

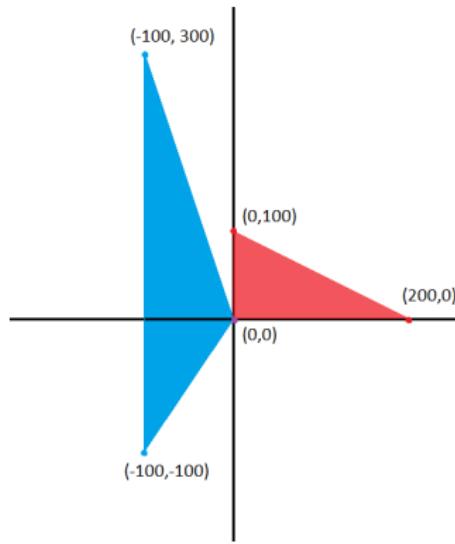
그래픽스 강의노트 04 - OpenGL 변환

강영민

동명대학교

OpenGL의 변환

- 변환을 자유자재로 할 수 있다고 가정: $T(\mathbf{p}, \vec{d})$, 크기를 변경 $S(\mathbf{p}, s)$
- 필요한 것은 반지름 1의 원 그리기 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



대표적인 어파인 변환

이동(translate)	주어진 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.

동차좌표계

- 동차 좌표(homogeneous coordinate)은 n 차원의 사영공간을 $n + 1$ 차원의 좌표로 나타내는 좌표계
- 1827년 아우구스트 페르디난드 뮌비우스(August Ferdinand Möbius)가 그의 저작 “Der barycentrische Calül”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 3×3 행렬의 곱으로 표현
- 이동과 회전이 누적되면 벡터 덧셈과 행렬 곱셈이 연속적 적용됨
- 동차좌표(homogeneous coordinate)을 사용하면 이동과 회전 모두 4×4 행렬의 곱으로 표현 가능
- 누적된 이동, 회전 변환을 하나의 행렬로 표현 가능

변환 행렬

- 동차 좌표계에서 어파인 변환은 행렬로 표현
- 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

glTranslatef(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 M이라고 하고, glTranslate의 이동을 표현하는 행렬이 T
- glTranslate 함수가 불렸을 때, M을 MT로 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	$\text{CTM} := \mathbf{I}$
LoadMatrix(M)	$\text{CTM} := M$
T: Translate*	$\text{CTM} := \text{CTM} * T$
R: Rotate*	$\text{CTM} := \text{CTM} * R$
S: Scale*	$\text{CTM} := \text{CTM} * S$

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	$\text{CTM} := \mathbf{I}$
LoadMatrix(M)	$\text{CTM} := M$
T: Translate*	$\text{CTM} := \text{CTM} * T$
R: Rotate*	$\text{CTM} := \text{CTM} * R$
S: Scale*	$\text{CTM} := \text{CTM} * S$

변환의 적용에 따른 CTM의 변화

- 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();
```

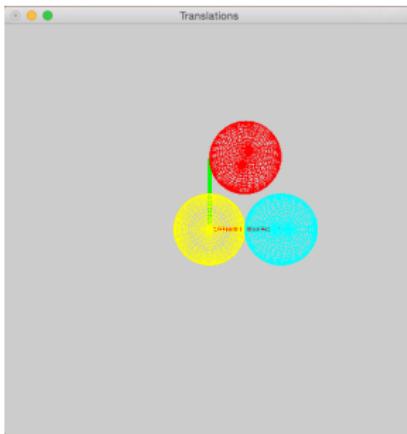
코드 구현 내용

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. 다시 말해 우리가 호출한 변환의 역순으로 변환이 객체에 적용된다.

구(sphere) 그리기에 변환 적용 예제

```
void draw() {
    glutWireSphere(0.5, 30, 30); // 반지름 0.5의 구를 원점을 중심으로 그림
}
void 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();
}
```

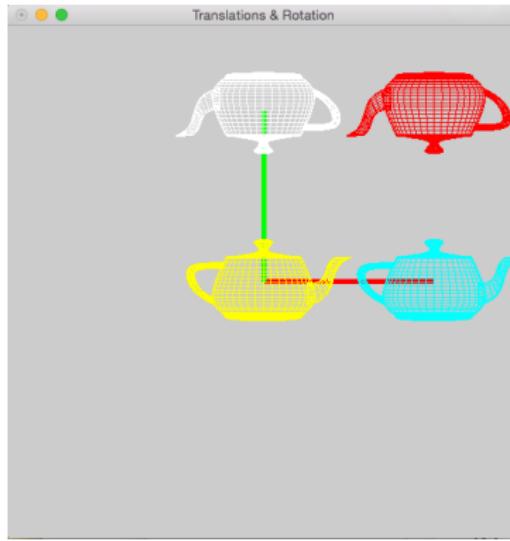
구(sphere) 그리기에 변환 적용 예제 실행 결과



- 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

복합변환의 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 해야할지를 고민해 보자.



CTM 모델로 원하는 변환 구현해 보기

- 노란색 주전자는 아무런 변환도 적용되지 않은 상태. 바로 그림

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

- 하늘색 주전자는 x 축으로 1만큼 이동하여 그려야 하므로 `glTranslate*` 함수를 부른 뒤 주전자를 그린다. 이 이동 변환을 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 T_1 이므로 (0,1,0)만큼의 이동 T_2 를 추가 적용하면 (1,1,0) 이동이 되며, 여기에 회전 R 을 적용

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

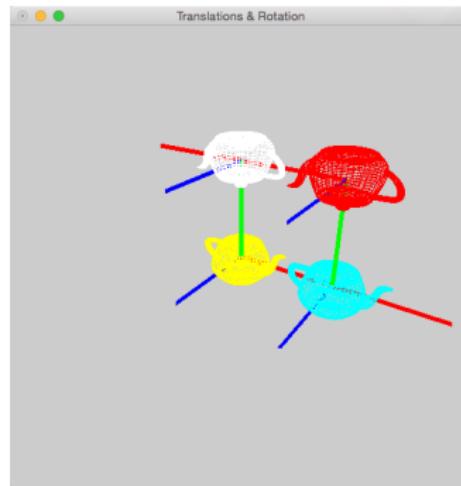
CTM 모델로 원하는 변환 구현해 보기

- 붉은 색 주전자는 $T_1 T_2 R$ 의 변환
- 흰색 주전자를 그리는 방법은 까다롭다
- 이 주전자는 180도 회전 시킨 주전자를 $(0,1,0)$ 만큼 이동한 것
- 현재의 변환 $T_1 T_2 R$ 에 어떤 변환을 추가적으로 호출하면 이것이 가능할지 생각해야 함
- 회전이 호출된 상태라 까다롭다.
- 새롭게 적용할 변환이 X 라고 하고, $(0,1,0)$ 만큼 이동하는 변환을 T 라고 하면 다음과 같은 관계 성립
 - $T_1 T_2 R X = T R$
 - $X = R^T T_2^{-1} T_1^{-1} T R$
- 위의 계산을 수행하면 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);
```

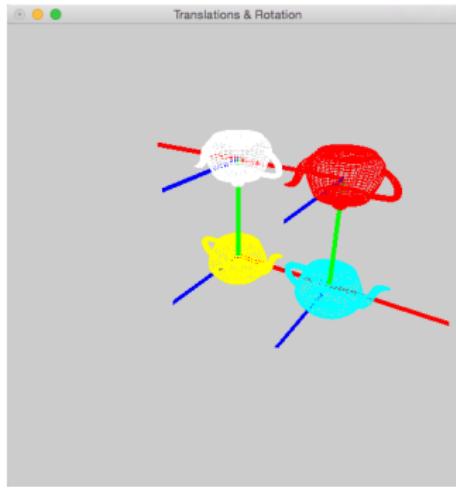
지역 좌표계 변환을 중심으로 생각하기 1/6

- 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);
```

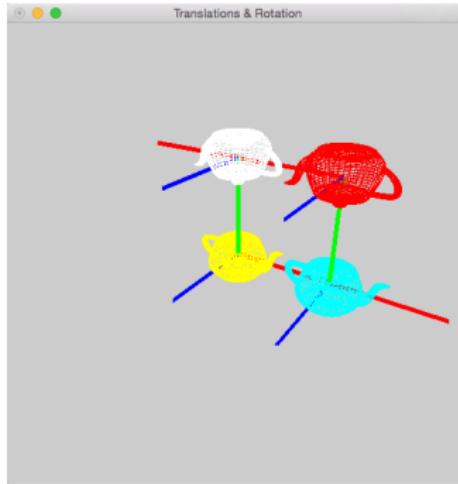
지역 좌표계 변환을 중심으로 생각하기 2/6



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);
```

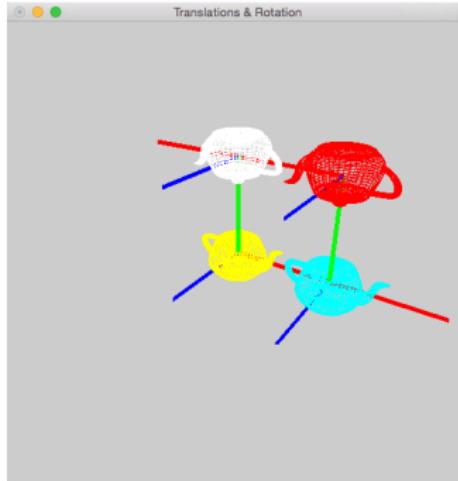
지역 좌표계 변환을 중심으로 생각하기 3/6



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);
```

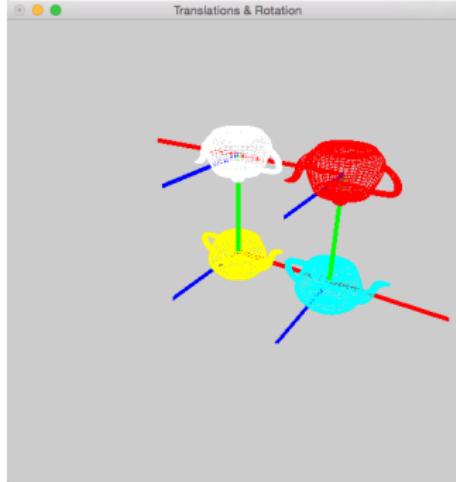
지역 좌표계 변환을 중심으로 생각하기 4/6



붉은 주전자의 변환은 순서를 바꾸어 하늘색 주전자에서 z 축 회전을 먼저 적용한 뒤에 y 축 기준으로 -1만큼 이동하는 것과도 같다. 따라서 다음과 같이 적용해도 동일한 결과

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

지역 좌표계 변환을 중심으로 생각하기 5/6



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

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

지역 좌표계 변환을 중심으로 생각하기 6/6

```
void drawLocalAxes() { drawAxes(); }
void draw() { glutWireTeapot(0.3);}

void display() {
    [[카메라 설정과 버퍼 지우기 등의 코드]]

    drawLocalAxes();
    glColor4f(1, 1, 0, 0.25);
    draw();

    glTranslatef(1.0, 0.0, 0.0);
    drawLocalAxes();
    glColor4f(0, 1, 1, 0.25);
    draw();

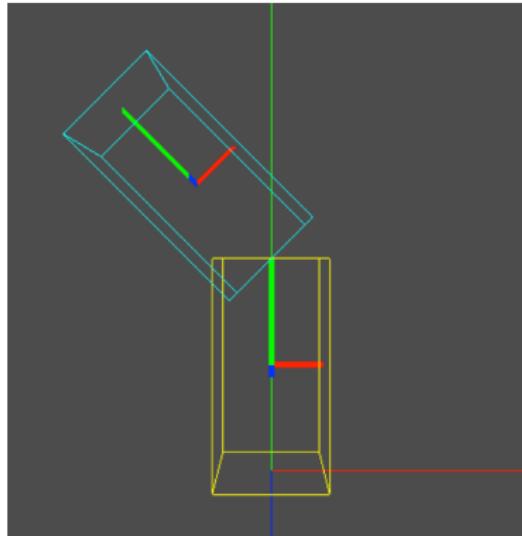
    glRotatef(180, 0.0, 0.0, 1.0);
    glTranslatef(0.0, -1.0, 0.0);
    drawLocalAxes();
    glColor4f(1, 0, 0, 0.25);
    draw();

    glTranslatef(1.0, 0.0, 0.0);
    drawLocalAxes();
    glColor4f(1, 1, 1, 0.25);
    draw();

    glutSwapBuffers();
}
```

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 놓이려면 y 축 방향으로 0.5 이동
 - 이미 지역좌표계가 glScalef에 의해 변환되었기 때문에 0.5만 들어올려도 전역좌표계에서는 1만큼 올라가게 된다.
- 노란색 육면체는 다음과 같이 그림

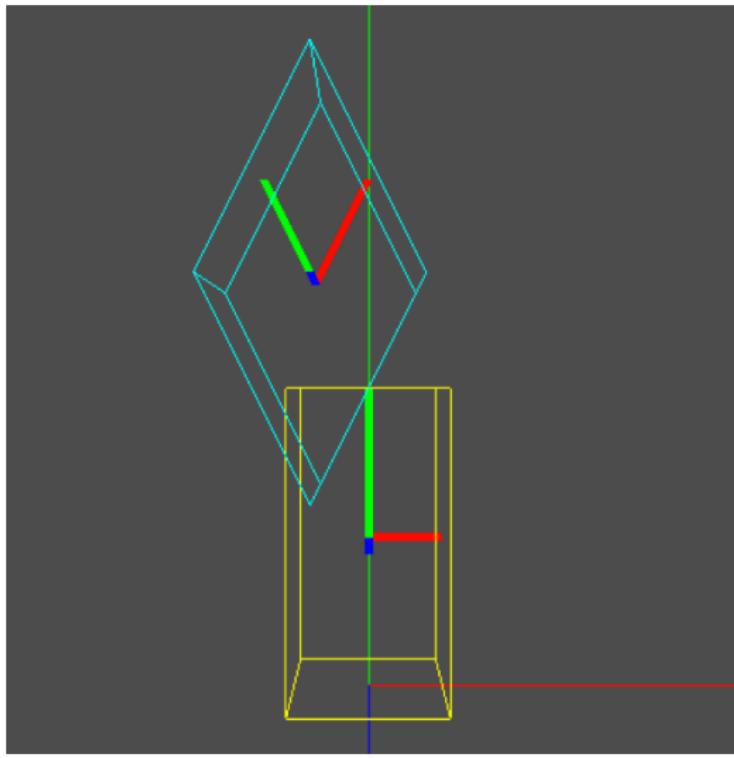
```
glScalef(1.0, 2.0, 1.0); // S  
glTranslatef(0.0, 0.5, 0.0); // T1  
draw(yellow);
```

- 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

결과는...



크기 변경이 포함된 경우

- 실제로 하늘색 육면체에 적용된 변환
- **ST₁T₂RT₃**
- 가장 마지막에 적용되는 **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

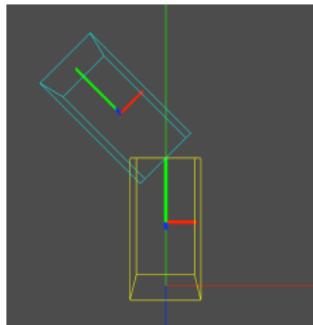
함수명	역할
<code>glPushMatrix</code>	현재의 CTM을 스택에 저장한다.
<code>glPopMatrix</code>	행렬 스택을 pop하여 CTM을 갱신한다.

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

크기 변경이 포함된 경우

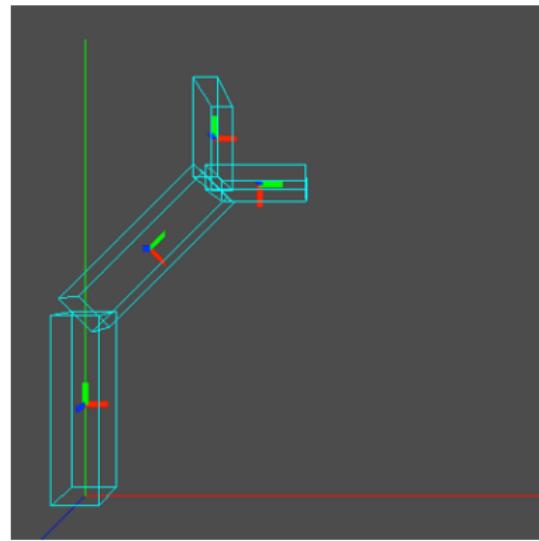
```
glTranslatef(0.0, 1.0, 0.0);
glPushMatrix();
glScalef(1.0, 2.0, 1.0);
draw(yellow);
glPopMatrix();

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();
```



계층적 구조

- 컴퓨터 그래픽스로 표현하는 객체: 구조를 가짐
- 많은 경우 이 구조는 인간이나 로봇과 같은 구조체에서 볼 수 있는 바와 같이 계층적(hierarchical)



계층적 모델링(hierarchical modeling) 1/2

- 계층적 모델링은 앞서 살펴본 glPushMatrix와 glPopMatrix를 활용하여 구현
- 팔 구성 요소는 길이가 4이며, 손을 구성하는 요소는 길이가 2라고 가정
- 팔 구성요소를 그리는 함수를 drawArm()이라 하자.
- 손을 그리는 함수는 drawHand()
- 루트(root) 객체인 가장 아래쪽 육면체는 팔 요소의 길이 4를 고려하여 그 반에 해당하는 2만큼 위로 올림
 - glTranslatef(0.0, 2.0, 0.0);
 - drawArm();
- 다음으로 45° 로 누워 있는 팔은 부모 노드의 변환을 반영해야 하므로 앞서 적용된 변환을 유지한 상태로 팔의 길이의 반만큼 올라간 뒤, 45° 로 회전하고, 나머지 반을 더 올라가서 끝을 맞춤
 - glTranslatef(0.0, 2.0, 0.0);
 - glRotatef(-45, 0.0, 0.0, 1.0);
 - glTranslatef(0.0, 2.0, 0.0);
 - drawArm();

계층적 모델링(hierarchical modeling) 2/2

- 손의 중심을 팔의 끝에 두기 위해 팔의 길이 반에 해당하는 2만큼 이동시킨뒤 45° 회전하고, 손의 끝점을 관절부에 맞추기 위해 손의 길이 2의 반인 1만큼 이동하여 그림
- 두 번째 손을 그리려면 이 변환은 무효화되어야 함
- glPushMatrix를 이용하여 현재의 변환을 기록했다가 다시 복원
 - glPushMatrix();
 - glTranslatef(0.0, 2.0, 0.0);
 - glRotatef(-45, 0.0, 0.0, 1.0);
 - glTranslatef(0.0, 1.0, 0.0);
 - glPopMatrix();
- 다음으로 마지막 손 요소를 그림
 - glTranslatef(0.0, 2.0, 0.0);
 - glRotatef(45, 0.0, 0.0, 1.0);
 - glTranslatef(0.0, 1.0, 0.0);

로봇 팔 그리기 예제

Lines 1–25 / 50

```
void drawArm() { // 로봇 팔 그리기 코드. gl[Push|Pop]Matrix로 외부에 영향을 미치지 않음
    glPushMatrix();
    glScalef(1.0, 4.0, 1.0);
    glutWireCube(1.0);
    glPopMatrix();
}
void drawHand() { // 로봇 손 그리기 코드. gl[Push|Pop]Matrix로 외부에 영향을 미치지 않음
    glPushMatrix();
    glScalef(1.0, 2.0, 1.0);
    glutWireCube(1.0);
    glPopMatrix();
}
void display() {
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0, 0, 15, 0, 0, 0, 0, 1, 0); // 카메라의 위치를 설정하는 작업
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // 루트(root) 노드가 되는 팔을 하나 그리는 코드
    glTranslated(0.0, 2.0, 0.0);
    drawAxes();
    drawArm();
    // 루트에 달린 두 번째 팔을 그리는 코드
    glTranslated(0.0, 2.0, 0.0);
```

로봇 팔 그리기 예제

Lines 26–50 / 50

```
glRotatef(-45, 0.0, 0.0, 1.0);
glTranslated(0.0, 2.0, 0.0);
drawAxes();
drawArm();
// 손을 그리기. 이때 사용한 변환이 다른 손에 영향을 미치지 않도록 gl[Push|Pop]Matrix 사용
glPushMatrix();
glTranslated(0.0, 2.0, 0.0);
glRotatef(-45, 0.0, 0.0, 1.0);
glTranslated(0.0, 1.0, 0.0);
drawAxes();
drawHand();
glPopMatrix();
// 위의 glPushMatrix() 의해 현재 변환은 두 번째 팔을 그리고 난 상태와 동일함
glTranslated(0.0, 2.0, 0.0);
glRotatef(45, 0.0, 0.0, 1.0);
glTranslated(0.0, 1.0, 0.0);
drawAxes();
drawHand();

glutSwapBuffers();
}
```

가상의 태양계 그리기

Lines 1–25 / 75

```
void display() {
    static float t=10;      t+=1.0;
    [[카레라 설정, 버퍼 지우기, 중심 축 그리기 등을 수행]]

    draw(1, 1.0, 0.5, 0.0); // 태양

    glPushMatrix();
    glRotated(t, 0.0, 1.0, 0.0);
    glTranslated(3.0, 0.0, 0.0);
    draw(0.2, 0.5, 0.8, 1.0); // 수성
    glPopMatrix(); // 수성 영향권 끝

    glPushMatrix();
    glRotated(t*1.5, 0.0, 1.0, 0.0);
    glTranslated(5.0, 0.0, 0.0);
    draw(0.2, 1.0, 1.0, 0.0); // 금성
    glPopMatrix(); // 금성 영향권 끝

    glPushMatrix();
    glRotated(t*0.8, 0.0, 1.0, 0.0);
    glTranslated(7.0, 0.0, 0.0);
    draw(0.2, 0.2, 0.2, 1.0); // 지구
    glPushMatrix();
    glRotated(t*10.0, 0.0, 1.0, 0.0);
    glTranslated(0.5, 0.0, 0.0);
```

가상의 태양계 그리기

Lines 26–50 / 75

```
draw(0.05, 1.0, 1.0, 1.0); // 지구의 영향권 내에 달을 그림
glPopMatrix(); // 달의 영향권 끝
glPopMatrix(); // 지구의 영향권 끝

glPushMatrix();
glRotated(t*0.5, 0.0, 1.0, 0.0);
glTranslated(10.0, 0.0, 0.0);
draw(0.25, 1.0, 0.5, 0.4); // 화성
glPopMatrix(); // 화성 영향권 끝

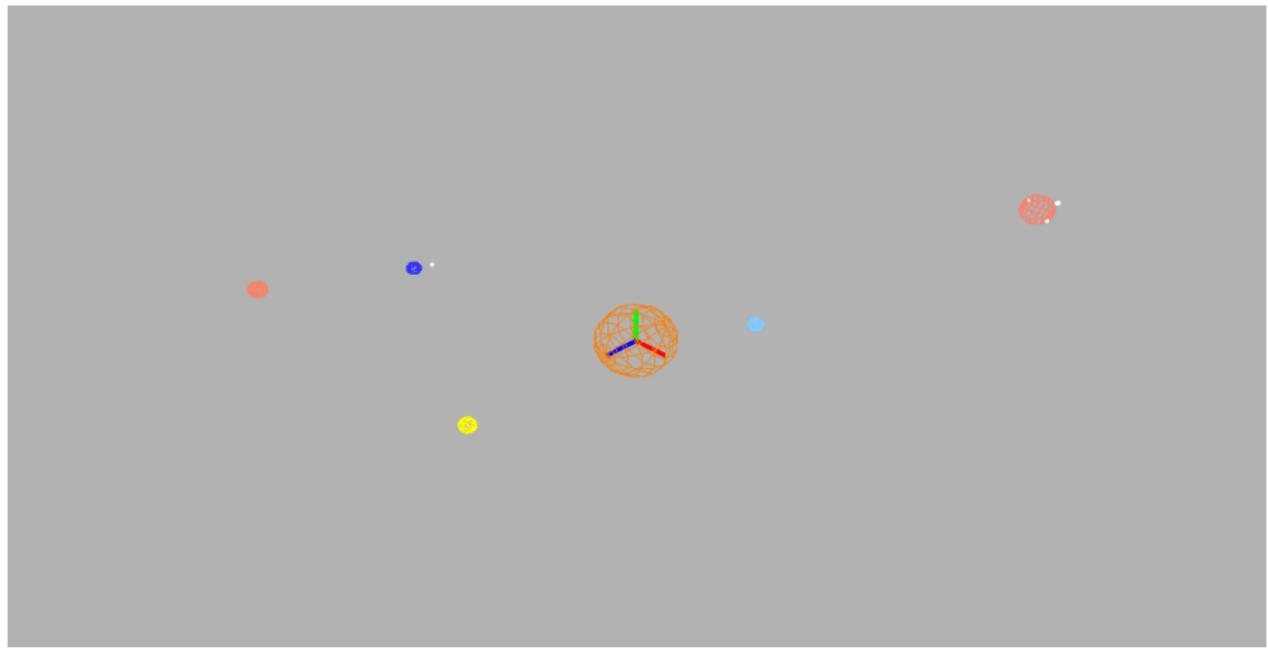
glPushMatrix(); // 목성 영향권의 시작
glRotated(t*0.7, 0.0, 1.0, 0.0);
glTranslated(14.0, 0.0, 0.0);
draw(0.5, 1.0, 0.5, 0.4); // 목성
glPushMatrix();
glRotated(t*10.0, 0.0, 1.0, 0.0);
glTranslated(0.8, 0.0, 0.0);
draw(0.05, 1.0, 1.0, 1.0); // 목성 위성 1번
glPopMatrix(); // 목성 위성 1번의 영향권 끝
glPushMatrix();
glRotated(t*12.0, 0.0, 1.0, 0.0);
glTranslated(0.9, 0.0, 0.0);
draw(0.07, 1.0, 1.0, 1.0); // 목성 위성 2번
glPopMatrix(); // 목성 위성 2번의 영향권 끝
glPushMatrix();
```

가상의 태양계 그리기

Lines 51–75 / 75

```
glRotated( t * 13.0, 0.0, 1.0, 0.0 );
glTranslated( 0.7, 0.0, 0.0 );
draw(0.03, 1.0, 1.0, 1.0); // 목성 위성 3번
glPopMatrix(); // 목성 위성 2번의 영향권 끝
glPopMatrix(); // // 목성의 영향권 끝
glutSwapBuffers();
}
```

가상의 태양계 그리기



궤도 그려 넣기

- 각 천체들이 움직이는 궁전궤도를 표현하고 싶다면
- 궁전 궤도를 표현할 drawCircle(radius) 함수를 다음과 같이 구현

```
void drawCircle( float radius ) {  
    glColor3f( 1.0 , 1.0 , 1.0 );  
    glBegin(GLLINE_LOOP);  
    for ( int i=0; i<50; i++ ) {  
        glVertex3f(  
            radius*cos( 6.28*float(i)/50.0 ),  
            0.0,  
            radius*sin( 6.28*float(i)/50.0 ));  
    }  
    glEnd();  
}
```

- 이제 o] drawCircle 함수를 어디에서 부를 것인지를 결정

궤도 그려 넣기

Lines 1–25 / 50

```
static float t=10;      t+=1.0;  
[[카메라 설정, 버퍼 지우기, 중심 축 그리기 등을 수행]]
```

```
draw(1, 1.0, 0.5, 0.0); // 태양  
drawCircle(3.0); // 수성 궤도 그리기  
drawCircle(5.0); // 금성 궤도 그리기  
drawCircle(7.0); // 지구 궤도 그리기  
drawCircle(10.0); // 화성 궤도 그리기  
drawCircle(14.0); // 목성 궤도 그리기
```

```
glPushMatrix();  
[[수성 변환과 그리기]]  
glPopMatrix(); // 수성 영향권 끝
```

```
glPushMatrix();  
[[금성 변환과 그리기]]  
glPopMatrix(); // 금성 영향권 끝
```

```
glPushMatrix();  
[[지구 변환과 그리기]]  
drawCircle(0.5); // 달의 궤도 그리기  
glPushMatrix();  
[[달의 변환과 그리기]]  
glPopMatrix(); // 달의 영향권 끝  
glPopMatrix(); // 지구의 영향권 끝
```

궤도 그려 넣기

Lines 26–50 / 50

```
glPushMatrix();  
    [[화성 변환과 그리기]]  
    glPopMatrix(); // 화성 영향권 끝  
  
glPushMatrix(); // 목성 영향권의 시작  
    [[목성 변환과 그리기]]  
    drawCircle(0.7); // 목성 위성 1의 궤도 그리기  
    drawCircle(0.9); // 목성 위성 2의 궤도 그리기  
    drawCircle(1.1); // 목성 위성 3의 궤도 그리기  
    glPushMatrix();  
        [[목성 위성 1의 변환과 그리기]]  
    glPopMatrix(); // 목성 위성 1번의 영향권 끝  
    glPushMatrix();  
        [[목성 위성 2의 변환과 그리기]]  
    glPopMatrix(); // 목성 위성 2번의 영향권 끝  
    glPushMatrix();  
        [[목성 위성 3의 변환과 그리기]]  
    glPopMatrix(); // 목성 위성 2번의 영향권 끝  
    glPopMatrix(); // // 목성의 영향권 끝
```

궤도 그려 넣기

