CSCI3260 PRINCIPLES OF COMPUTER GRAPHICS

Tutorial 2

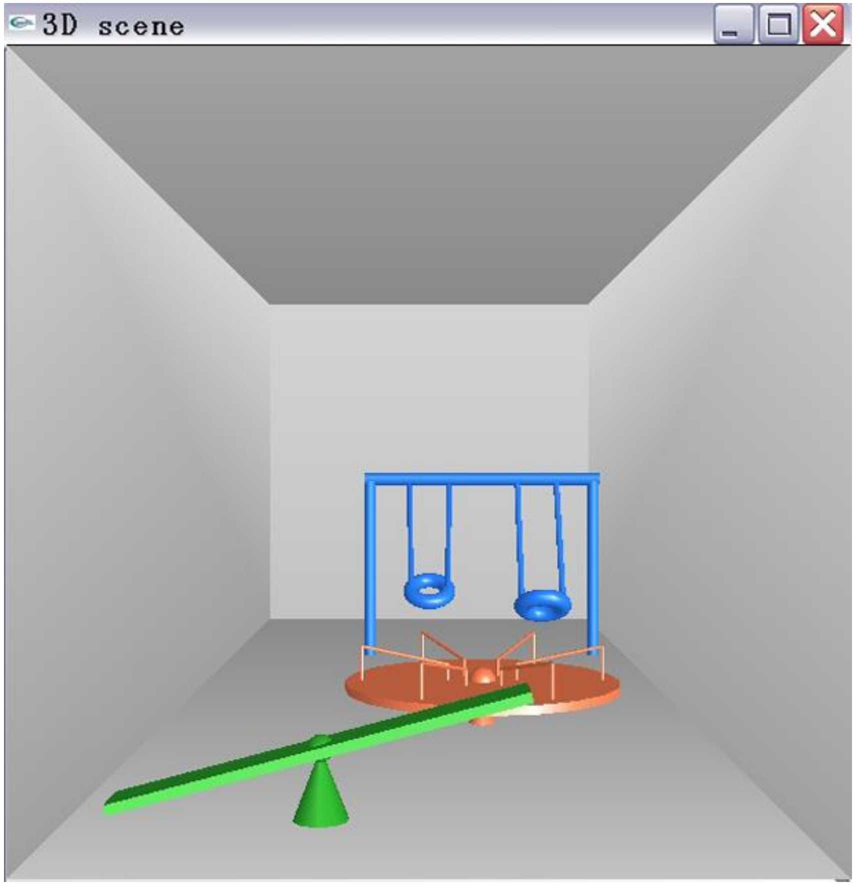
Colman Leung

Outline

- Introduction on Assignment 1
- OpenGL Transformation

Assignment 1

- Creating a 3D scene
  - Draw objects using different geometric primitives
  - Simple animation



Assignment 1

- You will experience with
  - Transformation
  - Draw 3D objects
  - Color and Lighting
  - Handling keyboard events
  - Basic animation

Assignment 1

- Details about assignment 1
  - Announced: 19th January 2016
  - Deadline: 15th February 2016

  - Setup OpenGL environment on you machine

  - We will give you an assignment kit which includes some basic OpenGL function for you to get started. All you need to do is to add some lines to fulfill the requirement of the assignment.

Assignment 1

- Primitive Drawing
  - Planes (left wall, right wall, ceiling, floor, back wall)

Tutorial 1

- Modeling Transformation
  - Rotation, Translation, and Scaling
  - Matrix Stacks

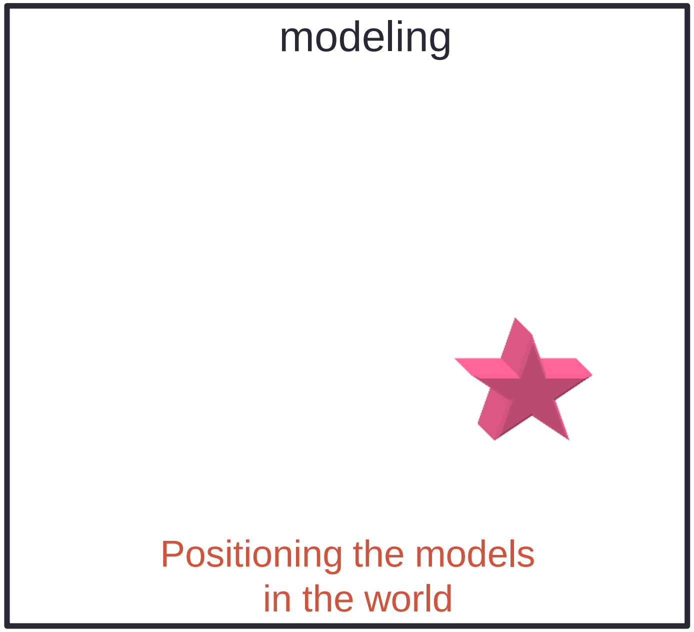
Tutorial 2

- Perspective & Orthographic Projection
- Complex 3D objects drawing
- Lighting & Material
- Keyboard and mouse interaction
- Animation

OpenGL Transformation

The transformation process is analogous to taking a photo with camera

1. Arrange the scene into the desired composition: Modeling transformation



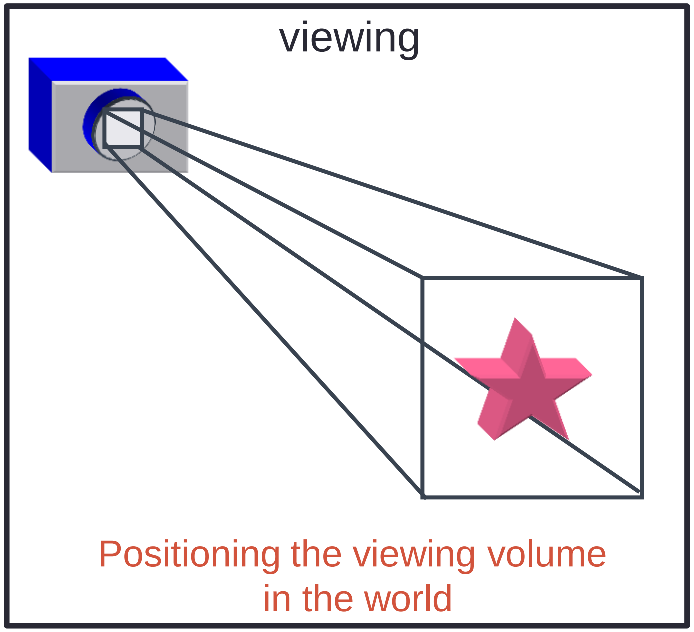
modeling

Positioning the models in the world

OpenGL Transformation

The transformation process is analogous to taking a photo with camera

1. Arrange the scene into the desired composition: Modeling transformation
2. Point the camera at the scene: Viewing transformation

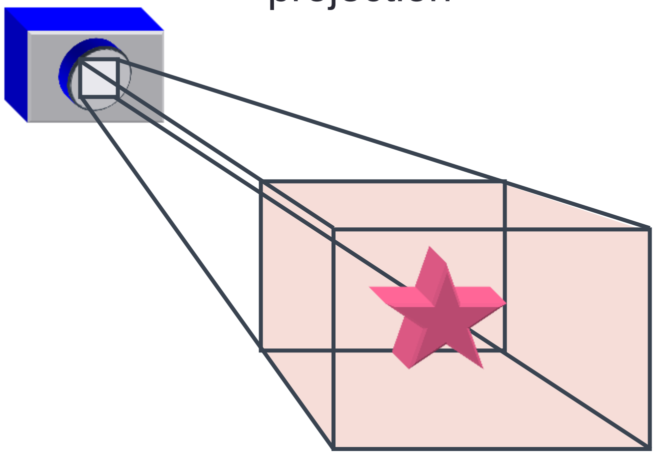


OpenGL Transformation

The transformation process is analogous to taking a photo with camera

1. Arrange the scene into the desired composition: Modeling transformation
2. Point the camera at the scene: Viewing transformation
3. Adjust the lens of the camera: Projection transformation

- projection

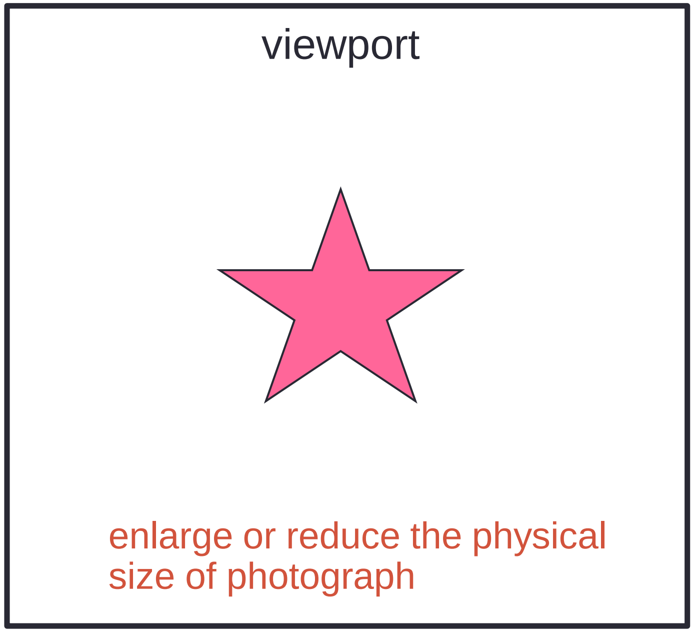


Determining shape of view volume

OpenGL Transformation

The transformation process is analogous to taking a photo with camera

1. Arrange the scene into the desired composition: Modeling transformation
2. Point the camera at the scene: Viewing transformation
3. Adjust the lens of the camera: Projection transformation
4. Determine how large your photo to be: Viewport transformation



viewport

enlarge or reduce the physical size of photograph

Matrix operations

- The OpenGL transformations (viewing/model/projection/viewport transformation) are achieved by the 4×4 matrix operations.

- Before matrix operation, we need to specify which transformation to be operated.

General-Purpose Transformation Commands

void glMatrixMode**(GLenum *mode*);**

Specify which matrix will be modified, then subsequent transformation commands affect the specified matrix

mode	matrix
GL_MODELVIEW →	modeview matrix
GL_PROJECTION →	projection matrix
GL_TEXTURE →	texture matrix

void glLoadIdentity**(void);**

Set the currently modifiable matrix to the 4X4 identity matrix, i.e. reset the matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

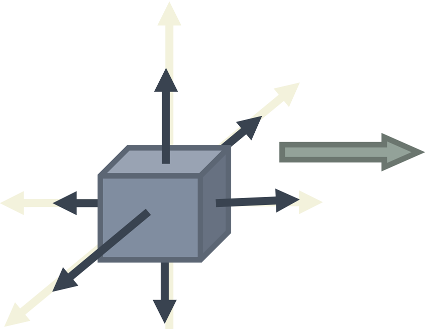
Viewing and Modeling Transformations

- Viewing and modeling transformation are combined into a single modelview matrix
- Move the camera in one direction = Move the object in opposite direction
- Call **glMatrixMode(GL_MODELVIEW)** before performing modeling or viewing transformation

Three modeling transformations

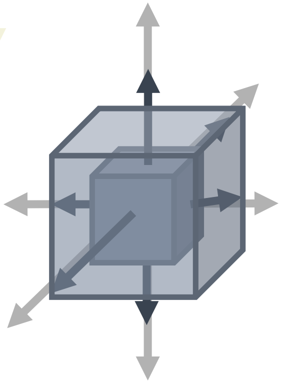
- **glTranslate{fd}(*x, y, z*)**
  Translate the object by (x,y,z).

```
glTranslatef(5.0f, 0.0f, 0.0f)
glutSolidCube(1.0f);
```



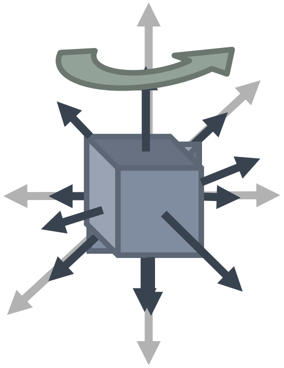
- **glScale{fd}(*x, y, z*)**
  Scale the object by factor (x,y,z).

```
glScalef(2.0f, 2.0f, 2.0f)
glutSolidCube(1.0f);
```



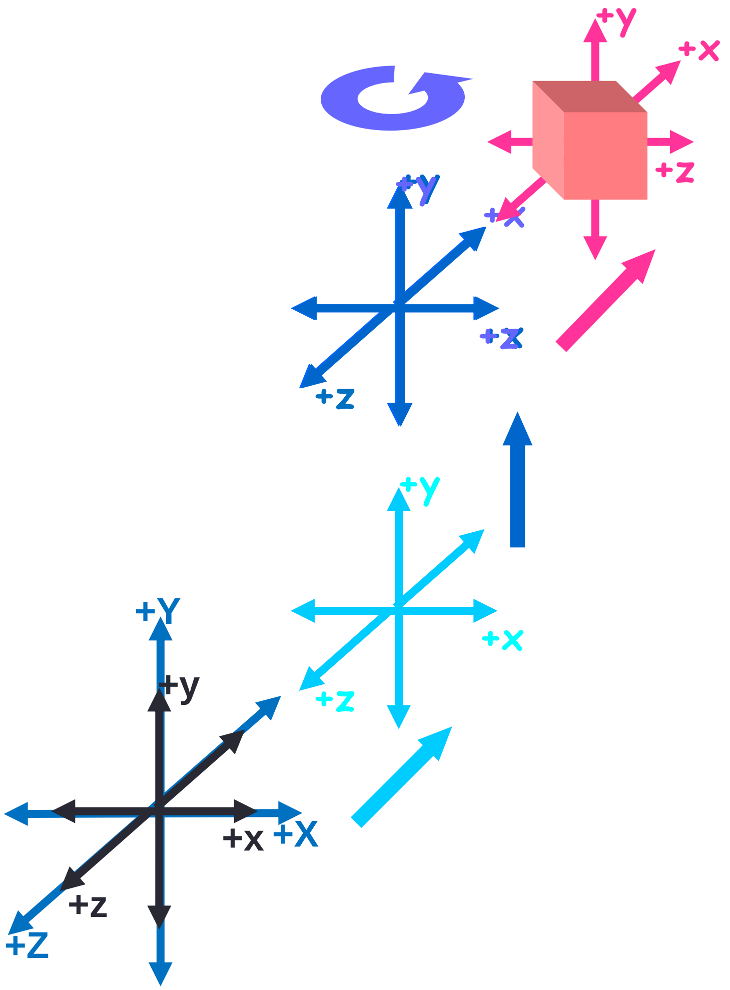
- **glRotate{fd}(angle, *x, y, z*)**
  Rotate the object about vector (x, y, z) by angle (degree)

```
glRotatef(45.0f, 0.0f, 1.0f, 0.0f)
glutSolidCube(1.0f);
```



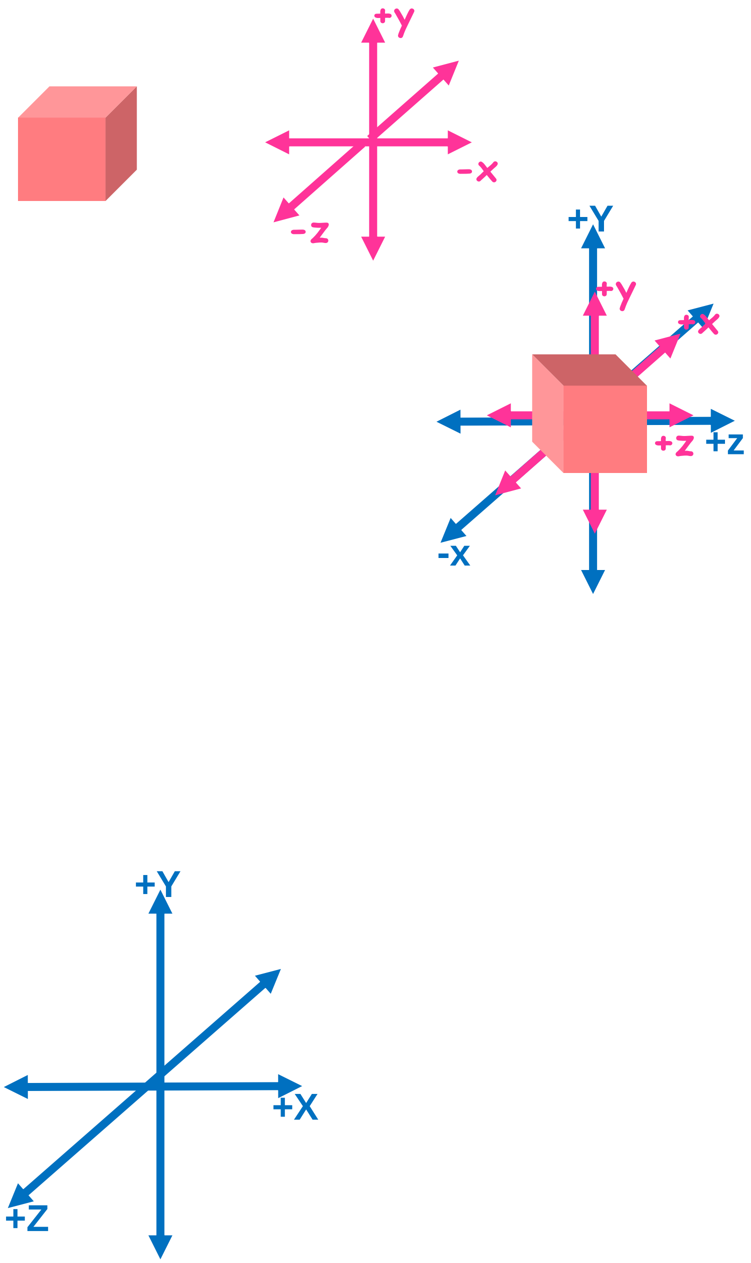
3D Transformation (An example)

```
void display()
{
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslated(0.0, 0.0, -2.0);
  glTranslated(0.0, 2.0, 0.0);
  glRotated(90, 0, 1, 0);
  glTranslatef(1.0, 0, 0);
  glutSolidCube(1.0);
}
```



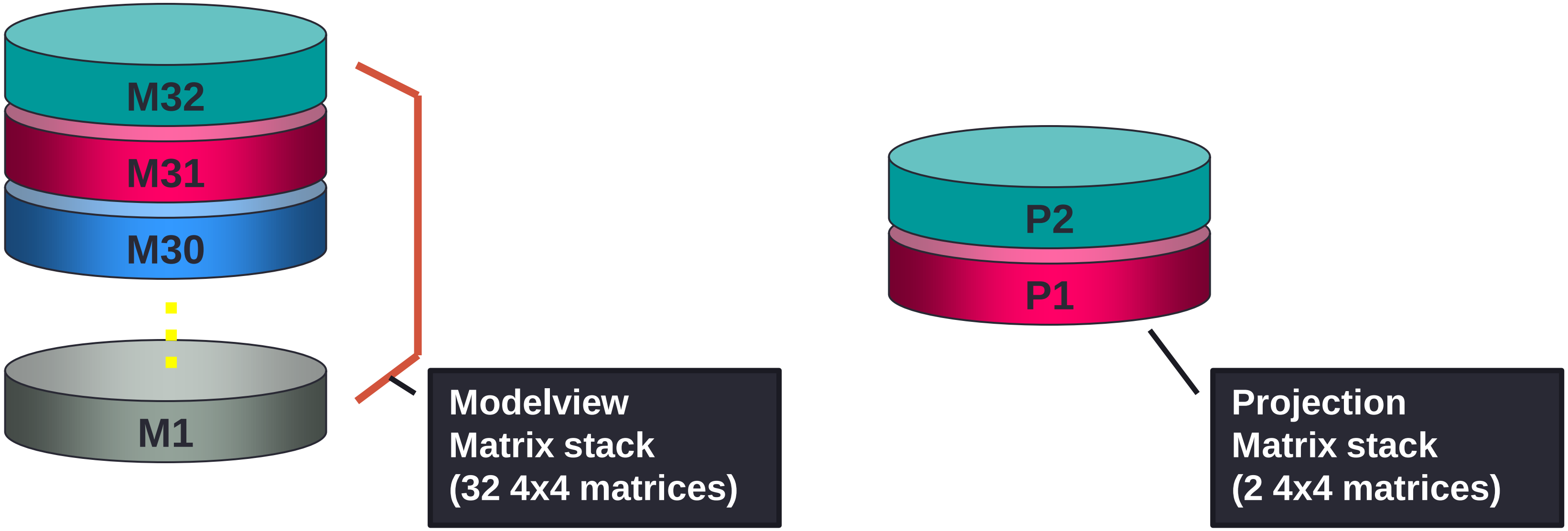
3D Transformation

```
void display()
{
  glMatrixMode(GL_MODELVIEW);
  //glLoadIdentity();
  glTranslated(0.0, 0.0, -2.0);
  glTranslated(0.0, 2.0, 0.0);
  glRotated(90, 0, 1, 0);
  glTranslatef(1.0, 0, 0);
  glutSolidCube(1.0);
}
```



Manipulating the Matrix Stacks

- ModelView and Projection matrices are actually the topmost member of their stack of matrices



Modelview
Matrix stack
(32 4x4 matrices)

Projection
Matrix stack
(2 4x4 matrices)

❖ Understanding the mechanism of stack of matrices is useful for constructing multiple models

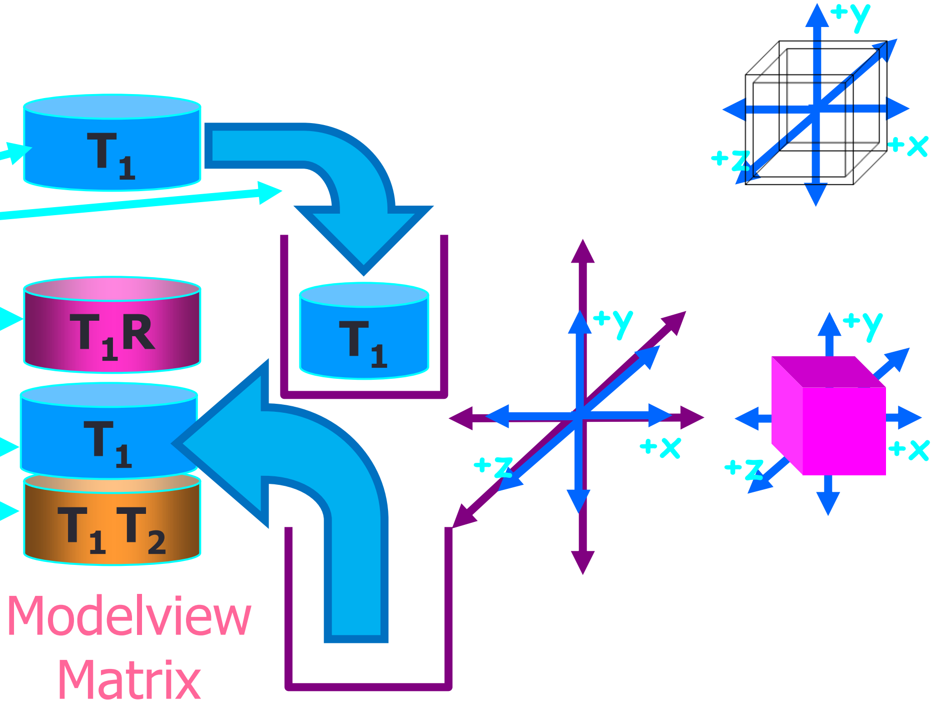
glPushMatrix() and glPopMatrix()

- glPushMatrix()
  - Copy the current matrix and adds the copy to the top of the stack
  - “remember where you are”

- glPopMatrix()
  - Discard the top matrix on the stack
  - “go back to where you were”

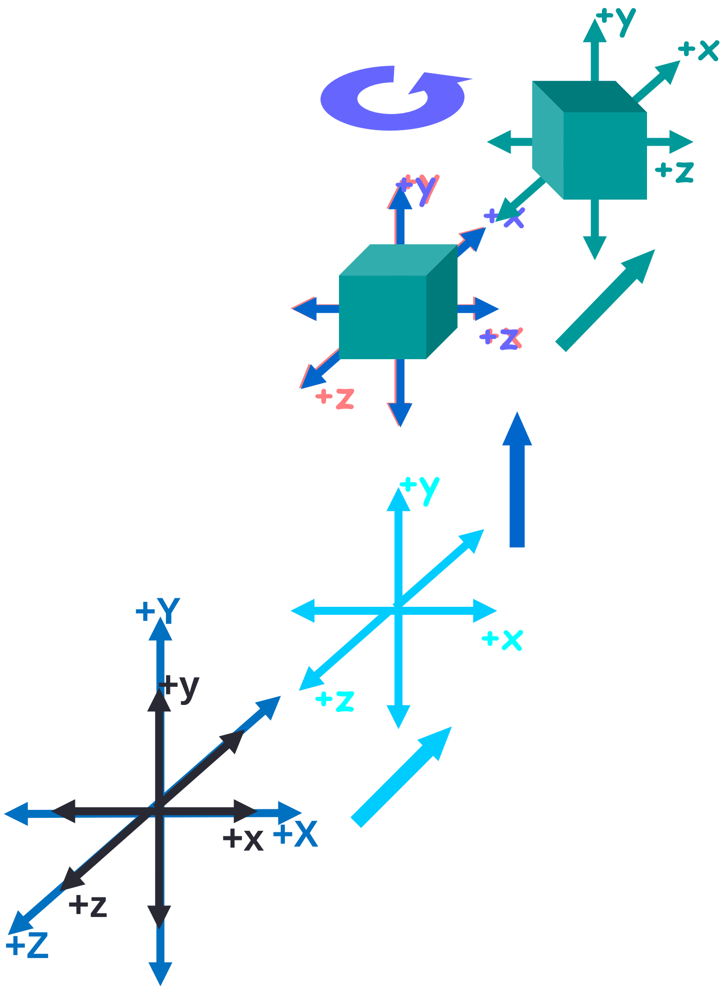
glPushMatrix() and glPopMatrix()

```
glTranslated(3.0, 0.0, 0.0);
glPushMatrix();
glRotated(90.0, 0.0, 1.0, 0.0);
glutSolidCube(1.0);
glPopMatrix();
glTranslated(0.0, 3.0, 0.0);
glutWireCube(2.0);
```



Another example

```
void display()
{
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslated(0.0, 0.0, -2.0);
  glTranslated(0.0, 2.0, 0.0);
  glPushMatrix();
    glRotated(90, 0, 1, 0);
    glTranslatef(1.0, 0, 0);
    glutSolidCube(1.0);
  glPopMatrix();
  glutSolidCube(1.0);
}
```



Next Tutorial

- Complex 3D objects drawing (GLU & GLUT Lib)
- Perspective and Orthographic Projection
  http://www.glprogramming.com/red/chapter03.html
- Lighting
  http://www.glprogramming.com/red/chapter05.html

The End