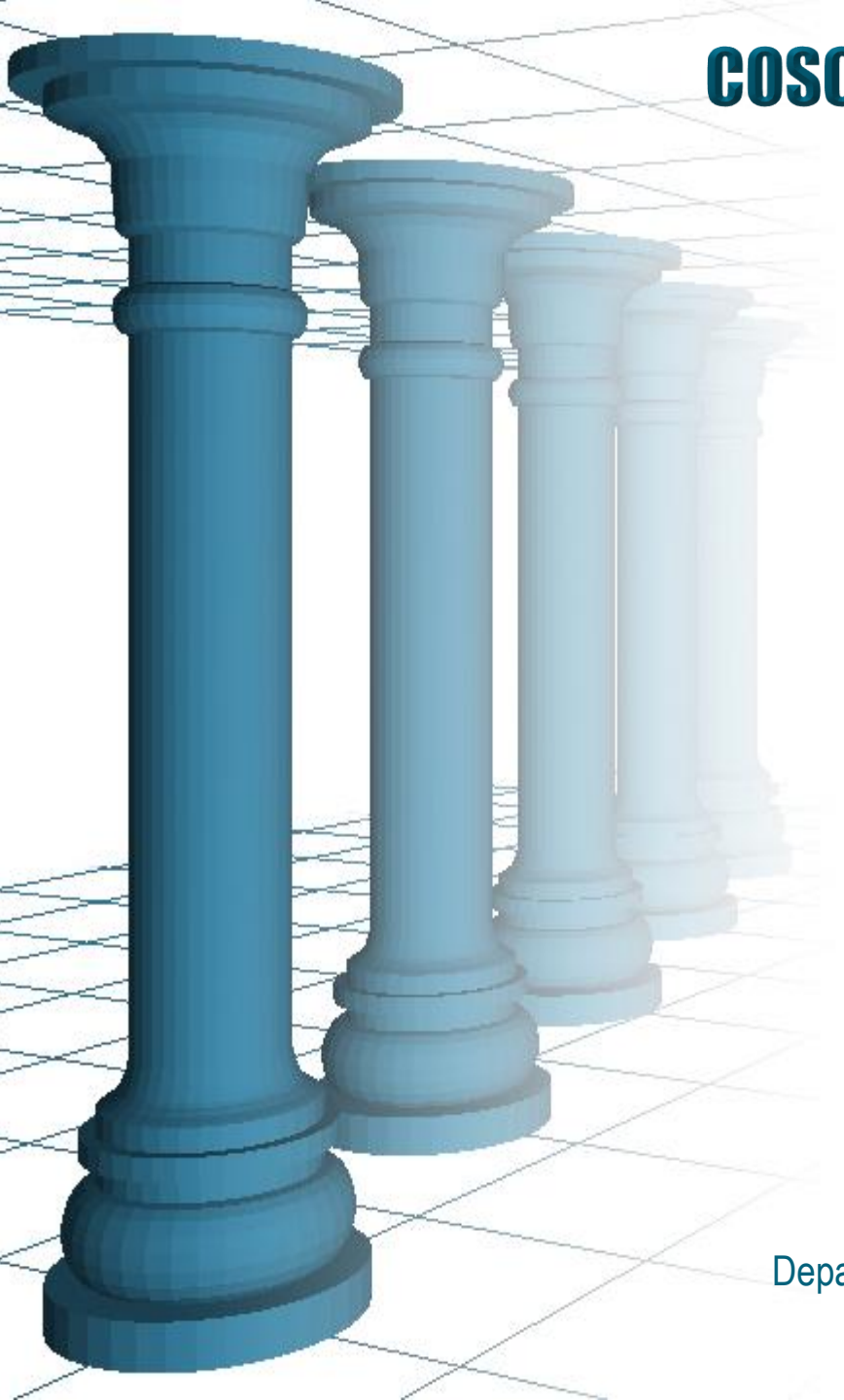


COSC363 Computer Graphics

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.

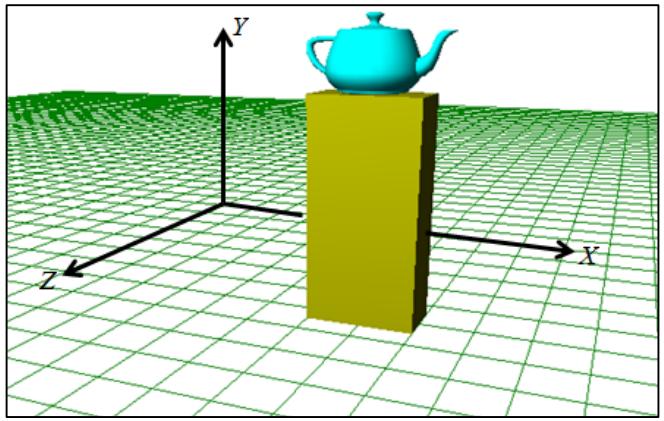
Transformations

- 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.

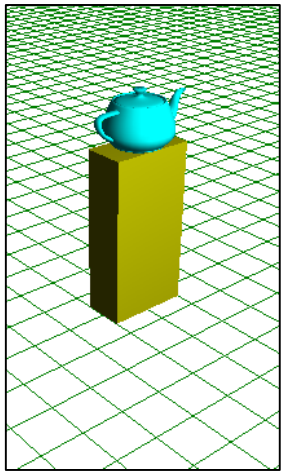
Model Transformation. Eg. **Scale, Rotation, Translation**



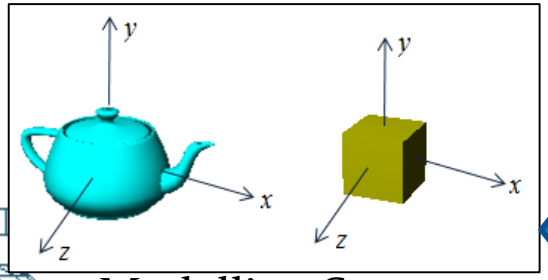
World Space

View Transformation

`gluLookAt(..)`



Camera Space



Modelling Space

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

Transformations in OpenGL

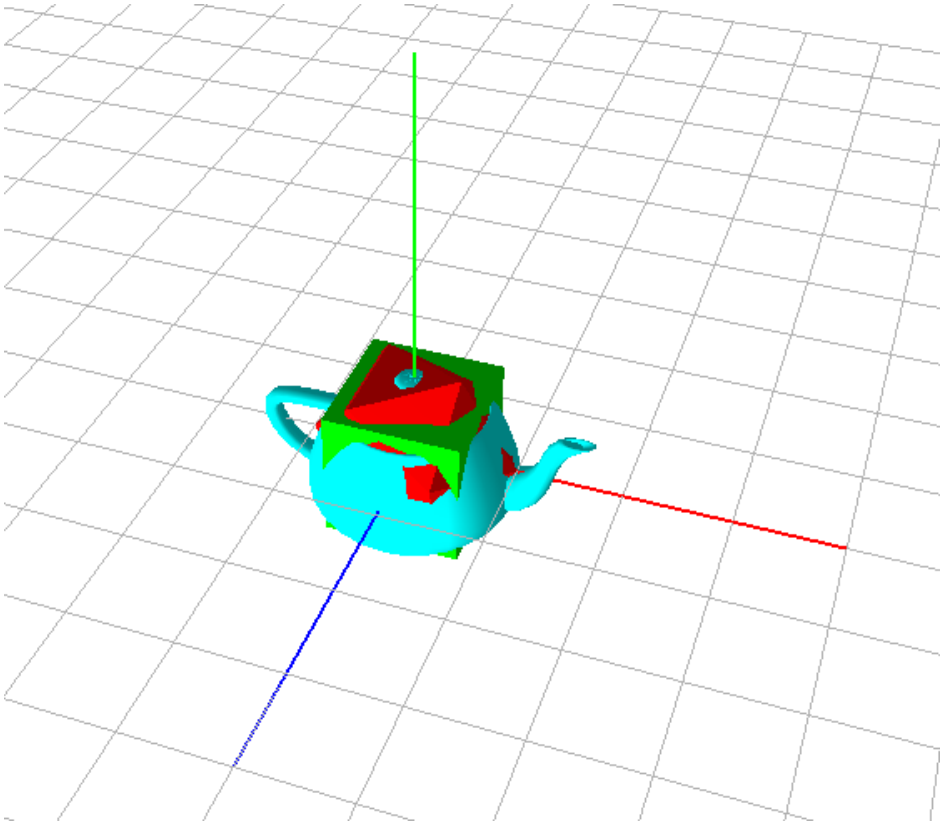
- OpenGL supports the following types of three-dimensional model transformations:
  - Translations: `glTranslatef(a, b, c);`
  - Rotations: `glRotatef(angle, l, m, n);`
  - Scale Transformations: `glScalef(sx, sy, sz);`
  - Generalized transformation: `glMultMatrixf(mat);`
- *All* transformations are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

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

Model Creation

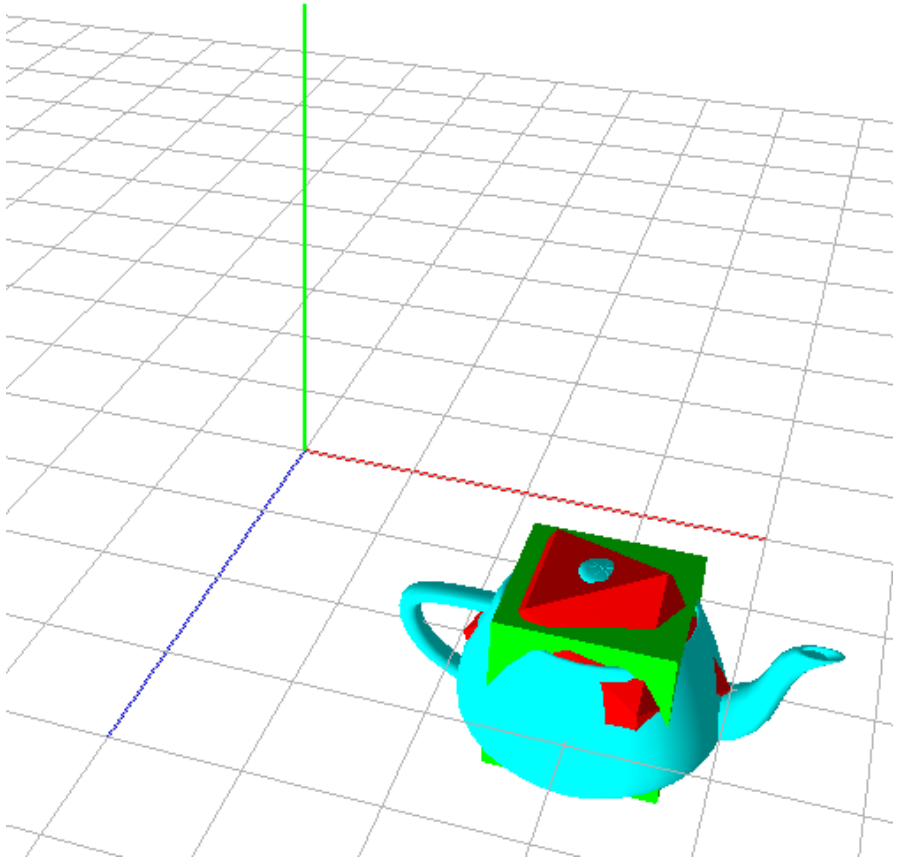
- A model's vertex coordinates are *usually* defined with the origin at its centre.
- The display of a model, without any transformation, therefore appears at or near the origin.

```
void display()
{
 ...
 ...
 glColor3f(0.,1.,1.);
 glutSolidTeapot(1);
 glColor3f(0., 1., 0.);
 glutSolidCube(1.3);
 glColor3f(1, 0, 0);
 glutSolidIcosahedron();
 glFlush();
}
```



Model Transformation

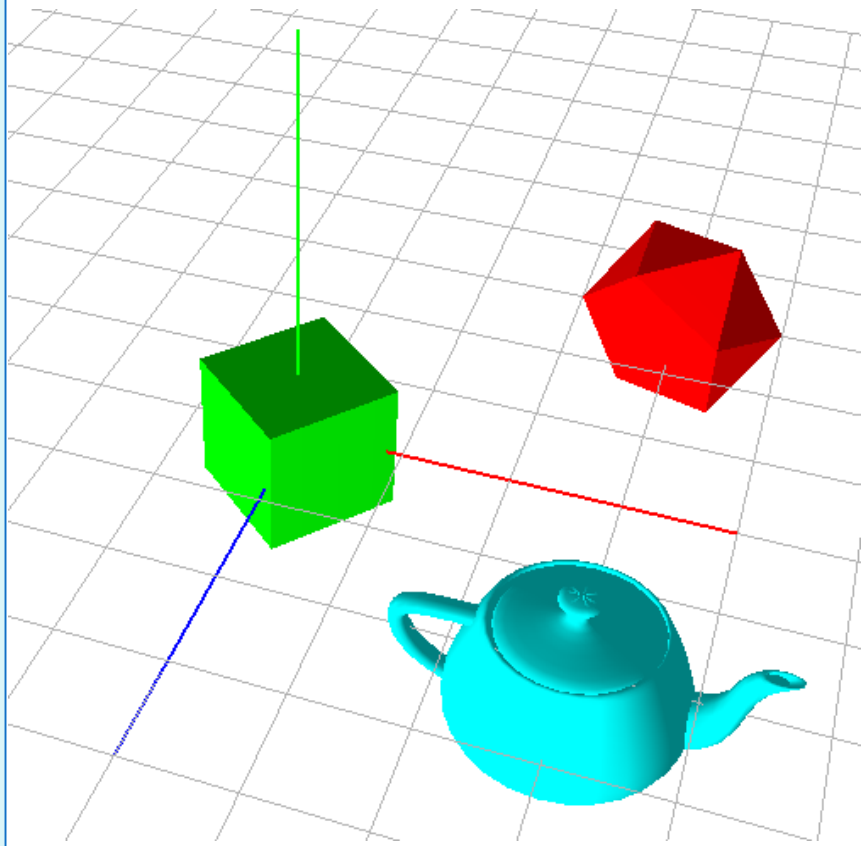
```
void display()
{
 ...
 ...
 glTranslatef(3, 0, 2);
 glColor3f(0.,1.,1.);
 glutSolidTeapot(1);
 glColor3f(0., 1., 0.);
 glutSolidCube(1.3);
 glColor3f(1, 0, 0);
 glutSolidIcosahedron();
 glFlush();
}
```



- A transformation function to translate an object by 3 units along x-direction, and 2 units along z-direction is used.
- In the above example, the transformation is applied to all three objects.

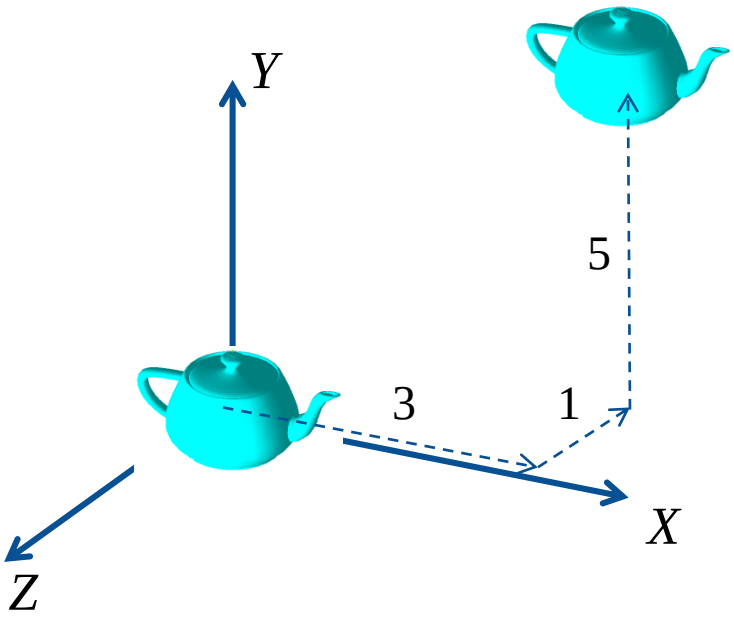
Independent Transformations

```
void display()
{
 ...
 glPushMatrix();
   glTranslatef(3, 0, 2);
   glColor3f(0.,1.,1.);
   glutSolidTeapot(1);
 glPopMatrix();

 glPushMatrix();
   glRotatef(50, 0, 1, 0);
   glColor3f(0., 1., 0.);
   glutSolidCube(1.3);
 glPopMatrix();

 glPushMatrix();
   glTranslatef(3, 0, -3);
   glColor3f(1, 0, 0);
   glutSolidIcosahedron();
 glPopMatrix();
glFlush();
}
```



Translation

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



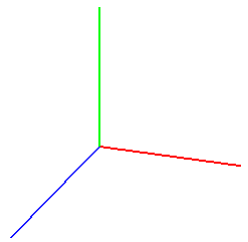
Example:

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

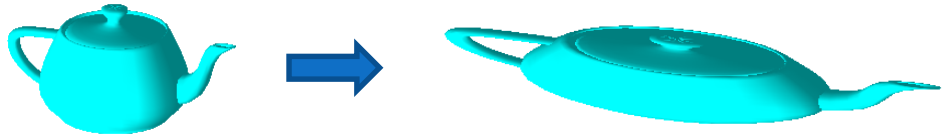
$(x, y, z) \xrightarrow{T(a, b, c)} (x + a,\ \ y + b,\ \ z + c)$

Scaling

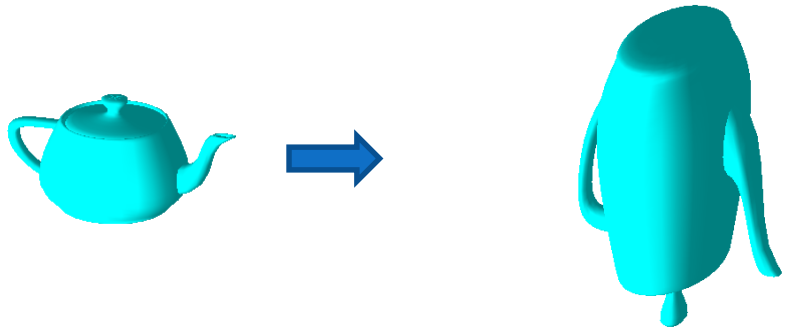
- OpenGL function: **`glScalef`**`(a, b, c)`
- A negative scale factor corresponds to a reflection.
- A zero scale factor corresponds to a projection



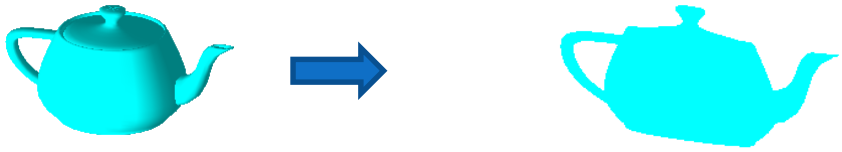
```
glScalef(2.0, 0.5, 1);
glutSolidTeapot(1);
```



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



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



$S(a, b, c)$

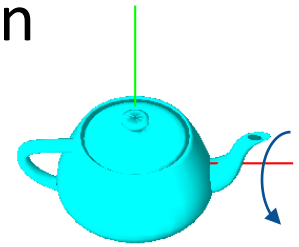
(x, y, z) → $(xa\ , yb, zc)$

Rotations

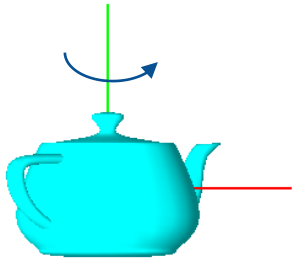
Axis of rotation

- OpenGL function: **`glRotatef`**`(theta, l, m, n)`
- A positive angle corresponds to a rotation in the anti-clockwise sense about the axis of rotation

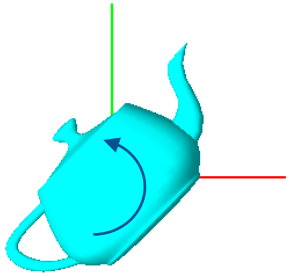
```
glRotatef(45, 1, 0, 0);
glutSolidTeapot(1);
```



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

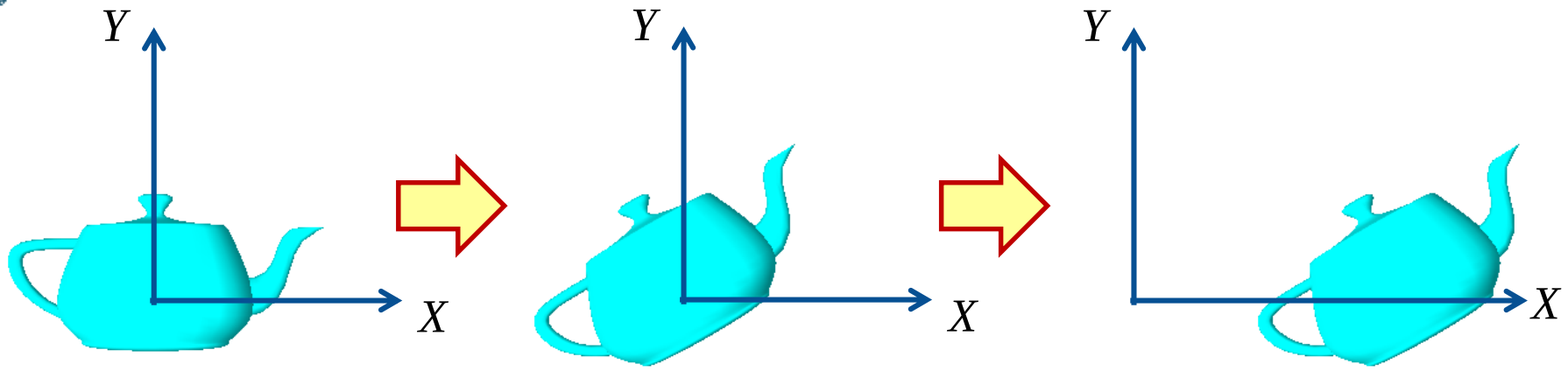


```
glRotatef(45, 0, 0, 1);
glutSolidTeapot(1);
```



Composite Transformations

Example: A rotation followed by a translation



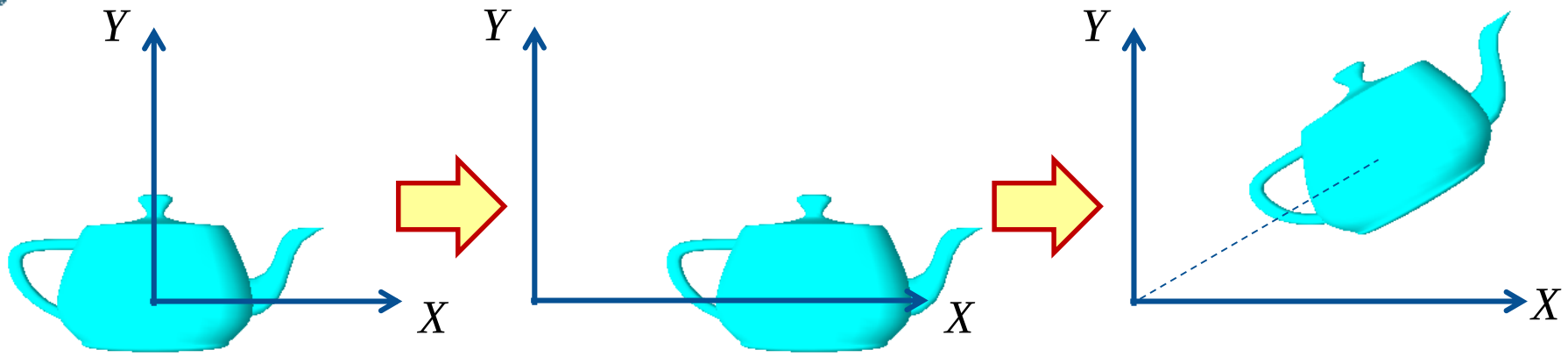
`glRotatef(30,0,0,1);` `glTranslatef(2,0,0);`

```
glTranslatef(2., 0., 0.);
glRotatef(30., 0., 0., 1.);
glutSolidTeapot(1);
```

Order of Transformations

Composite Transformations

Example: A translation followed by a rotation



`glTranslatef(2,0,0);`

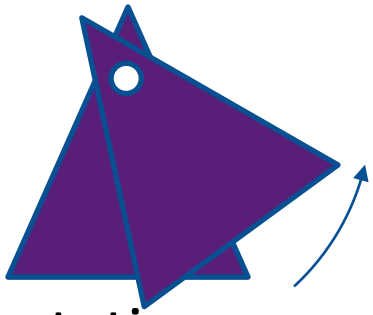
`glRotatef(30,0,0,1);`

```
glRotatef(30., 0., 0., 1.);
glTranslatef(2., 0., 0.);
glutSolidTeapot(1);
```

Order of Transformations

Rotations about a pivot point

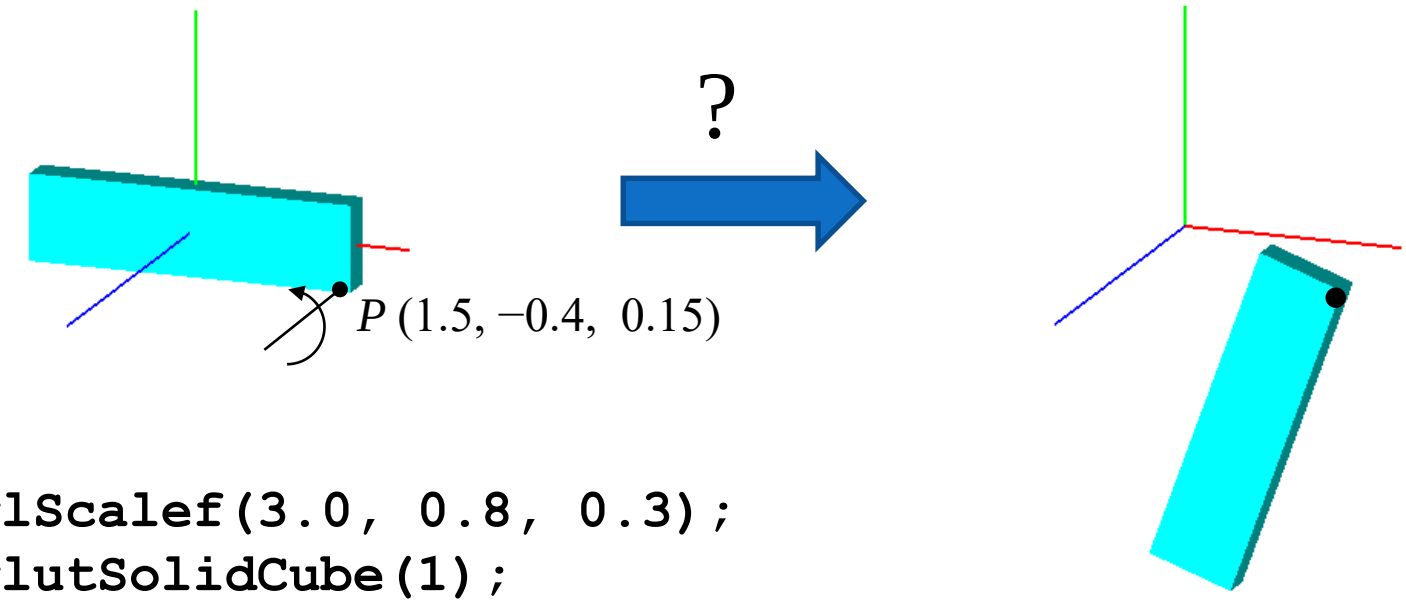
- Rotations of an object about an axis passing through a pivot point are often required.
- A pivot point remains fixed during the rotational transformation.



- 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

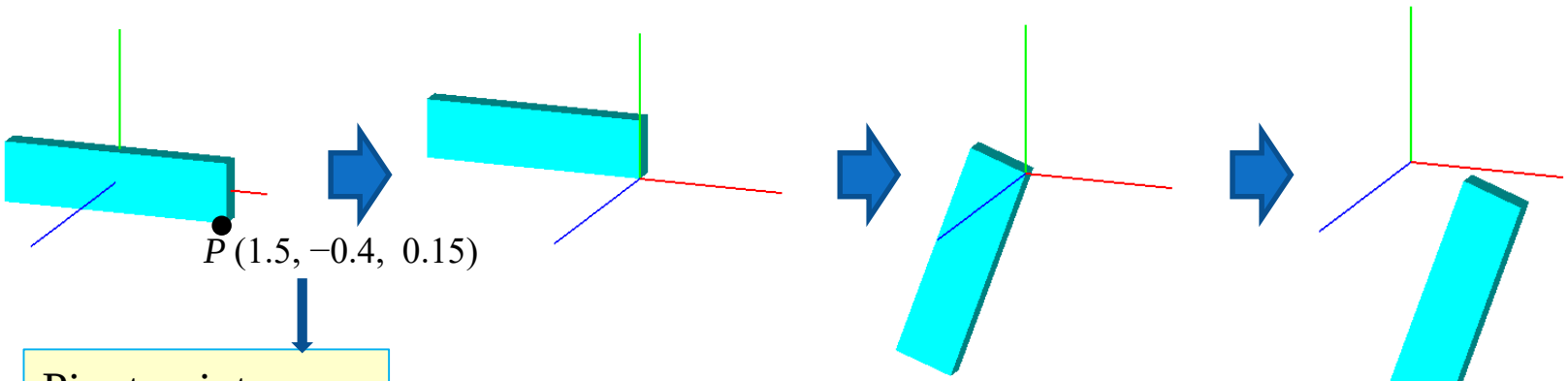
Example: We require a 70 deg rotation of the parallelopiped in the following figure about the pivot point *P* and axis of rotation parallel to *z*.




```
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)



Pivot point = (1.5, −0.4, 0.15)

sign reversed

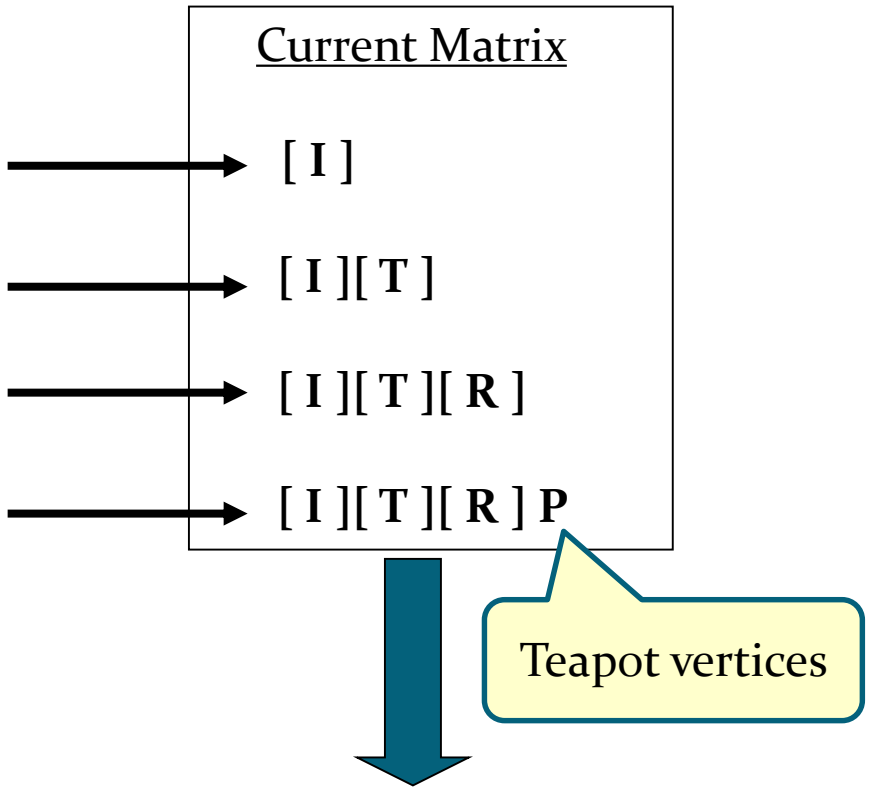
```
glTranslatef(1.5, -0.4, 0.15);
glRotatef(70.0, 0, 0, 1);
glTranslatef(-1.5, 0.4, -0.15);
glScalef(3.0, 0.8, 0.3);
glutSolidCube(1);
```

Trivia Quiz

OpenGL Transformations

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

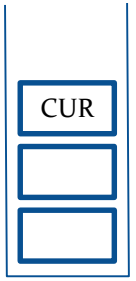
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(tx, ty, tz);
glRotatef(theta, 0, 0, 1.0);
glutSolidTeapot(1);
```



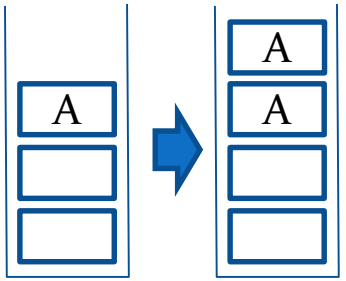
Rotation followed by translation !!

Matrix Stack

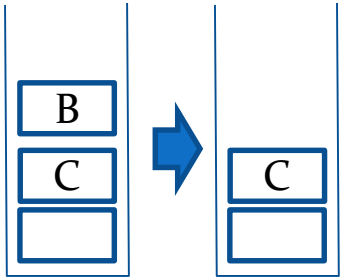
- A transformation matrix can be pushed into the matrix stack, saving it for later use.
- The top of stack represents the current transformation matrix



- 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

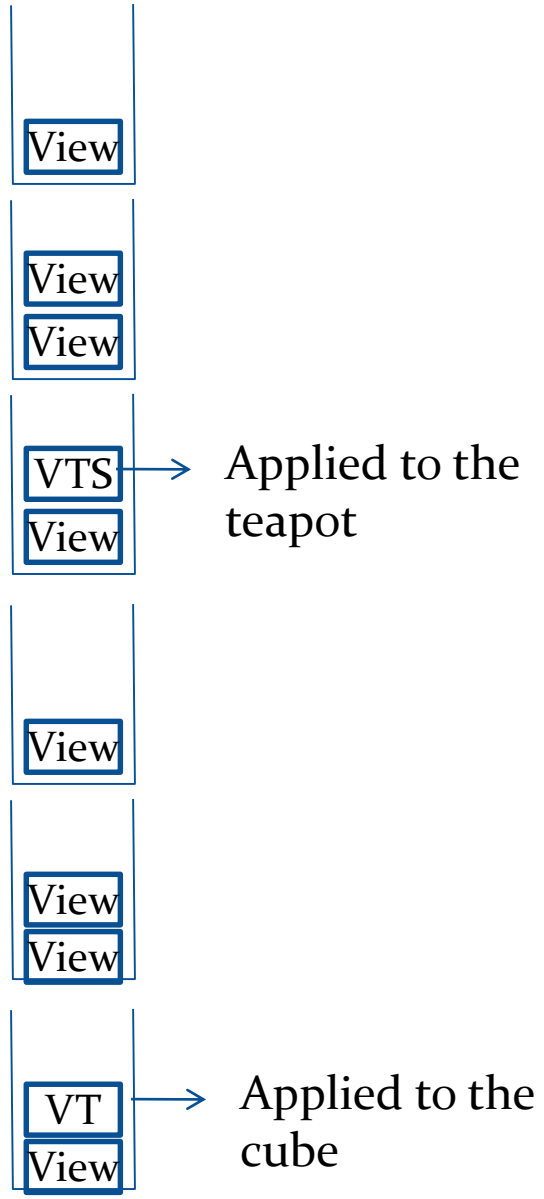


  - **`glPopMatrix()`**: Remove the matrix at the top of stack



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();
```



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