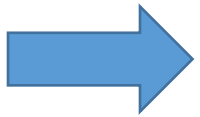
카메라Camera와 투영Projection

동명대학교 게임공학과
강영민

간단한 OpenGL 코드를 만들어 보자

```
from OpenGL.GL import *
from OpenGL.GLU import *

from PyQt6.QtWidgets import QApplication, QWidget
from PyQt6.QtOpenGLWidgets import QOpenGLWidget
import sys

def drawAxes():
    glBegin(GL_LINES)
    # x축 (0,0,0) - (1,0,0)
    glColor(1, 0, 0) # 빨간색
    glVertex3f(0,0,0)
    glVertex3f(1,0,0)
    # y축
    glColor(0, 1, 0) # 녹색
    glVertex3f(0,0,0)
    glVertex3f(0,1,0)
    # z축
    glColor(0, 0, 1) # 파란색
    glVertex3f(0,0,0)
    glVertex3f(0,0,1)
    glEnd()
```



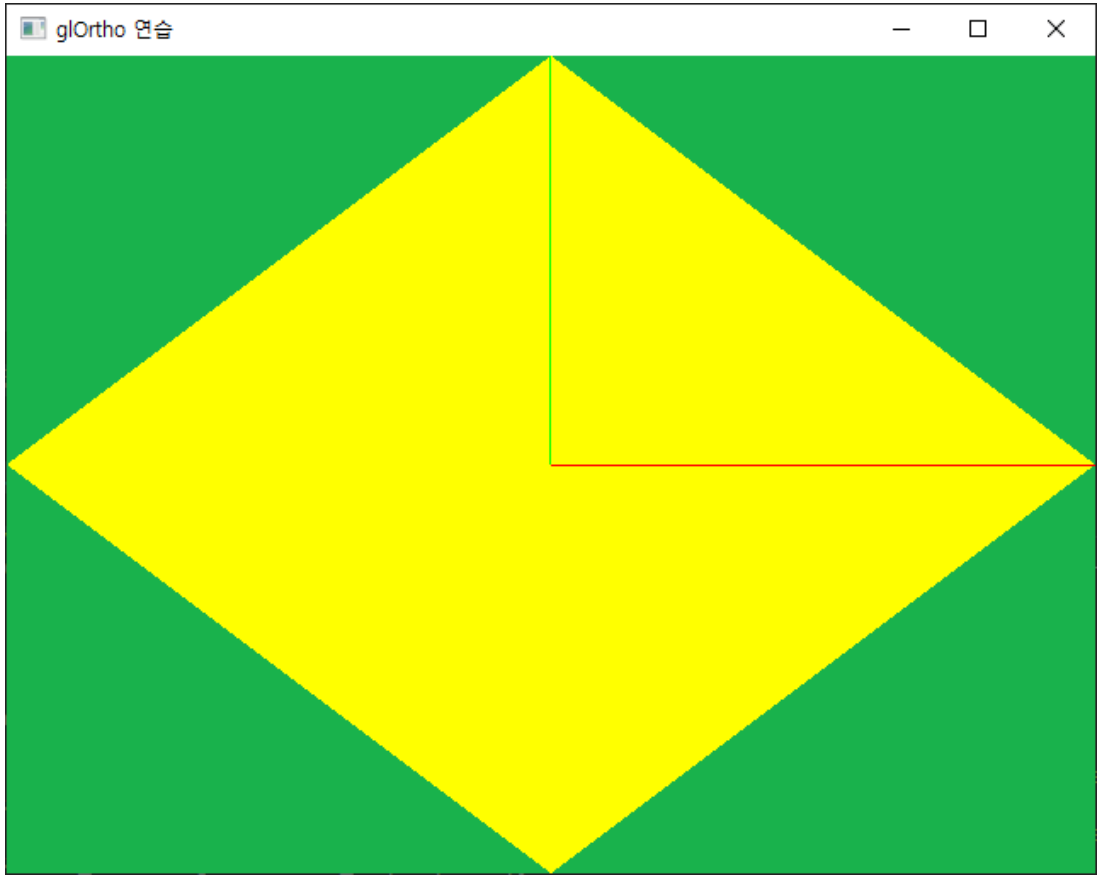
OpenGL 프리미티브를 이용하여
x, y, z 축을 그리는 함수

OpenGL 윈도우 클래스의 구현

```
class MyGLWindow(QOpenGLWidget) : # QOpenGLWidget 상속

    def __init__(self):
        super().__init__()  # 슈퍼클래스 QMainWindow 생성자 실행
        self.setWindowTitle('glOrtho 연습')

    def initializeGL(self) :
        glClearColor(0.1, 0.7, 0.3, 1.0)

    def resizeGL(self, w: int, h: int) :
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()

    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

        glBegin(GL_POLYGON)
        glColor3f(1, 1, 0)
        glVertex3f(1, 0, 0)
        glVertex3f(0, 1, 0)
        glVertex3f(-1, 0, 0)
        glVertex3f( 0, -1, 0)
        glEnd()

        drawAxes()
```

윈도우 생성 및 OpenGL 위짓 포함

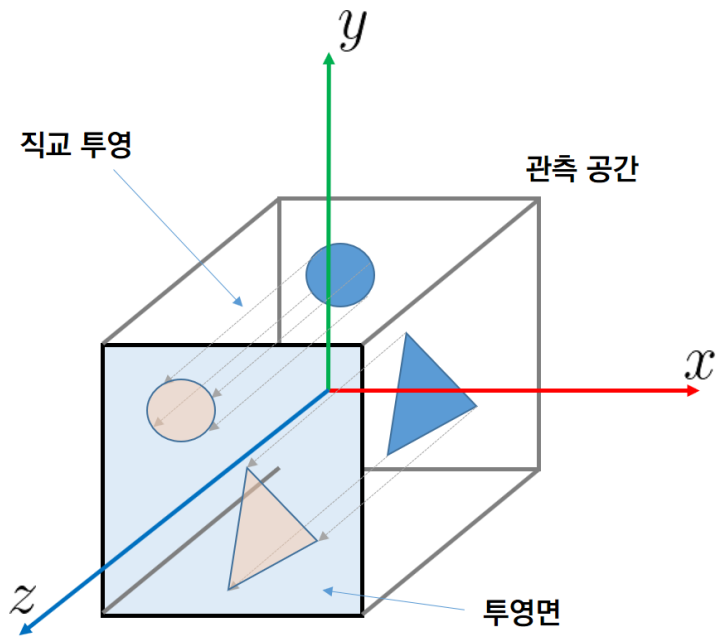
```
def main(argv = sys.argv) :
    ## 윈도우 생성하기
    app = QApplication(argv)
    window = MyGLWindow()
    window.show()

    app.exec()


if __name__ == '__main__' :
    main(sys.argv)
```



왜 이런 장면을 보게 된 것일까?

카메라 모델

- 직교 투영 공간



$$f : \mathbf{V} \rightarrow \mathbf{V}$$

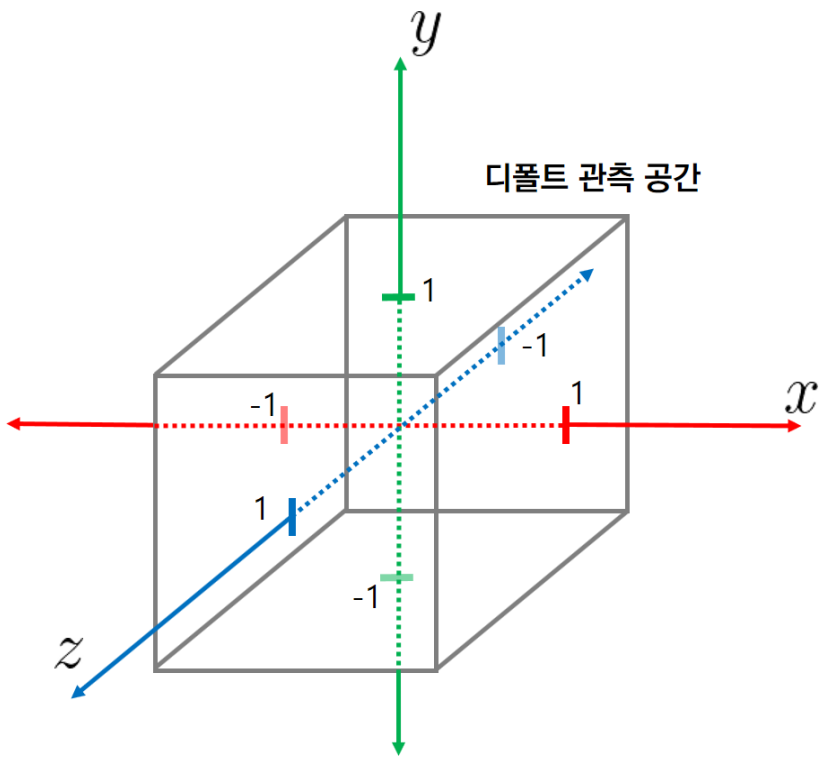
$$f(x, y, z) = (x, y, z_{proj})$$

이게 다 뭔 소리냐…

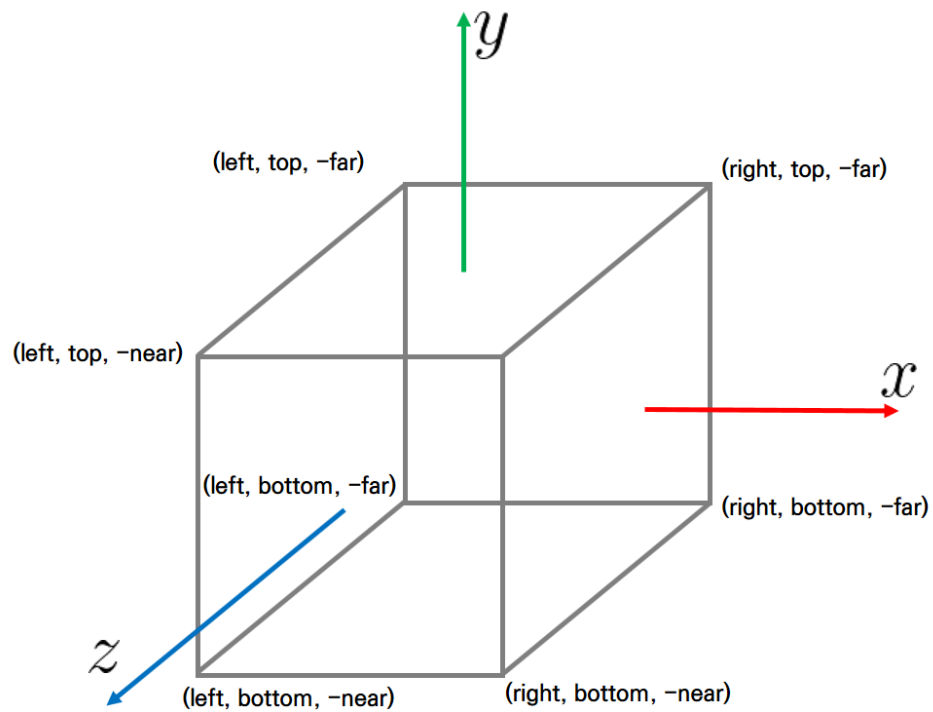
x, y의 좌표는 그대로 유지
z만 그리는 면에 떨어트림

카메라 모델

- 디폴트 직교 투영 공간과 변경



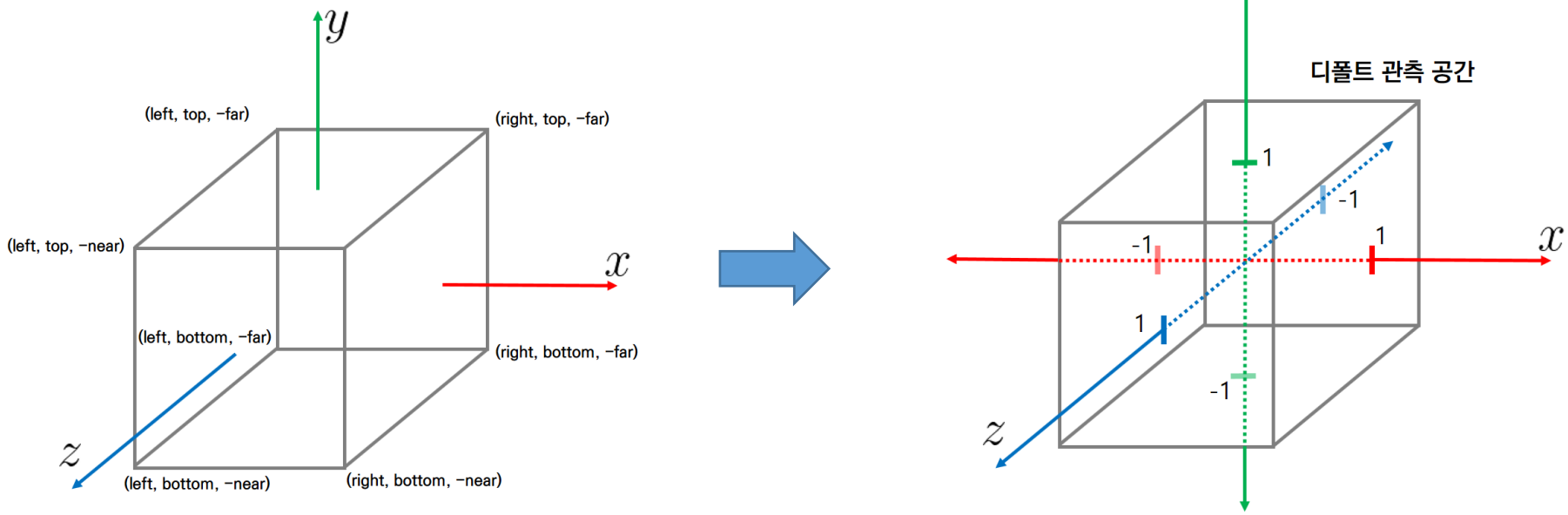

```
glOrtho(left, right, bottom, top, near, far)
```



화면에 그리기 위해 좌표는 어떻게 바뀌나

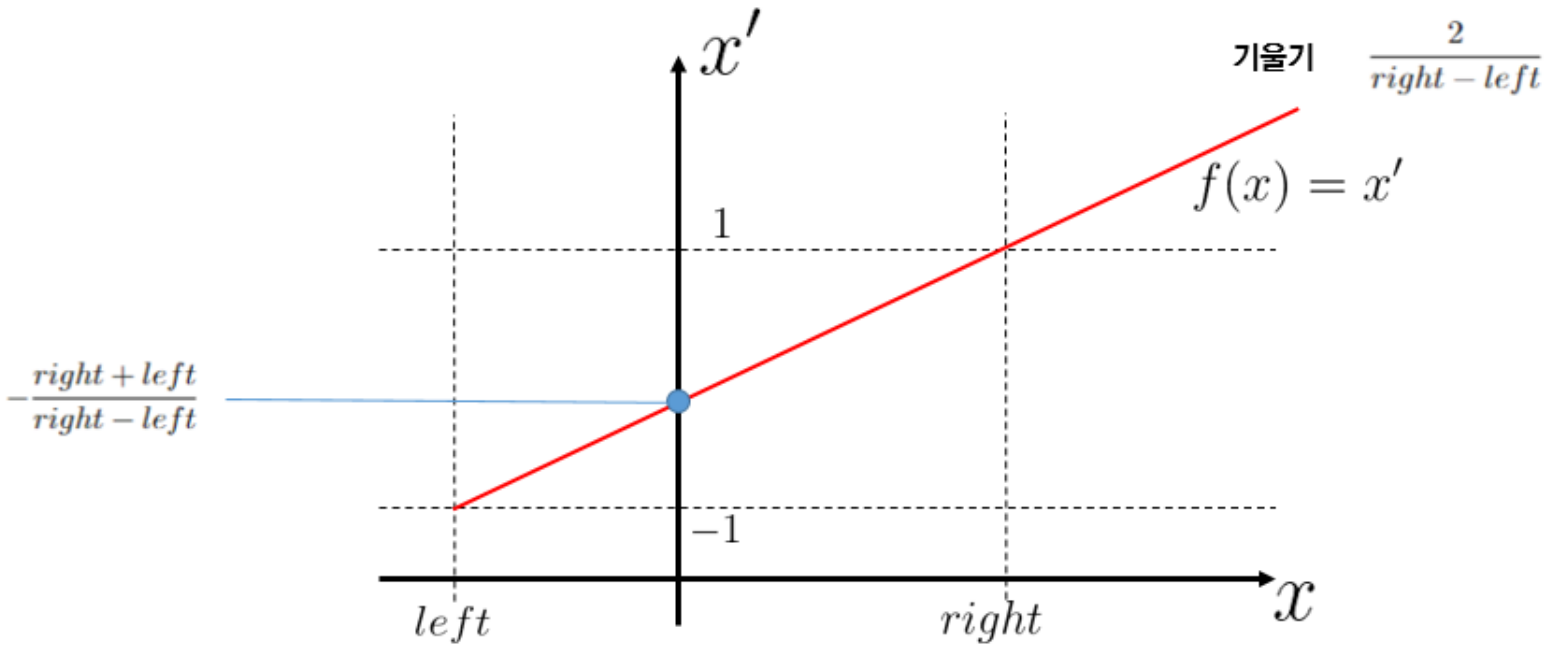
- 정규 관측 공간으로 옮기기
  - 정규 관측 공간
    - x, y, z 축 방향으로 [-1, 1] 사이의 공간
  - 좌표를 정규 관측 공간으로 옮긴다는 것 ➔

$$x : [\text{left, right}] \Rightarrow x' : [-1, 1]$$
$$y : [\text{bottom, top}] \Rightarrow y' : [-1, 1]$$
$$z : [\text{near, far}] \Rightarrow z' : [-1, 1]$$



X 좌표를 바꾸어 보자

- left에 해당하는 값은 -1, right에 해당하는 값은 1
- right-left의 간격은 길이 2의 간격으로 변환



X 좌표를 바꾸어 보자

- left에 해당하는 값은 -1, right에 해당하는 값은 1
- right-left의 간격은 길이 2의 간격으로 변환

$$x' = f(x) = \left(\frac{2}{right - left}\right) x - \frac{right + left}{right - left}$$

x,y,z 좌표를 바꾸어 보자

$$x' = f(x) = \left(\frac{2}{right - left}\right) x - \frac{right + left}{right - left}$$

$$y' = \left(\frac{2}{top - bottom}\right) y - \frac{top + bottom}{top - bottom}$$

$$z' = -\left(\frac{2}{far - near}\right) z - \frac{far + near}{far - near}$$

부호 바뀜 주의

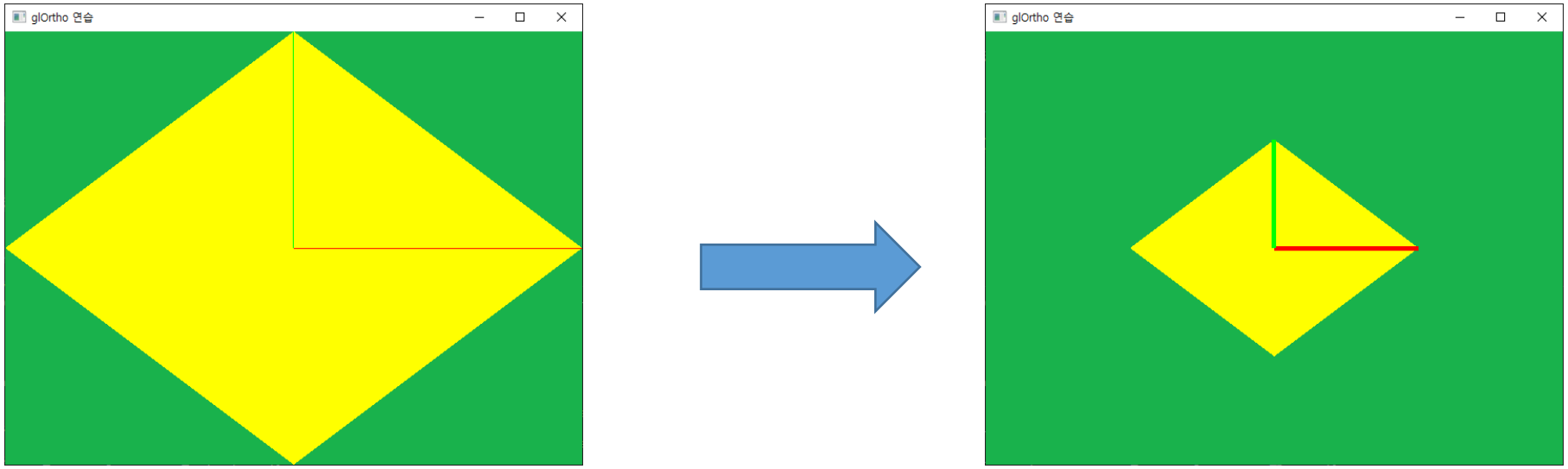
직교 투영 = 선형 변환 행렬을 적용하는 것

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

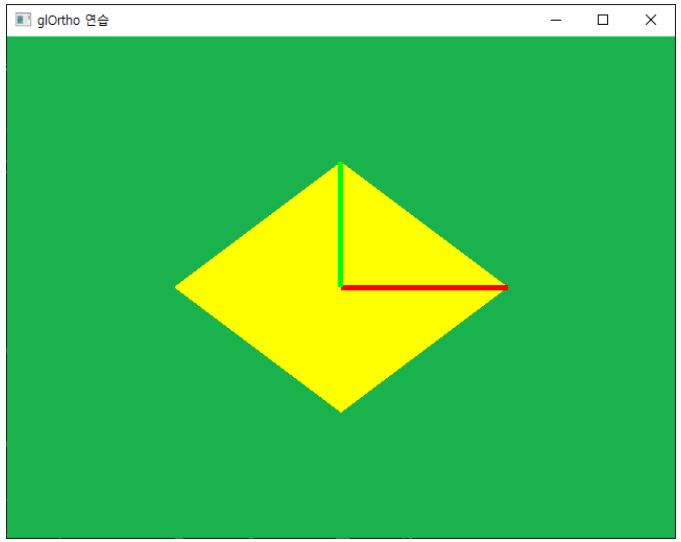
투영행렬: 모든 점들은 이 행렬이 곱해져서 최종 좌표가 결정된다

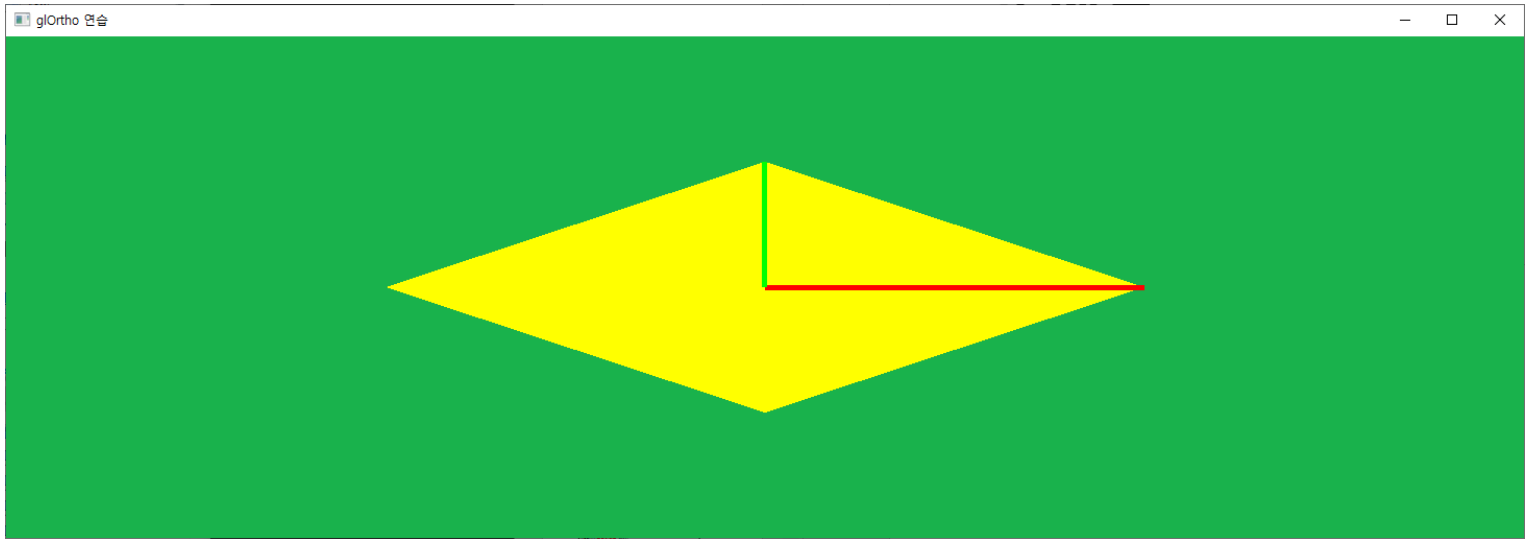
glOrtho 사용하기

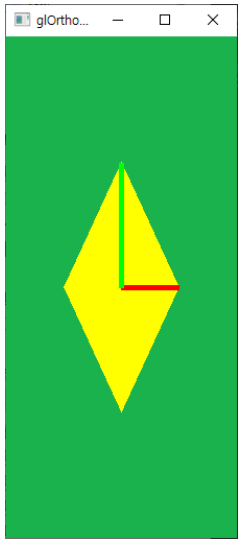
```
    def resizeGL(self, w: int, h: int) :
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        glOrtho(-2, 2, -2, 2, -1, 1)
```



문제점



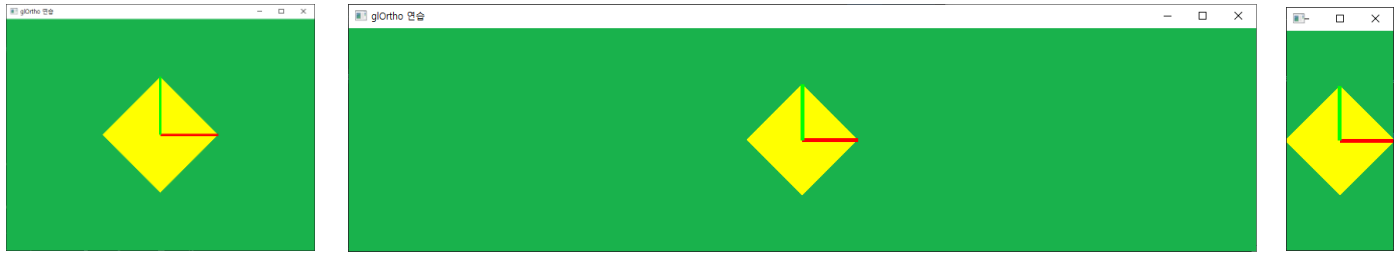




왜곡이 없는 투영

- 화면의 크기에 맞춰 투영행렬 설정

```
    def resizeGL(self, w: int, h: int) :

        aspRatio = w / h # 종횡비를 계산한다.
        range = 2
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        glOrtho(-range*aspRatio, range*aspRatio, -range, range, -range, range)
```



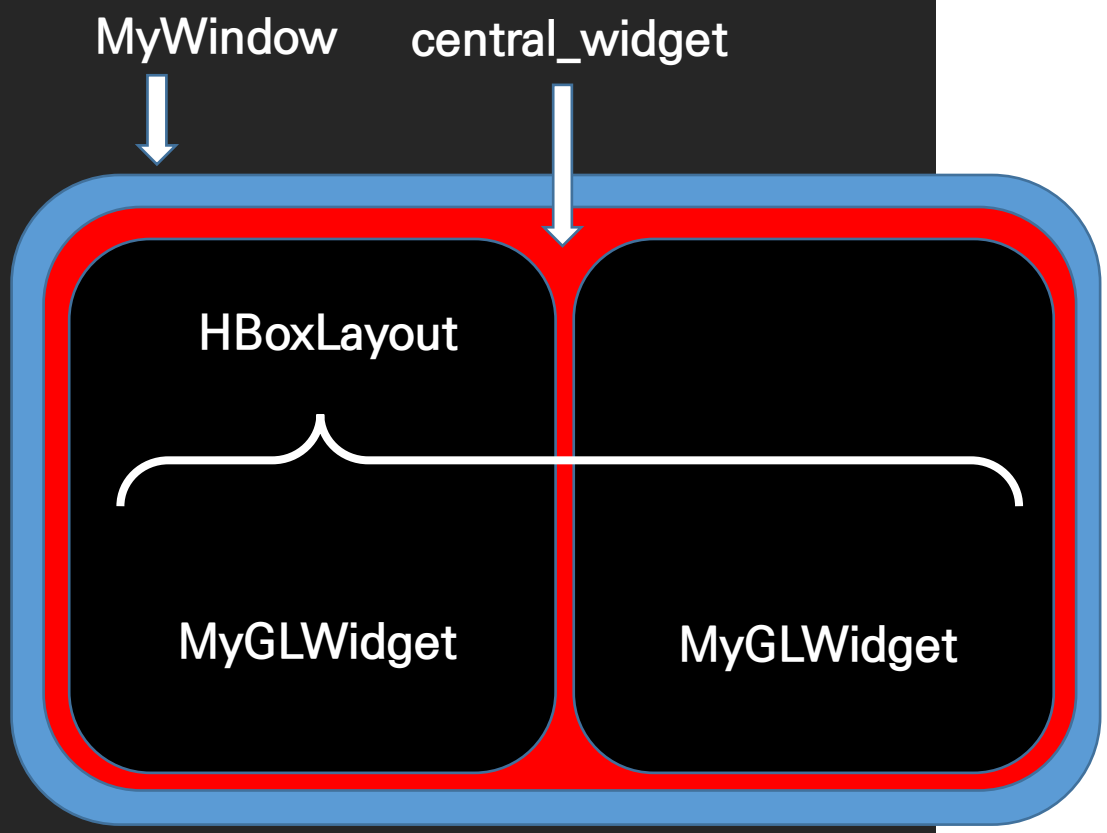
투영의 관찰을 위한 두 개의 OpenGL 위짓

- OpenGL 위짓 클래스 정의

```
from OpenGL.GL import *
from OpenGL.GLU import *
import sys

from PyQt6.QtWidgets import *
from PyQt6.QtOpenGLWidgets import QOpenGLWidget

class MyGLWidget(QOpenGLWidget):
    def __init__(self, parent=None, observation = False):
        super().__init__(parent)
        self.observation = observation

    def initializeGL(self):
        pass

    def resizeGL(self, w, h):
        pass

    def paintGL(self):
        pass
```

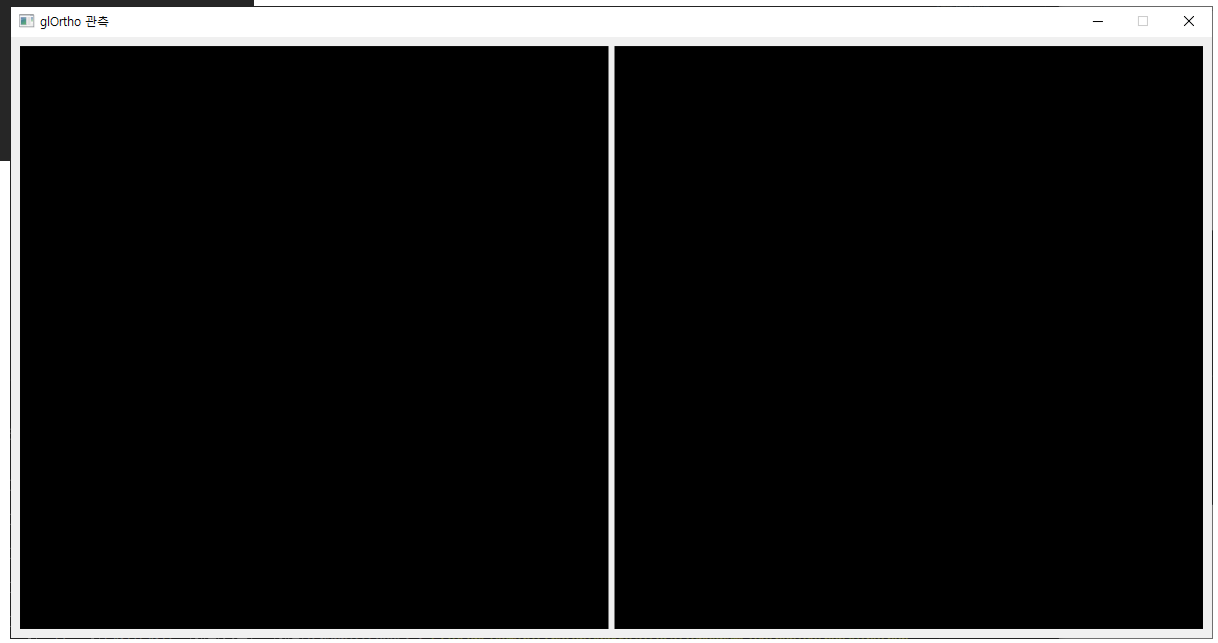
메인 윈도우 생성 – 두 개의 OpenGL 위짓 배치

```
class MyWindow(QMainWindow):
    def __init__(self, title=''):
        super().__init__()
        self.setWindowTitle(title)

        ## GUI 설정
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        gui_layout = QHBoxLayout()

        central_widget.setLayout(gui_layout)

        self.glWidget1 = MyGLWidget()
        self.glWidget2 = MyGLWidget()

        gui_layout.addWidget(self.glWidget1)
        gui_layout.addWidget(self.glWidget2)
```



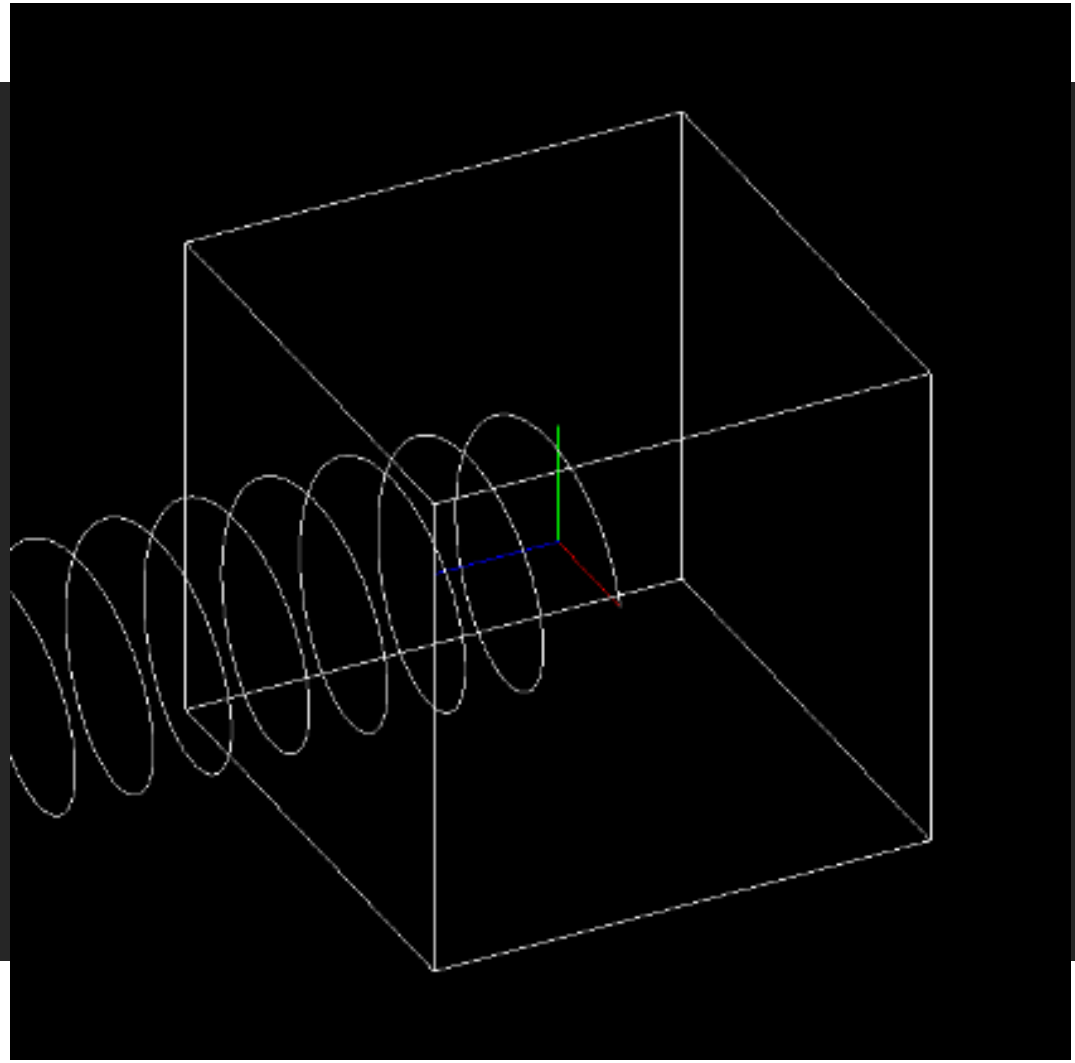
윈도 띄우기

```
def main(argv = sys.argv):
    app = QApplication(argv)
    window = MyWindow('glOrtho 관측')
    window.setFixedSize(1200, 600)
    window.show()
    app.exec()

if __name__ == '__main__':
    main(sys.argv)
```



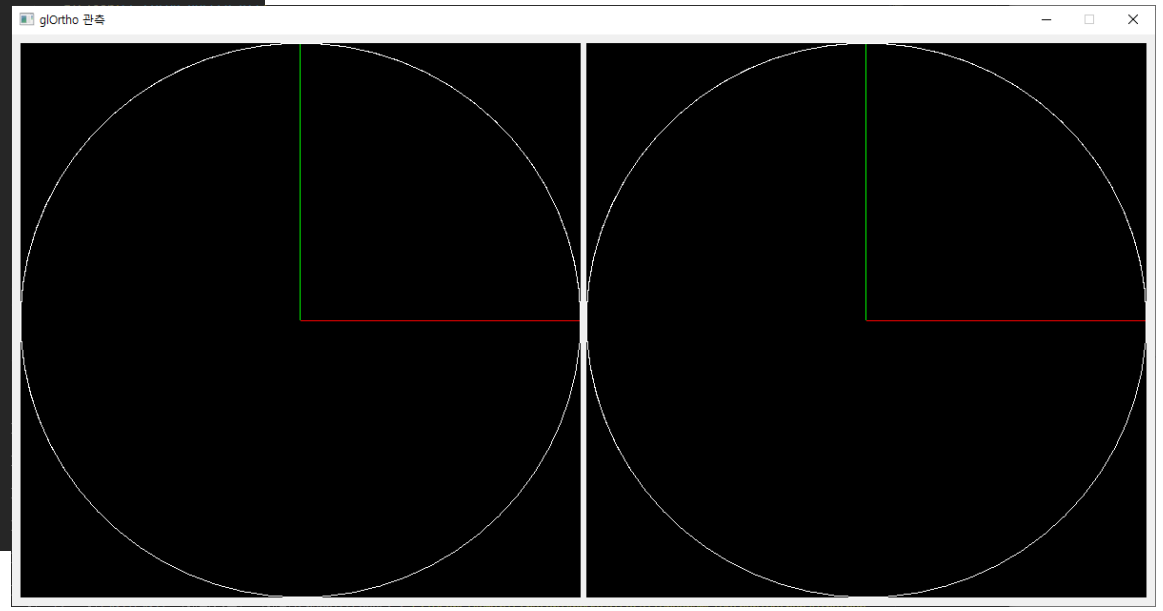
두 창에 나선 그리기

```
def drawAxes():
    glBegin(GL_LINES)
    glColor3f(1,0,0) # red x axis
    glVertex3f(0,0,0); glVertex3f(1,0,0)
    glColor3f(0,1,0) # green y axis
    glVertex3f(0,0,0); glVertex3f(0,1,0)
    glColor3f(0,0,1) # blue z axis
    glVertex3f(0,0,0); glVertex3f(0,0,1)
    glEnd()


def drawHelix():
    glColor3f(1,1,1)
    glBegin(GL_LINE_STRIP)
    for i in range(1000):
        angle = i/10
        x, y = math.cos(angle), math.sin(angle)
        glVertex3f(x, y, angle/10)
    glEnd()
```



OpenGL 위짓에 나선 그리기 추가

```
class MyGLWidget(QOpenGLWidget):
    def __init__(self, parent=None, observation = False):
        super().__init__(parent)
        self.observation = observation

    def initializeGL(self):
        pass

    def resizeGL(self, w, h):
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()

    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT)
        glMatrixMode(GL_MODELVIEW)
        glLoadIdentity()
        drawAxes()
        drawHelix()
```



glOrtho 영역 확인하기

```
def drawBox(l, r, b, t, n, f): # glOrtho가 만드는 공간(육면체)을 가시화
    glColor3f(1, 1, 1)
    glBegin(GL_LINE_LOOP)
    # 앞면
    glVertex3f(l,t,n)
    glVertex3f(l,b,n)
    glVertex3f(r,b,n)
    glVertex3f(r,t,n)
    glEnd()

    glBegin(GL_LINE_LOOP)
    # 뒷면
    glVertex3f(l,t,f)
    glVertex3f(l,b,f)
    glVertex3f(r,b,f)
    glVertex3f(r,t,f)
    glEnd()
```

OpenGL 위짓에 observation 속성 추가

```
class MyGLWidget(QOpenGLWidget):

    left = bottom = near = -2
    right = top = far = 2

    def __init__(self, parent=None, observation = False):
        super().__init__(parent)
        self.observation = observation

    def initializeGL(self):
        pass

    def resizeGL(self, w, h):
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        if self.observation:
            glOrtho(-4, 4, -4, 4, -100, 100)
        else:
            glOrtho(self.left, self.right, self.bottom, self.top, self.near, self.far)
```

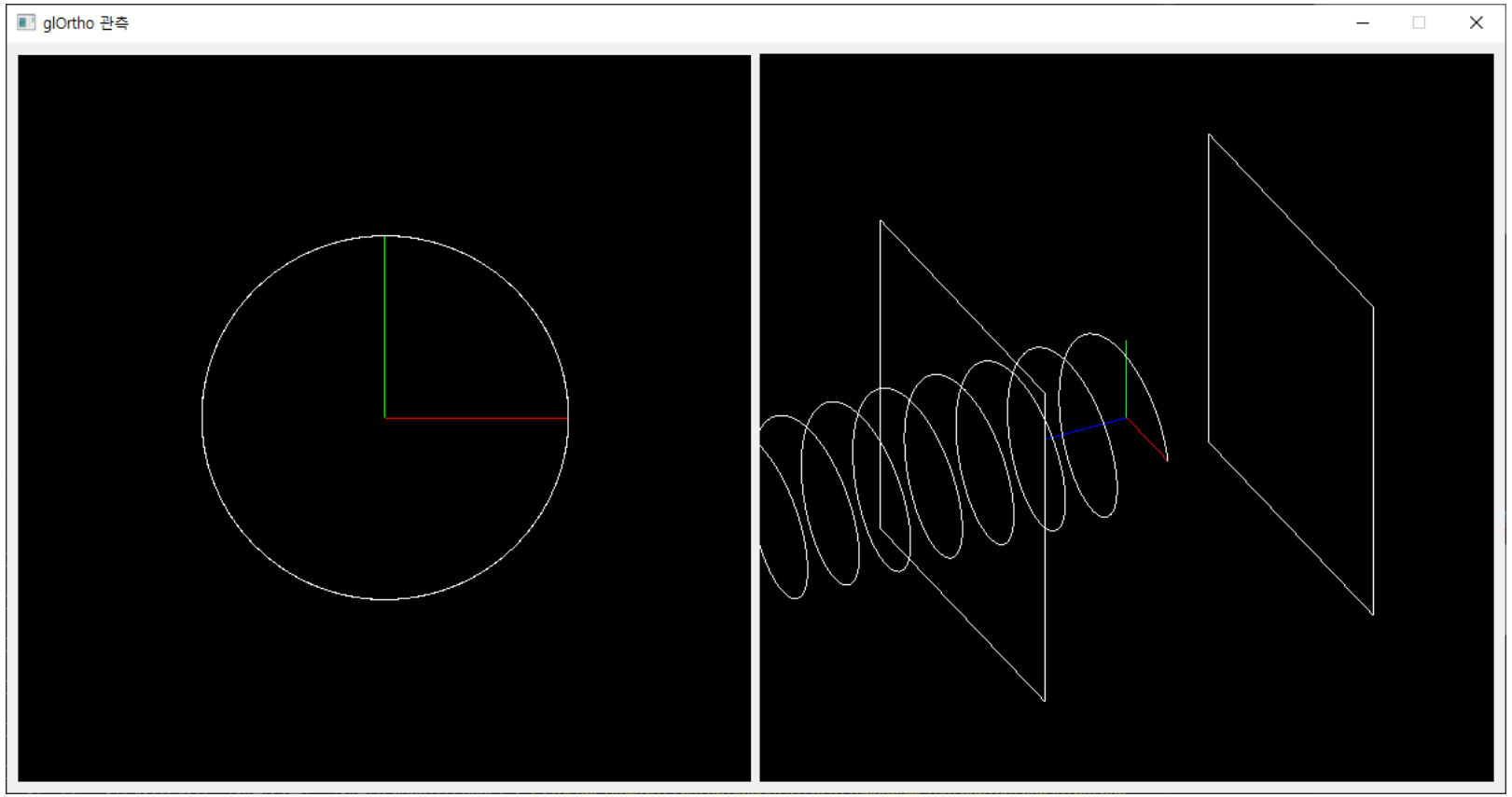
관찰용 위짓은 눈의 위치를 변경

```
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT)
        glMatrixMode(GL_MODELVIEW)
        glLoadIdentity()
        if self.observation:
            gluLookAt(1, 0.7, 0.5, 0, 0, 0, 0, 1, 0)
        drawAxes()
        drawHelix()
        drawBox(self.left, self.right, self.bottom, self.top, self.near, self.far)
```

MyWindow에서 오픈지엘 위짓 생성시에 관찰용인지 여부를 지정

```
class MyWindow(QMainWindow):
    def __init__(self, title=''):
        . . .

        self.glWidget1 = MyGLWidget()
        self.glWidget2 = MyGLWidget(observation = True) # 관측용 OpenGL 위짓
```

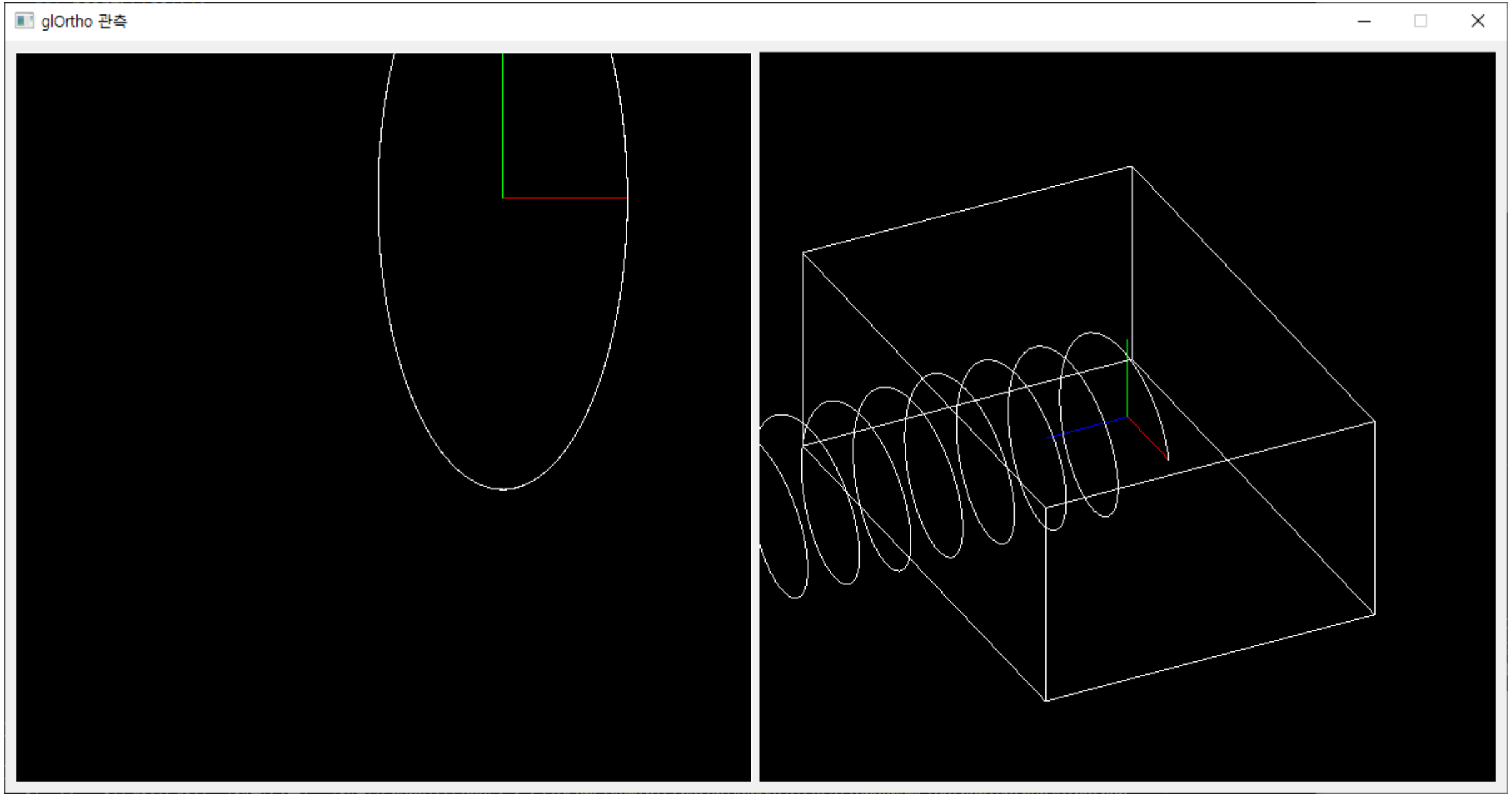
결과



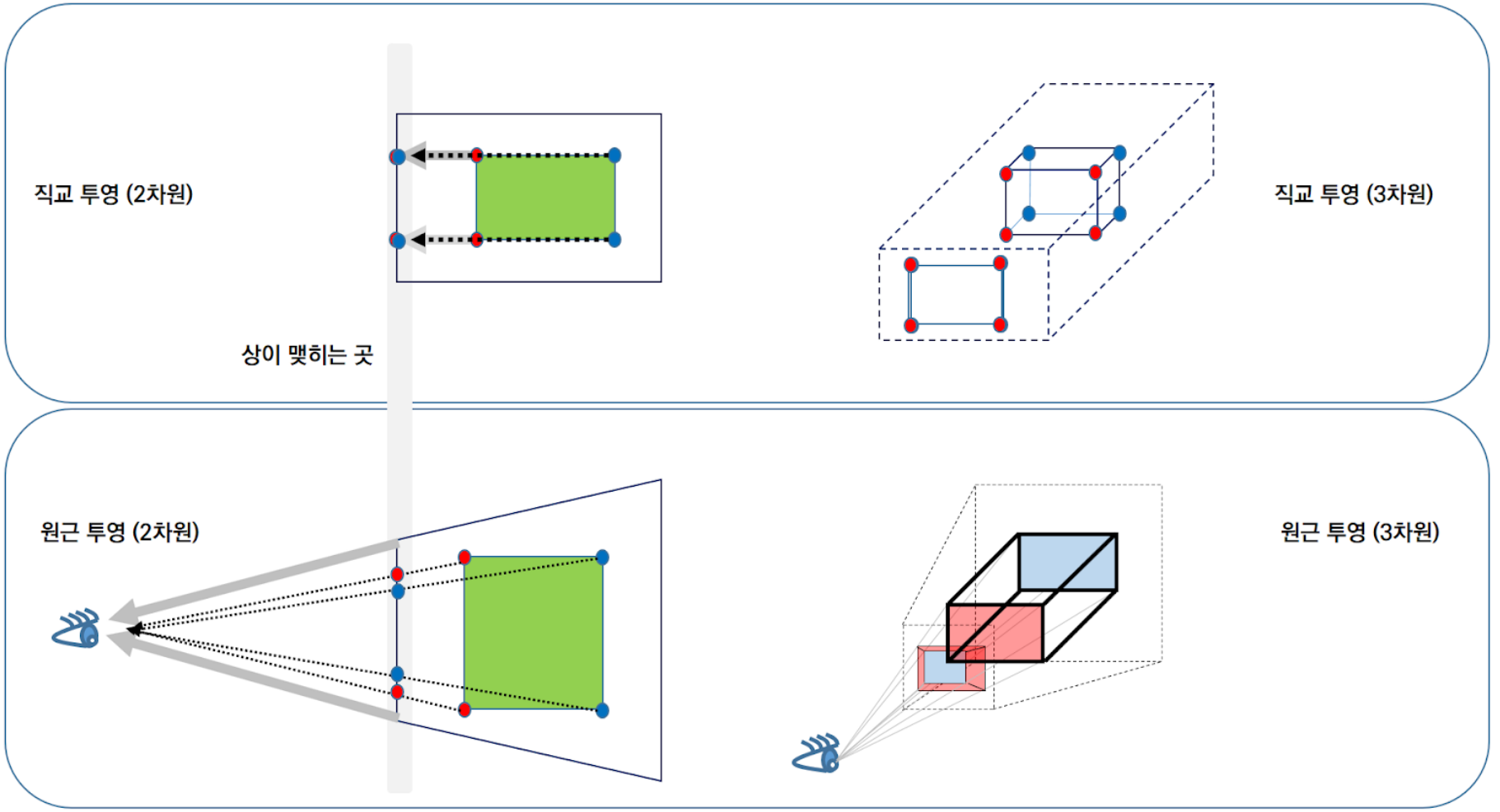
키보드를 통한 관측 공간 제어

```
class MyWindow(QMainWindow):
    . . .

    def keyPressEvent(self, e):
        if e.key() == Qt.Key.Key_A:
            MyGLWidget.left -= 0.1
        elif e.key() == Qt.Key.Key_S:
            MyGLWidget.left += 0.1
        elif e.key() == Qt.Key.Key_D:
            MyGLWidget.right -= 0.1
        elif e.key() == Qt.Key.Key_F:
            MyGLWidget.right += 0.1
        elif e.key() == Qt.Key.Key_Q:
            MyGLWidget.top += 0.1
        elif e.key() == Qt.Key.Key_W:
            MyGLWidget.top -= 0.1
        elif e.key() == Qt.Key.Key_Z:
            MyGLWidget.near += 0.1
        elif e.key() == Qt.Key.Key_X:
            MyGLWidget.near -= 0.1
        elif e.key() == Qt.Key.Key_V:
            MyGLWidget.far += 0.1
        elif e.key() == Qt.Key.Key_C:
            MyGLWidget.far -= 0.1

        self.glWidget1.update()
        self.glWidget2.update()
```

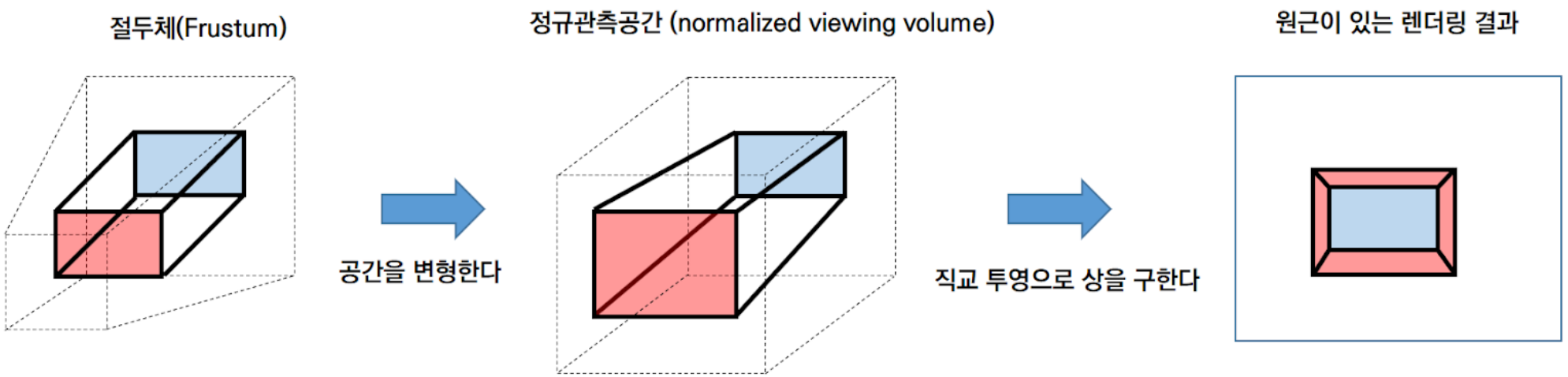
```
class MyGLWidget(QOpenGLWidget):
    . . .

    def paintGL(self):

        self.projection_update()

        glClear(GL_COLOR_BUFFER_BIT)
        . . .

    def projection_update(self):
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        if self.observation:
            glOrtho(-4, 4, -4, 4, -100, 100)
        else:
            glOrtho(self.left, self.right,
                    self.bottom, self.top,
                    self.near, self.far)
```


glOrtho 관측

원근이 파악되는 투영 – glFrustum



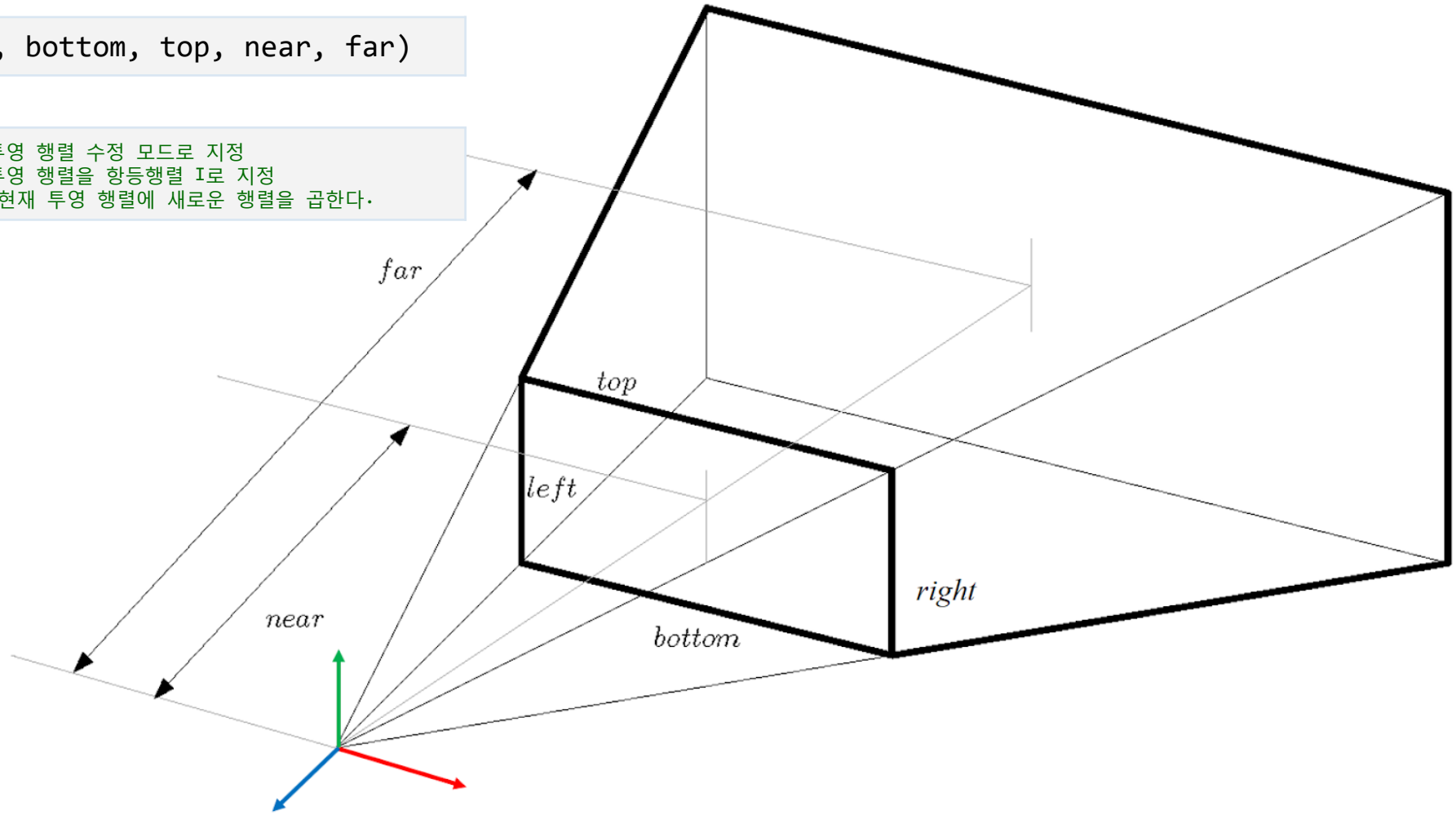
원근이 파악되는 투영 – glFrustum



원근이 파악되는 투영 – glFrustum

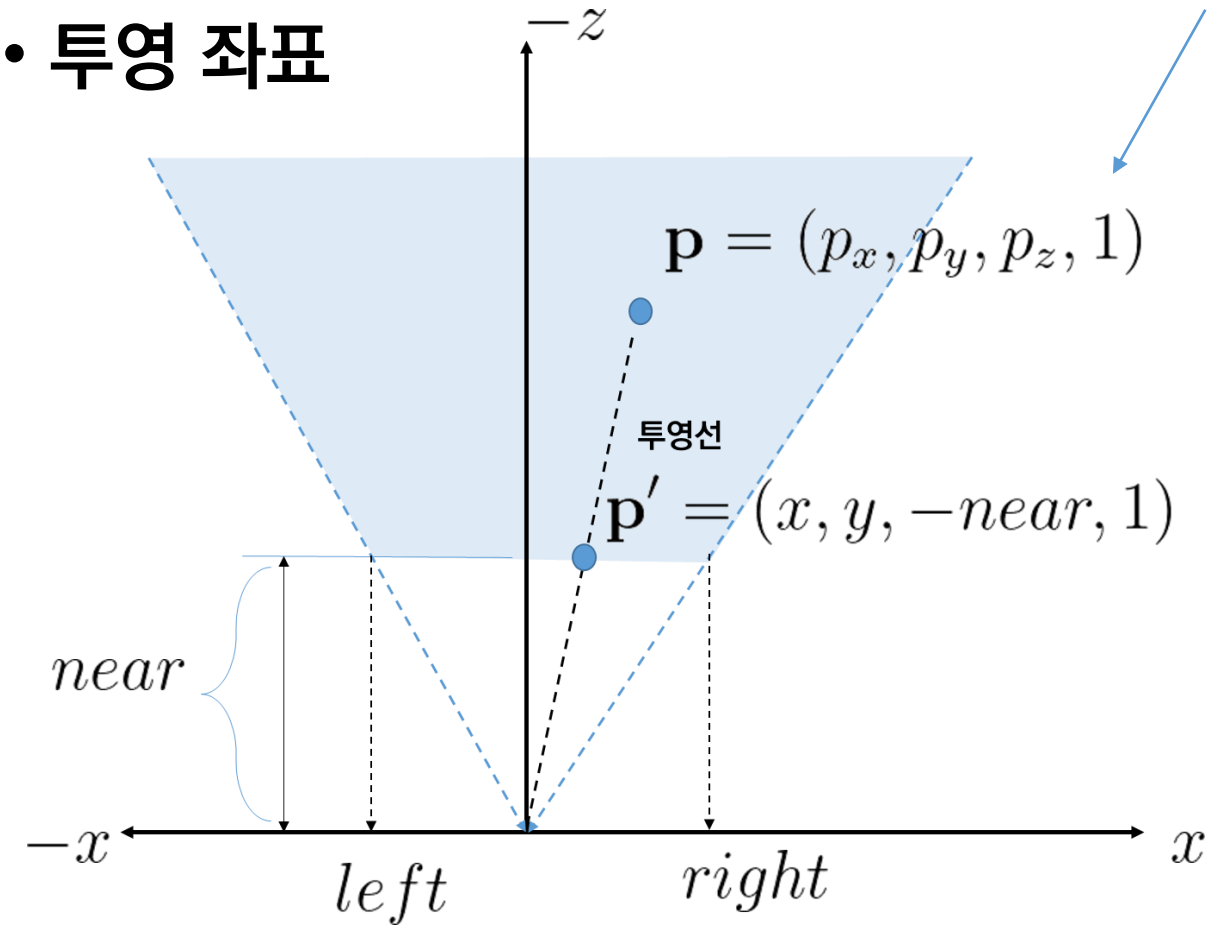
```
glFrustum(left, right, bottom, top, near, far)
```

```
glMatrixMode(GL_PROJECTION)   # 투영 행렬 수정 모드로 지정
glLoadIdentity();             # 투영 행렬을 항등행렬 I로 지정
glFrustum(l, r, b, t, n, f)    # 현재 투영 행렬에 새로운 행렬을 곱한다.
```



원근 투영 행렬 구하기

- 투영 좌표



동차좌표homogeneous coordinate $\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x/w \\ y/w \\ z/w \\ 1 \end{pmatrix}$

$$p_x : p_z = x : -near$$

$$x = -near\frac{p_x}{p_z}$$

정규장치좌표계로 옮겨야 함
[left, right] ➔ [-1, 1]

정규 장치좌표계로 옮기기

- 직교 투영에서 다루어 본 계산

$$x' = \left(-\frac{2near}{right - left}\right)\frac{p_x}{p_z} - \frac{right + left}{right - left}$$

y 축 방향으로도 동일한 계산

$$y' = \left(-\frac{2near}{top - bottom}\right)\frac{p_y}{p_z} - \frac{top + bottom}{top - bottom}$$

z 좌표의 투영 변환

- x, y 좌표의 변환을 보면 z 좌표로 나누는 모양
  - 동일한 꼴로 표현하기 위해 미지수 도입

$$z' = \frac{\alpha}{p_z} + \beta$$

z 축 좌표가 far라면 1로, near라면 −1로 옮겨져야 함

$$\left.\begin{aligned} -1 &= \frac{\alpha}{near} + \beta \\ 1 &= \frac{\alpha}{far} + \beta \end{aligned}\right\} \quad \alpha = \frac{near \cdot far}{far - near} \qquad \beta = \frac{far + near}{far - near}$$

$$z' = \frac{near \cdot far}{(far - near)p_z} + \frac{far + near}{far - near}$$

투영행렬 유도

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} -x' \cdot p_z \\ -y' \cdot p_z \\ -z' \cdot p_z \\ -p_z \end{pmatrix} = \begin{pmatrix} \frac{2near}{right-left} p_x + \frac{right+left}{right-left} p_z \\ \frac{2near}{top-bottom} p_y + \frac{top+bottom}{top-bottom} p_z \\ -\frac{far+near}{right-left} p_z - \frac{2near \cdot far}{far-near} \\ -p_z \end{pmatrix}$$

$$\mathbf{M} = \begin{pmatrix} \frac{2near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & \frac{-2near \cdot far}{far-near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Frustum 그리기

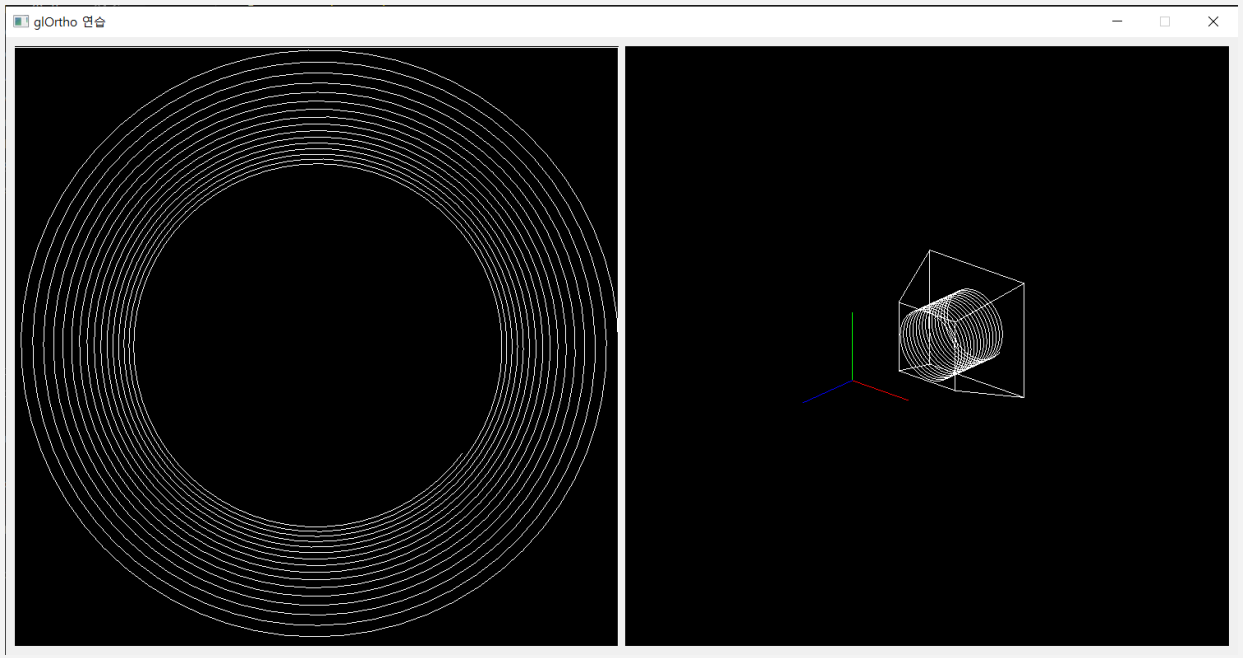
- l, r, b, t = 가까운 쪽 면의 left, right, bottom, top
- 먼 쪽 면의 left, right, bottom, top: 간단한 비례식으로 구할 수 있음
  - L, R, B, T

$$L = \frac{l}{n}f \qquad R = \frac{r}{n}f \qquad B = \frac{b}{n}f \qquad T = \frac{t}{n}f$$

- 가까운 면의 z 좌표는 −n, 먼 쪽 면의 z 좌표는 −f

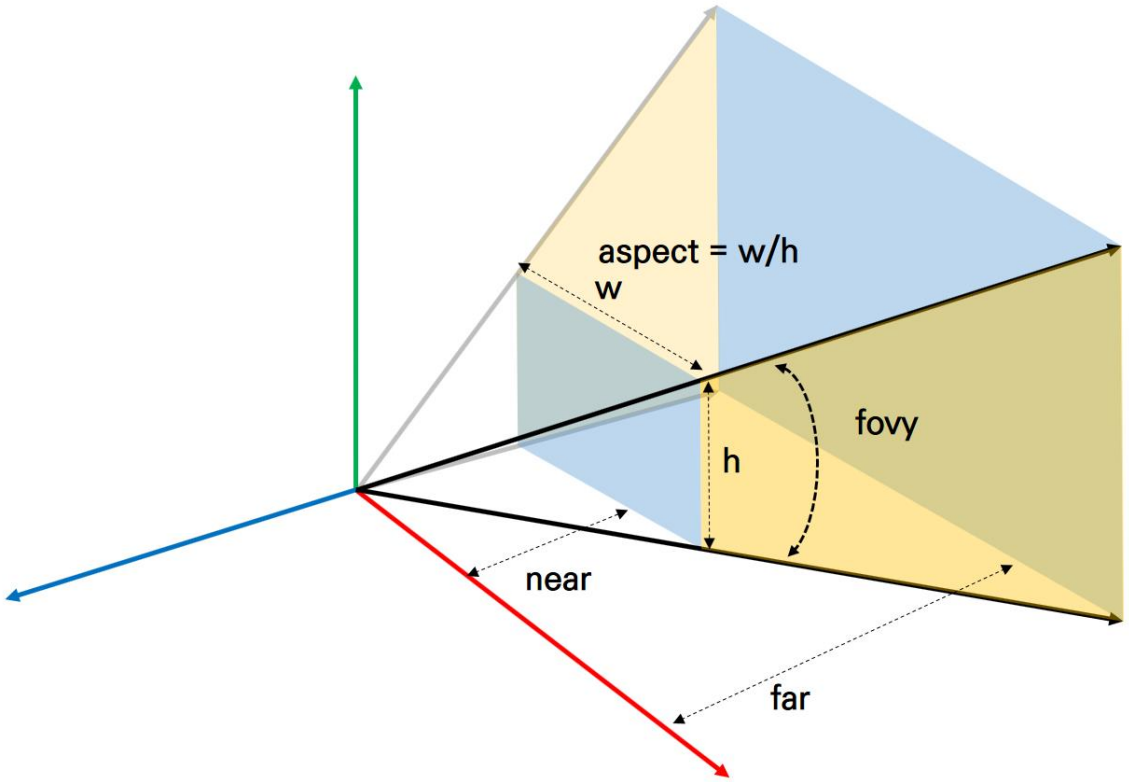
Frustum 그리기

```
def drawFrustum(l, r, b, t, n, f):
    L = l * (f/n)
    R = r * (f/n)
    B = b * (f/n)
    T = t * (f/n)
    glColor3f(1,1,1)
    glBegin(GL_LINE_LOOP)
    glVertex3f(l,t,-n)
    glVertex3f(l,b,-n)
    glVertex3f(r,b,-n)
    glVertex3f(r,t,-n)
    glEnd()
    glBegin(GL_LINE_LOOP)
    glVertex3f(L,T,-f)
    glVertex3f(L,B,-f)
    glVertex3f(R,B,-f)
    glVertex3f(R,T,-f)
    glEnd()
    glBegin(GL_LINES)
    glVertex3f(l,t,-n)
    glVertex3f(L,T,-f)
    glVertex3f(l,b,-n)
    glVertex3f(L,B,-f)
    glVertex3f(r,b,-n)
    glVertex3f(R,B,-f)
    glVertex3f(r,t,-n)
    glVertex3f(R,T,-f)
    glEnd()
```



gluPerspective

- 직관적인 파라미터로 원근 투영 제어



glFrusum(left, right, bottom, top, near, far)
gluPerspective(fovy, aspect, near, far)

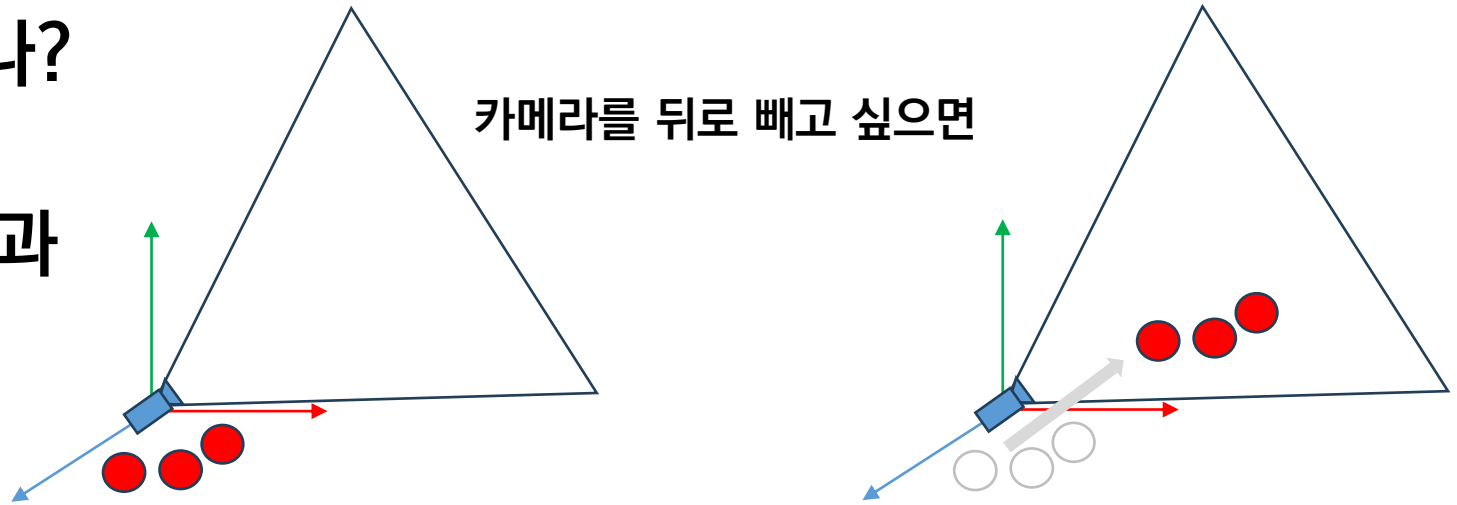
$$top = near \cdot \tan \frac{\theta}{2}$$
$$bottom = -top$$
$$right = top \cdot aspect$$
$$left = -right$$

- 표현력은 glFrustum이 더 높음
- gluPerspective는 좌우 대칭인 Frustum만 가능

카메라의 이동

- OpenGL의 카메라는 언제나 원점에 존재
- 카메라의 방향은 언제나 z 축 음의 방향

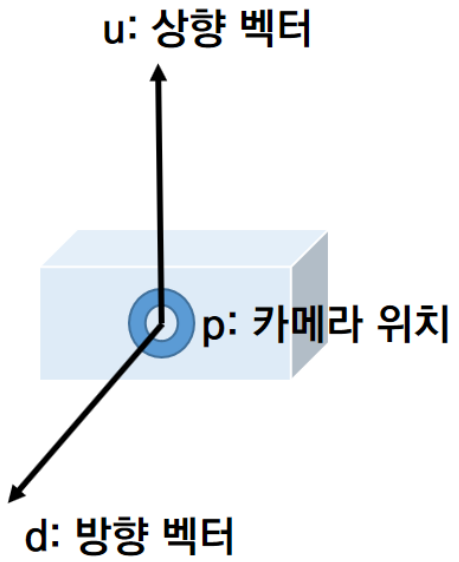
- 카메라는 어떻게 옮기나?
  - 못 옮김
- 카메라 이동과 같은 효과
  - 세상을 옮기기



물체의 좌표를 옮기는 동작은 모델뷰model-view 행렬을 사용

카메라 위치 제어

- gluLookAt(px, py, pz, px+dx, py+dy, pz+dz, ux, uy, uz)




```
gluLookAt(
        p.x, p.y, p.z,
        p.x+ d.x, p.y + d.y, p.z + d.z,
        u.x, u.y, u.z
)
```

실제로 카메라를 옮기는 것이 아니라 동일한 효과가 나도록 세상을 옮기는 함수