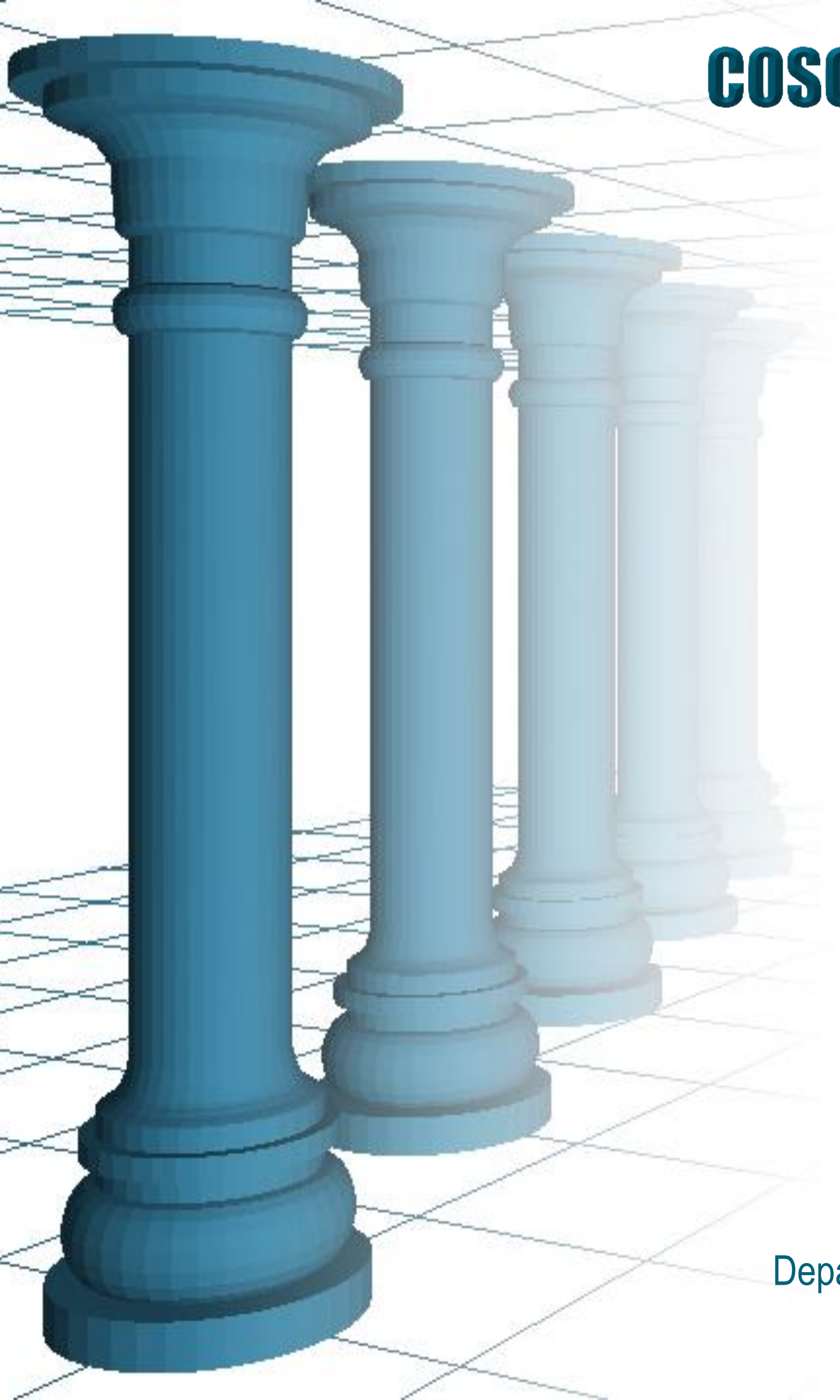


COSC363 Computer Graphics

2

Transformations

Set the wheel in motion!

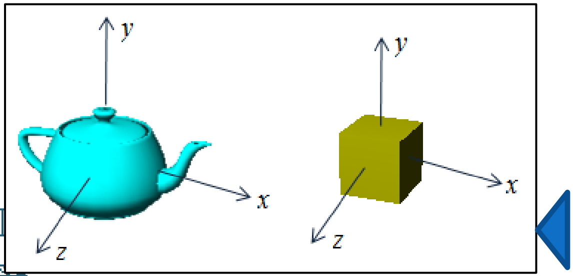
R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

Transformations

- A transformation changes the coordinates of a vertex or the components of a vector.
- Several types of transformations are commonly used in a graphics application:
  - Transformations of objects within the same frame: Egs. Translations, rotations and scale transformations. These transformations are called **Model Transformations**.
  - Transformation of an object from one reference frame to another. Eg. The transformation from world space to the camera space **(View Transformation)**
  - Projection transformations. These are based on the camera’s frustum parameters.
  - In this section, we will consider model transformations.

Model-View Transformation

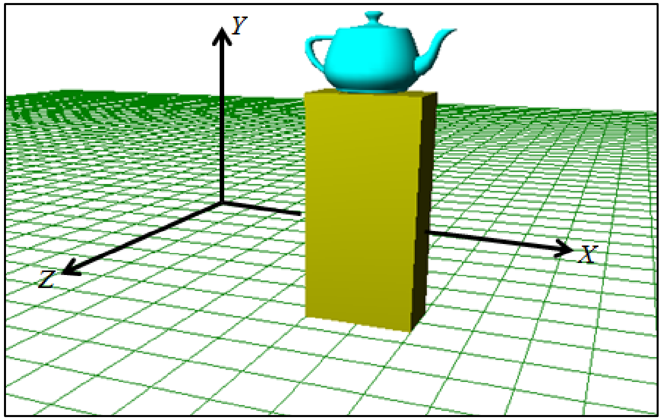
- Objects are created in their own local coordinate space and then transformed into the world coordinate space.
- They are transformed again into the coordinate space of the camera to generate the view as seen by the camera.



Modelling Space

`glutSolidCube(..)`
`glutSolidTeapot(..)`

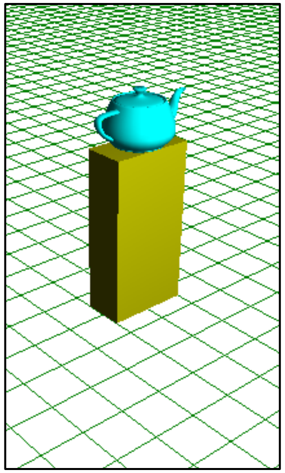
Model Transformation. Eg. Scale, Rotn, Translation



World Space

View Transformation

`gluLookAt(..)`



Camera Space

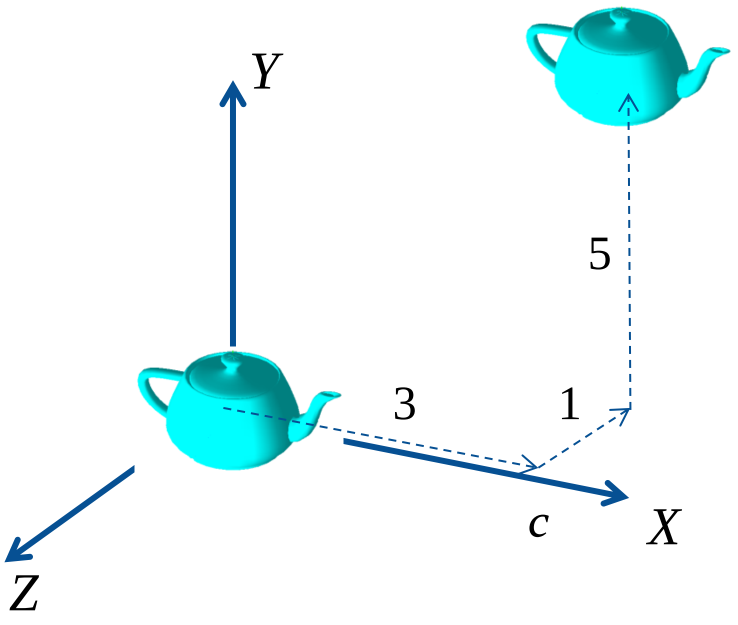
Transformations in OpenGL

- OpenGL supports the following types of three-dimensional model transformations:
  - Translations: **`glTranslatef`**`(a, b, c);`
  - Rotations: **`glRotatef`**`(angle, l, m, n);`
  - Scale Transformations: **`glScalef`**`(sx, sy, sz);`
  - Generalized transformation: **`glMultMatrixf`**`(mat);`
- *All* transformations are stored as 4x4 matrices (discussed later in the course).
- Model transformations form part of the **model-view matrix**.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Translation

OpenGL function: **glTranslatef**(a, b, c);



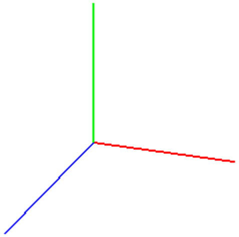
Example:

```
void display()
{
 ...
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(…)
  glLightfv(…)
  glTranslatef(3, 5, -1);
  glutSolidTeapot(1.0);
  glFlush();
}
```

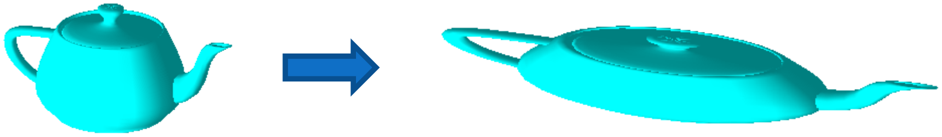
$$(x, y, z) \xrightarrow{T(a, b, c)} (x + a,\ \ y + b,\ \ z + c)$$

Scaling

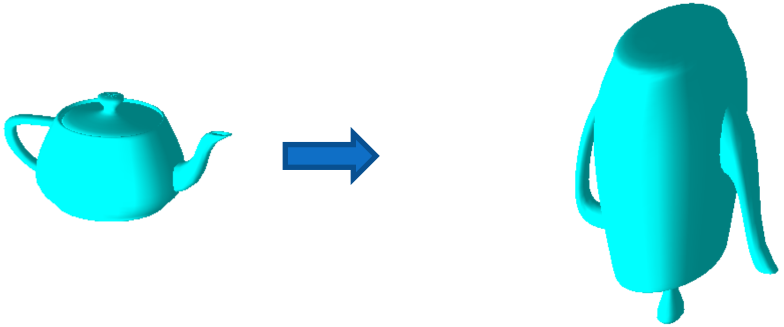
- OpenGL function: **`glScalef`**`(a, b, c)`
- A negative scale factor corresponds to a reflection.
- A zero scale factor corresponds to a projection



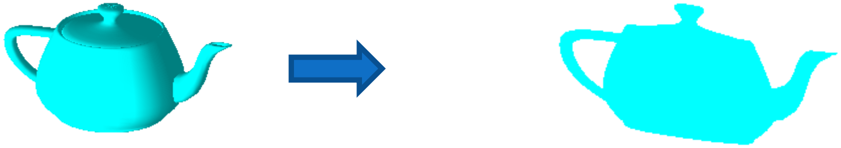
```
glScalef(2.0, 0.5, 1);
glutSolidTeapot(1);
```



```
glScalef(1, -3, 2);
glutSolidTeapot(1);
```



```
glScalef(1, 1, 0);
glutSolidTeapot(1);
```

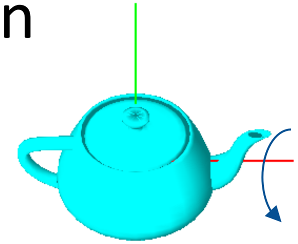


$$(x, y, z) \xrightarrow{S(a, b, c)} (xa\ , yb, zc)$$

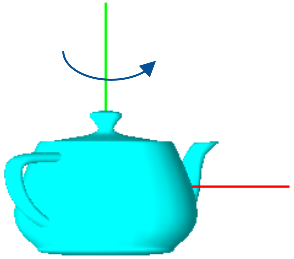
Rotations

- OpenGL function: **`glRotatef`**`(theta,` $\overbrace{\texttt{l, m, n}}^{\text{Axis of rotation}}$`)`
- A positive angle corresponds to a rotation in the anti-clockwise sense about the axis of rotation

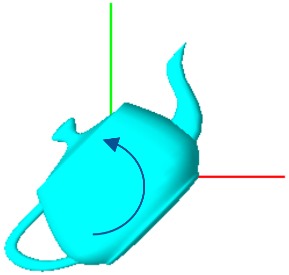
```
glRotatef(45, 1, 0, 0);
glutSolidTeapot(1);
```



```
glRotatef(45, 0, 1, 0);
glutSolidTeapot(1);
```



```
glRotatef(45, 0, 0, 1);
glutSolidTeapot(1);
```



Rotations

- Rotation about the *x*-axis:

$$(x, y, z) \Rightarrow (\,x,\quad y\cos\theta - z\sin\theta\,,\quad y\sin\theta + z\cos\theta\,)$$

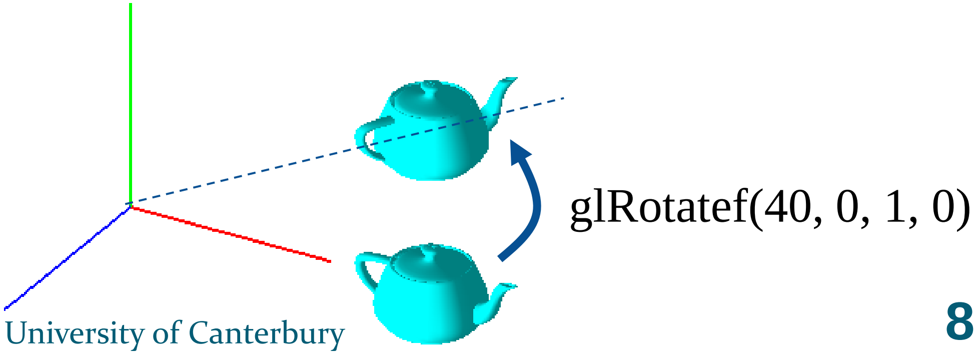
- Rotation about the *y*-axis:

$$(x, y, z) \Rightarrow (\mathrm{x}\cos\theta + \mathrm{z}\sin\theta\,,\quad y,\quad -\mathrm{x}\sin\theta + \mathrm{z}\cos\theta\,)$$

- Rotation about the *z*-axis:

$$(x, y, z) \Rightarrow (x\cos\theta - y\sin\theta,\quad x\sin\theta + y\cos\theta,\quad z)$$

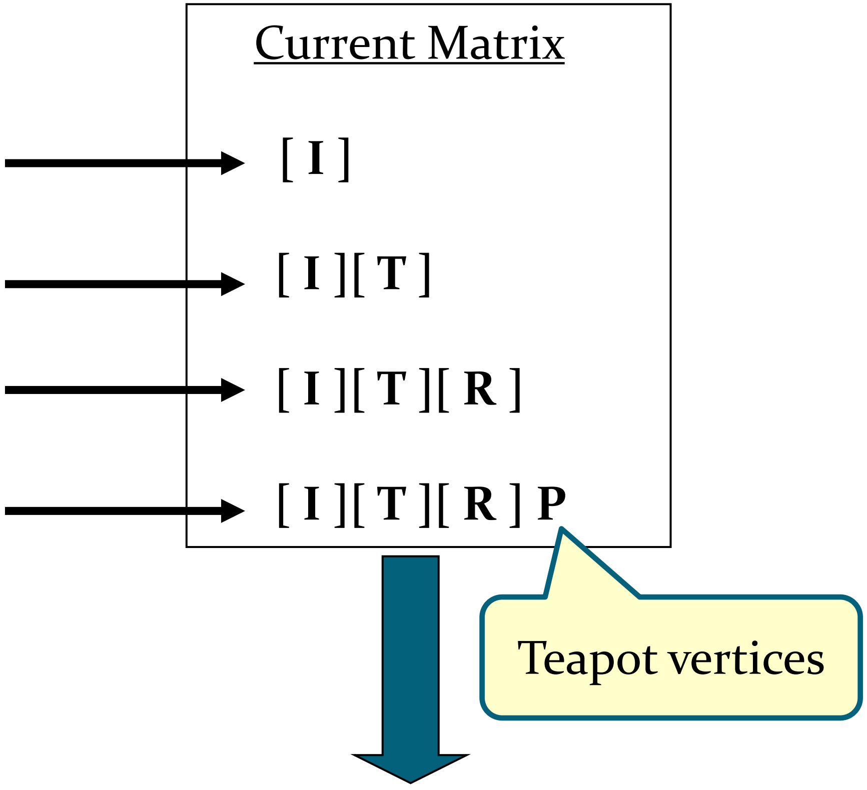
Note: Rotations are always performed about an axis through origin.



OpenGL Transformations

OpenGL ***post***-multiplies the current transformation matrix with the new transformation matrix

```
glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

glTranslatef(tx, ty, tz);

glRotatef(theta, 0, 0, 1.0);

glutSolidTeapot(1);
```



Current Matrix

[I]

[I][T]

[I][T][R]

[I][T][R] P

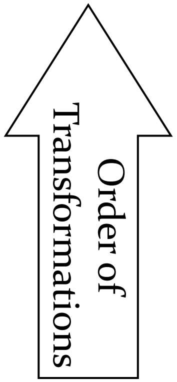
Teapot vertices

Rotation followed by translation !!

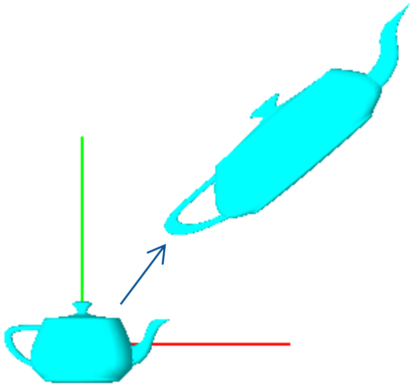
Composite Transformations

The order in which a sequence of transformations is applied to an object is very important.

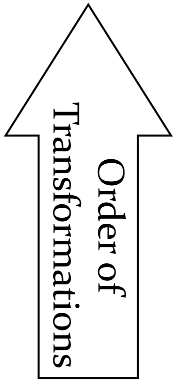
```
glTranslatef(2.0, 2.0, 0.0);
glRotatef(40.0, 0., 0.0, 1.0);
glScalef(2.0, 1.0, 0.5);
glutSolidTeapot(0.5);
```


Order of Transformations

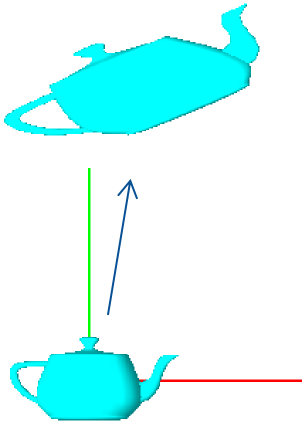
Scaling → Rotation → Translation



```
glScalef(2.0, 1.0, 0.5);
glRotatef(40.0, 0., 0.0, 1.0);
glTranslatef(2.0, 2.0, 0.0);
glutSolidTeapot(0.5);
```

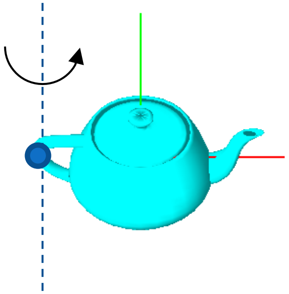

Order of Transformations

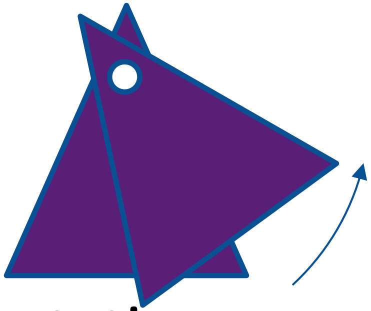
Translation → Rotation → Scaling



Rotations about a pivot point

- Rotations of an object about an axis passing through a pivot point are often required.
- A pivot point remains fixed during the rotational transformation.

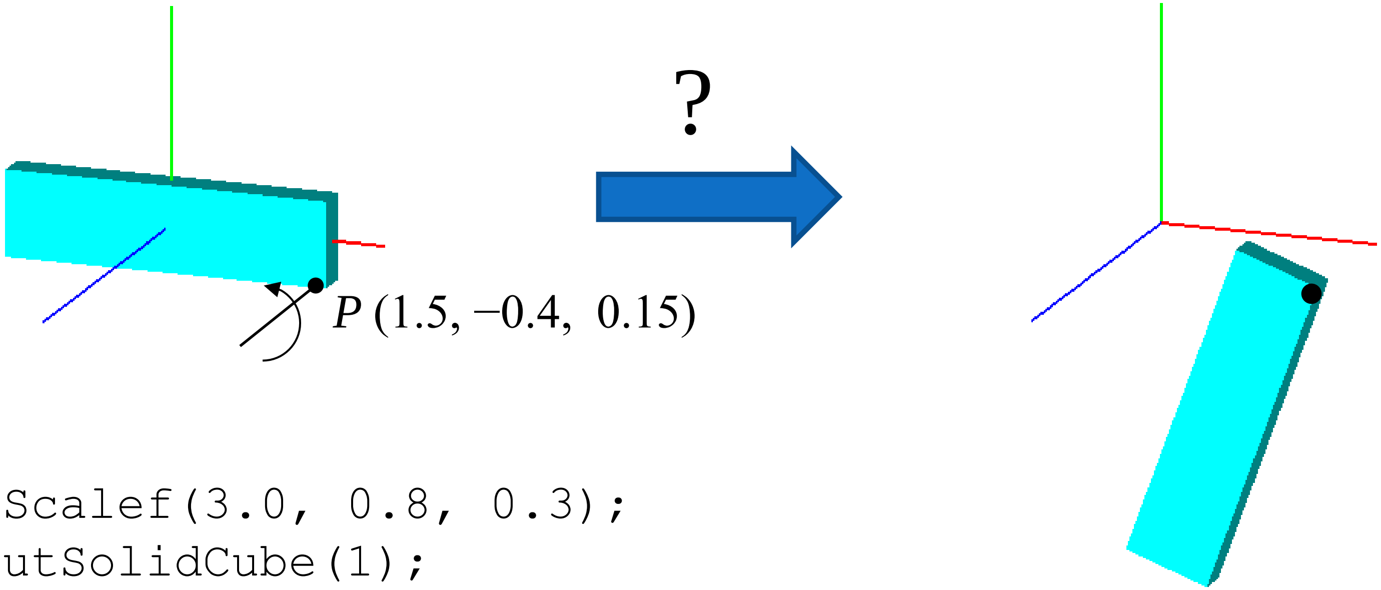




- OpenGL rotations are always performed with the origin as the pivot point
- How can we perform rotations about an arbitrary pivot point? (Eg. Rotation of an arm about the shoulder joint, rotation of a wheel about its centre)

Rotations about a pivot point

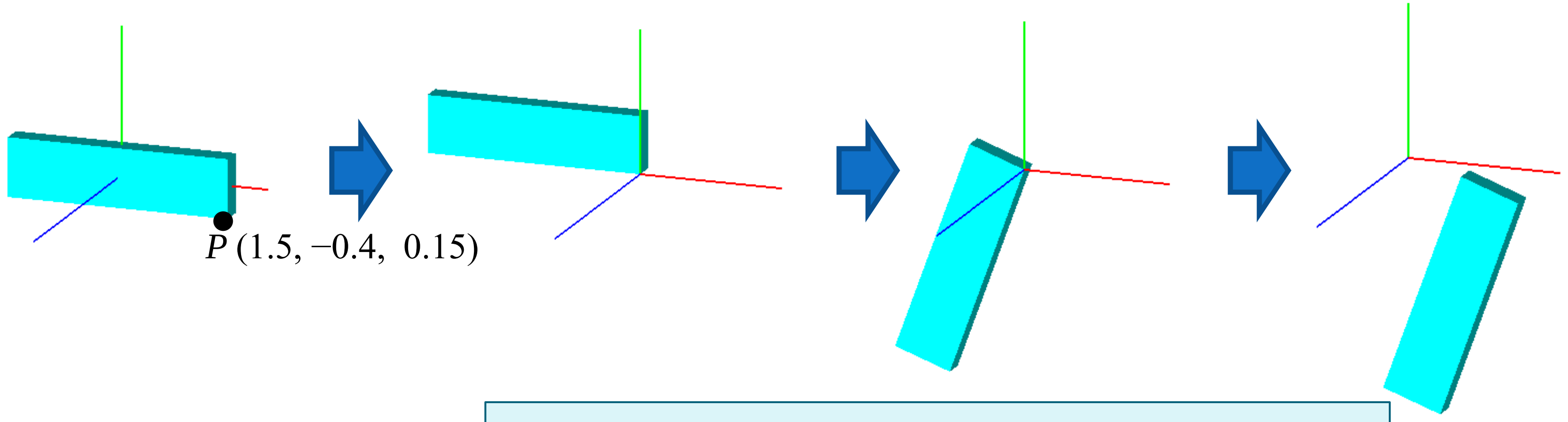
Example: We require a 70 deg rotation of the parallelopiped in the following figure about the pivot point *P* and axis of rotation parallel to *z*.



```
glScalef(3.0, 0.8, 0.3);
glutSolidCube(1);
```

Rotations about a pivot point

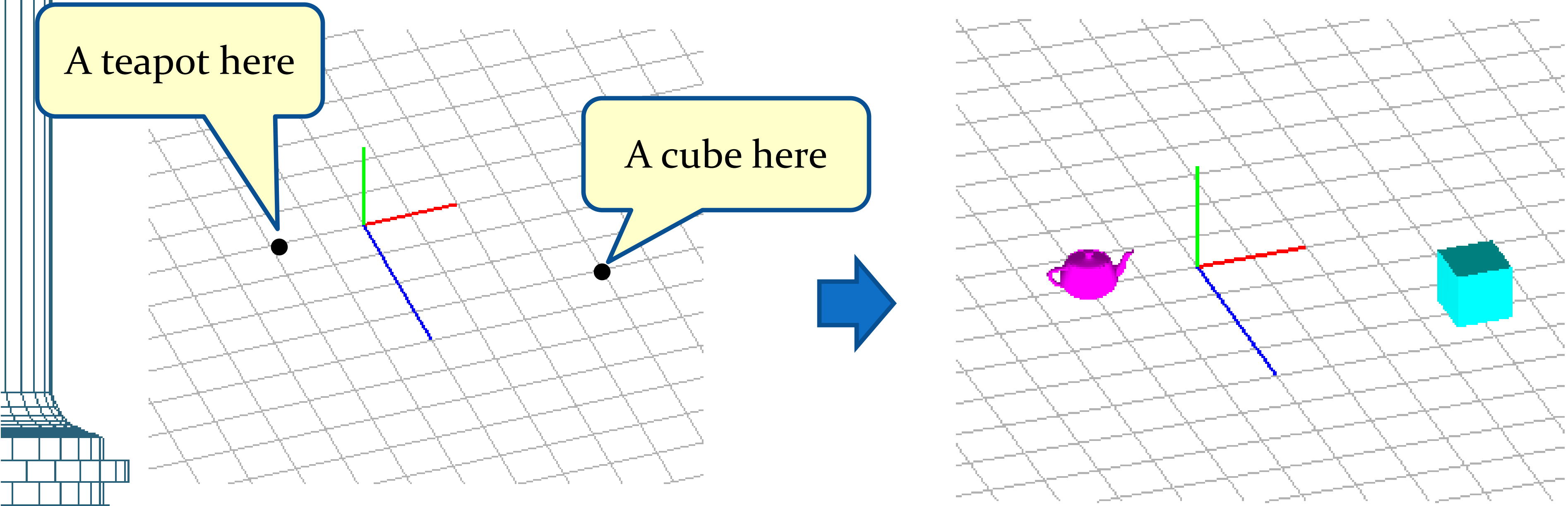
- First, translate the object such that pivot point (p_x, p_y, p_z) goes to the origin: glTranslatef($-p_x$, $-p_y$, $-p_z$)
- Perform the required rotation: glRotatef(theta, *l*, *m*, *n*)
- Translate the object so that the pivot point goes back to its original position: glTranslatef(p_x, p_y, p_z)



```
glTranslatef(1.5, -0.4, 0.15);
glRotatef(70.0, 0, 0, 1);
glTranslatef(-1.5, 0.4, -0.15);
glScalef(3.0, 0.8, 0.3);
glutSolidCube(1);
```

Independent Transformations

- Each object in a scene normally requires its *own* set of transformations
- Example: A scaled teapot at location (-2, 0, 0) and a cube at location (4, 0, 3).

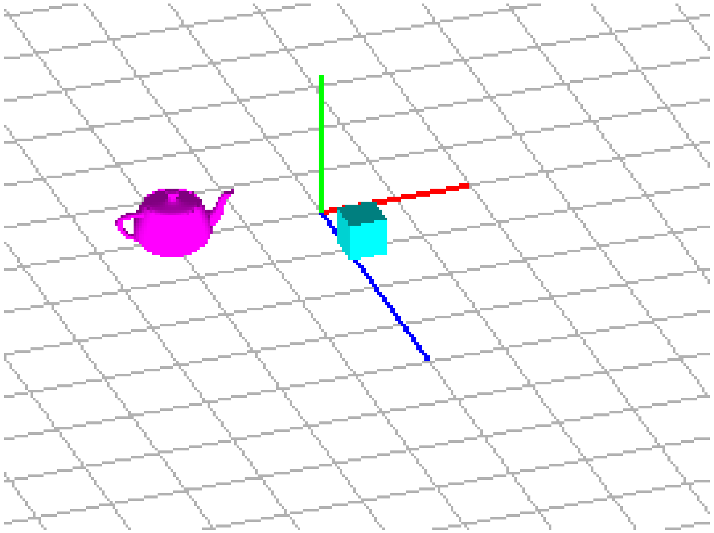


A teapot here

A cube here

Incorrect transformation

```
glTranslatef(-2.0, 0.3, 0.0);
glScalef(0.5, 0.5, 0.5);
glutSolidTeapot(1);

glTranslatef(4, 0.5, 3);
glutSolidCube(1);
```



The cube is in the wrong position, and has a reduced size!

The transformations defined for the teapot are also applied to the cube.

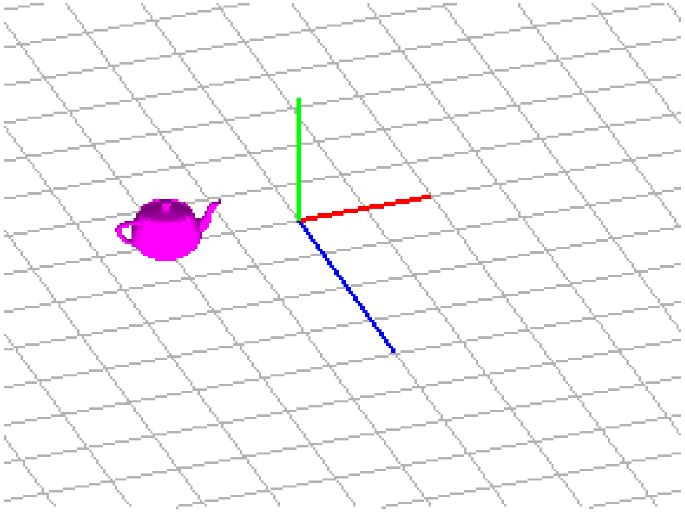
We need to isolate the transformations applied to each object.

Independent Transformations

One possible solution: Reset the transformation matrix.

Often, this method will not work. (Why?)

```
glTranslatef(-2.0, 0.3, 0.0);
glScalef(0.5, 0.5, 0.5);
glutSolidTeapot(1);

glLoadIdentity();

glTranslatef(4, 0.5, 3);
glutSolidCube(1);
```

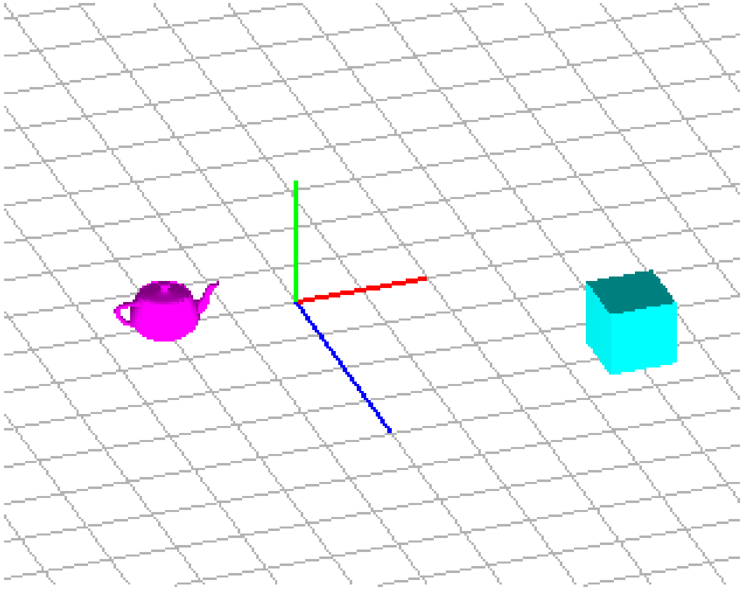


Where did the cube go?

Independent Transformations

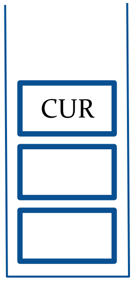
Correct approach to specifying independent transformations:

```
glPushMatrix();
  glTranslatef(-2.0, 0.3, 0.0);
  glScalef(0.5, 0.5, 0.5);
  glutSolidTeapot(1);
glPopMatrix();

glPushMatrix();
  glTranslatef(4, 0.5, 3);
  glutSolidCube(1);
glPopMatrix();
```

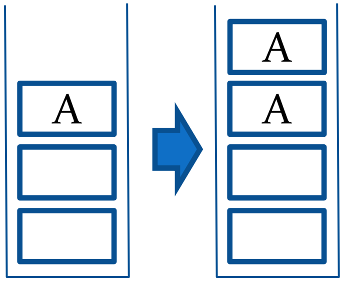


Matrix Stack

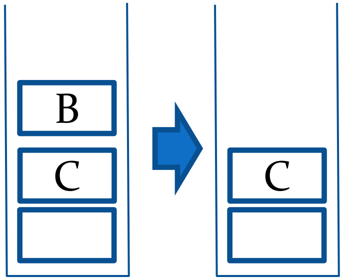
- A transformation matrix can be pushed into the matrix stack, saving it for later use.
- The top of stack represents the current transformation matrix



- The matrix stack is useful for applying different and independent sets of transformations to different objects.
- OpenGL:
  - **`glPushMatrix():`** Create a copy of the current transformation matrix and push into the stack

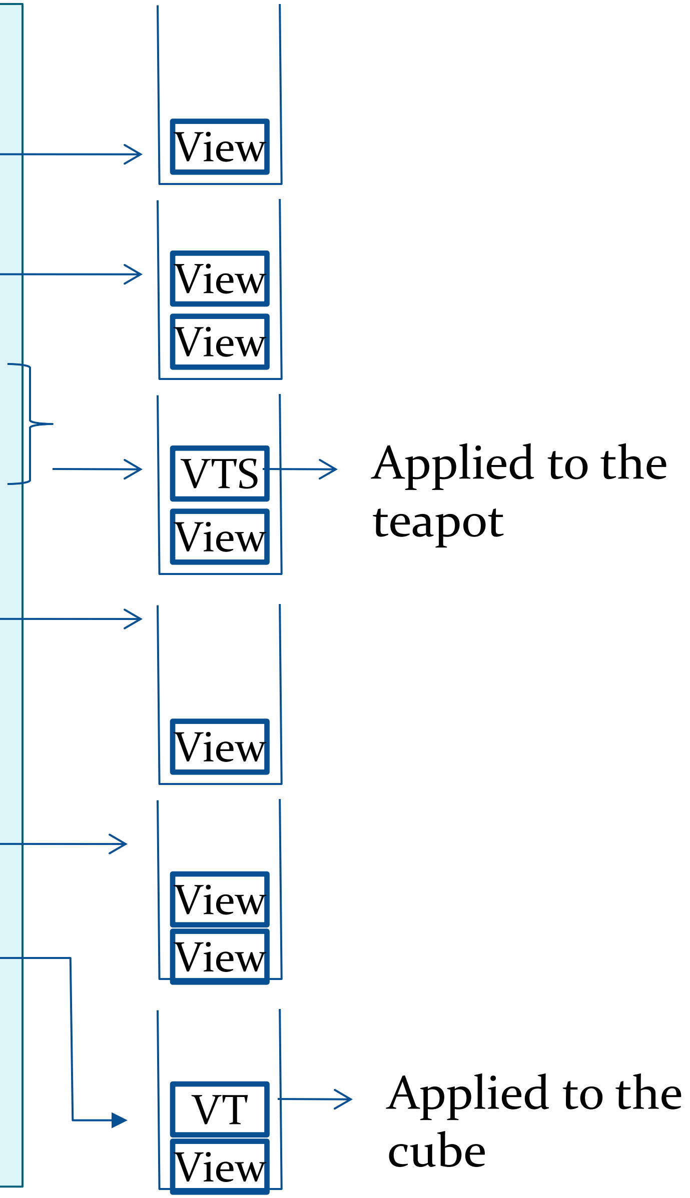


  - **`glPopMatrix():`** Remove the matrix at the top of stack



Independent Transformations

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(...);

glPushMatrix();

  glTranslatef(-2.0, 0.3, 0.0);
  glScalef(0.5, 0.5, 0.5);
  glutSolidTeapot(1);

glPopMatrix();


glPushMatrix();

  glTranslatef(4, 0.5, 3);
  glutSolidCube(1);

glPopMatrix();
```



Applied to the teapot

Applied to the cube

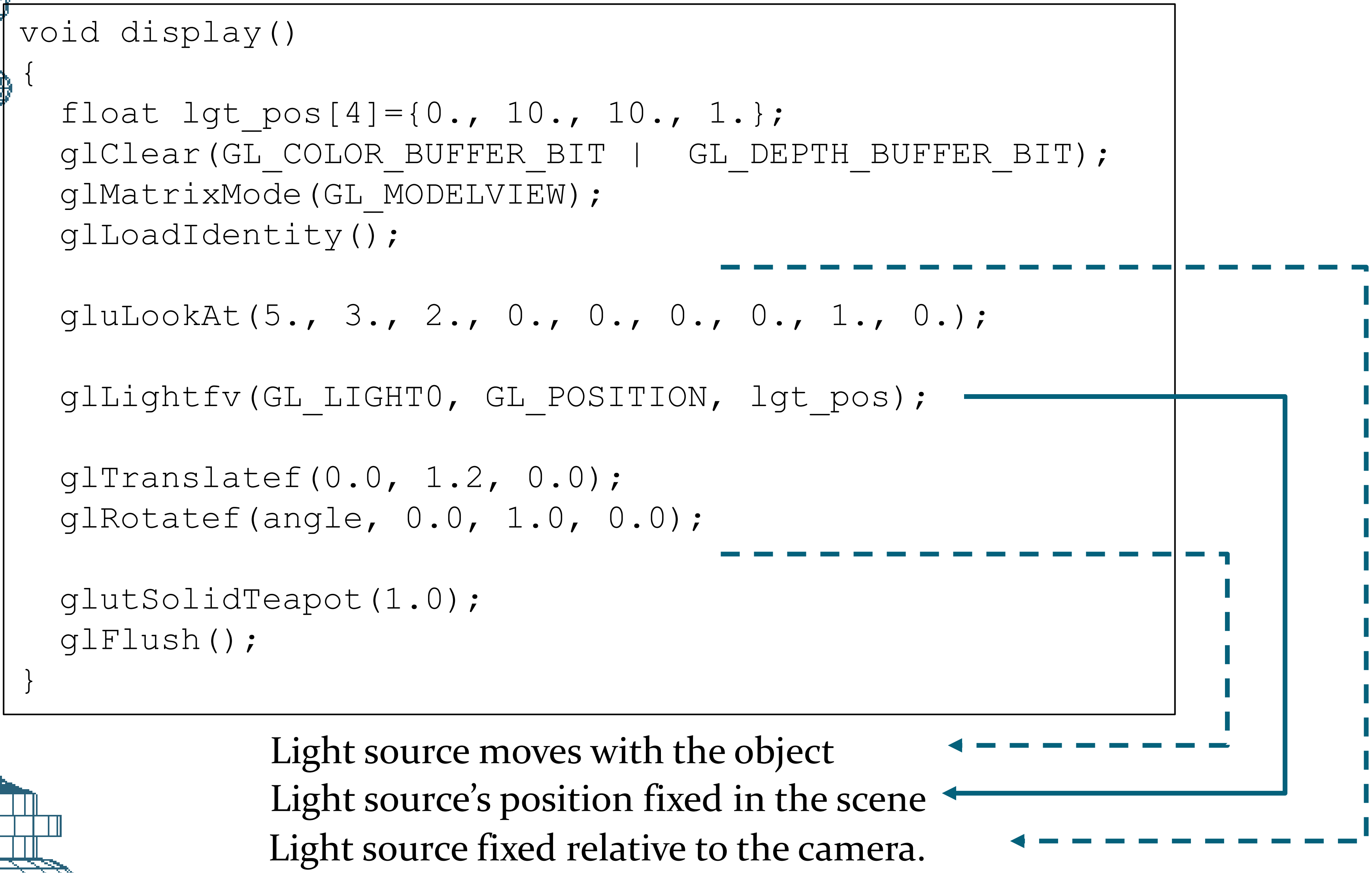
Independent Transformations

A modified example:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(...);

glRotatef(30, 0, 1, 0);

glPushMatrix();
  glTranslatef(-2.0, 0.3, 0.0);
  glScalef(0.5, 0.5, 0.5);
  glutSolidTeapot(1);
glPopMatrix();

glPushMatrix();
  glTranslatef(4, 0.5, 3);
  glutSolidCube(1);
glPopMatrix();
```

This transformation will be applied to both the teapot and the cube.

Transformation of Light Sources



```
void display()
{
   float lgt_pos[4]={0., 10., 10., 1.};
   glClear(GL_COLOR_BUFFER_BIT |  GL_DEPTH_BUFFER_BIT);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();

   gluLookAt(5., 3., 2., 0., 0., 0., 0., 1., 0.);

   glLightfv(GL_LIGHT0, GL_POSITION, lgt_pos);

   glTranslatef(0.0, 1.2, 0.0);
   glRotatef(angle, 0.0, 1.0, 0.0);

   glutSolidTeapot(1.0);
   glFlush();
}
```

Light source moves with the object

Light source's position fixed in the scene

Light source fixed relative to the camera.

A point light source is transformed like any other point