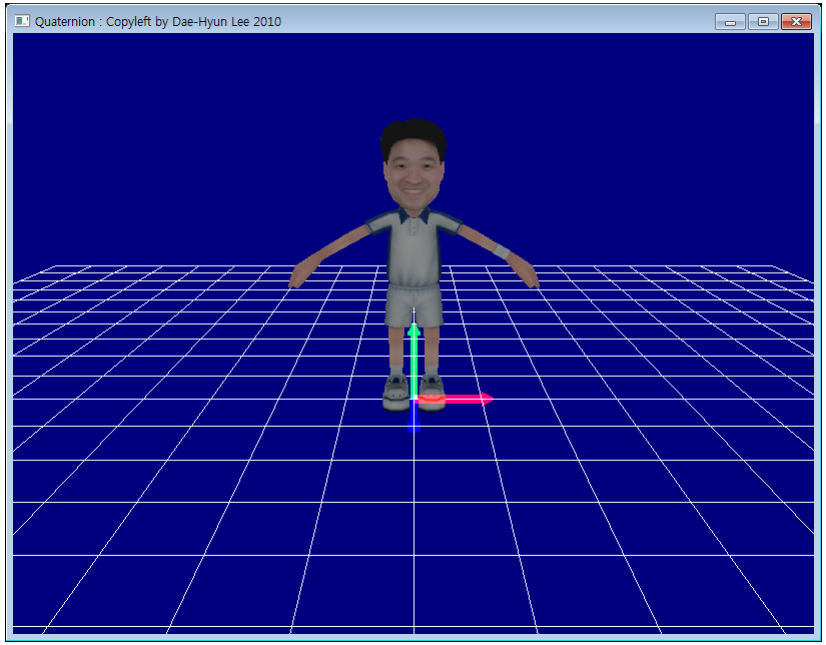
- x축 중심 회전을 먼저하고, 그리고 z축 중심 회전을 수행함.

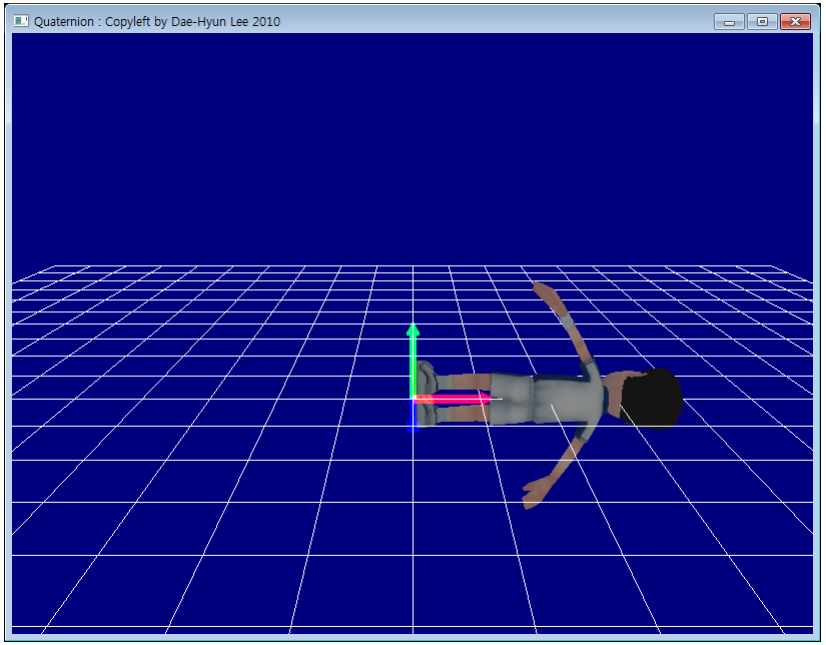
```
mProfessorNode->rotate(p);
```



```
mProfessorNode->setOrientation(Ogre::Quaternion::IDENTITY);
Quaternion q(Degree(180), Vector3(1, 1, 0));
mProfessorNode->rotate(q);
```

- (1, 1, 0) 벡터를 축으로 하여 180도 회전.





노드의 회전 관련 함수

- const Quaternion& Ogre::Node::getOrientation ()
  - 노드에 적용된 최종 회전값(노드의 초기 회전값을 기준으로 한)를 구한다.

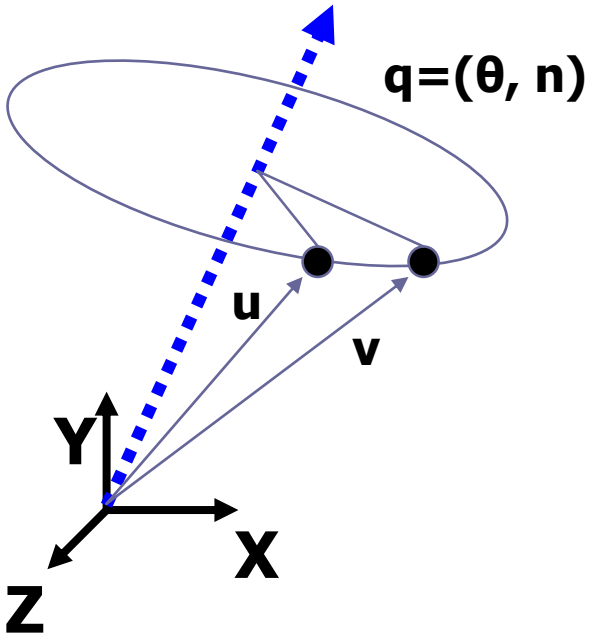
- void Ogre::Node::setOrientation (const Quaternion & q)
  - 노드의 회전값을 정한다.

```
rotate(q) == setOrientation( q * getOrientation() )
```

사원수와 벡터의 곱

- Vector3 Ogre::Quaternion::operator * (const Vector3 & rkVector)
  - 벡터 rKVector를 사원수를 이용하여 회전한다.

```
Vector3 v = q * u;
```

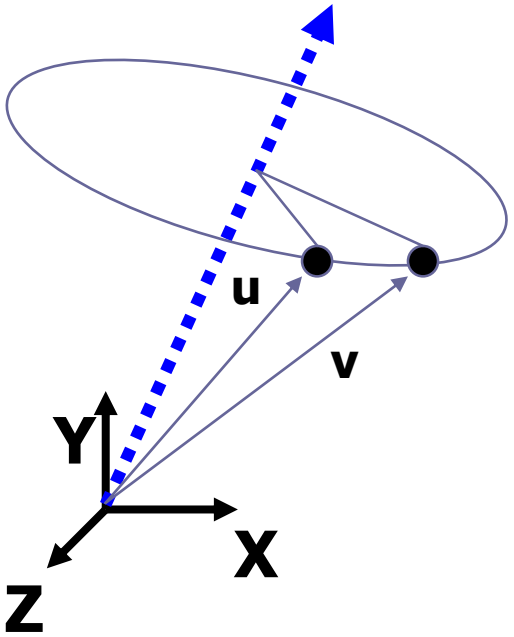


두벡터로부터 회전값 사원수를 얻는 함수

■ Quaternion Ogre::Vector3::getRotationTo (const Vector3 & dest, const Vector3 & fallbackAxis = Vector3::ZERO)

□ 현재 벡터로부터 dest 벡터를 얻기 위해 필요한 회전을 나타내는 사원수 q를 구한다.

```
Quaternion q = u->getRotationTo(v);
```



• 단위벡터가 되어야 함.

학습 정리

3D 오브젝트의 확대 및 축소

- setScale() - 엔터티의 절대 크기(원래 모델의 크기를 기준) 설정
- scale() - 현재 엔터티의 크기를 기준으로 한 상대적 크기 설정

3D 오브젝트의 회전

- 기본축을 중심으로 한 회전 - yaw(), roll(), pitch()
- 임의축을 중심으로 한 회전 - rotate

오일러 회전

- x,y,z 축을 회전축으로 한 회전.
- 이해하기 쉬우나, gimbal lock 등의 문제가 있음.

사원수

- 한 개의 실수 성분과 세 개의 허수 성분으로 이루어진 개념의 수.
- 임의의 회전축을 중심으로 한 회전을 하나의 사원수 값으로 표현할 수 있음.