

# 변환과 계층적 모델링

동명대학교 게임공학과

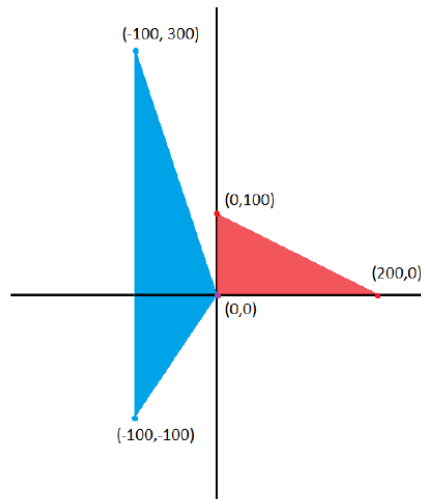
강영민

# 변환

- 여러 개의 원을 그리는 방법

- 옵션 1: 서로 다른 그리기를 사용하는 방법
- 옵션 2: 기본적인 원 그리기를 구현하고, 변환을 적용하는 방법
  - 변환: 좌표 집합을 다른 어떤 좌표 집합으로 변경하는 일

- 변환을 자유자재로 할 수 있다고 가정:  $T(\mathbf{p}, \vec{d})$ , 크기를 변경  $S(\mathbf{p}, s)$
- 필요한 것은 반지름 1의 원 그리기  $\text{draw}()$ 만 있으면 어떤 위치에 어떤 크기로든 원을 그릴 수 있음
  - $T(S(\text{draw}(), s), \vec{d})$
- 어파인(affine) 변환 - 직선은 직선으로, 평행선은 평행선으로 유지



# OpenGL의 변환

- Affine Transformation
- 대표적인 Affine Transformation

이동(translate)	주어진 변위 벡터만큼 좌표를 동일하게 옮겨 놓는다.
회전(rotate)	2차원에서는 기준점, 3차원에서는 기준축을 중심으로 주어진 각도만큼 돌아간다.
크기변경(scale)	각 축 방향으로 주어진 비율에 따라 좌표 값이 커지거나 줄어든다.


# 동차 좌표계

- 동차 좌표(homogeneous coordinate)은  $n$  차원의 사영공간을  $n + 1$  차원의 좌표로 나타내는 좌표계
- 1827년 아우구스트 페르디난드 뫼비우스(August Ferdinand Möbius)가 그의 저작 “Der barycentrische Calcul”에서 처음으로 소개
- 사영기하학에서 사용되는 좌표계
- 무한의 위치에 있는 점을 유한 좌표로 표현하는 데에 적합

## 그래픽스에서 동차좌표계를 사용하는 이유

- 3차원 데카르트 좌표를 사용할 경우 이동은 벡터의 덧셈으로 표현되고, 회전은  $3 \times 3$  행렬의 곱으로 표현
- 이동과 회전이 누적되면 벡터 덧셈과 행렬 곱셈이 연속적 적용됨
- 동차좌표(homogeneous coordinate)을 사용하면 이동과 회전 모두  $4 \times 4$  행렬의 곱으로 표현 가능
- 누적된 이동, 회전 변환을 하나의 행렬로 표현 가능

# 변환 행렬

- 동차 좌표계에서 어파인 변환은 **행렬로 표현**  **선형변환**
- OpenGL 역시 변환을 행렬로 표현
- OpenGL은 정점 데이터를 그래픽 카드로 보내는데, 각각의 정점들은 필요한 변환 행렬에 곱해져서 최종적인 화면 좌표 생성
- 곱해지는 행렬은 두 종류: 모델뷰(ModelView) 행렬, 투영(Projection) 행렬
- 하나의 정점이 화면 좌표로 바뀌는 과정
  - 지역좌표계 내의 좌표를  $coord_{local}$ 이라고 하고, 모델뷰 행렬은  $ModelViewMatrix$ , 투영행렬을  $ProjMatrix$ 라고 하면, 화면 좌표  $coord_{screen}$ 은 다음과 같이 구한다.
  - $coord_{screen} = ProjMatrix \times ModelViewMatrix \times coord_{local}$

# OpenGL에서 변환은 어떻게 이루어지나

- OpenGL의 변환: *ModelViewMatrix*, *ProjMatrix* 행렬을 변경하는 것
- 모델 뷰 행렬, 투영 행렬 중 어느 것을 바꿀 것인지는 행렬 모드로 결정
- 행렬모드: GL\_MODELVIEW와 GL\_PROJECTION
  - 당분간 GL\_TEXTURE는 고려하지 않음
- 투영 행렬을 변경 = 카메라의 속성을 변경: glOrtho나 gluPerspective
- 모델뷰 행렬을 변경하는 함수
  - 카메라 위치 변경: gluLookAt
  - 객체의 위치, 방향, 크기 변경 - 표 참고

glTranslate3f(dx, dy, dz)	(dx,dy,dz) 만큼 좌표 이동
glRotatef(angle, axisX, axisY, axisZ)	축(axisX, axisY, axisZ)를 기준으로 angle만큼 회전
glScalef(scaleX, scaleY, scaleZ)	(scaleX, scaleY, scaleZ)를 성분별로 좌표에 곱함

- 이상의 변환들은 좌표를 변경하지 않고 행렬을 변경하는 역할을 수행
- 현재의 행렬이 **M**이라고 하고, glTranslate의 이동을 표현하는 행렬이 **T**
- glTranslate 함수가 불렸을 때, **M**을 **MT**로 변경

# 현재 변환 행렬

- 현재 변환 행렬(current transform matrix)은 줄여서 CTM이라 부름
- 모든 정점은 CTM에 곱해진다
- 따라서 변환 함수는 이 CTM을 변경하는 것
- CTM을 변경하는 함수로 다음과 같은 것들이 있다

함수	CTM의 변경
LoadIdentity	$CTM := I$
LoadMatrix(M)	$CTM := M$
T: Translate*	$CTM := CTM * T$
R: Rotate*	$CTM := CTM * R$
S: Scale*	$CTM := CTM * S$

# 변환의 적용에 따른 CTM 변화 과정

- 다음 코드와 같이 변환이 적용될 경우
- CTM은 이 코드에 주석 처리된 부분에 나타난 것과 같이 변경

```
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()           // CTM:=I
glTranslatef(1,2,1)        // CTM:=IT  T는 glTranslate가 표현하는 행렬
glRotatef(45, 1,0,0)       // CTM:=ITR  R은 glRotate가 표현하는 행렬
glScalef(2.0, 1.0, 1.0)    // CTM:=ITRS S는 glScale이 표현하는 행렬
drawObjects()
```

## 코드 구현 내용

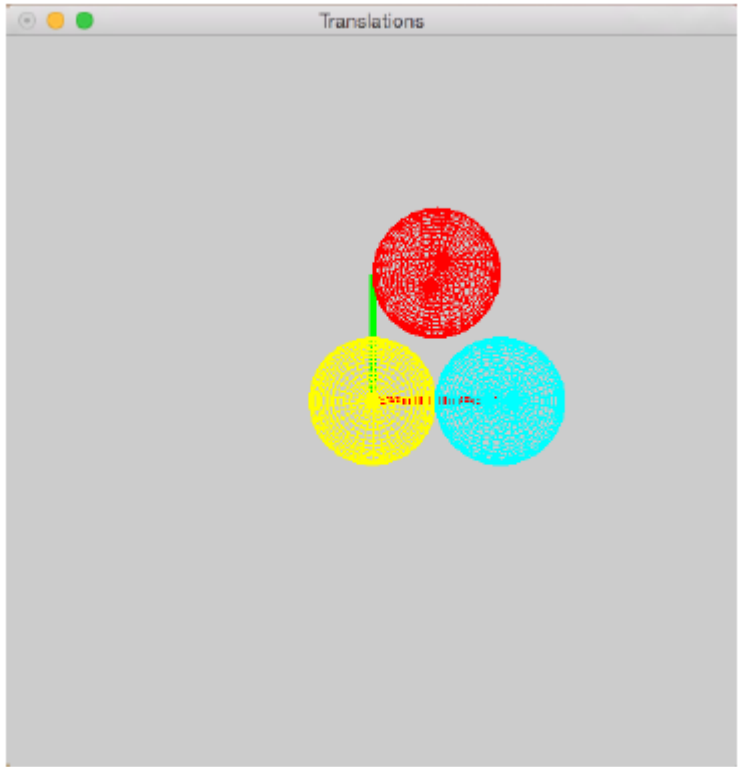
`drawObjects` 내에서 그려지는 모든 기하 객체의 좌표에 대해 크기변환 (**S**)을 적용한 뒤, 이 값을  $x$  축 기준으로 45도 회전하게 되며, 이후 (1,2,1) 만큼의 이동을 한다. 다시말해 우리가 호출한 변환의 역순으로 변환이 객체에 적용된다.



# 구 그리기에 변환 적용 예제

```
def draw() :  
    drawSphere( r=0.5, at=(0,0,0) )           // 반지름 0.5의 구를 원점을 중심으로 그림  
  
def display() :  
    [[여기에 카메라 설정(gluLookAt 등), 버퍼 클리어(glClear) 코드]]  
    drawAxes() // 축을 그림  
    glColor3f(1, 1, 0)  
    draw() // 아무런 변환 없이 원점에 구를 그림  
    glTranslatef(1.0, 0.0, 0.0) //(1,0,0) 벡터만큼 이동을 함  
    glColor3f(0, 1, 1)  
    draw() //적용된 변환을 이용하여 구를 그림  
    glTranslatef(-0.5, 1.0, 0.0) /(-0.5,1,0) 벡터만큼 이동을 함  
    glColor3f(1, 0, 0)  
    draw() //적용된 변환을 이용하여 옮겨진 구를 그림  
    glutSwapBuffers()
```

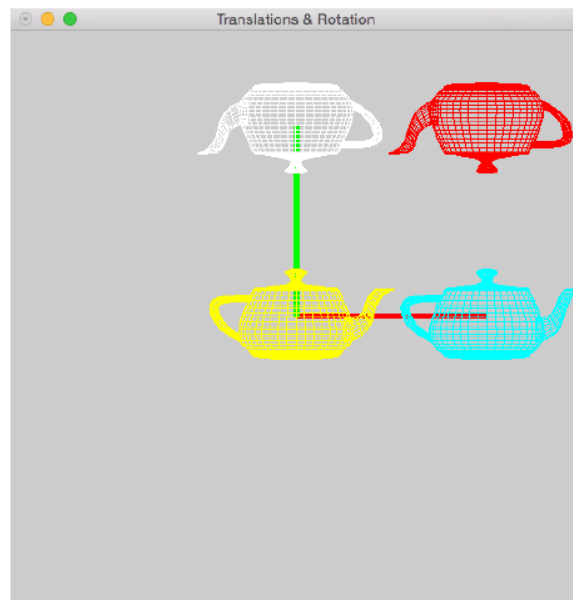
# 구 그리기에 변환 적용 예제



- 노란 색, 하늘 색, 붉은 색 공이 차례로 그려진다.
- 노란 색 공에는 변환이 적용되지 않았으므로 원점에
- 하늘 색 공에는  $(1, 1, 0)$  만큼의 이동
- 붉은 색 공에  $(-0.5, 1, 0)$  적용 + 하늘 색에 적용된  $(1, 1, 0)$  이동이 추가

# 복합 변환의 이해

- 실제 객체에 적용되는 변환은 호출 순서의 역으로 적용
- CTM을 활용한 변환 적용 기법을 사용하기 때문
- 이러한 방식으로 이해하면 적용된 변환을 상상하기가 어려움
- 다음 그림과 같이 네 개의 주전자를 배치하고 싶을 때에 OpenGL로 구현하려면 어떻게 해야할 지를 고민해 보자.



# CTM 모델로 원하는 변환 구현하기

- 노란색 주전자는 아무런 변환도 적용되지 않은 상태. 바로 그림

```
glLoadIdentity()  
draw(yellow)           // CTM = I
```

- 하늘색 주전자는  $x$  축으로 1만큼 이동하여 그려야 하므로  $\text{glTranslate}^*$  함수를 부른 뒤 주전자를 그린다. 이 이동 변환을  $\mathbf{T}_1$ 이라고 하자.

```
glTranslatef(1.0, 0.0, 0.0) // T1  
draw(cyan)
```

- 붉은 주전자는  $z$  축을 기준으로 180도 회전한 뒤에 (1,1,0)만큼 이동
- 현재의 CTM이  $\mathbf{T}_1$ 이므로 (0,1,0)만큼의 이동  $\mathbf{T}_2$ 를 추가 적용하면 (1,1,0) 이동이 되며, 여기에 회전  $\mathbf{R}$ 을 적용

```
glTranslate(0.0, 1.0, 0.0) // T2  
glRotatef(180, 0.0, 0.0, 1.0) // R  
draw(red)
```

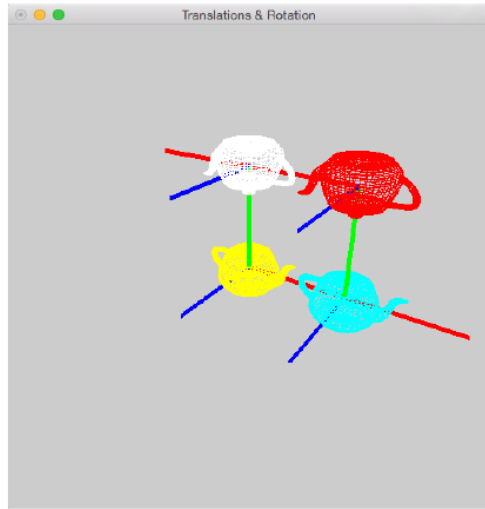
# CTM 모델로 원하는 변환 구현하기

- 붉은 색 주전자는  $\mathbf{T}_1\mathbf{T}_2\mathbf{R}$ 의 변환
- 흰색 주전자를 그리는 방법은 까다롭다
- 이 주전자는 180도 회전 시킨 주전자를 (0,1,0) 만큼 이동한 것
- 현재의 변환  $\mathbf{T}_1\mathbf{T}_2\mathbf{R}$ 에 어떤 변환을 추가적으로 호출하면 이것이 가능할지 생각해야 함
- 회전이 호출된 상태라 까다롭다.
- 새롭게 적용할 변환이  $\mathbf{X}$ 라고 하고, (0,1,0) 만큼 이동하는 변환을  $\mathbf{T}$ 라고 하면 다음과 같은 관계 성립
  - $\mathbf{T}_1\mathbf{T}_2\mathbf{R}\mathbf{X} = \mathbf{T}\mathbf{R}$
  - $\mathbf{X} = \mathbf{R}^T\mathbf{T}_2^{-1}\mathbf{T}_1^{-1}\mathbf{T}\mathbf{R}$
- 위의 계산을 수행하면  $\mathbf{X}$ 는 (1,0,0) 만큼의 이동으로 나타나게 된다. 따라서 다음과 같이 적용하면 원하는 결과를 얻을 수 있다.

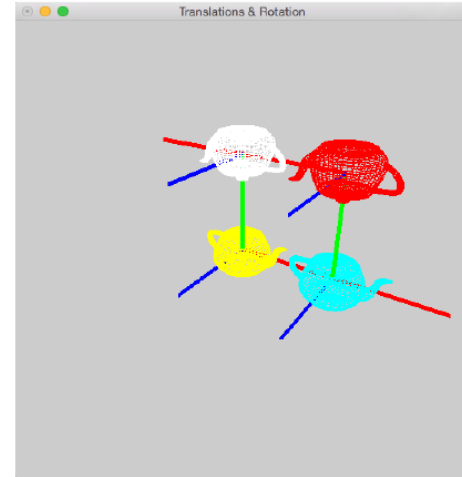
```
glTranslatef(1.0, 0.0, 0.0) // X
draw(white)
```

# 지역좌표계 변환을 중심으로 생각하기

- 앞 슬라이드의 방식은 너무 번거롭다.
- 쉬운 방법: 물체의 변환이 아니라 좌표계의 변환으로 이해
- 각각의 주전자가 가진 지역 좌표계를 함께 그리기
- 노란색 주전자는 변환 적용 없어 전역 좌표계와 일치



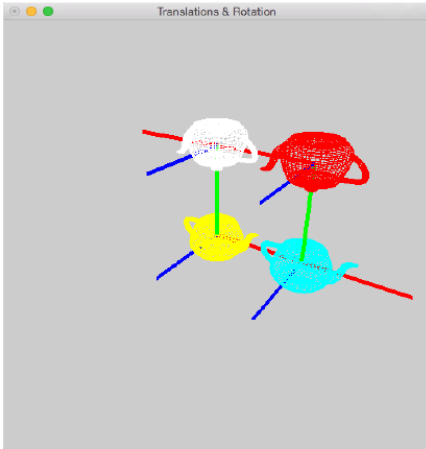
```
glLoadIdentity()  
draw(yellow)
```



하늘색 주전자는 노란색 주전자의 좌표계에서 보았을 때  $x$  축으로 1만큼 이동하여 있다. 따라서 다음과 같이 적용한다.

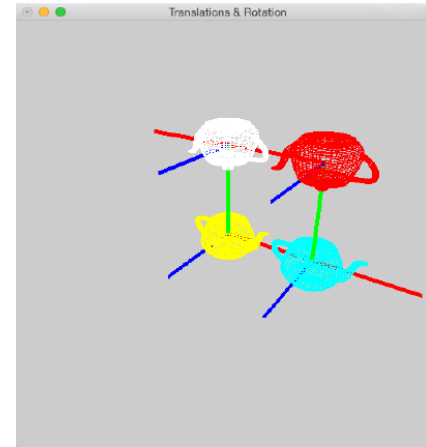
```
glTranslatef(1.0,0.0,0.0)  
draw(cyan)
```

# 지역좌표계 변환을 중심으로 생각하기



붉은 색 주전자는 하늘색 주전자에서  $y$  축으로 1만큼 올라간 뒤에, 이동된 좌표계에서  $z$  축을 기준으로 180도 회전

```
glTranslatef(0.0, 1.0, 0.0)
glRotatef(180, 0.0, 0.0, 1.0)
draw(red)
```

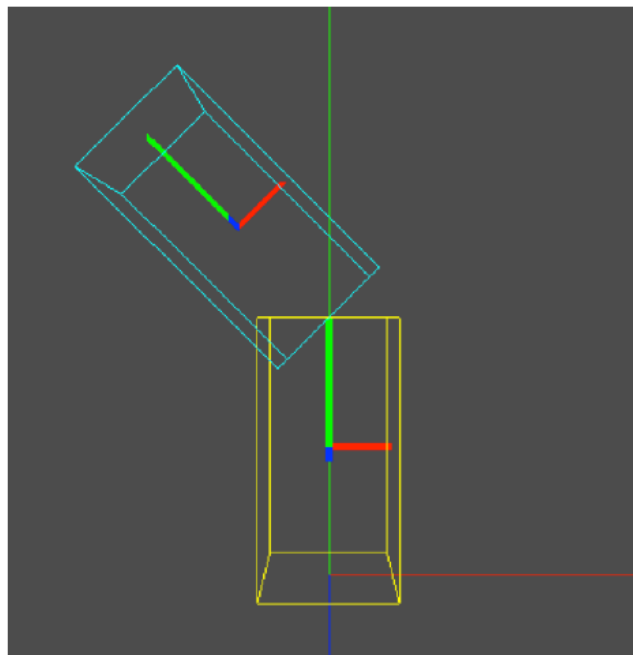


마지막으로 흰색 주전자는 붉은 색 주전자에서  $x$  축으로 1만큼 이동한 것

```
glTranslatef(1.0, 0.0, 0.0)
draw(white)
```

# 크기 변경이 포함된 경우

- 크기 변경이 포함될 경우에는 문제가 발생
  - 지역좌표계 변환을 통한 이해는 강체(rigid) 변환에 대해서만 적용
- 다음의 그림과 같은 장면을 그린다고 가정
  - 각 변 길이 1인 정육면체를 높이 2, 너비 1, 깊이 1로 만들 경우
  - `glScalef(1.0, 2.0, 1.0)`





# 크기 변경이 포함된 경우

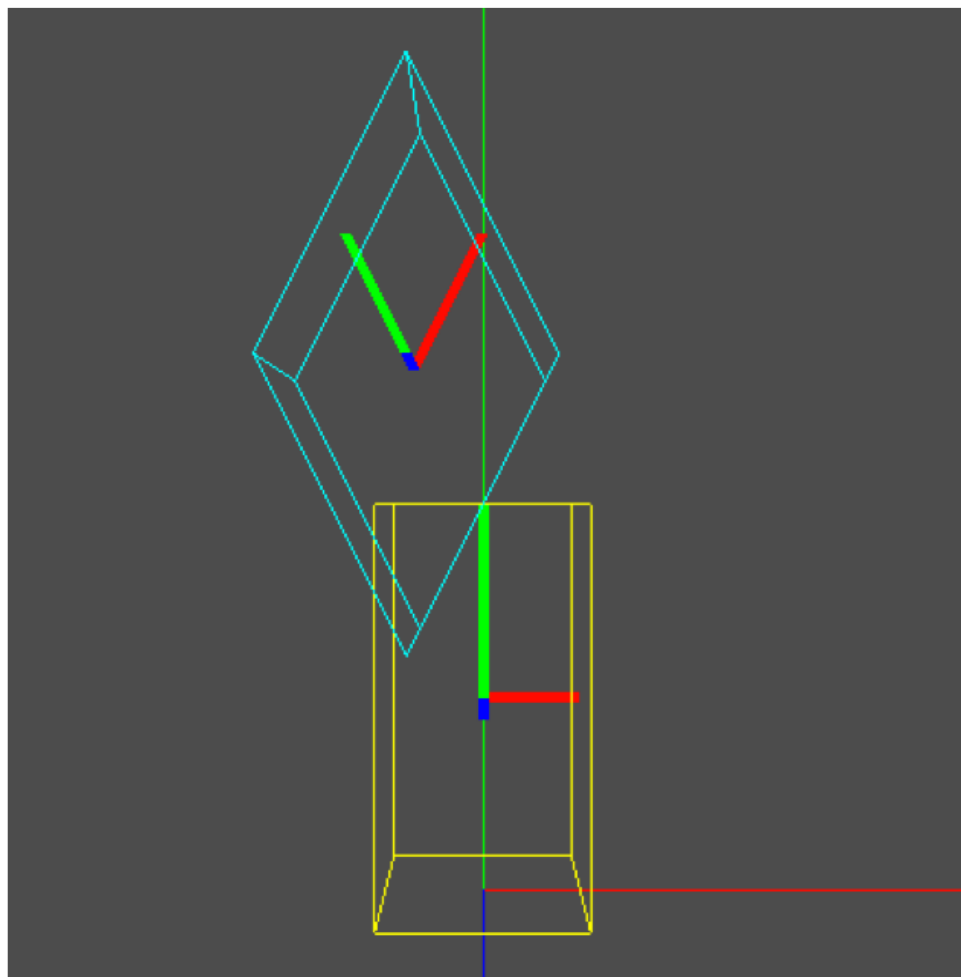
- 육면체가  $xz$  평면위에 놓으려면  $y$ 축 방향으로 0.5 이동
  - 이미 지역좌표계가 glScale에 의해 변환되었기 때문에 0.5만 들어올려도 전역좌표계에서는 1만큼 올라가게 된다.
- 노란색 육면체는 다음과 같이 그림

```
glScalef(1.0, 2.0, 1.0) // S
glTranslatef(0.0, 0.5, 0.0) // T1
draw(yellow)
```

- 문제는 하늘색 육면체
- 이 육면체는 노란색 육면체를 0.5만큼 들어올린 뒤에  $z$ 축 기준으로 회전을 시키고, 이렇게 변환된 공간에서 다시  $y$ 축 기준으로 0.5만큼 이동시키면 됨
- 코드로는 다음과 같은 변환 적용으로 구현이 가능할 것처럼 보임

```
glTranslatef(0.0, 0.5, 0.0) // T2
glRotatef(45, 0.0, 0.0, 1.0) // R
glTranslatef(0.0, 0.5, 0.0) // T3
draw(cyan)
```

# 크기 변경이 포함된 경우



# 크기 변경이 포함된 경우

- 실제로 하늘색 육면체에 적용된 변환
- $\mathbf{ST}_1\mathbf{T}_2\mathbf{RT}_3$
- 가장 마지막에 적용되는  $\mathbf{S}$ 는 전역 좌표계를 기준으로 크기변경
- 하늘색 육면체의 방향(orientation)을 고려하지 않고 크기 변경
- 해결하는 방법은 행렬 스택 연산인 `glPushMatrix()`와 `glPopMatrix()`를 활용하여 크기변경 변환은 적용된 객체 이외에서 영향을 미치지 않도록 제한

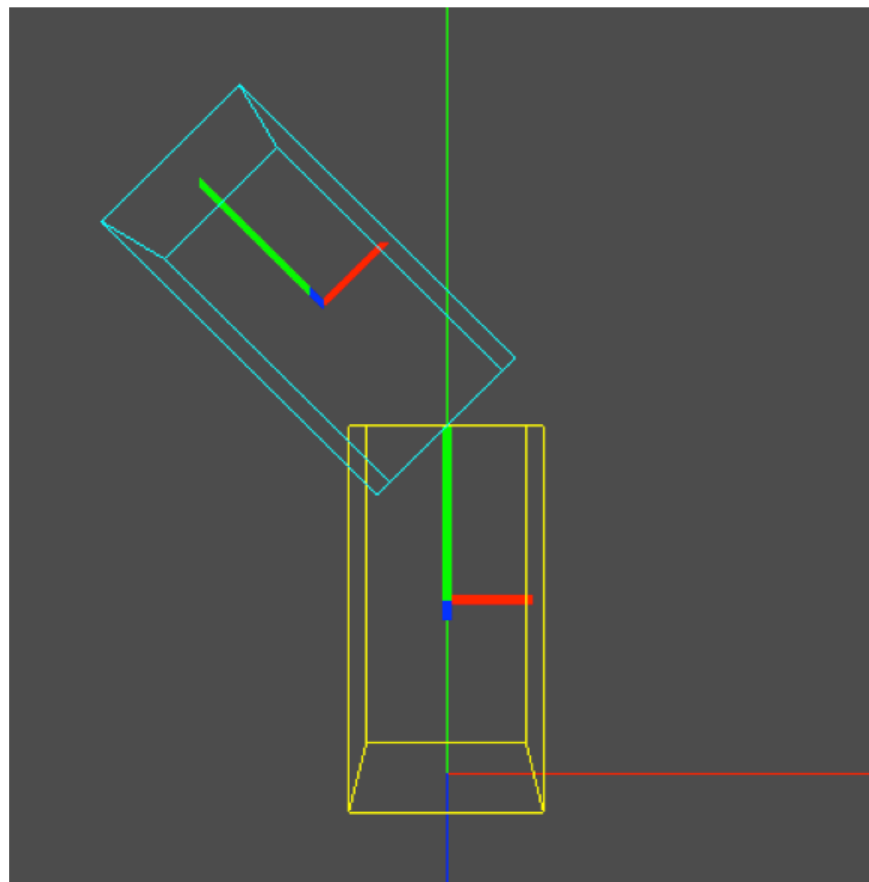
함수명	역할
<code>glPushMatrix</code>	현재의 CTM을 스택에 저장한다.
<code>glPopMatrix</code>	행렬 스택을 pop하여 CTM을 갱신한다.

- `glPushMatrix`를 수행한 뒤에 `glPopMatrix`를 수행하면 직전의 `glPushMatrix` 수행 당시의 CTM으로 복원

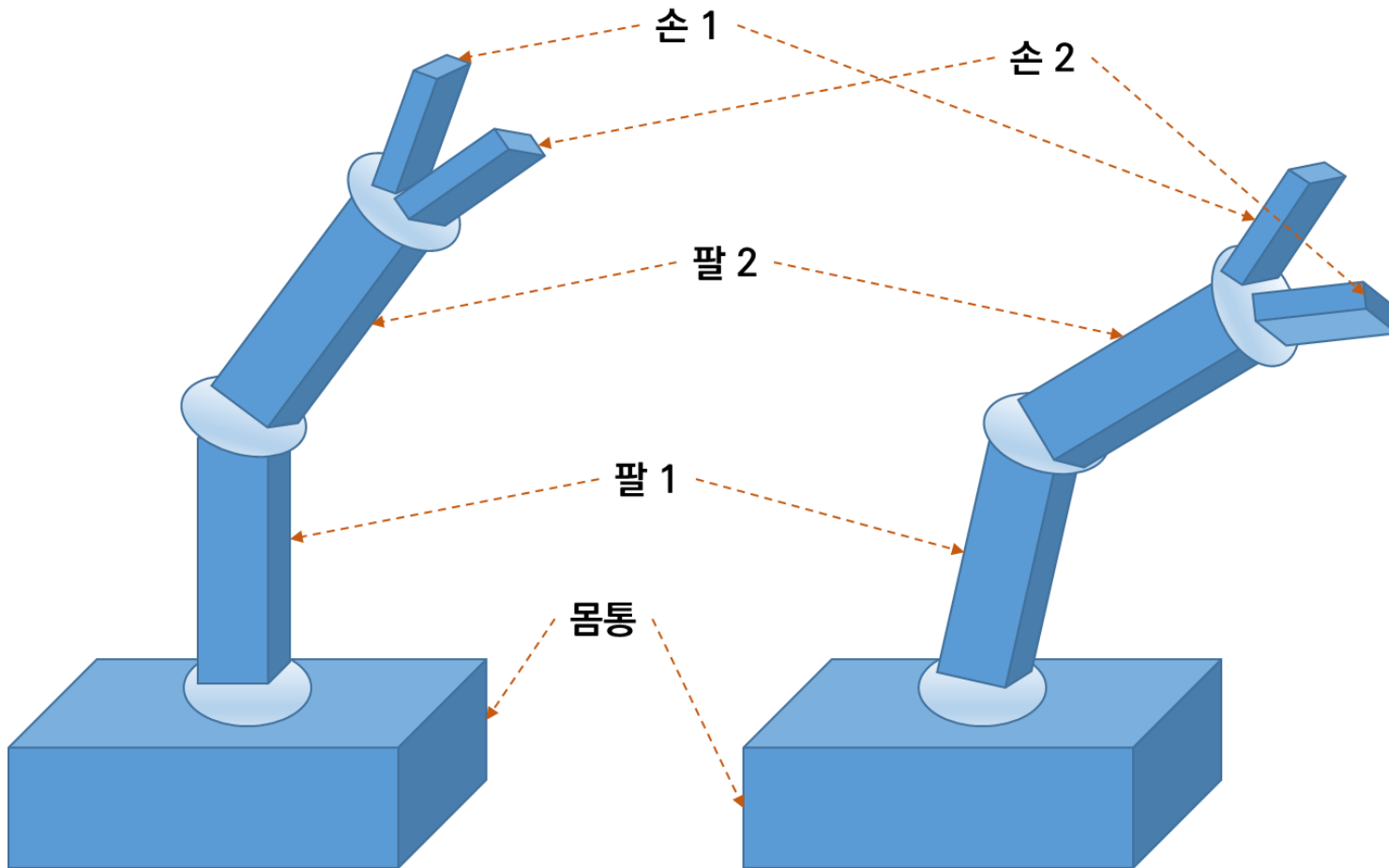
# 제대로 된 크기변경 적용 코드

```
glTranslatef(0.0, 1.0, 0.0)
glPushMatrix()
glScalef(1.0, 2.0, 1.0)
draw(yellow)
glPopMatrix()
```

```
glTranslatef(0.0, 1.0, 0.0)
glRotatef(45, 0.0, 0.0, 0.0)
glTranslatef(0.0, 1.0, 0.0)
glPushMatrix()
glScalef(1.0, 2.0, 1.0)
draw(cyan)
glPopMatrix()
```



# 연결 구조가 있는 객체

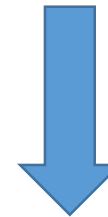


몸통은 손에 영향을 미치지만,  
손의 움직임은 몸통에 영향을 미치지 않는다



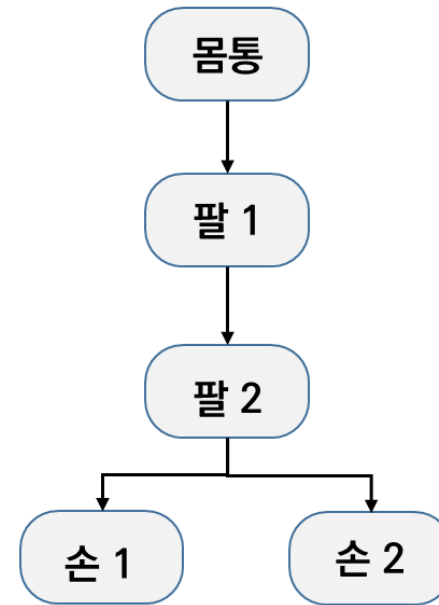
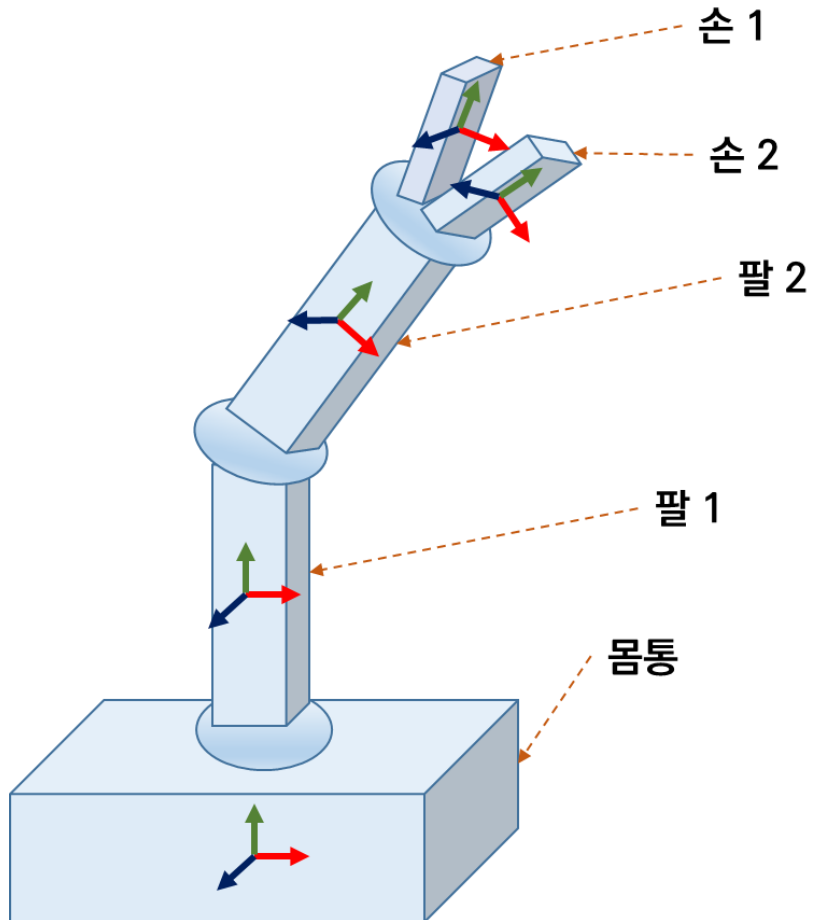
몸통 → 손

팔은 몸통의 영향을 받지만,  
손의 영향은 받지 않고,  
팔은 손에 영향을 미친다.



몸통 → 팔 → 손

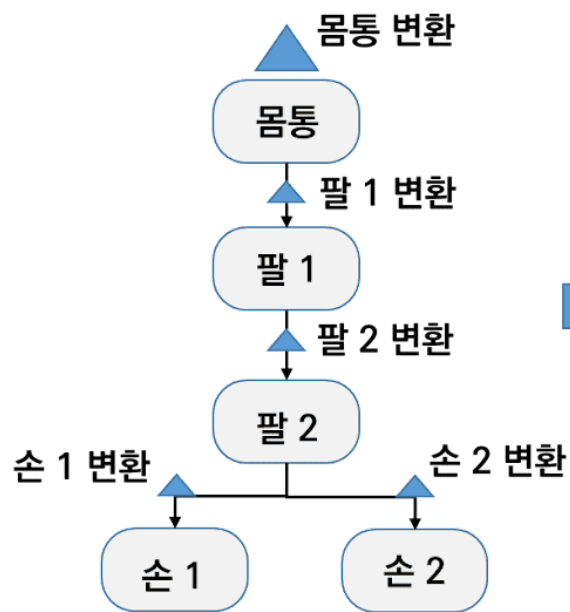
# 연결 구조가 있는 객체 각 요소의 관계



트리tree 구조  
= 계층구조

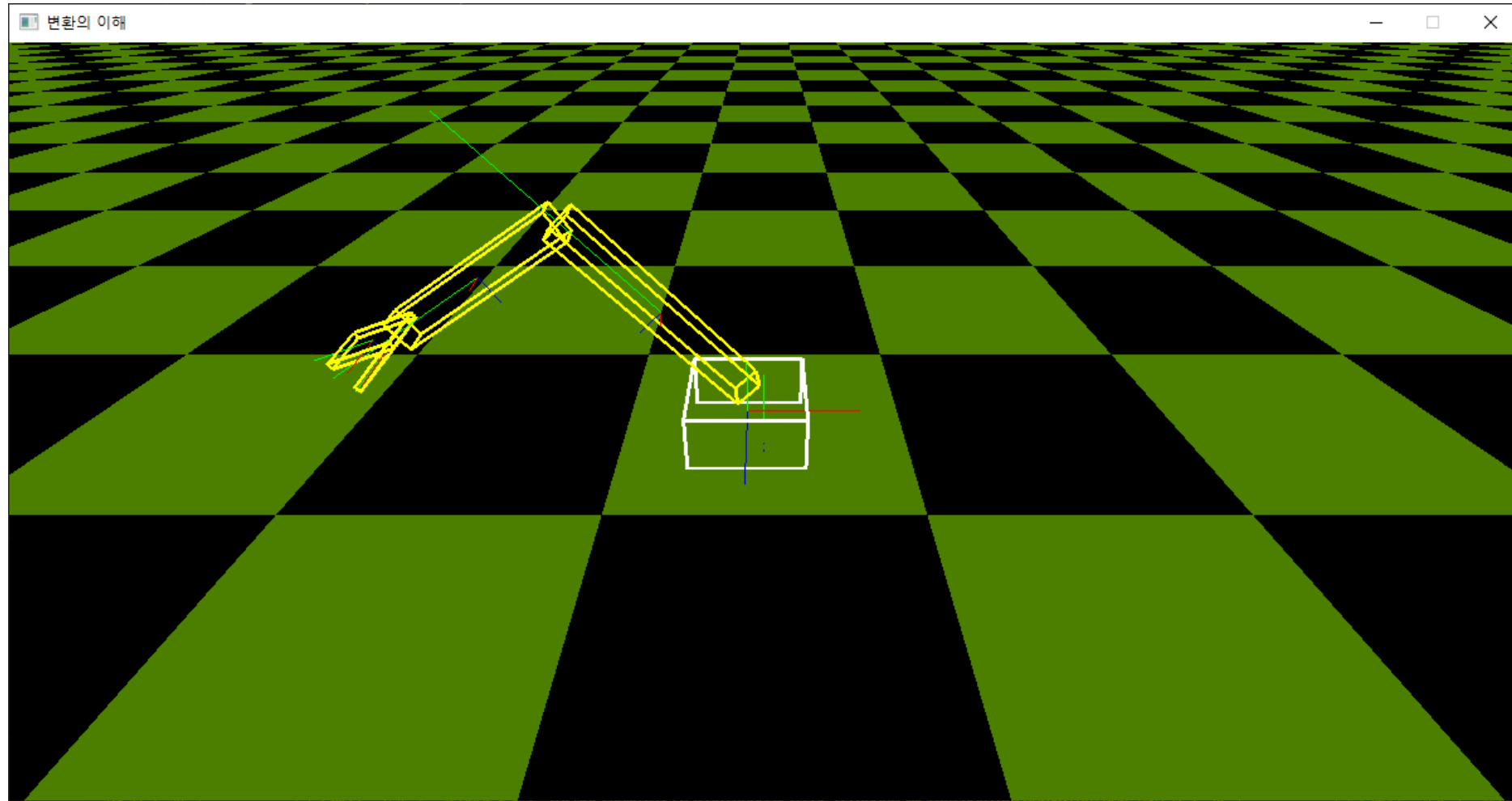
# 계층적 모델의 그리기

- glPushMatrix / glPopMatrix 활용



Code	CTM 변화
몸통변환 Draw(몸통)	CTM ← 몸통변환
팔 1 변환 Draw(팔 1)	CTM ← 몸통변환 * 팔 1 변환
팔 2 변환 Draw(팔 2)	CTM ← 몸통변환 * 팔 1 변환 * 팔 2 변환
Push Matrix 손 1 변환 Draw(손 1) Pop Matrix	Stack top = 몸통변환 * 팔 1 변환 * 팔 2 변환 CTM ← 몸통변환 * 팔 1 변환 * 팔 2 변환 * 손 1 변환 CTM ← 몸통변환 * 팔 1 변환 * 팔 2 변환
손 2 변환 Draw(손 2)	CTM ← 몸통변환 * 팔 1 변환 * 팔 2 변환 * 손 1 변환

# 계층적 모델링 프로젝트





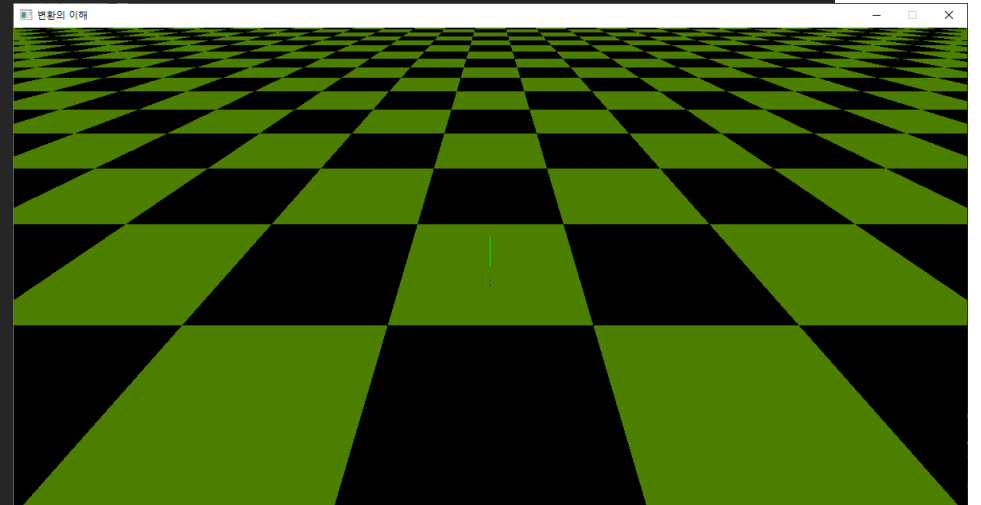
# 사용할 패키지들

```
from OpenGL.GL import *
from OpenGL.GLU import *
import sys
from PyQt6.QtWidgets import *
from PyQt6.QtOpenGLWidgets import QOpenGLWidget
from PyQt6.QtCore import *

import math
import numpy as np
```

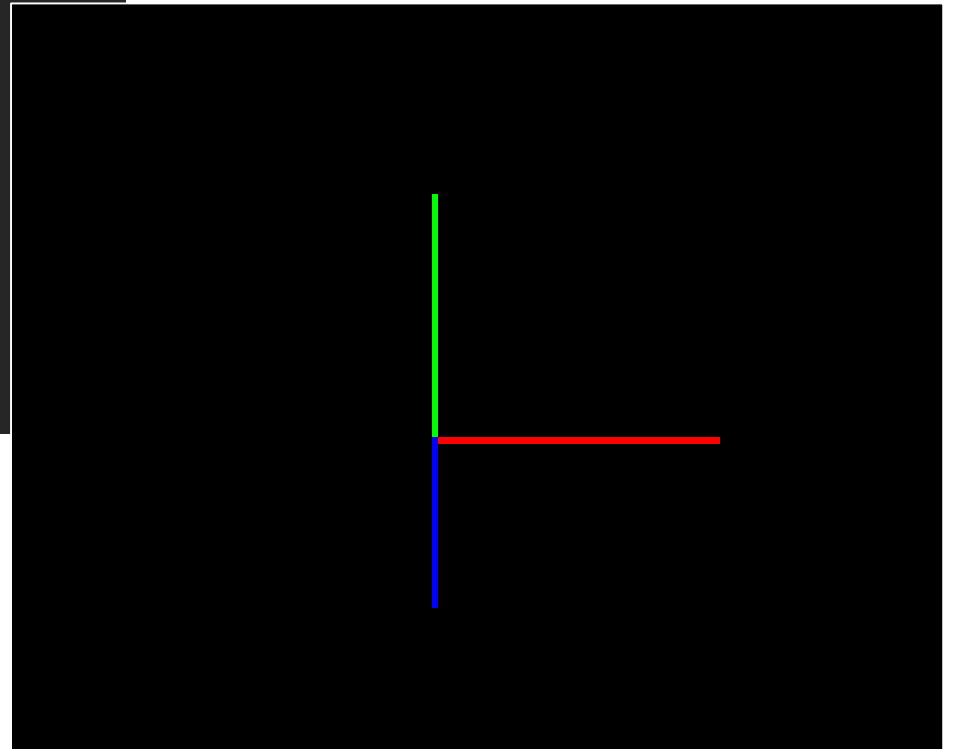
# 평면 그리기

```
def drawPlane():  
    n, w = 100, 500  
    # n: 체스판 한면의 정점수, w: 체스판 한면의 길이  
  
    d = w / (n-1) # 인접한 두 정점 사이의 간격  
  
    # 체스판 그리기  
    glColor3f(0.3,0.5,0)  
    glBegin(GL_QUADS)  
    for i in range(n):  
        for j in range(n):  
            if (i+j)%2 == 0:  
                startX = -w/2 + i*d  
                startZ = -w/2 + j*d  
                glVertex3f(startX, 0, startZ)  
                glVertex3f(startX, 0, startZ+d)  
                glVertex3f(startX+d, 0, startZ+d)  
                glVertex3f(startX+d, 0, startZ)  
  
    glEnd()
```



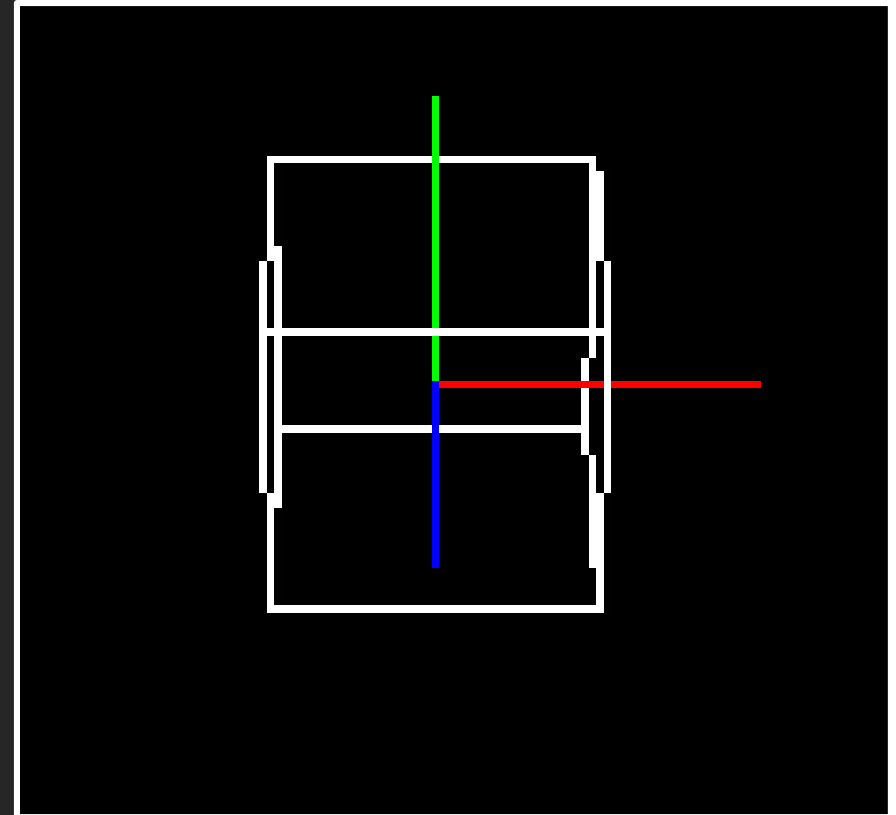
# 축 그리기

```
def drawAxes():  
    glBegin(GL_LINES)  
    glColor3f(1,0,0)  
    glVertex3f(0,0,0)  
    glVertex3f(1,0,0)  
    glColor3f(0,1,0)  
    glVertex3f(0,0,0)  
    glVertex3f(0,1,0)  
    glColor3f(0,0,1)  
    glVertex3f(0,0,0)  
    glVertex3f(0,0,1)  
    glEnd()
```



# 큐브 그리기 (폭, 높이, 깊이 모두 1인 표준 큐브)

```
def drawCube():  
    v0 = [-0.5, 0.5, 0.5]  
    v1 = [ 0.5, 0.5, 0.5]  
    v2 = [ 0.5, 0.5, -0.5]  
    v3 = [-0.5, 0.5, -0.5]  
    v4 = [-0.5, -0.5, 0.5]  
    v5 = [ 0.5, -0.5, 0.5]  
    v6 = [ 0.5, -0.5, -0.5]  
    v7 = [-0.5, -0.5, -0.5]  
    glBegin(GL_LINES)  
    glVertex3fv(v0); glVertex3fv(v1)  
    glVertex3fv(v1); glVertex3fv(v2)  
    glVertex3fv(v2); glVertex3fv(v3)  
    glVertex3fv(v3); glVertex3fv(v0)  
    glVertex3fv(v4); glVertex3fv(v5)  
    glVertex3fv(v5); glVertex3fv(v6)  
    glVertex3fv(v6); glVertex3fv(v7)  
    glVertex3fv(v7); glVertex3fv(v4)  
    glVertex3fv(v0); glVertex3fv(v4)  
    glVertex3fv(v1); glVertex3fv(v5)  
    glVertex3fv(v2); glVertex3fv(v6)  
    glVertex3fv(v3); glVertex3fv(v7)  
    glEnd()  
    drawAxes()
```



# GL Widget – initialize & resize

```
class MyGLWidget(QOpenGLWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.base_position = [0.0,0.0]

    def initializeGL(self):
        glClearColor(0.0, 0.0, 0.0, 1.0)
        self.planeList = glGenLists(1)
        glNewList(self.planeList, GL_COMPILE)
        # 그리기 코드
        drawPlane()
        glEndList()

        glEnable(GL_DEPTH_TEST)

    def resizeGL(self, width, height):
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(60, width/height, 0.01, 100)
```

# GL Widget – paint

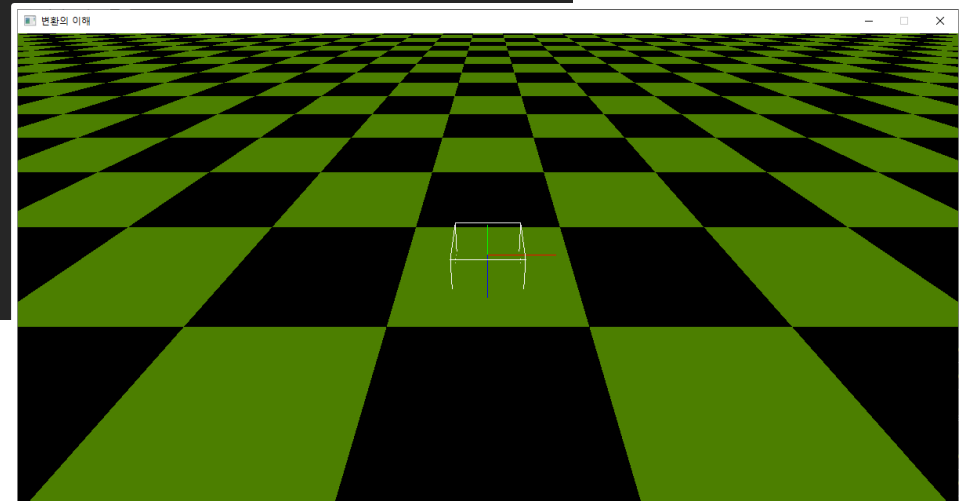
```
def paintGL(self):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    gluLookAt(0,7,10, 0,0,0, 0,1,0)

    glCallList(self.planeList)
    drawAxes()

    ### Base: 전후 좌우로 이동 가능

    # 제어를 통해 옮겨간 위치
    glTranslatef(self.base_position[0], 0, self.base_position[1])

    glTranslatef(0, 0.5, 0) # 몸통을 평면으로 들어올리는 변환
    glPushMatrix()
    glScalef(2, 1, 2) # 몸통의 크기 변경
    drawAxes()
    glColor3f(1,1,1)
    drawCube()
    glPopMatrix()
```

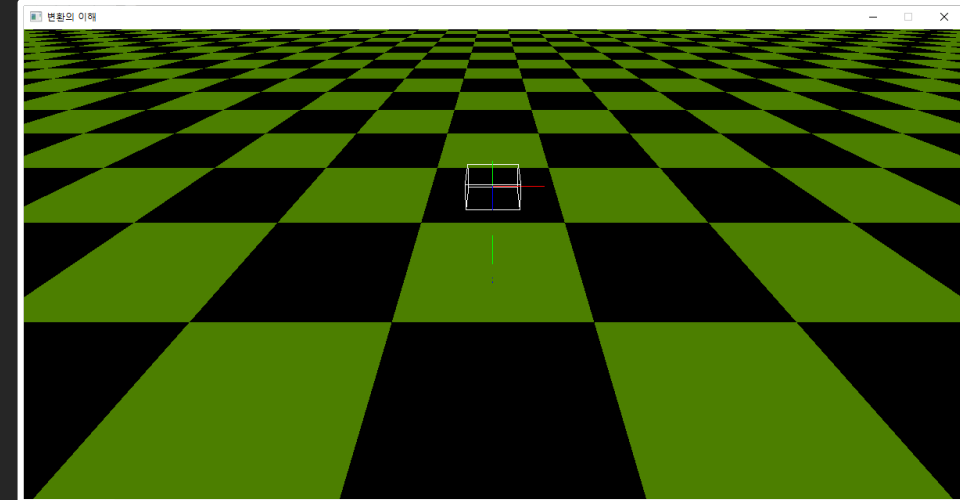


# QT Main Window

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

    def keyPressEvent(self, e):
        step = 0.1
        if e.key() == Qt.Key.Key_W:
            self.glWidget.base_position[1] -= step
        elif e.key() == Qt.Key.Key_S:
            self.glWidget.base_position[1] += step
        elif e.key() == Qt.Key.Key_A:
            self.glWidget.base_position[0] -= step
        elif e.key() == Qt.Key.Key_D:
            self.glWidget.base_position[0] += step

        self.glWidget.update()
```

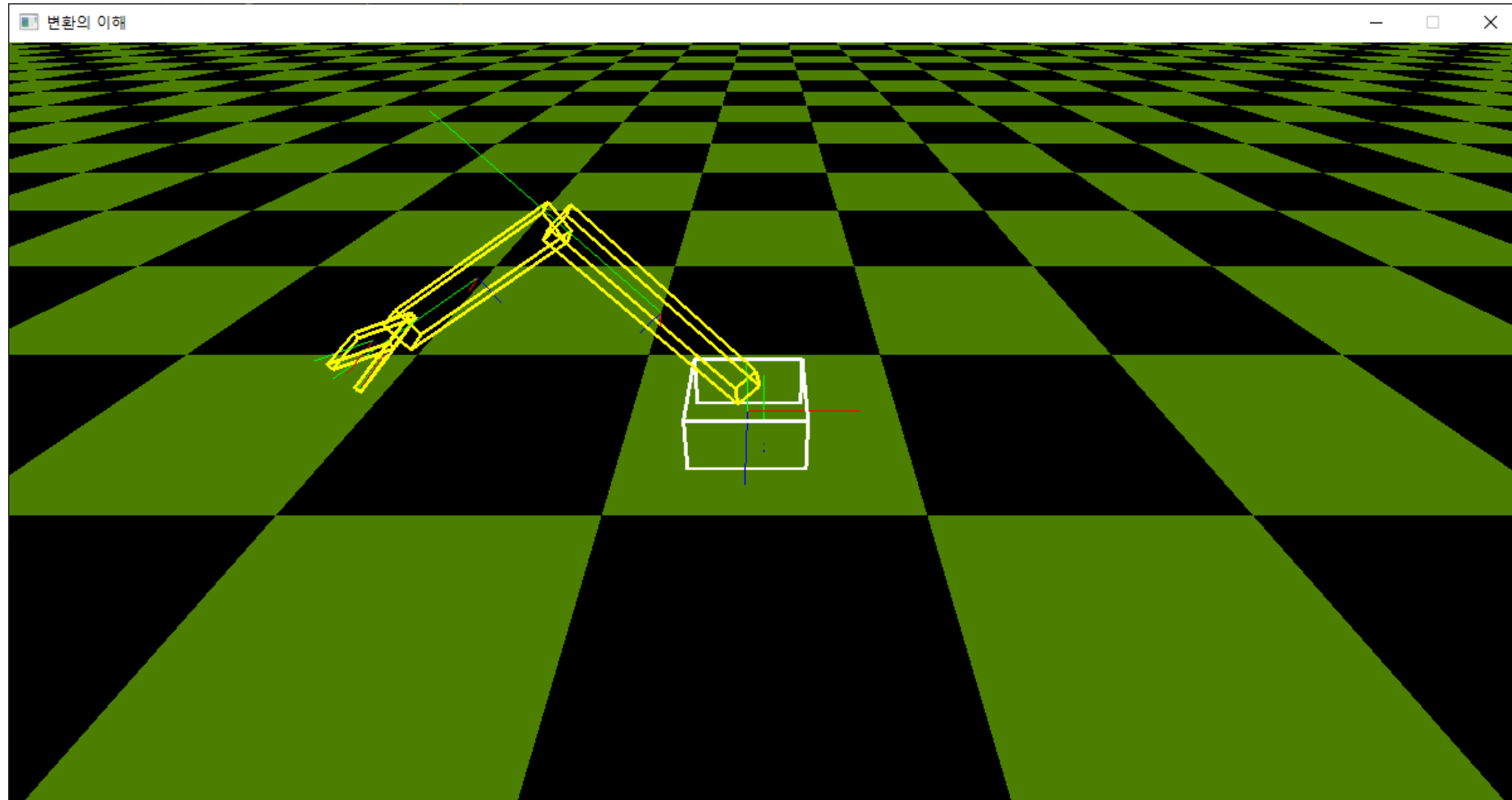


# Application 실행

```
def main(argv = []):  
    app = QApplication(argv)  
    window = MyWindow('변환의 이해')  
    window.setFixedSize(1200, 600)  
    window.show()  
    app.exec()  
  
if __name__ == '__main__':  
    main(sys.argv)
```



# 구조에 따라 다른 큐브도 추가하고 제어



# 추가 객체를 다룰 변수

```
def __init__(self, parent=None):  
    super().__init__(parent)  
    self.base_position = [0.0,0.0]  
    ### 팔 1: y축 기준 회전과, x 축 기준 회전이 가능  
    self.arm1Y = 35  
    self.arm1X = 15  
    self.arm2X = 80  
    self.handAngle = 45
```

# 추가 객체를 다룰 그리기

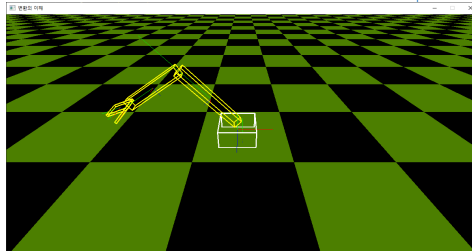
```
def paintGL(self):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    gluLookAt(0,7,10, 0,0,0, 0,1,0)

    glCallList(self.planeList)
    drawAxes()

    ### Base: 전후 좌우로 이동 가능

    # 제어를 통해 옮겨간 위치
    glTranslatef(self.base_position[0], 0, self.base_position[1])
    glTranslatef(0, 0.5, 0) # 몸통을 평면으로 들어올리는 변환
    glPushMatrix()
    glScalef(2, 1, 2) # 몸통의 크기 변경
    drawAxes()
    glColor3f(1,1,1)
    drawCube()
    glPopMatrix()

    ### 팔 1: y축 기준 회전과, x 축 기준 회전이 가능
    glTranslatef(0, 0.5, 0) # 몸통 높이의 반 이동
    glRotatef(self.arm1Y, 0, 1, 0)
    glRotatef(self.arm1X, 1, 0, 0)
    glTranslatef(0, 2, 0) # 팔 1의 높이 반 이동
    glPushMatrix()
    glScalef(0.5, 4, 0.5) # 팔 1의 크기 변경
    drawAxes()
    glColor3f(1,1,0)
    drawCube()
    glPopMatrix()
```



```
### 팔 2: x 축 기준 회전이 가능
glTranslatef(0, 2, 0) # 팔 1 높이의 반 이동
glRotatef(self.arm2X, 1, 0, 0)
glTranslatef(0, 1.5, 0) # 팔 2의 높이 반 이동
glPushMatrix()
glScalef(0.5, 3, 0.5) # 팔 2의 크기 변경
drawAxes()
glColor3f(1,1,0)
drawCube()
glPopMatrix()
```

```
### 손 1: x 축 기준 회전이 가능
glPushMatrix() ##### 손 1 변환 이전 상태 기록
glTranslatef(0, 1.5, 0) # 팔 2 높이의 반 이동
glRotatef(self.handAngle, 1, 0, 0)
glTranslatef(0, 0.5, 0) # 손 1 높이의 반 이동
glPushMatrix()
glScalef(1, 1, 0.1) # 손 1의 크기 변경
drawAxes()
glColor3f(1,1,0)
drawCube()
glPopMatrix()
glPopMatrix() ##### 손 1 변환 무효화
```

```
### 손 2: x 축 기준 회전이 가능
glPushMatrix() ##### 손 2 변환 이전 상태 기록
glTranslatef(0, 1.5, 0) # 팔 2 높이의 반 이동
glRotatef(-self.handAngle, 1, 0, 0)
glTranslatef(0, 0.5, 0) # 손 1 높이의 반 이동
glPushMatrix()
glScalef(1, 1, 0.1) # 손 1의 크기 변경
drawAxes()
glColor3f(1,1,0)
drawCube()
glPopMatrix()
glPopMatrix() ##### 손 2 변환 무효화
```

# 키보드 제어 추가

```
def keyPressEvent(self, e):  
  
    step = 0.1  
    angle_step = 1  
  
    if e.key() == Qt.Key.Key_W:  
        self.glWidget.base_position[1] -= step  
    elif e.key() == Qt.Key.Key_S:  
        self.glWidget.base_position[1] += step  
    elif e.key() == Qt.Key.Key_A:  
        self.glWidget.base_position[0] -= step  
    elif e.key() == Qt.Key.Key_D:  
        self.glWidget.base_position[0] += step  
  
    elif e.key() == Qt.Key.Key_Q:  
        self.glWidget.arm1Y -= angle_step  
    elif e.key() == Qt.Key.Key_E:  
        self.glWidget.arm1Y += angle_step  
    elif e.key() == Qt.Key.Key_1:  
        self.glWidget.arm1X -= angle_step  
    elif e.key() == Qt.Key.Key_2:  
        self.glWidget.arm1X += angle_step  
    elif e.key() == Qt.Key.Key_3:  
        self.glWidget.arm2X -= angle_step  
    elif e.key() == Qt.Key.Key_4:  
        self.glWidget.arm2X += angle_step  
    elif e.key() == Qt.Key.Key_Z:  
        self.glWidget.handAngle -= angle_step  
    elif e.key() == Qt.Key.Key_X:  
        self.glWidget.handAngle += angle_step  
  
    self.glWidget.update()
```