OpenGL 6

가상현실론

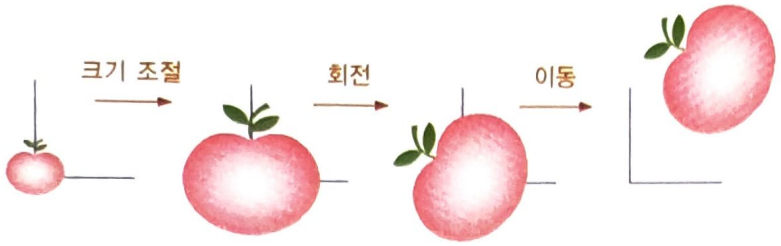
2021/03/18

목차

- 복습
  - 복합변환
  - 모델 좌표계에서 전역 좌표계로 변환
  - 모델변환
- 복합변환에 의한 모델링
- 행렬스택 (Matrix Stack)

복습

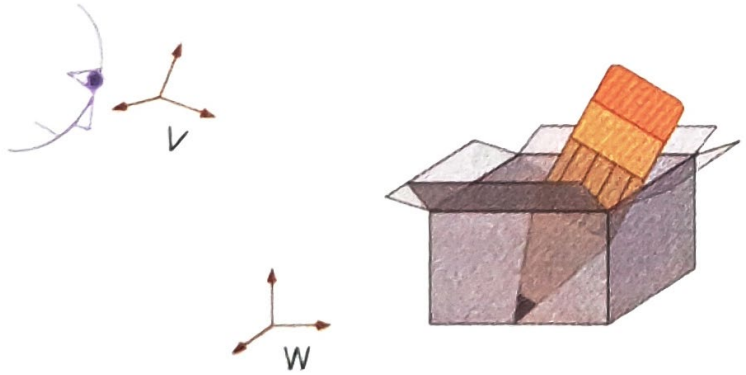
- 복합변환



[그림 6-31] 크기 조절, 회전, 이동

- 변환된 물체 하나를 원래 물체의 인스턴스(instance)라고 함
- 물체 인스턴스는 크기 조절, 회전 이동 순으로 변환을 가함
- 크기 조절에이나 회전을 먼저 가하는 이유는 이 상태에서 물체 중심이 피벗과 원점이 일치한 상태이므로 이를 일치시키기 위한 변환이 불필요하기 때문.
- C=T·R·S

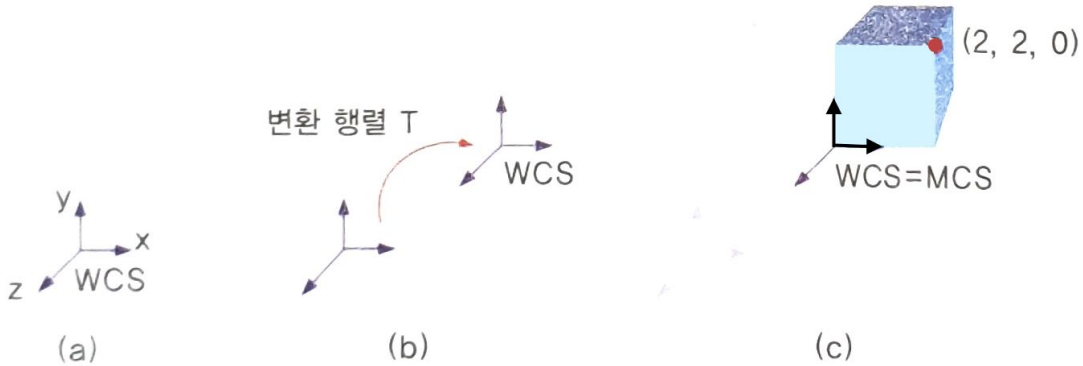
복습



- GL의 모델 변환
  - 연필과 박스가 각각 M1, M2의 지역좌표계를 가지고 있기 때문에 모든 물체를 한꺼번에 아우를 수 있는 기준 좌표계가 필요. 이를 전역 좌표계 (WCS: World Coordinate System)이라고 함.
  - 그림에서는 W를 전역좌표계로 설정. 전역 좌표계가 설정되면 각 모델 좌표계를 기준으로 표시된 물체 좌표가 전역 좌표계를 기준으로 바뀌어 물체 간의 상대적인 위치가 명확해짐.
  - 화면에 보이는 것은 시점(Viewpoint)가 좌우함. 사용자의 시점을 기준으로하는 시점 좌표계(VCS: View Coordinate System)으로 WCS가 다시 변환됨 (그림에서 V).

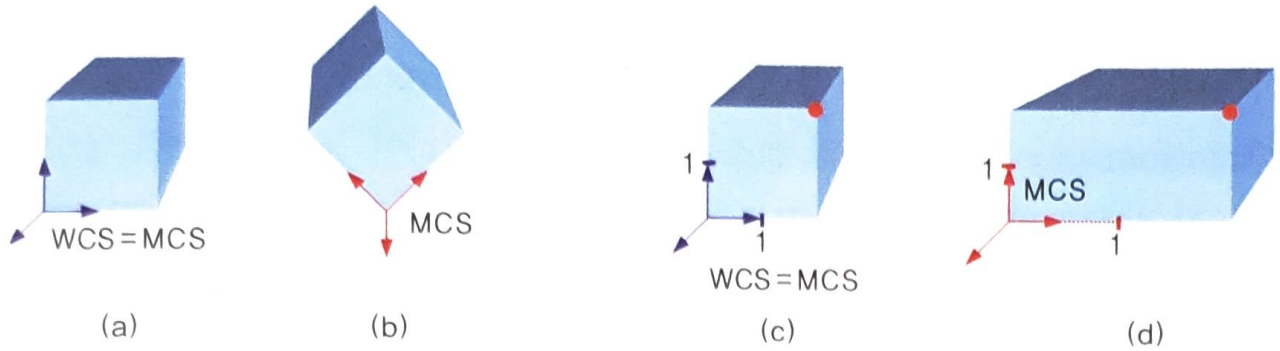
복습-모델 좌표계에서 전역 좌표계로 변환

$$P' = TP = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 2 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \\ 0 \\ 1 \end{bmatrix}$$



- GL의 입장에서는 물체의 이동보다 '좌표계의 이동'으로 바라봄. 따라서 전역좌표계를 (3, 2, 0)만큼 이동시 GL은 "전역 좌표계를 모델 좌표계로 일치시키기 위한 것이 변환 행렬임 " 으로 인식.
- 그림의 예에서 GL은 (a)의 전역좌표계에 변환 행렬 T를 가하여 (b)로 모델 좌표계로 일치시킴. 모델 좌표계에서 정점을 그려내면 변환된 위치에 육면체가 그리쳔다는 것.
- 따라서 GL 입장에서 볼 때 물체에 변환을 가한다는 것은 결국 전역 좌표계를 모델 좌표계와 일치시키기 위해서 변환을 가한다는 것.

복습-모델 좌표계에서 전역 좌표계로 변환



- (b): (a)의 물체를 원점 중심으로 45도 회전. 모델 좌표계 기준 물체 좌표계는 변함이 없음
- (c)의 우상단 점의 좌표는 (2, 2, 0). WCS=MCS 상태이기 때문에 모델좌표계와 전역 좌표가 서로 일치
- (d)처럼 변형이 일어날 경우 x축의 길이에서 WCS와 MCS에서 차이가 생김. MCS에서는 (2, 2, 0)인 좌상단 점이 WCS에서는 (4, 2, 0)이 됨.
- GL 입장에서 보았을 때 물체에 가해지는 모든 변환을 전역 좌표계를 모델좌표계와 일치시키기 위한 변환. 물체 이동은 원점의 이동, 물체 회전은 축 방향의 회전, 쿨체 크기 조절은 축 눈금 길이의 조절. 각각의 변환을 위한 변환행렬은 전역 좌표계를 변화된 모델 좌표계로 일치시키기 위한 행렬로 사용.

복습-모델변환

- void glMatrixMode(GLenum mode);
  - 행렬모드를 설정하는 함수 프로토타입
  - mode에 들어가는 상수는 GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE 등임
- GL에서는 상태 변수를 사용하기 때문에 현재의 상태 변수 값이 중요함. 모든 파이프라인 프로세스가 현 상태 변수를 기준으로 하기 때문. 현 변환 행렬 (CTM: Current Transformation Matrix)이 중요
- void glLoadIdentity();
  - 현 변환행렬을 항등 행렬(identity matrix)로 초기화 함.
  - $CTM = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

복습-모델변환

- 변환 후의 정점 P’와 변환 전의 정점 P의 관계
  - $P' = CTM \cdot P$
  - 프로그램이 시작할 때는 CTM=I이므로 기본적으로 모델 좌표계=전역 좌표계= 시점좌표계
- void glLoadMatrixf(const GLfloat *M);
  - 배열 M에 있는 행렬 요소 값을 현 변환 행렬로 올림.
  - GL은 모든 행렬이 열 우선 순위(Column Major Order)로 저장됨에 주의
  - $M = \begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix}$
  - 즉, 위 행렬을 배열에 저장할 때는 GLfloat M[16]={a, b, c,..., p}의 순서로 저장해야 함.

모델 변환

- 모든 변환은 전역 좌표계를 기준으로 하는 모델 좌표계의 변환
- 현 변환 행렬의 오른쪽에 해당 변환 행렬이 곱해짐.
  - Ex) 현 변환 행렬이 identity matrix로 초기화된 직후 glTranslatef(1, 2, 0)이 호출
  - $\mathrm{CMT} = \mathrm{CMT} \cdot \mathrm{T} = \mathrm{I} \cdot \mathrm{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
  - 기준 물체의 정점좌표 P를 위의 식 우변에 곱하면 이동 변환의 결과 전역 좌표 P‘가 나오게 됨
  - $\mathrm{P}' = \mathrm{I} \cdot \mathrm{T} \cdot \mathrm{P} = \mathrm{T} \cdot \mathrm{P}$

복합 변환에 의한 모델링

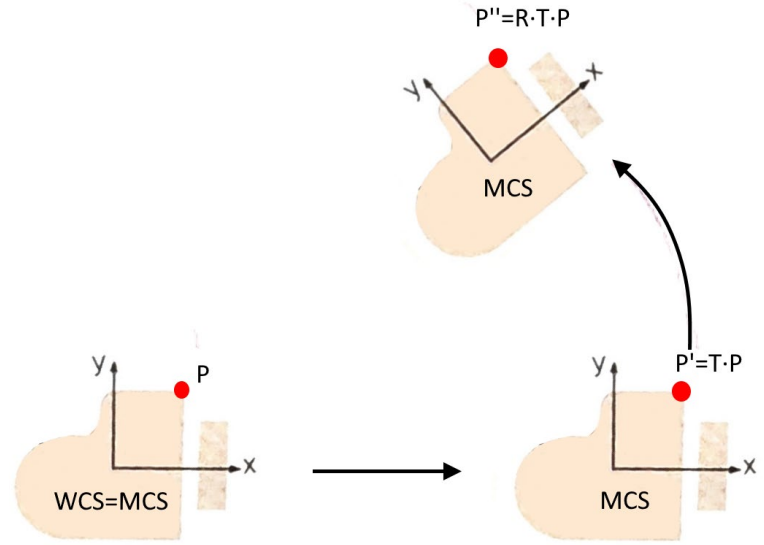
- example)

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(sx, sy, sz);
glRotatef(theta, vx, vy, vz);
glBegin(GL_POINTS);
  glVertexef(px, py, pz);
glEnd();
```

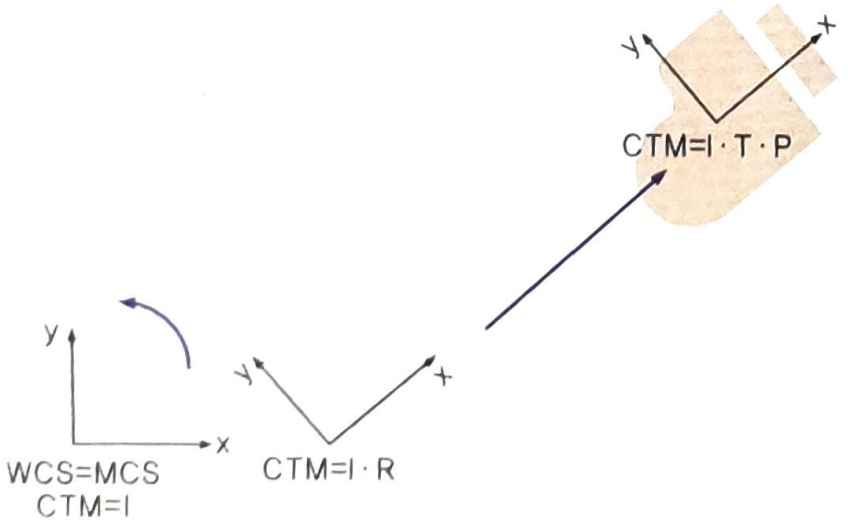
함수 종류	GL 함수	matrix computation
matrix initialization	glLoadIdenty();	CTM = I
scale	glScalef();	CTM = CTM·S=I·S
rotation	glRotatef();	CTM=CTM·R=I·S·R
point declaration	glVertex3f();	P'=CTM·P=I·S·R·P

- GL의 정점좌표는 열행렬(column)으로 표시됨. 전위 곱셈(pre-multiplication) 규칙으로 계산됨에 유의
- 연산순서와 코드 나열 순서와는 반대인 것에 유의.

복합 변환에 의한 모델링

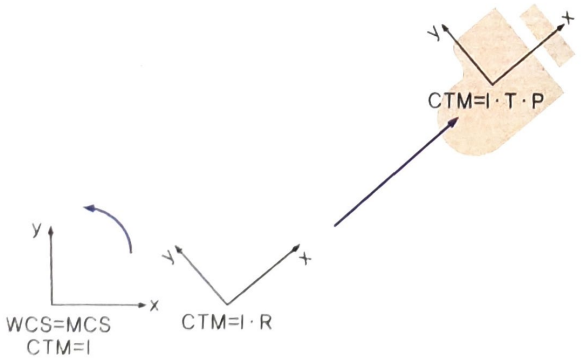


- 전역좌표계 기준으로 물체 이동
- Pre-multiplication
- 물체 자체를 이동시킨 후 회전
  - P’=T·P
  - P’’=R·P’=R·T·P



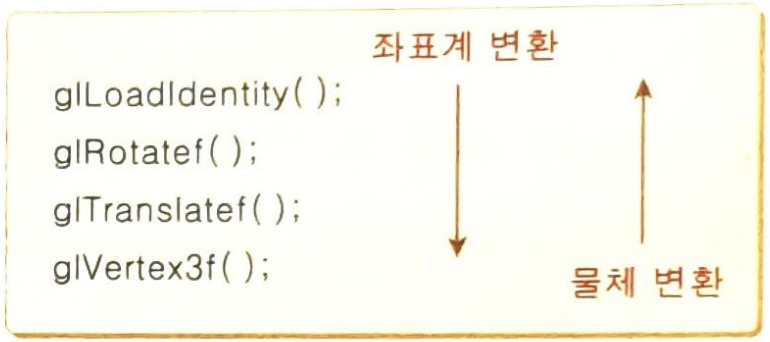
- 모델 좌표계 자체를 이동
- Post-mulitiplication
  - CTM=I·R
  - CTM= I· R·T
  - P’’=I· R·T·P= R·T·P

복합 변환에 의한 모델링



- 변환행렬로 표현
  - glMatrixMode(GL_MODELVIEW);
  - glLoadIdentity();
  - glRotatef(45, 0.0, 0.0, 1.0);
  - glTranslatef(10.0, 0.0, 0.0);
  - glVertex3f(Px, Py, Pz);

복합 변화에 의한 모델링



- 물체변환 (Object Transformation)
  - 고정된 전역 좌표계를 기준으로 물체를 연속적으로 움직이는 방법. 물체가 전역 좌표계에 원점에 이미 그려져 있다고 가정하고 전역 좌표계 기준으로 물체를 이동, 회전, 크기조절하여 원하는 위치와 방향으로 이동
- 좌표계 변환 (Coordinate Trnaformation)
  - 모델 좌표계를 연속적으로 움직이는 방법. 물체가 아직 그려지지 않았다고 가정하고 대신 모델 좌표계를 연속적으로 움직여 원하는 위치와 방향으로 가져다 놓은 다음, 최종적인 모델 좌표계를 기준으로 물체를 그림. GL의 명령어 순서는 이 방법을 기준으로 함.
  - 일반적으로는 물체에 대한 변환으로 파악하는 것이 수월함.

예제

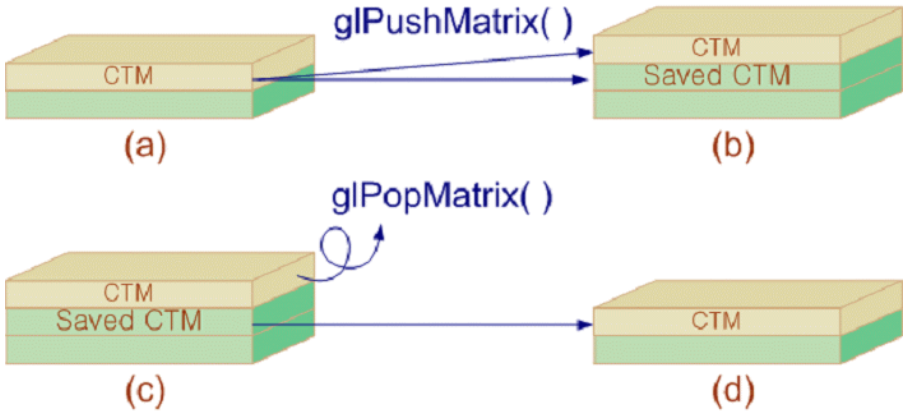
```
void display();
void myInit();
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGB);
   glutInitWindowSize(300, 300);
   glutInitWindowPosition(300, 300);
   glutCreateWindow("OpenGL sample drawing");
   glutDisplayFunc(display);
   glutKeyboardFunc(keyboard);
   myInit();
   glutMainLoop();
   return 0;
}
void myInit()
{
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
```

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glViewport(0, 0, 300, 300);
   glColor3f(1.0, 0.0, 0.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glRotatef(45.0, 0.0, 0.0, 1.0);
   glTranslatef(0.6, 0.0, 0.0);
   glutSolidCube(0.3);
   glFlush();
}
```

행렬 스택 (Matrix Stack)

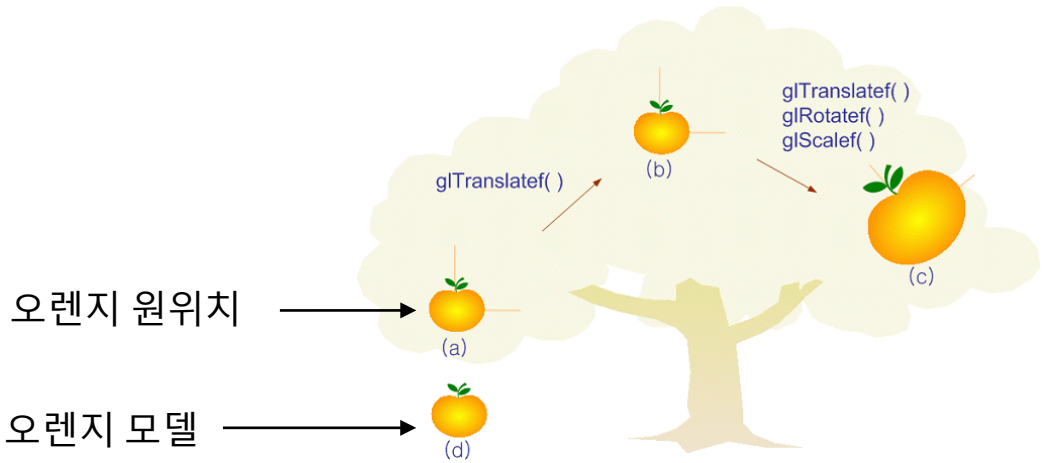
- GL은 행렬 스택을 제공. 이를 사용하여 좌표계가 변화한 과정을 파악할 수 있음.
- 스택은 LIFO (Last-In-First-Out) 방식의 자료 구조
- 모델 뷰 행렬 스택의 깊이는 최소 32개, 투상 행렬 스택의 깊이는 최소 2개. 행렬 스택을 조작하기 위한 함수는
  - void glPushMatrix();
    - 현 변환행렬(CTM)을 스택 top에 삽입하여 그것이 새로운 현 변환행렬이 되게 함.
  - void glPopMatrix();
    - 스택 top의 행렬을 삭제함으로써 바로 아래에 있던 이전의 현 변환 행렬을 현 변환 행렬로 복원함.

행렬 스택 (Matrix Stack)



- (a)~(b): glPushMatrix()를 호출함으로써 현 CTM을 스택 top에 삽입하면서 이전의 CTM (saved CTM)은 현 CTM 바로 아래 위치하게 함
- (c)~(d): glPopMatrix()를 호출함으로써 스택 top의 CTM을 삭제함. 이에 따라 saved CTM이 현재의 CTM이 됨. 즉, glPushMatrix() 이전 상태로 원상복귀함.
- 즉, 현 상태의 CTM을 저장하고 싶으면 glPushMatrix()를 호출하고, 저장된 이전 상태로 원상복귀하고 위해서는 glPopMatrix()를 호출하면 됨.

행렬 스택 (Matrix Stack)

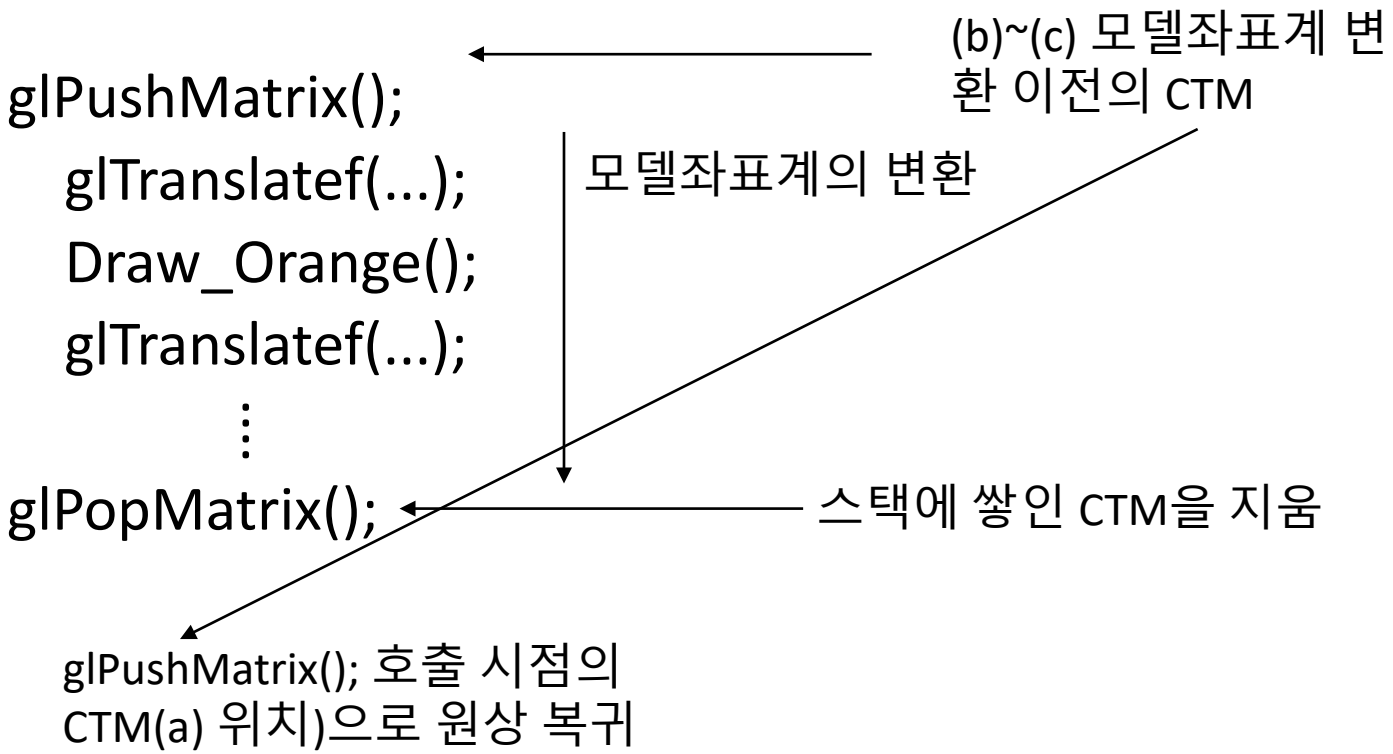


- 여러 개의 오렌지를 모델 좌표계 이동을 통해서 그림

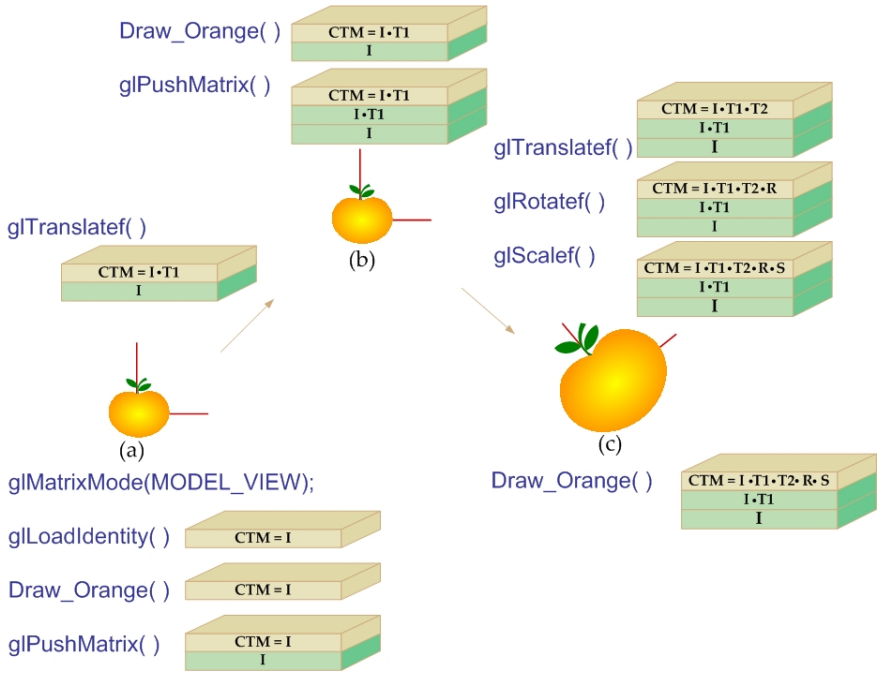
호출 함수	현 변환행렬	작업
glMatrixMode(GL_MODELVIEW);	CTM ← ModelView CTM	모델뷰 행렬 선택
① glLoadIdentity( );	CTM = I	초기화
② Draw_Orange( );	P' = CTM • P	(a)의 오렌지 그리기
③ glTranslatef(4.0, 4.0, 0.0);	CTM = CTM • T1	좌표계 이동
④ Draw_Orange( );	P' = CTM • P	(b)의 오렌지 그리기
⑤ glTranslatef(6.0, -2.0, 0.0);	CTM = CTM • T2	좌표계 이동
⑥ glRotatef(45, 0.0, 0.0, 1.0);	CTM = CTM • R	좌표계 회전
⑦ glScalef(2.0, 2.0, 2.0);	CTM = CTM • S	좌표계눈금 크기조절
⑧ Draw_Orange( );	P' = CTM • P	(c)의 오렌지 그리기

행렬 스택 (Matrix Stack)

- 오렌지를 원래 위치 (d)로 되돌리기 위해서는
  - 행렬 스택 사용



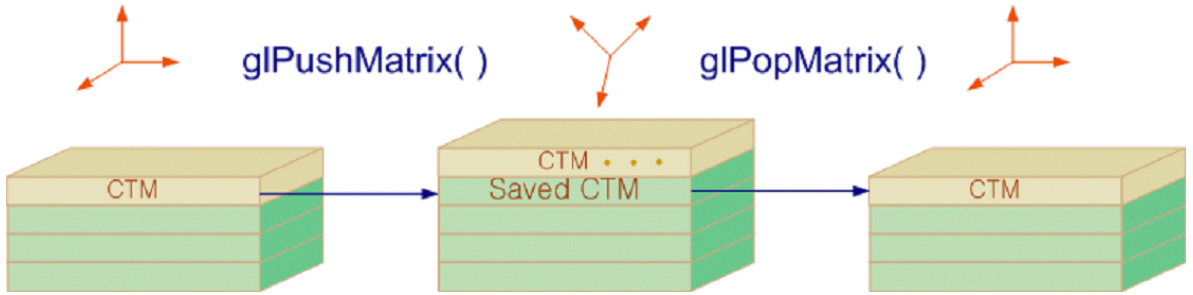
행렬 스택 (Matrix Stack)



- 예제에서 (a)와 (b)를 그리고 각각 glPushMatrix()를 호출함으로써 (a), (b) 위치로 돌아갈 수 있음.
- (a), (b), (c)의 좌표계가 각각 I, I·T1, I·T1 ·T2 · R · S 의 변환행렬로 나타남
- (c)를 그리는 시점의 행렬 스택에 해당 변환행렬이 저장된 것을 알 수 있음.

행렬 스택 (Matrix Stack)

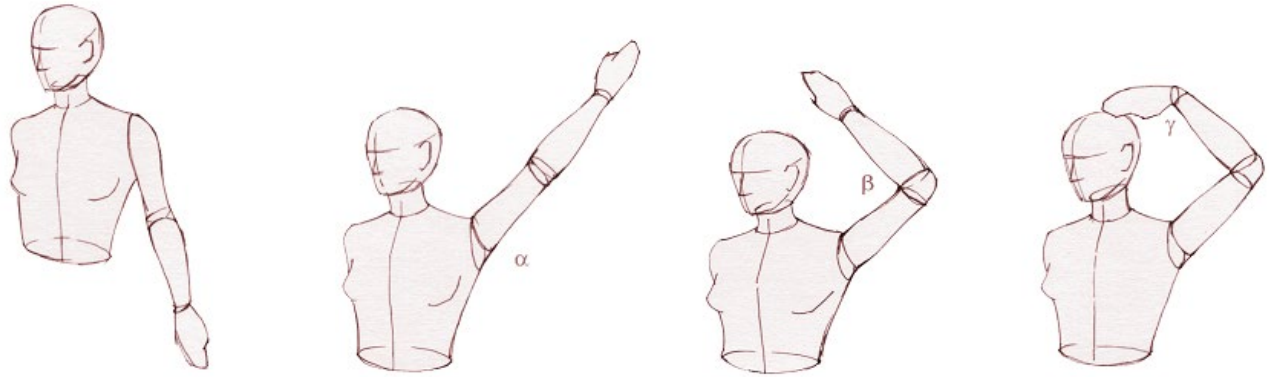
- 일반적인 TM push-pop 호출
  - glPushMatrix( );
    - glTranslatef( );
    - glRotatef( );
    - glScalef( );
    - ...
    - Draw_TransformedObject( );
  - glPopMatrix( );



계층구조 모델링

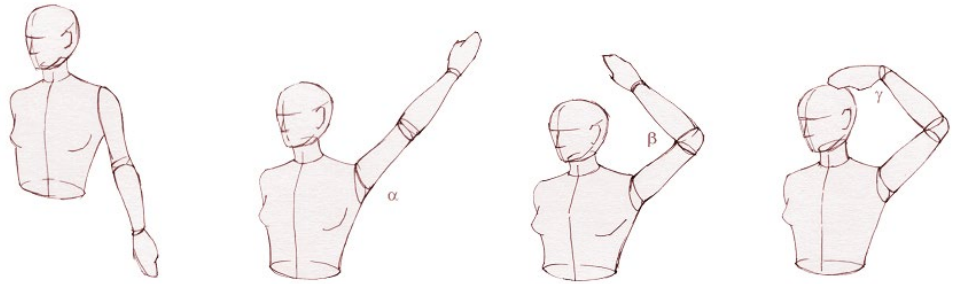
- 인체와 같은 다관절 객체를 그릴 때는 계층 구조(Hierarchy)를 설정하는 것이 유리.
- 계층 구조를 설정함으로써 중심만 이동함으로써 중심에 연결된 하위 객체들의 모델좌표계가 함께 변할 수 있어 계산에 유리.
- 몸의 각 부분은 몸체-위팔-아래팔-손바닥-손가락 순의 계층 구조로 연결.
- 즉, 몸체만 움직여도 종속된 다른 객체들이 자동으로 따라 움직임. 상속(Inheritance) 특성 때문.

계층구조 모델링



- 상체를 몸체-위팔-아래팔-손의 계층구조로 묘사할 경우 어깨, 팔꿈치, 손목 등은 객체 사이를 연결하는 관절(joint)에 해당.
- 중심축 기준 회전(Pivot Point Rotation)을 통해서 경례포즈를 구현할 수 있음.
- 어깨 관절 α 회전, 아래팔 β, 손목 γ 회전을 통해 상위 객체는 영향을 받지 않고 하위 객체의 모델 좌표가 변화하게 됨.
- 이러한 동작 구현을 순방향 키네마틱스(Forward Kinematics)라고 함.

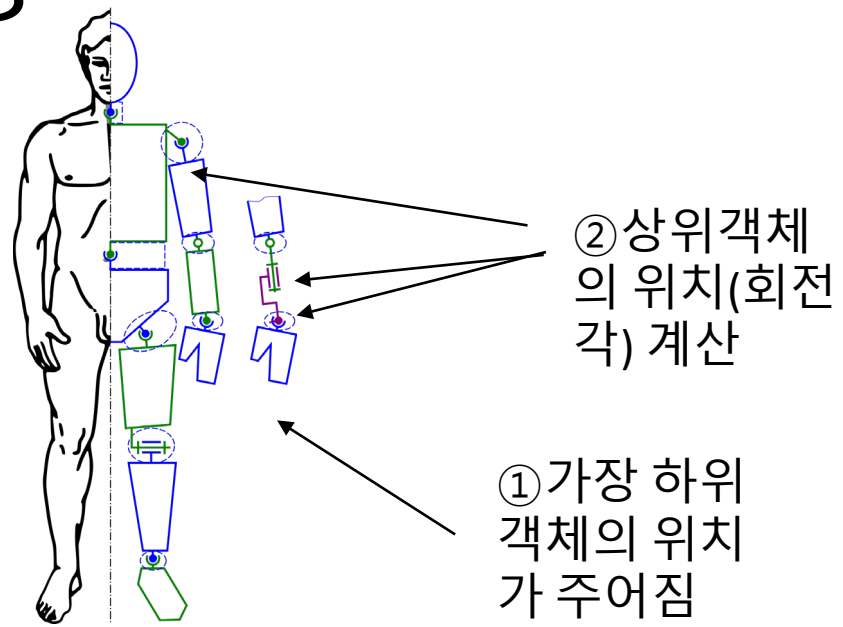
계층구조 모델링



Push-Pop을 사용한 Forward Kinematics의 구현

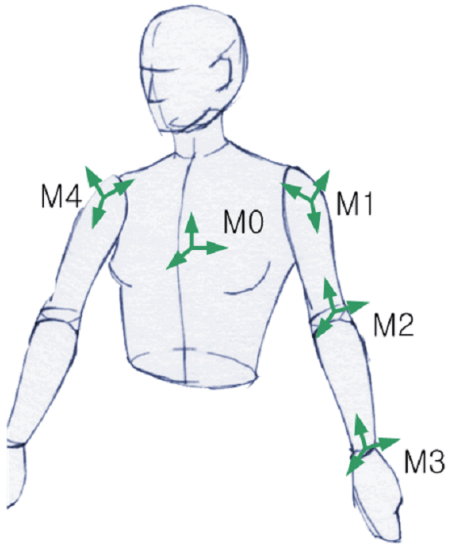
```
void drawArm( ){
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity( );                    전역 좌표계 = 모델 좌표계
  Draw_Body( );                         몸체 그리기
  glPushMatrix( );                      전역 좌표계 저장
     GoToShoulderCoordinates( );                어깨 기준 모델 좌표계
     Draw_UpperArm( );                  위 팔 그리기
     glPushMatrix( );                   어깨 기준 모델 좌표계 저장
        GoToElbowCoordinates( );                팔꿈치 기준 모델 좌표계
        Draw_LowerArm( );               아래팔 그리기
        glPushMatrix( );                팔꿈치 기준 모델 좌표계 저장
           GoToWristCoordinates( );             손목 기준 모델 좌표계
           Draw_Hand( );                손 그리기
        glPopMatrix( );                 팔꿈치 좌표계 복원
     glPopMatrix( );                    어깨 좌표계 복원
   glPopMatrix( );                      몸체 좌표계 복원
}
```

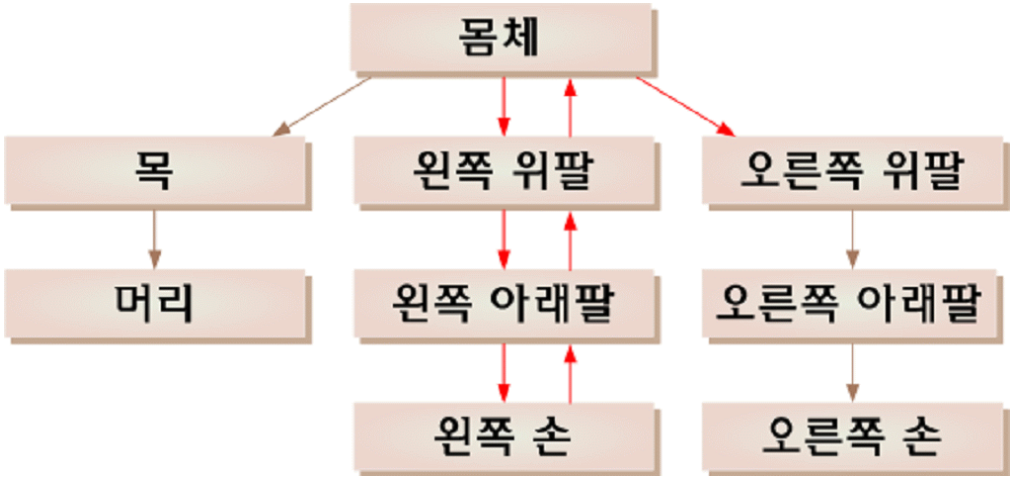
계층구조 모델링



- 역방향 키네마틱스 (Inverse Kinematics: IK)
  - 가장 하위 객체의 위치가 주어지면 상위객체들의 위치(회전각)를 계산.
  - 여러 개의 해가 존재할 수 있으며 관절각 범위 등의 제약조건으로 원하는 해를 찾음.
  - 변환행렬의 Jacobian을 계산해야 하기 때문에 GL 파트에서는 다루지 않으나 향후 다룰 예정.

계층구조 모델링





- 계층구조로 연결된 객체를 그릴 때,
  1. 상위의 객체를 먼저 그림
  2. Push-Pop을 통해서 하위의 객체들을 그림
     - 몸체를 그리고 나서 목을 그리기 전에 push, 머리까지 그리고 나서 pop, 왼쪽 위팔을 그리기 전에 push, 왼쪽 손 그리고 pop의 순서

예제 – solar system

```
static int Day = 0, Time = 0;

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(0, 0);
   glutCreateWindow("OpenGL Solar System Example");
   myInit();
   glutDisplayFunc(display);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
void myInit()
{
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
```

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glColor3f(1.0, 0.3, 0.3);
   glutWireSphere(0.2, 20, 16);// Draws the Sun
   glPushMatrix();
      glRotatef((GLfloat)Day, 0.0, 1.0, 0.0);// rotation of Earth
      glTranslatef(0.7, 0.0, 0.0);// far from the Sun by 0.7
      glRotatef((GLfloat)Time, 0.0, 1.0, 0.0);
      glColor3f(0.5, 0.6, 0.7);
      glutWireSphere(0.1, 10, 8);// Draws Earth
      glPushMatrix();// Moon part
         glRotatef((GLfloat)Time, 0.0, 1.0, 0.0);
         glTranslatef(0.2, 0.0, 0.0);
         glColor3f(0.9, 0.8, 0.2);
         glutWireSphere(0.04, 10, 8);
     glPopMatrix();
   glPopMatrix();
   glutSwapBuffers();
}
```

```
void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
   case 27:
      exit(0);
      break;
   case 'd' :
   case 'D' :
      Day = (Day + 10) % 360;
      glutPostRedisplay();
      break;
   case 't' :
   case 'T' :
      Time = (Time + 5) % 360;
      glutPostRedisplay();
      break;
   }
}
```