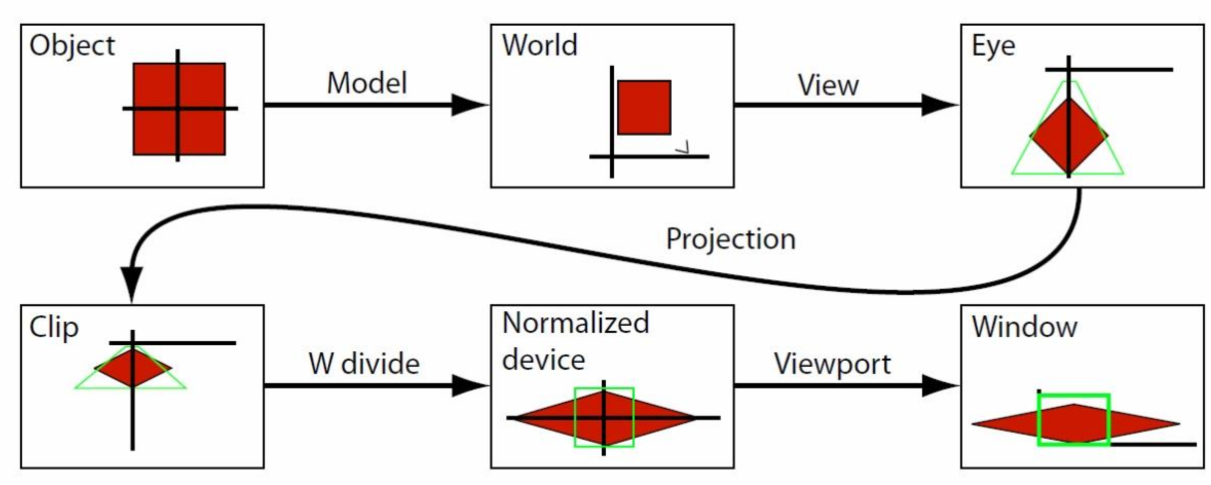
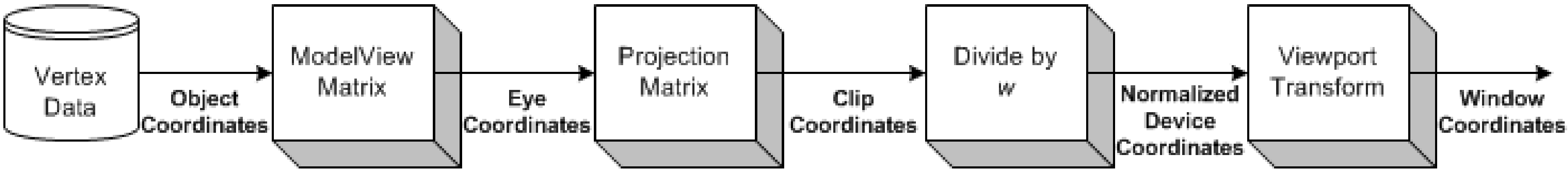
Transformations

OpenGL Steps

- **Every step in the graphics pipeline is related to the transformation.**

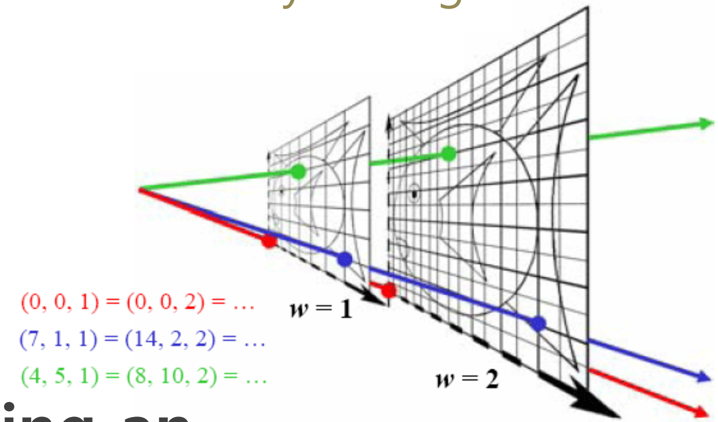




Homogeneous Coordinates

- 4 components $[x\ y\ z\ w]^T$ are used to represent a point in 3D.
  - Point $[x\ y\ z\ w]^T$
    - The result of transformation needs to be divided by w to give the 3D position. (homogeneous)
  - Vector $[x\ y\ z\ 0]^T$
    - w=0 represents a point at " infinity".

    (Only direction differentiates.)

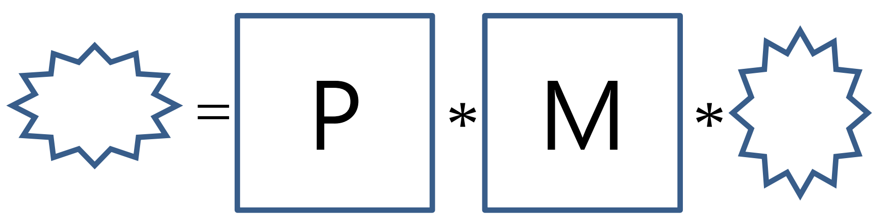


- 4x4 matrix is used for transforming an homogeneous vector.

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & t_1 \\ a_{10} & a_{11} & a_{12} & t_2 \\ a_{20} & a_{21} & a_{22} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Modelview & Projection matrix

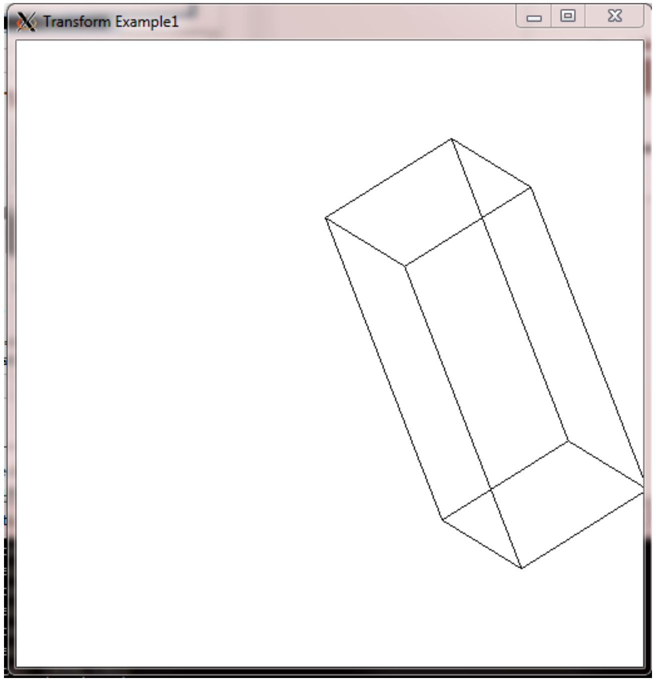
- **OpenGL helps us to change the two most important transformation matrices:**
  - Modelview Matrix
    - The relative transformation between object and camera
  - Projection Matrix
    - Clipping volume (viewing frustum)
    - Projection to screen
- **Vertices(primitives) are transformed by P*M.**



Transform Example

- **Draw a transformed box in 3d.**
  - Sample Program: "Transform_1.cpp"
  - Order: cp /home/share/Transform_1.cpp ./

```
void display() {
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glClearColor(1.0, 1.0, 1.0, 0.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-1.0, 1.0, -1.0,1.0, 0.1, 50.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glTranslatef(0.5, 0.0, -2.0);
        glRotatef(45.0, 1.0, 1.0, 1.0);
        glScalef(0.5, 1.2, 0.5);
        glColor3f(0.0,0.0,0.0);
        glutWireCube(1.0f);
        glXSwapBuffers(dpy, win);
}
```



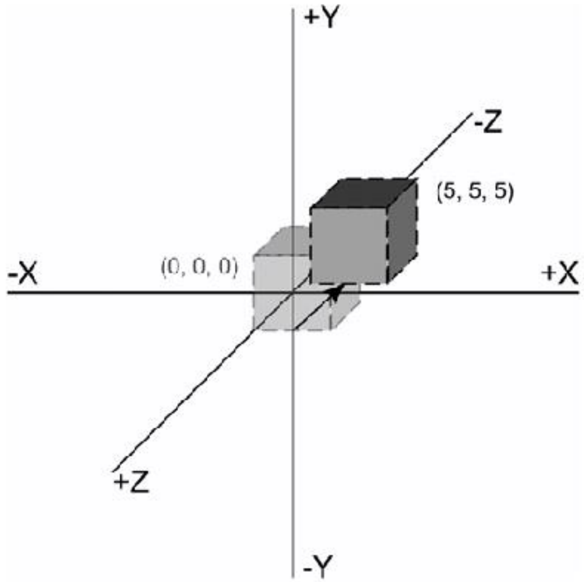
Transform Example-Code Details

- *glMatrixMode(GL_PROJECTION)* tells that we modify the projection matrix.
- *glMatrixMode(GL_MODELVIEW)* tells that we modify the modelview matrix.
- *glLoadIdentity()* replaces the current matrix with the identity matrix.
- *glOrtho(l,r,b,t,zn,zf)* sets the clipping space orthographically. This changes the projection matrix.
- *glClear(…)* clears video buffers.
- *glViewport(x,y,w,h)* sets the screen space.
- *glTranslatef(),glRotatef(),* and *glScalef()* change the modelview matrix.

Translation: glTranslatef()

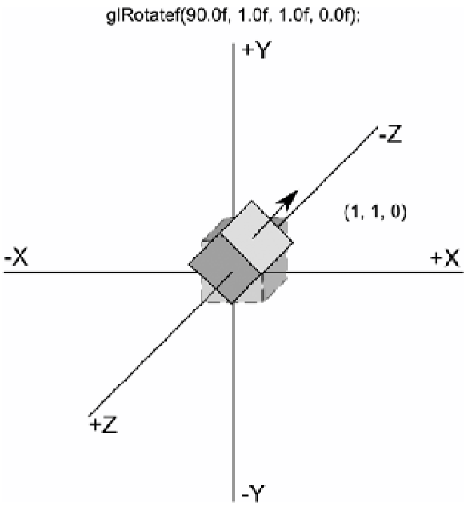
- **glTranslatef(x, y, z);**
  - translates the geometry by (x, y, z)
- **Example**
  - Moving a cube from (0,0,0) to (5,5,5)

```
glTranslatef(5.0, 5.0, 5.0);  // move to (5,5,5)
renderCube();                 // draw the cube
```



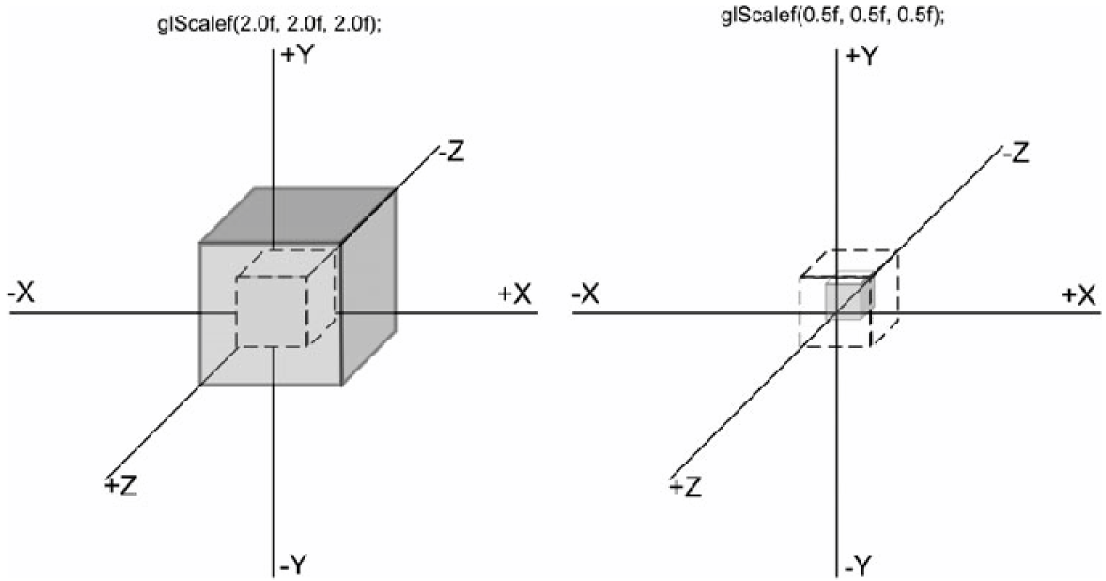
Rotation: glRotate

- **glRotatef(angle, axis_x, axis_y, axis_z);**
  - angle: rotation angle in degrees in CCW
  - (axis_x, axis_y, axis_z): the rotation axis
- **Example**
  - glRotatef(135.0f, 0.0f, 1.0f, 0.0f); // rotation around y-axis
  - glRotatef(90.0f, 1.0f, 1.0f, 1.0f); // around axis (1, 1, 1)



Scaling: glScale

- **glScalef (sx, sy, sz);**
  - sx, sy, sz: scale factors along x, y, and z directions
- **Examples**
  - glScalef(2.0f, 2.0f, 2.0f); // uniform scaling (doubling)
  - glScalef(2.0f, 1.0f, 1.0f); // doubling only along x direction

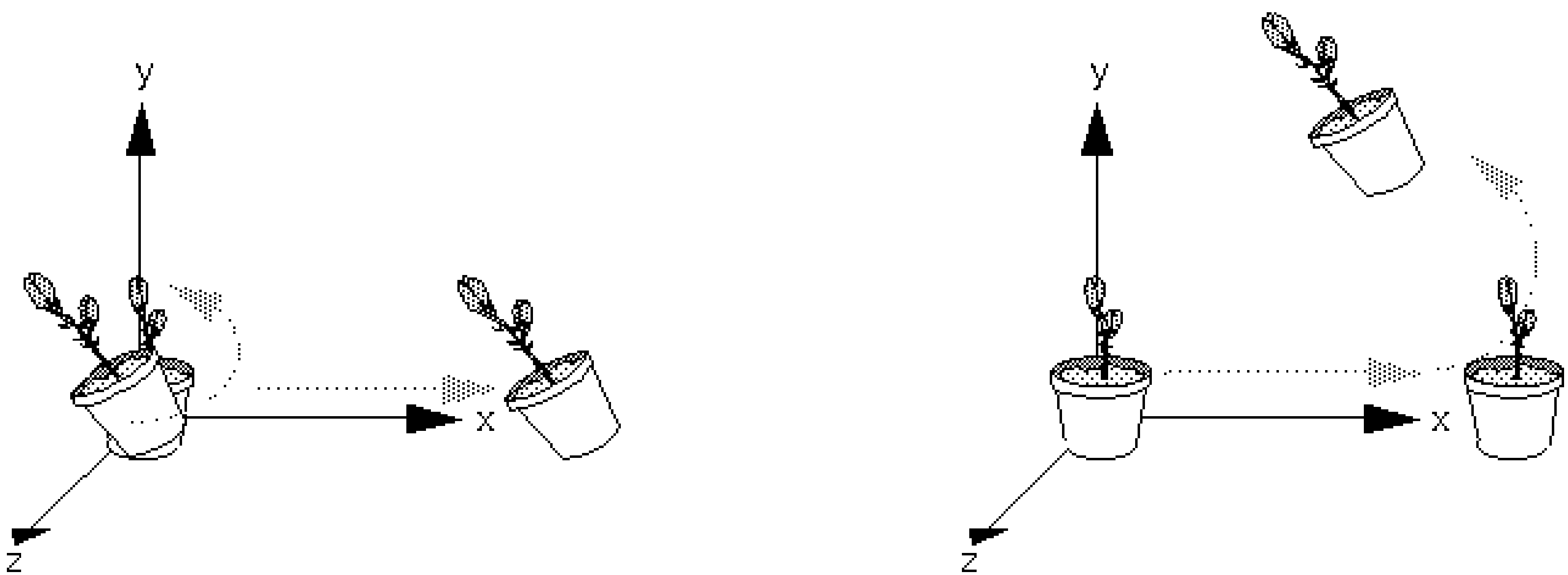


The Order of Transformation

```
glLoadIdentity();   // C = I
glMultMatrixf(N);   // C = N
glMultMatrixf(M);   // C = NM
glMultMatrixf(L);   // C = NML
glBegin(GL_POINTS);
  glVertex3f(v);    // NMLv
glEnd();
```

Result Comparison: Transform order

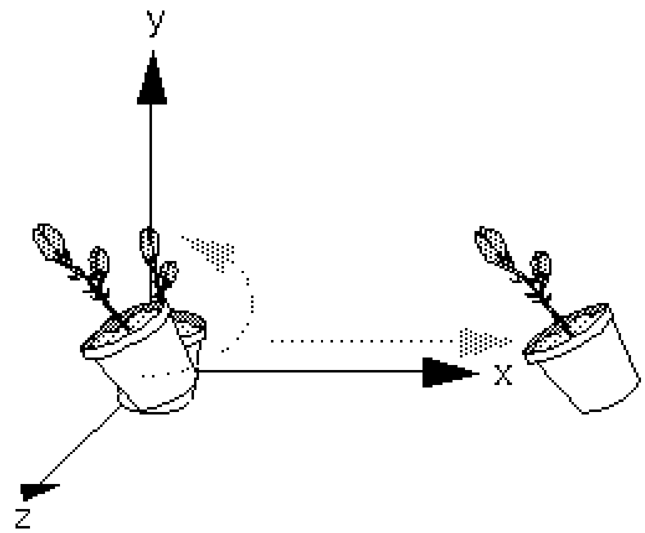
- **45 deg rotation around z-axis then 10 unit translation along +x, and vice versa.**



Result Comparison: Result 1

- Code for trans then rot with local coords.

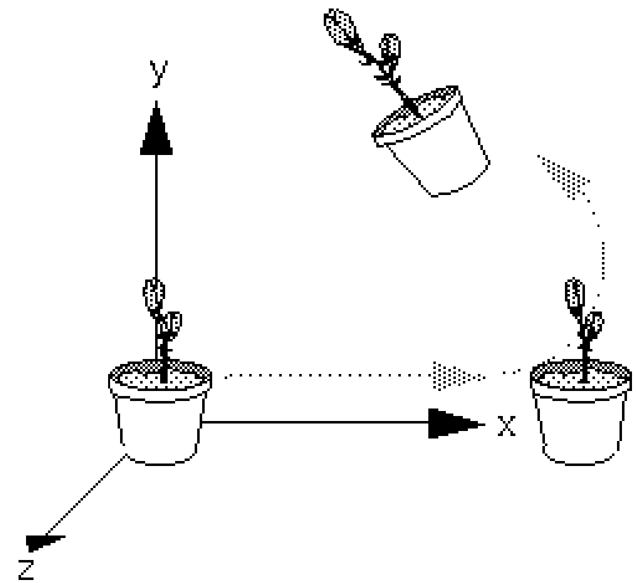
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(T);
glMultMatrixf(R);
draw_the_object();
```



Result Comparison: Result 2

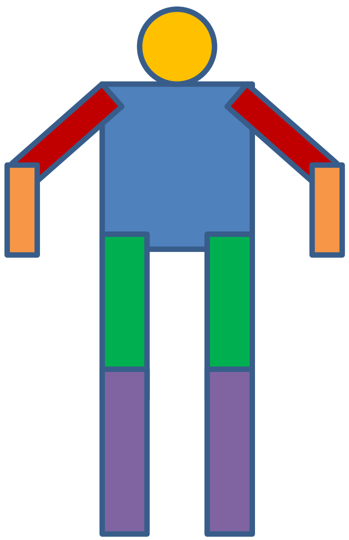
- Code for rot then trans with local coords.

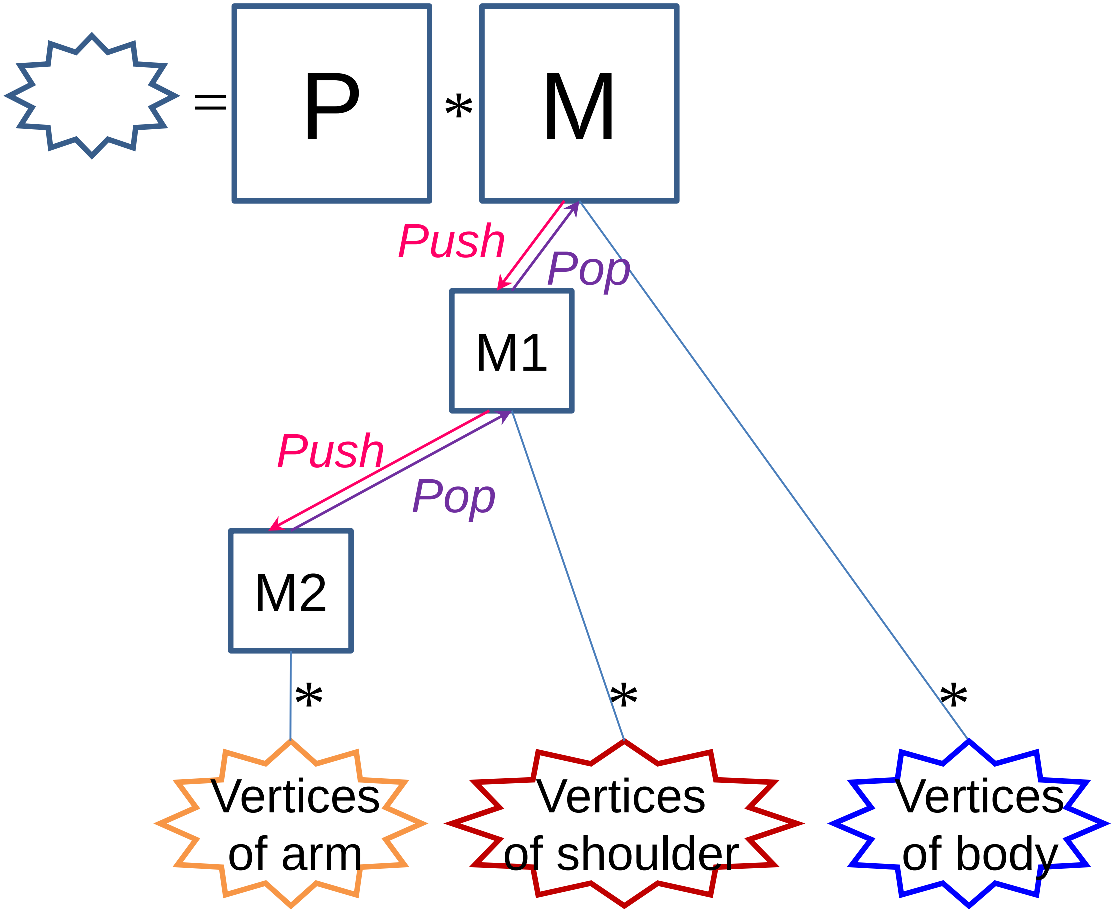
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(R);
glMultMatrixf(T);
draw_the_object();
```



Push & Pop

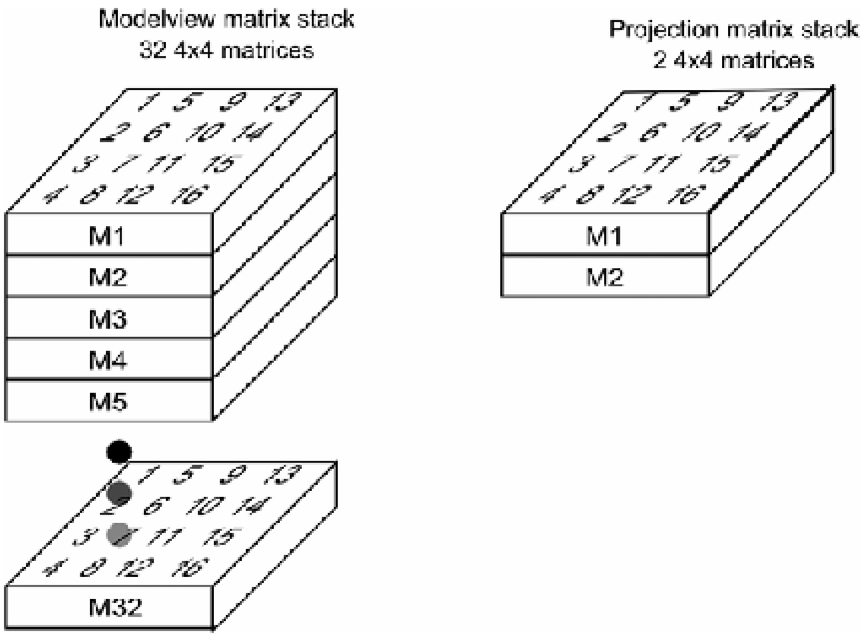
- **We can manage the hierarchy by** ***glPushMatrix(), glPopMatrix().***





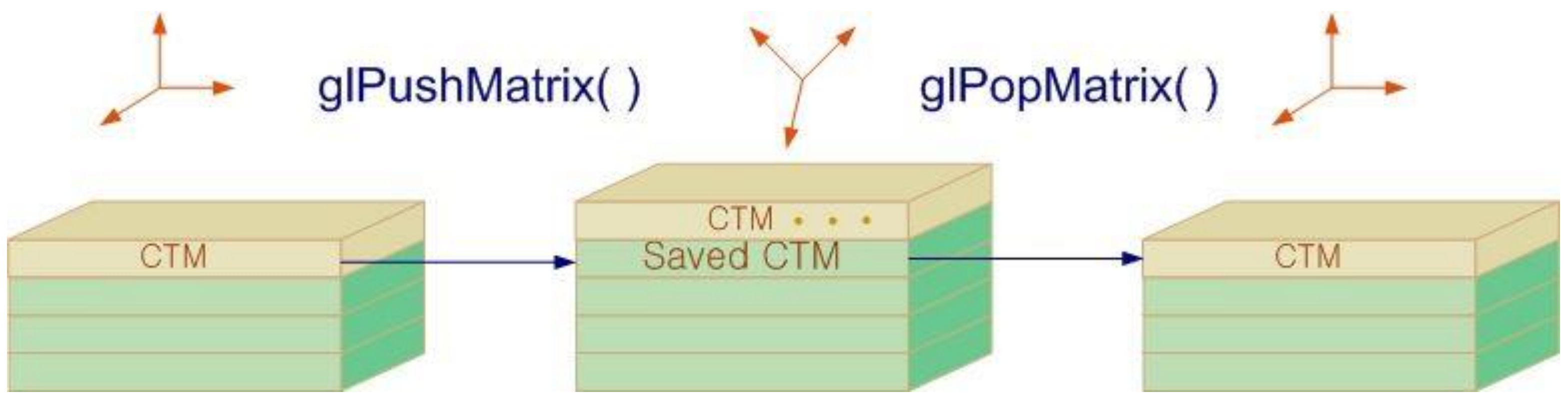
Matrix Stack

- **Allows you**
  - to save the current state of the transformation matrix
  - to perform other transformations
  - then, to return to the saved transformation matrix



Matrix Stack

- **Top matrix in the matrix stack**
  - is used as the current transformation matrix
- **Pushing and Popping the matrix stack**
  - void glPushMatrix( )
    - Push by duplicating the current top matrix
  - void glPopMatrix( )
    - Pop out the top matrix, the second top matrix



Transform Example : Robot Arm

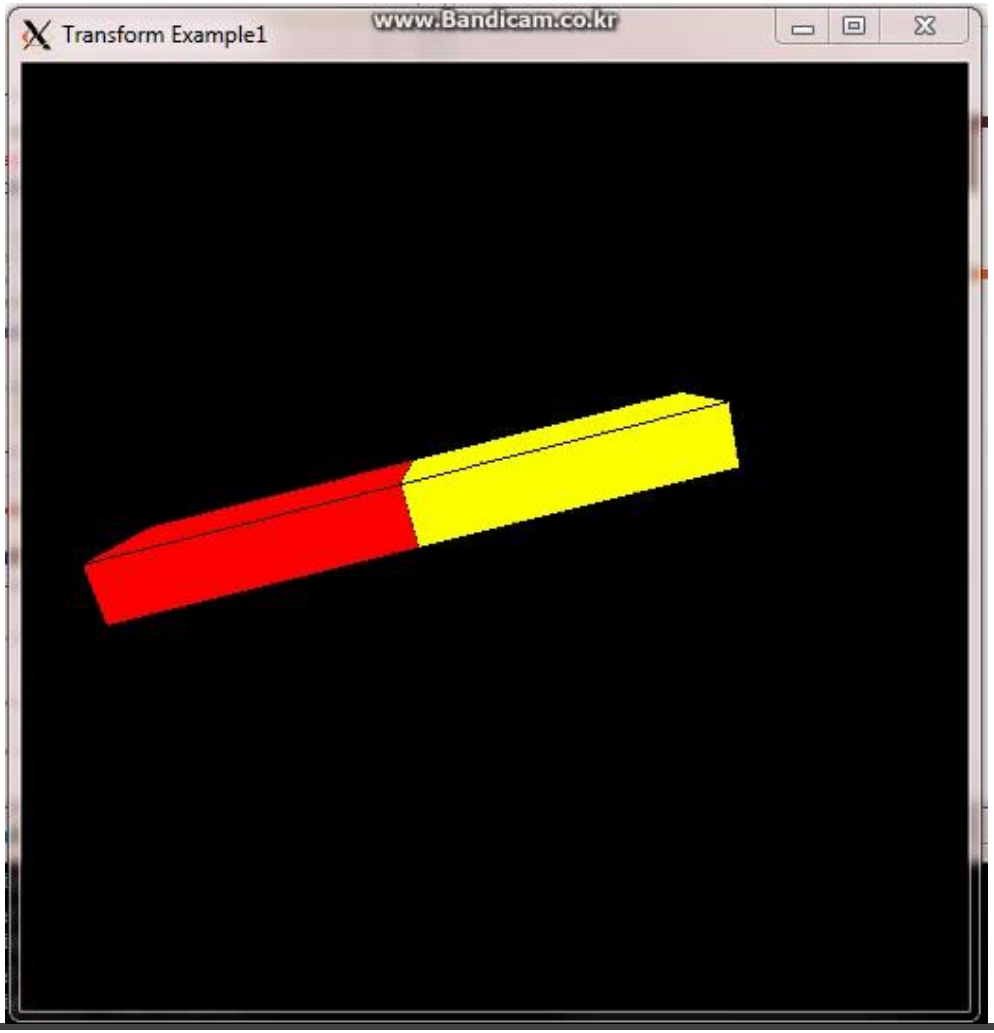
Sample Program: "Transform_2.cpp"
Order: cp /home/share/Transform_2.cpp ./

```
int shoulder = 0, elbow = 0;
void display() {
    /*Initialize Drawing*/
    glPushMatrix();
        glRotatef(20, 1, 0, 1);
        glPushMatrix();
            glTranslatef(-1.0, 0.0, 0.0);
            glRotatef(shoulder, 0.0, 0.0, 1.0);
            glTranslatef(1.0, 0.0, 0.0);

            glPushMatrix();
                glScalef(2.0, 0.4, 1.0);
                glColor3f(1,0,0);
                glutSolidCube(1.0);
            glPopMatrix();

            glTranslatef(1.0, 0.0, 0.0);
            glRotatef(elbow, 0.0, 0.0, 1.0);
            glTranslatef(1.0, 0.0, 0.0);

            glPushMatrix();
                glScalef(2.0, 0.4, 1.0);
                glColor3f(1,1,0);
                glutSolidCube(1.0);
            glPopMatrix();
        glPopMatrix();
    glPopMatrix();
    glXSwapBuffers(dpy, win);
}
```

```
void keyPressEvent(char* key_string){
        if(strncmp(key_string, "Up", 2) == 0){
                shoulder = (shoulder+5)%360;
        }else if(strncmp(key_string, "Down", 4) == 0){
                shoulder = (shoulder-5)%360;
        }else if(strncmp(key_string, "Right", 5) == 0){
                elbow = (elbow+5)%360;
        }else if(strncmp(key_string, "Left", 4) == 0){
                elbow = (elbow-5)%360;
        }
}

int main(int argc, char *argv[]) {
    /*CreateWindow*/
    XEvent xev;
    while(1) {
        display();
        XNextEvent(dpy, &xev);
        if(xev.type == KeyPress){
            char *key_string = XKeysymToString(
            XkbKeycodeToKeysym(dpy, xev.xkey.keycode, 0, 0));
            keyPressEvent(key_string);
        }
    }
}
```

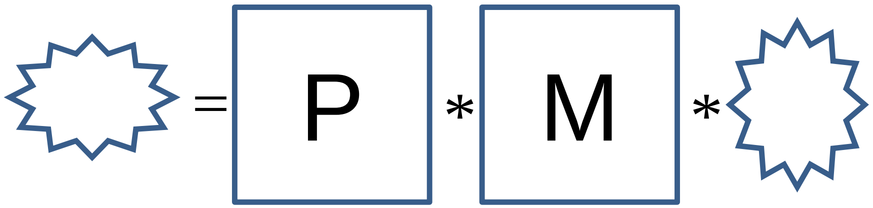
Robot Arm Result

Add library linking [-lGLU]



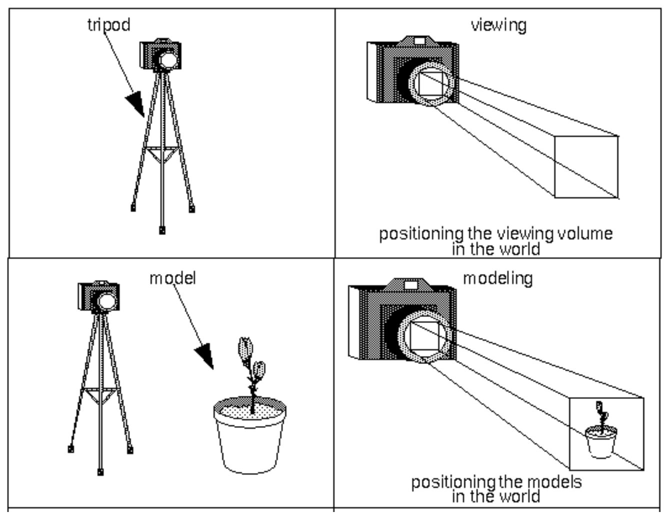
Review: Modelview & Projection matrix

- **OpenGL allows us to change the two most important transformation matrices:**
  - Modelview Matrix
    - The relative transformation between object and camera
  - Projection Matrix
    - Clipping volume (viewing frustum)
    - Projection to the normal space
- **Vertices(primitives) are transformed by P*M.**



Modelview Matrix

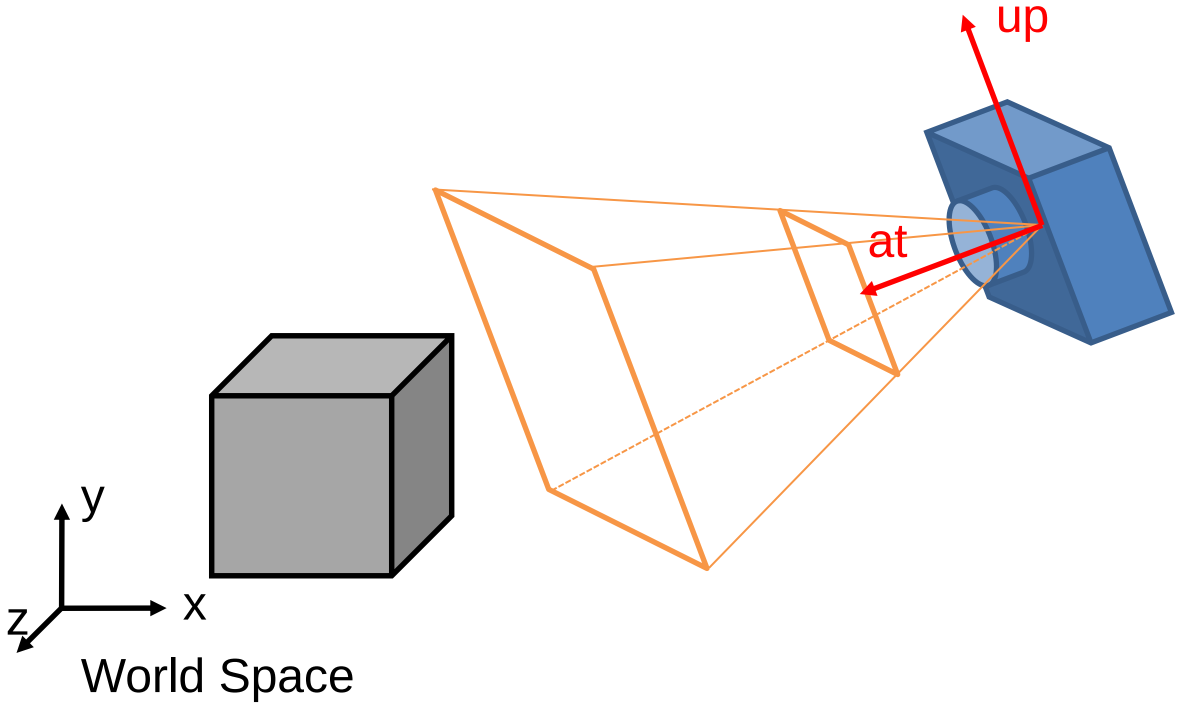
- **Modelview matrix is modified by**
  - Object movement: Modeling Transformation (M)
  - Camera movement: Viewing Transformation (V)



$$= \boxed{P} * \boxed{V * M} *$$

Viewing Transformation

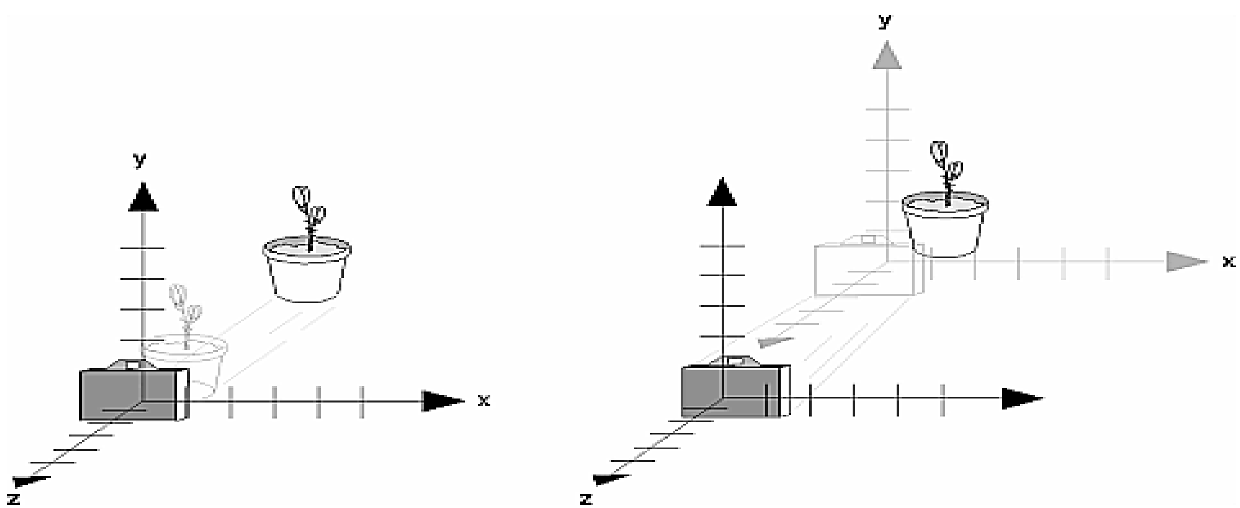
- ***gluLookAt(eyex,eyey,eyez, atx,aty,atz, upx,upy,upz)***
  - For example, *gluLookAt(0,0,2, 0,0,0, 0,1,0)* produces the same modelview matrix as *glTranslatef(0,0,-2).*



Duality of Modeling and Viewing Trans.

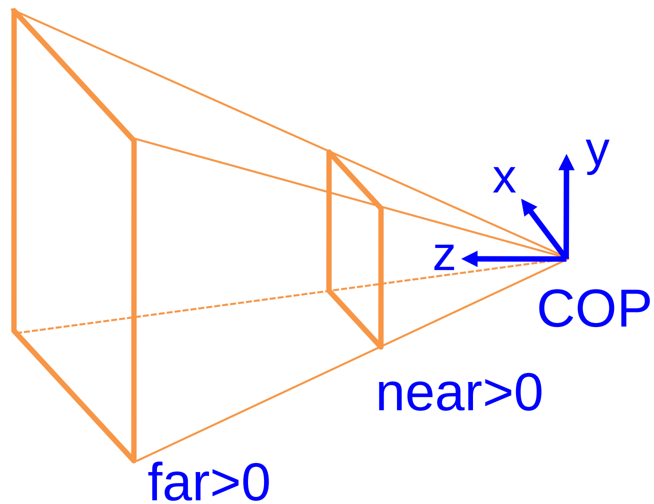
- **Relative effect**
  - Translate an object by (0,0,-5) has the same effect as translate camera by (0,0,5).
  - Viewing trans + Modeling trans

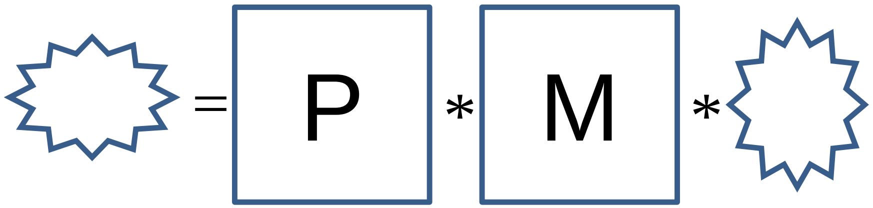
    => modelview transformation.



Projection Matrix

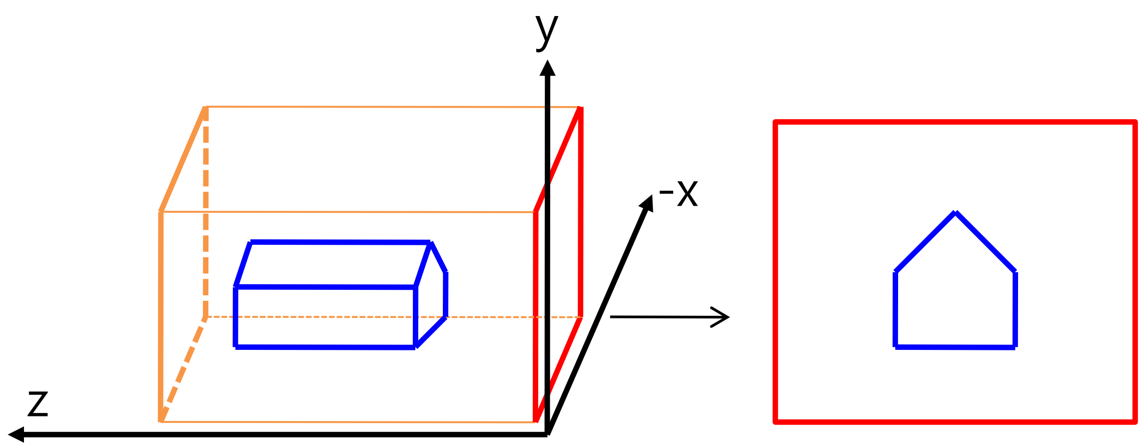
- **Projection matrix can be defined by giving**
  - Projection type (orthographic or perspective)
  - Frustum (clipping volume)





Simple Orthographic Projection

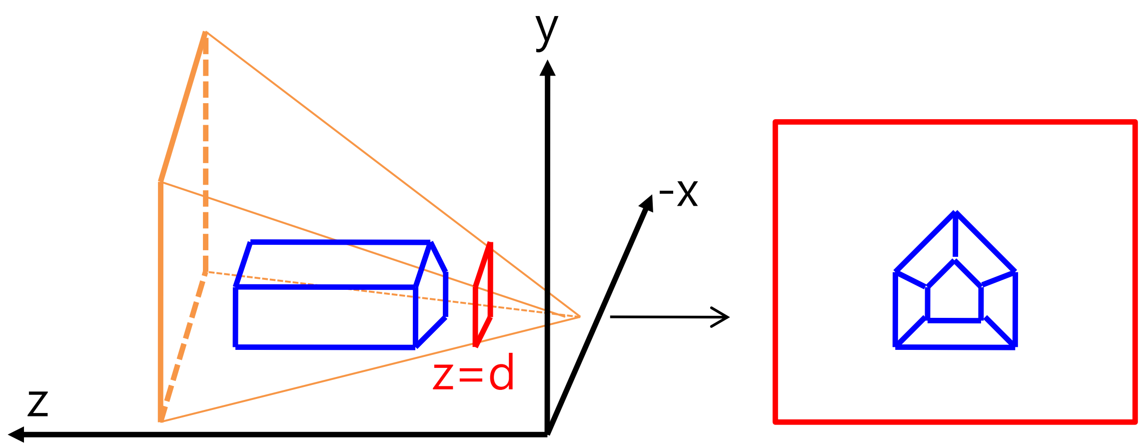
- Project all points to the z=0 plane.



$$\begin{bmatrix} x_p \\ y_p \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

A Simple Perspective Projection

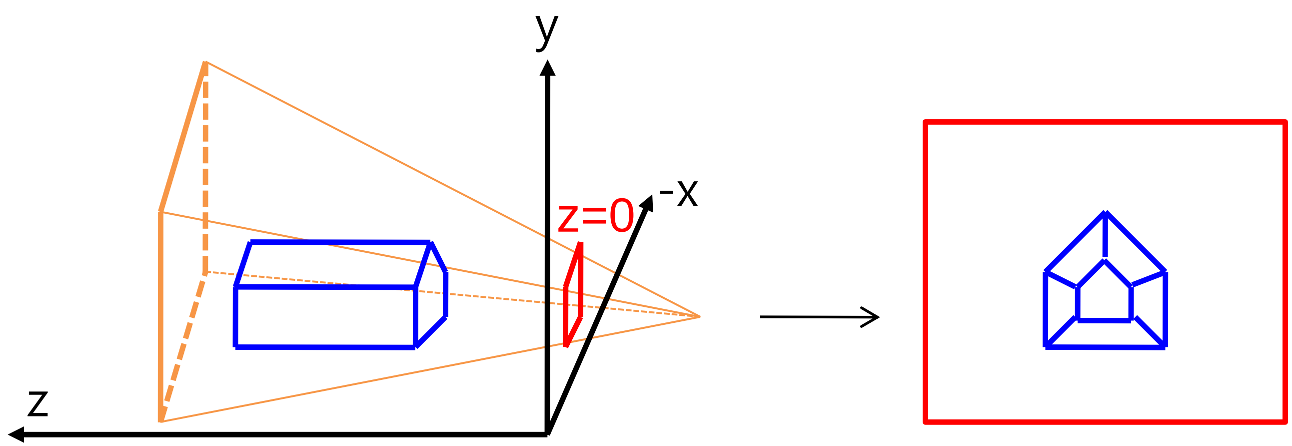
- **Project all points to the plane z=d with COP at z=0.**



$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} (d/z)x \\ (d/z)y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Another Perspective Projection

- Project all points to the plane z=0 with COP at z=-d.

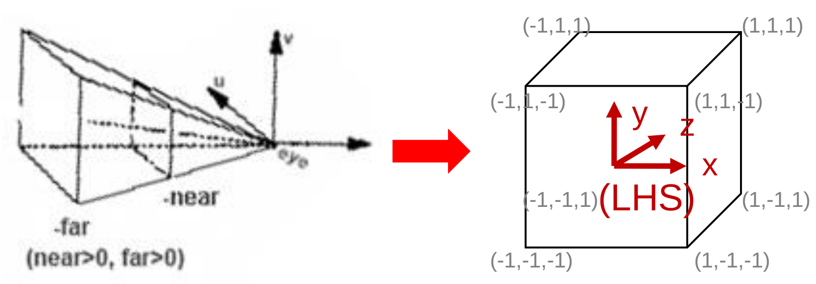


$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} \left(d/(z+d)\right)x \\ \left(d/(z+d)\right)y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ (z+d)/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

As $d \to \infty$, $(x,y) \to (x_p, y_p)$

OpenGL Projection Functions

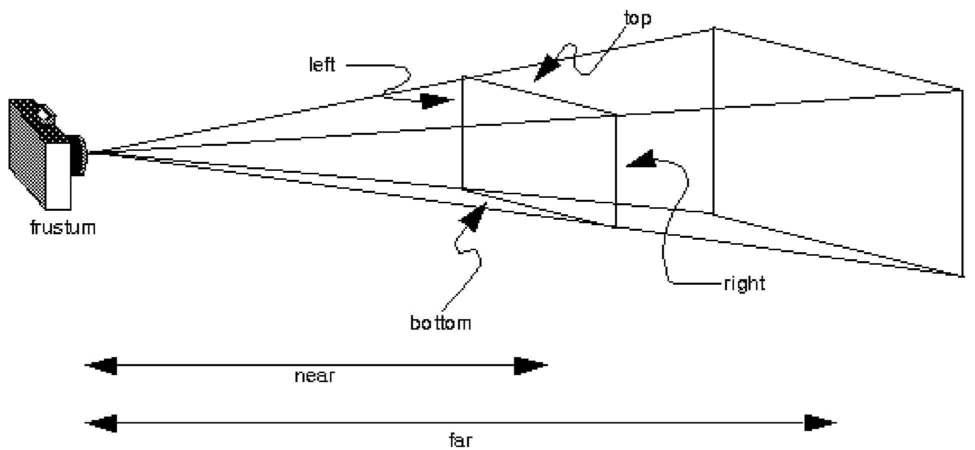
- In OpenGL, rather than performing the actual projection, the projection matrix causes the viewing volume to be transformed to the canonical view volume.
  (world coordinates→normalized device coordinates)



Original View Volume   Canonical View Volume

OpenGL Projection Types

- **OpenGL provides functions to construct the projection matrix.**
  - *glOrtho(left, right, bottom, top, near, far)*
  - *gluOrtho2D(left, right, bottom, top)*
  - *gluPerpective(field of view, aspect(=w/h), near, far)*
  - *glFrustum(left, right, bottom, top, near, far)*



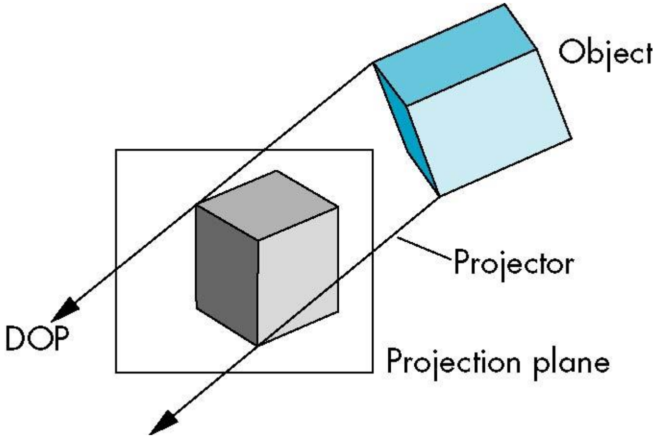
Orthographic Projection Matrix

- *glOrtho(l,r,b,t,n,f)* ***(left, right, bottom, top, near, far)***

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

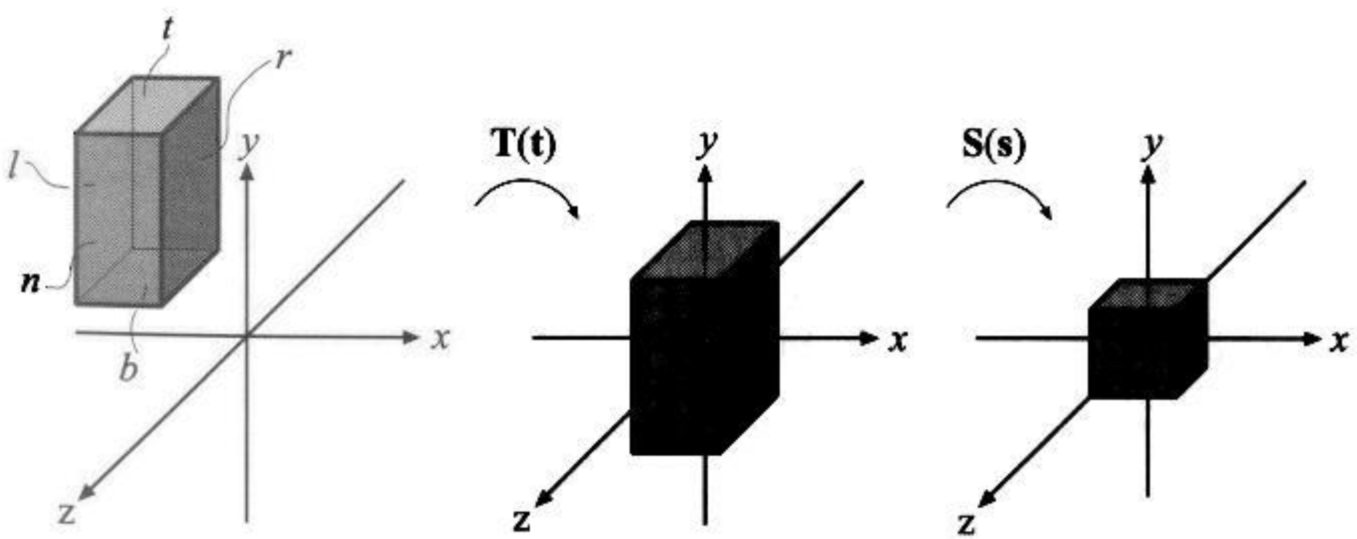
- *gluOrtho2D(l,r,b,t)* ***(left, right, bottom, top)***

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Derivation of glOrtho(l,r,b,t,n,f)

- **It reduces to finding a transformation which transforms the cuboid to the canonical view volume $[-1, +1]^3$.**
  - First center the cuboid by translating.
  - Then scale the result into the unit cube.



Transformation Matrix

Scale　　　　　　Translation (centering)

$$M = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A Negation Needs to be Applied

- **In OpenGL, cuboid (l,r; t,b; n,f) represents the volume with z ranging [-f, -n].**
  - OpenGL Convention: looking down –z
- **Therefore a negation needs to be applied internally.**

$$M = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad glOrtho = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \boxed{\frac{-2}{f-n}} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The Final Result

Perspective Projection Matrix

- *gluPerspective(fov,aspect(=w/h),n,f)*
  - *(field of view, aspect(=w/h), near, far)*

$$\begin{bmatrix} \frac{\cot(fov/2)}{aspect} & 0 & 0 & 0 \\ 0 & \cot(fov/2) & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- *glFrustum(l,r,b,t,n,f)*
  - *(left, right, bottom, top, near, far)*

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Referencing & Applying the Matrix

- ***glGetFloatv(GL_MODELVIEW_MATRIX, mat)***
  glGetFloatv(GL_PROJECTION_MATRIX, mat)
  - Get the 4x4 modelview matrix to the array columnwisely.

$$
\begin{array}{llll}
mat[0]=m00, & mat[4]=m01, & mat[8]=m02, & mat[12]=m03; \\
mat[1]=m10, & mat[5]=m11, & mat[9]=m12, & mat[13]=m13; \\
mat[2]=m20, & mat[6]=m21, & mat[10]=m22, & mat[14]=m23; \\
mat[3]=m30, & mat[7]=m31, & mat[11]=m32, & mat[15]=m33;
\end{array}
$$

- ***glMultMatrixf(mat)***
  - Multiply the current matrix with the specified matrix.

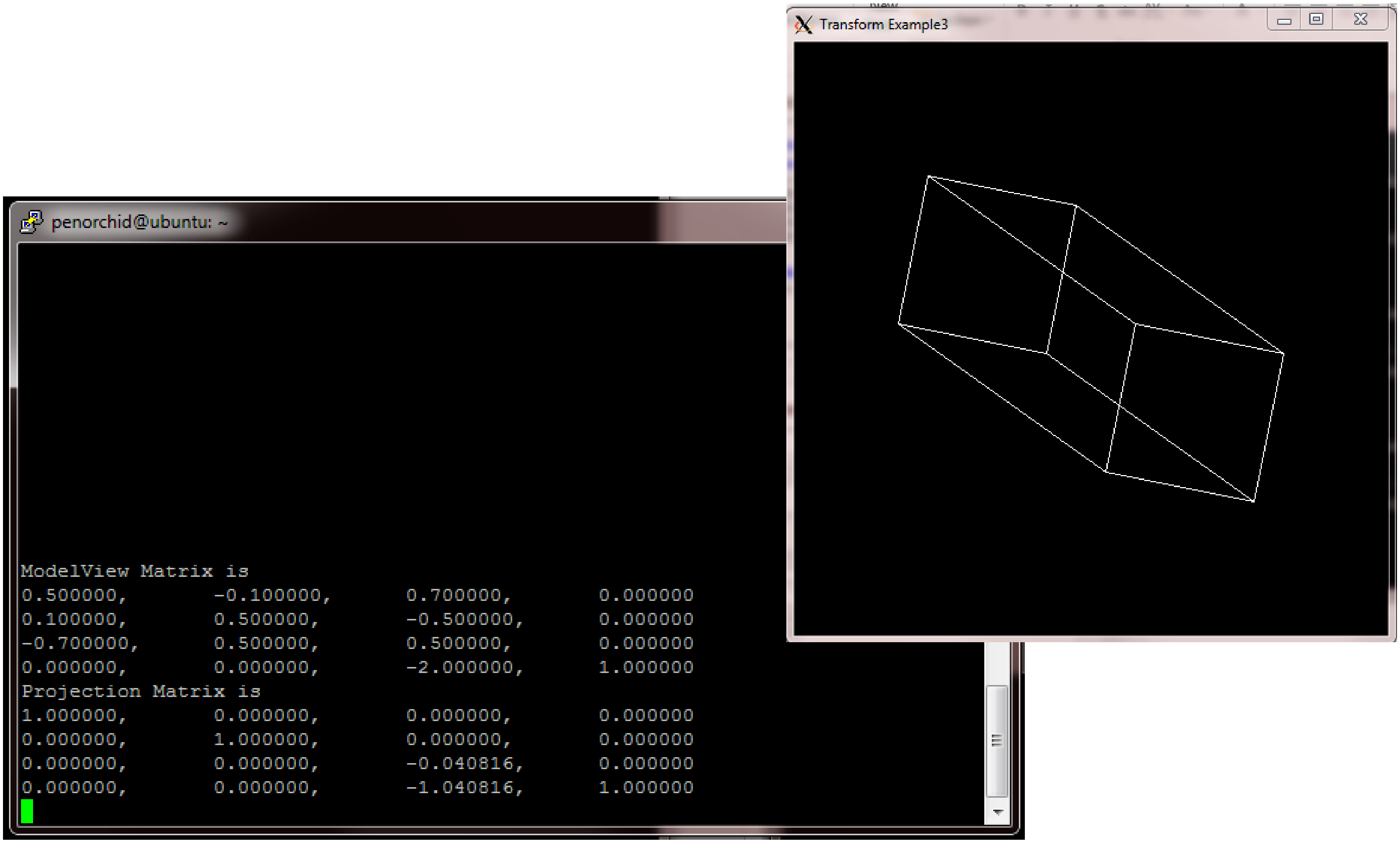
Matrix manipulation Example

Sample Program: "Transform_3.cpp"
Order: cp /home/share/Transform_3.cpp ./

```
void lookMatrix(){
        float m[16] = {0};
        float p[16] = {0};
        glGetFloatv(GL_MODELVIEW_MATRIX, m);
        glGetFloatv(GL_PROJECTION_MATRIX, p);
        /*print m&p*/

}

void applyModelviewMatrix(){
        float m[16] = {0.5,-0.1,0.7,0,
                0.1,0.5,-0.5,0,
                -0.7,0.50,0.50,0,
                0,0,-2,1};
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glMultMatrixf(m);
}
void applyProjectionMatrix(){
        float p[16] = {1,0,0,0,
        0,1,0,0,
        0,0,-2.0/(50.0-1.0),0,
        0,0,-(50.0+1)/(50.0-1.0),1};
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glMultMatrixf(p);
}
```

```
void display(){
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        applyProjectionMatrix();
        applyModelviewMatrix();
        glutWireCube(1.0);
        glXSwapBuffers(dpy, win);
}

void main(){
        /*createWindow*/
        while(1){
                display();
                /*etc*/
                if(/*keyPressEvent*/) lookMatrix();
        }
}
```

Matrix manipulation Result



Viewport Transformation

- **Final mapping to the screen space (pixels)**
- ***glViewport(x,y,w,h)***