Computer Graphics

Lecture 07

OpenGL Introduction

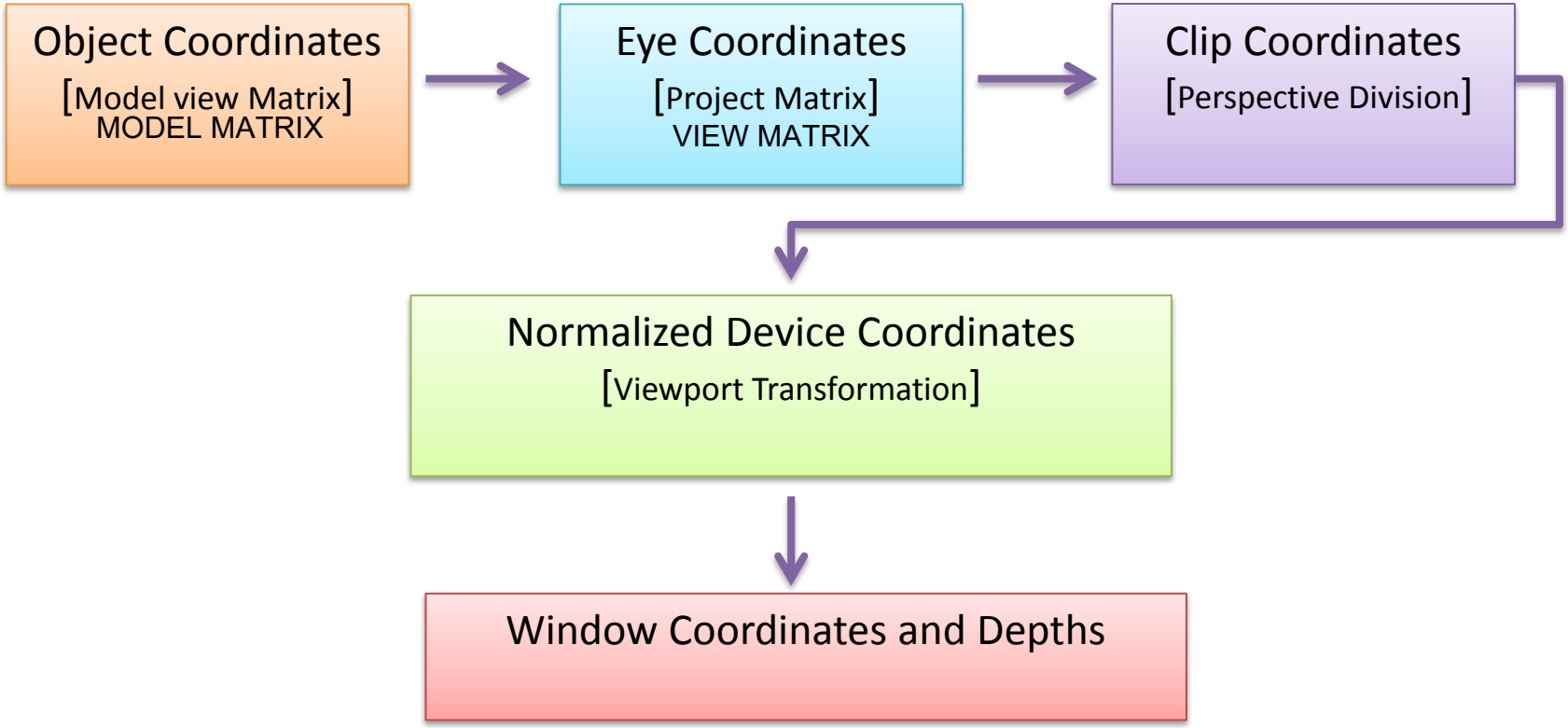
The most essential aspect of OpenGL is the vertex pipeline described in Chapter 3 of the Redbook. Objects are represented by a set of vertices in user deffined object coordinates, and these vertices are converted to window coordinates by a sequence of four transformations (state information). The primary goal of this course is to convey an understanding of these transformations.

The transformations of interest in graphics are linear transformations (scaling, rotation, refflection, shear, and orthographic projection), affine transformations (linear transformations, along with translation), and projective transformations (central projection). These transformations can be understood at three levels.

- The visual aspect of the geometry
- The algebraic matrix representation and operations
- The internal workings of the vertex pipeline

OpenGL Vertex Pipeline

The focus of these notes is on the vertex pipeline:



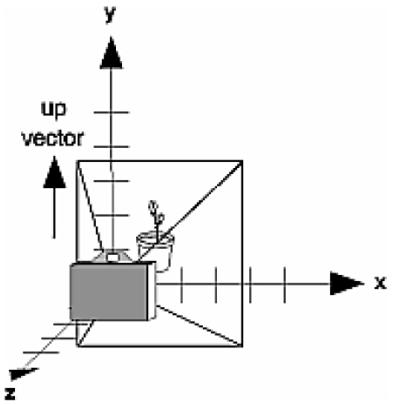
Modelview Matrix

The modelview matrix represents a sequence of *modeling transformations* followed by a sequence of *viewing transformations*.

The **modeling transformations** typically include scaling, rotation, and translation.

A tricycle might be rendered by drawing a wheel (perhaps stored as a display list) three times with different scaling and translations.

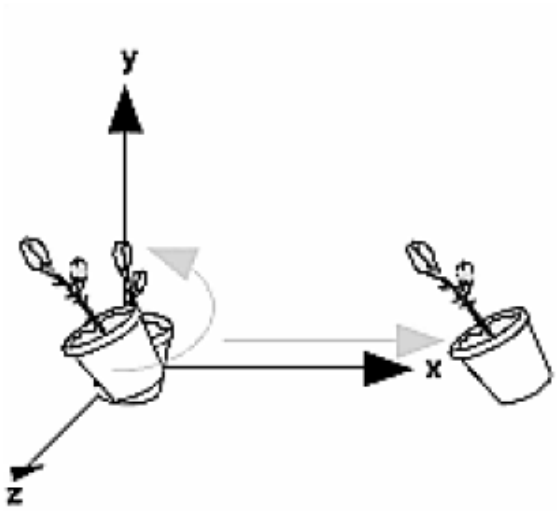
The parts of a robot arm might be constructed by scaling, rotating, and translating an axis-aligned cube centered at the origin.



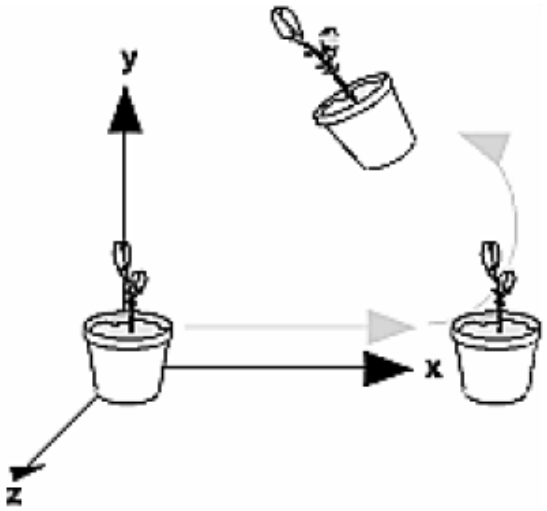
In a 2-D application no **viewing transformations** are necessary.
In a 3-D application with perspective projection, the eye position is at the origin, the viewing direction is the -z direction, and the projection plane is parallel to the x-y plane.
Only vertices with negative z components will be visible. A typical viewing transformation therefore includes at least a translation in the negative z direction.

Rotation and Translation

We will use a right-handed coordinate system



Rotate then Translate



Translate then Rotate

OpenGL Code

Consider the following code sequence, which draws a single point using three transformations

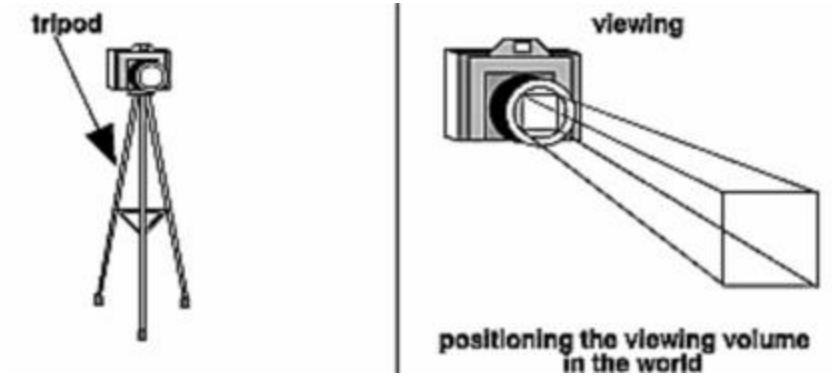
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(N);          /* apply transformation N */
glMultMatrixf(M);          /* apply transformation M */
glMultMatrixf(L);          /* apply transformation L */

glBegin(GL_POINTS);
glVertex3f(v);             /* draw transformed vertex v */
glEnd();
```

The Camera Analogy

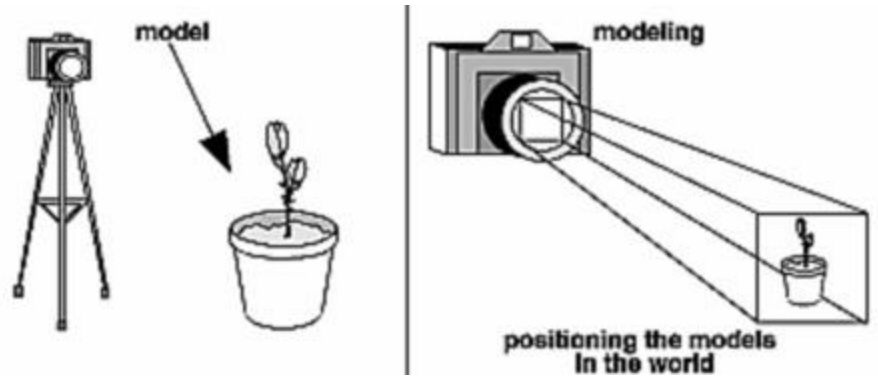
Set up your tripod and pointing the camera at the scene

→ viewing transformation



Arrange the scene to be photographed into the desired composition

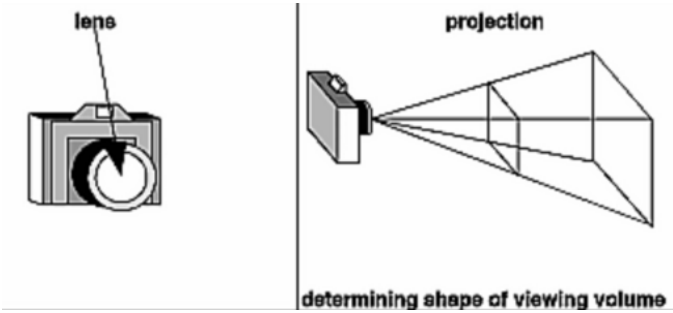
→ modeling transformation



The Camera Analogy

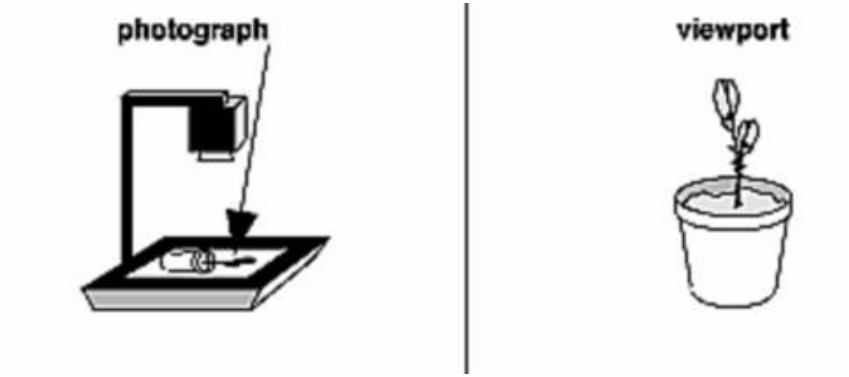
Choose a camera lens or adjust the zoom

→ projection transformation



Determine how large you want the final photograph to be ex: Enlarged

→ viewport transformation



Viewing Transformations

A viewing transformation changes the position and orientation of the viewpoint

Ex: positions the camera tripod, pointing the camera toward the model

viewing transformations are generally composed of translations and rotations.

you can either move the camera or move all the objects in the opposite direction.

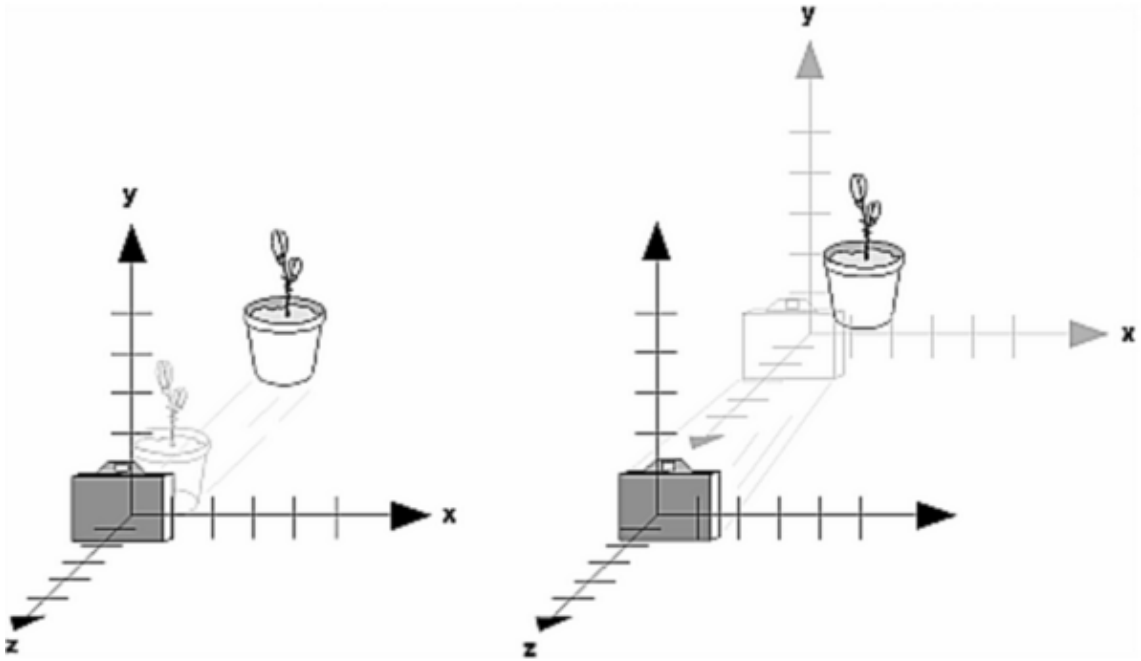
There are 3 ways of doing this

1. Use one or more modeling transformation commands (that is, glTranslate*() and glRotate*())
2. Use the Utility Library routine gluLookAt() to define a line of sight
3. Create your own utility routine that encapsulates rotations and translations

Using glTranslate*() and glRotate*()

glTranslatef(0.0, 0.0, -5.0);

This routine moves the objects in the scene -5 units along the z axis. This is also equivalent to moving the camera +5 units along the z axis.



Using the gluLookAt() Utility Routine

The **gluLookAt()** routine is particularly useful when you want to pan across a landscape

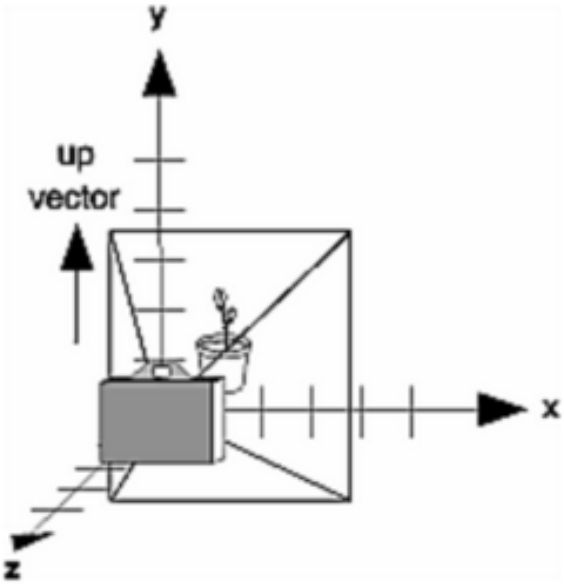
With a viewing volume that's symmetric in both x and y, the (eyex, eyey, eyez) point specified is always in the center of the image on the screen, so you can use a series of commands to move this point slightly, thereby panning across the scene.

*void **gluLookAt**(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, Gldouble upz);*

gluLookAt() Example

gluLookat (0.0, 0.0, 0.0, 0.0, 0.0, -100.0, 0.0, 1.0, 0.0);

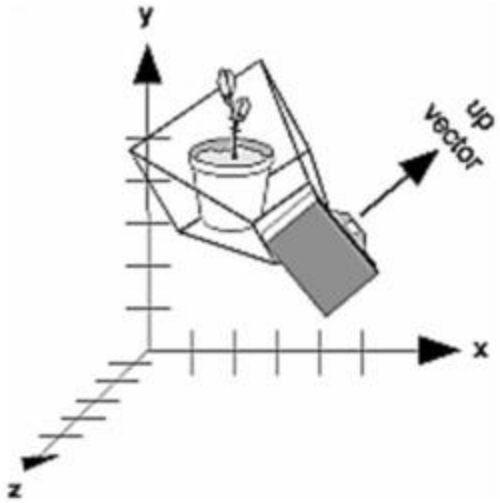
Default Camera Position



gluLookAt() Example

The camera position (*eyex, eyey, eyez*) is at (4, 2, 1). In this case, the camera is looking right at the model, so the reference point is at (2, 4, -3). An orientation vector of (2, 2, -1) is chosen to rotate the viewpoint to this 45-degree angle.

gluLookAt(4.0, 2.0, 1.0, 2.0, 4.0, -3.0, 2.0, 2.0, -1.0);

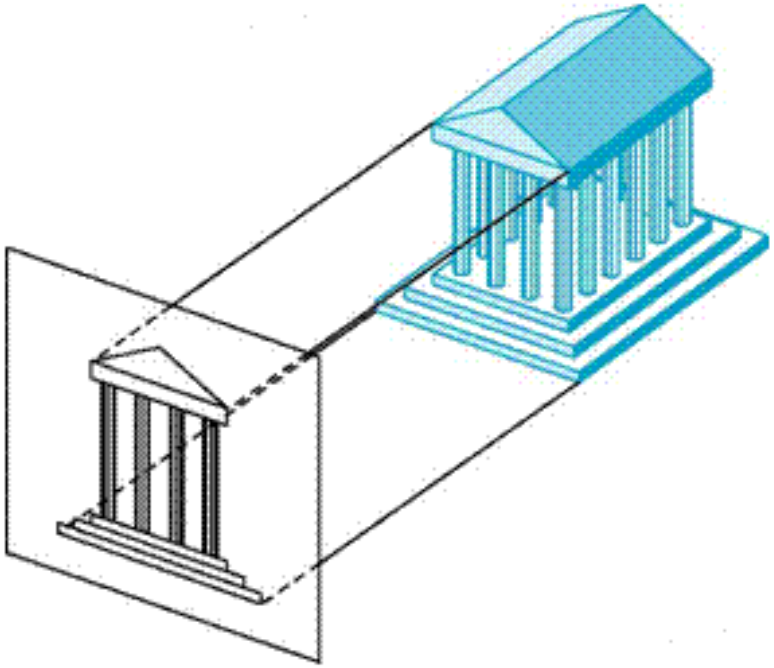


Projection Matrix

The projection plane is the near plane defined by z=-n for n > 0. There are two types of projection.

Orthogonal projection or (parallel) orthographic projection maps a point p to the nearest point in the projection plane by simply setting the z component to -n.

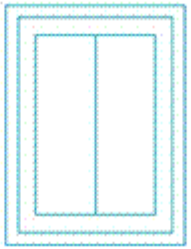
Orthogonal projection

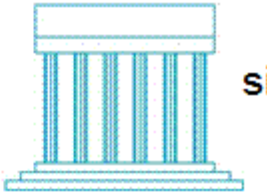


isometric (not multiview orthographic view)

front

top

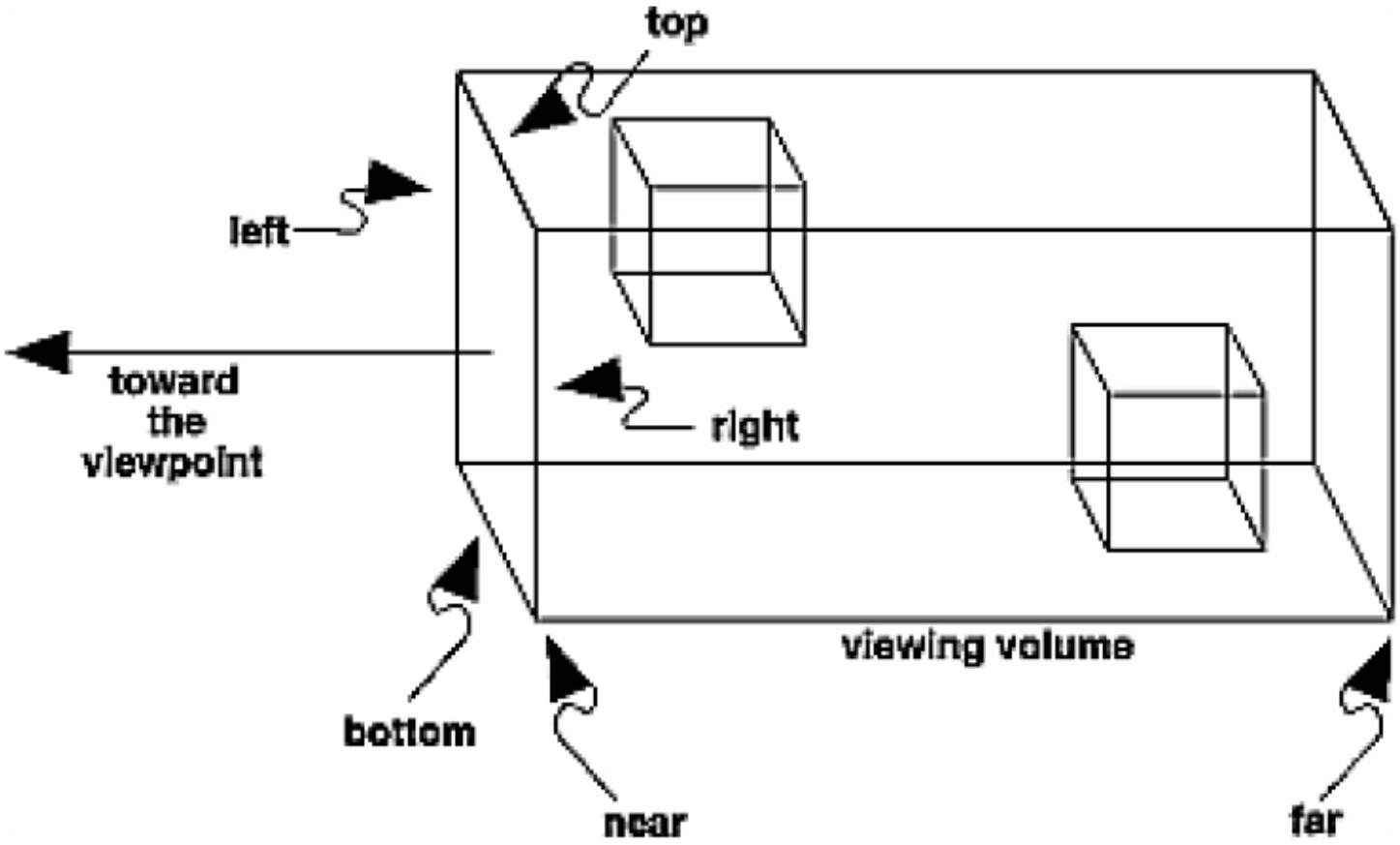




side

Useful for CAD and architectural drawing

Orthographic Projection by glOrtho



Demonstrates that orthographic projection preserves relative sizes.

Perspective projection

Perspective projection involves central projection through the origin (the center of projection). A point p is mapped to the point of intersection of the line defined by p and the origin with the projection plane.

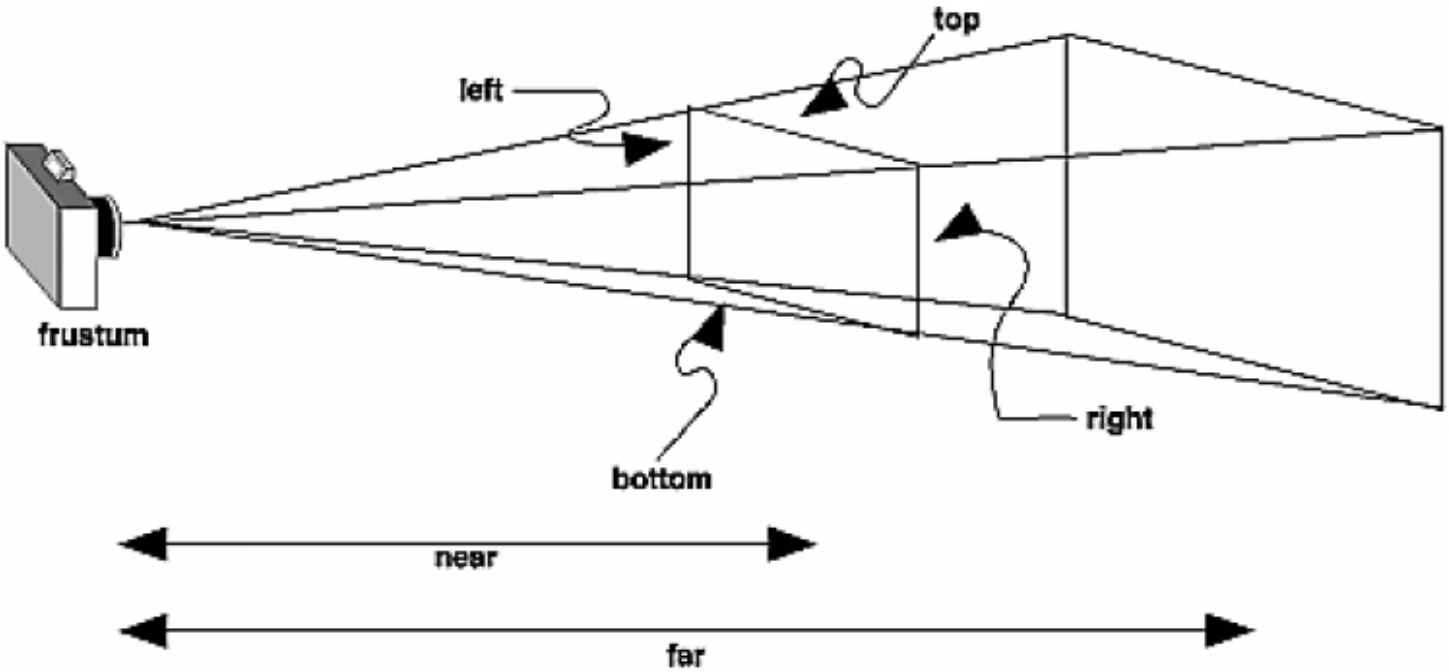
Perspective projection results in perspective foreshortening in which more distant lines project to shorter lines.

Orthographic projection may be thought of as the limit of perspective projection as the eye position is moved to (0; 0;1).

This is inappropriate for rendering real (or realistic appearing) objects, but is cheaper and has the advantage that relative lengths are preserved:

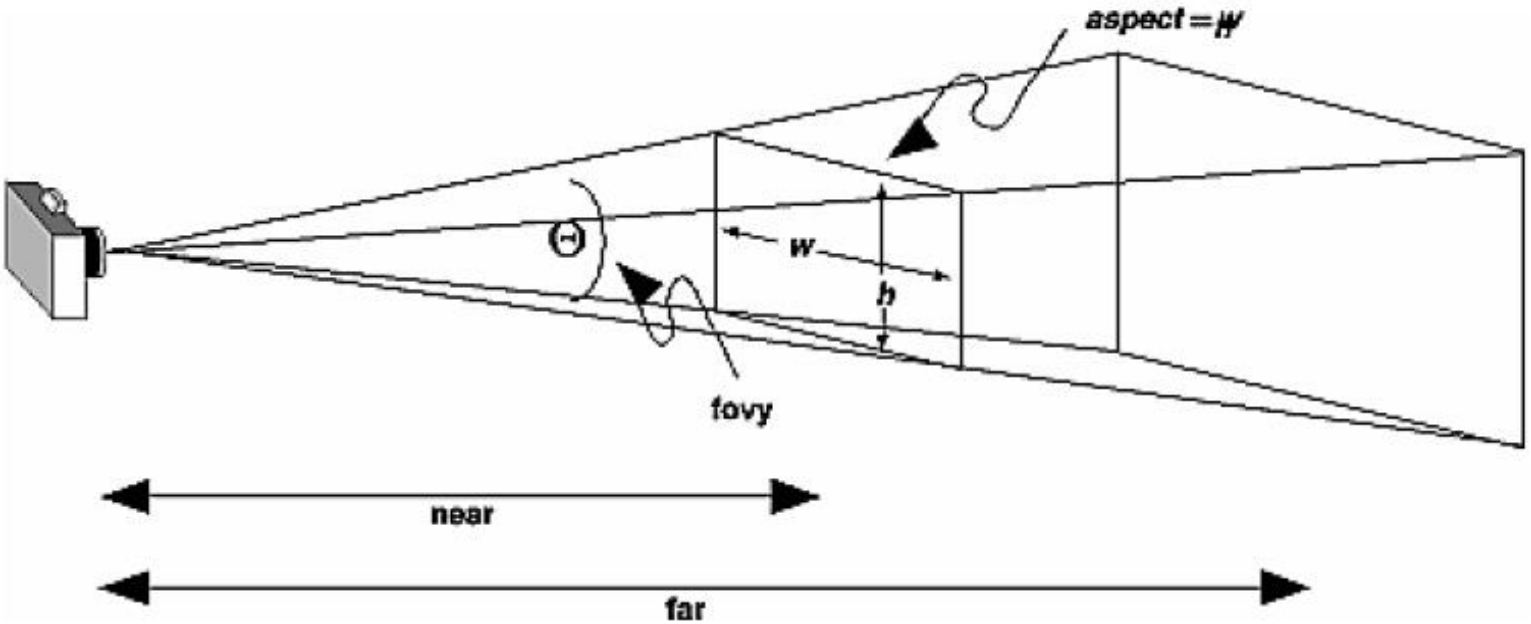
Perspective Projection by glFrustum

Depicts the six planes defining a frustum view volume



Perspective Projection by gluPerspective

gluPerspective(fovy, aspect, near, far).
The aspect ratio is w/h.



Include Files

#include <stlib.h>
#include <GL/glut.h> (This includes gl.h and glu.h)
#include <stdio.h> (if using C I/O)
#include <math.h> (if using C math library)

For portability, replace the second statement by the following:

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
```