German University in Cairo
Department of Computer Science
Dr. Rimon Elias
DMET 502 - Computer Graphics

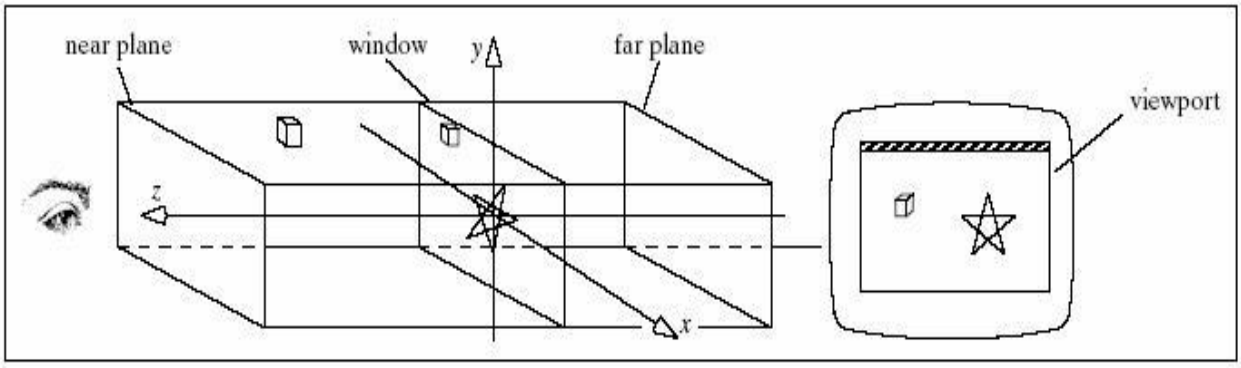
Lab 6

Projection:

Specifying the projection transformation is like choosing a lens for a camera. You can think of this transformation as determining the field of view and therefore which objects are inside it and, to some extent, how they should look. There are two basic types of projections provided for you by OpenGL, orthographic and perspective.

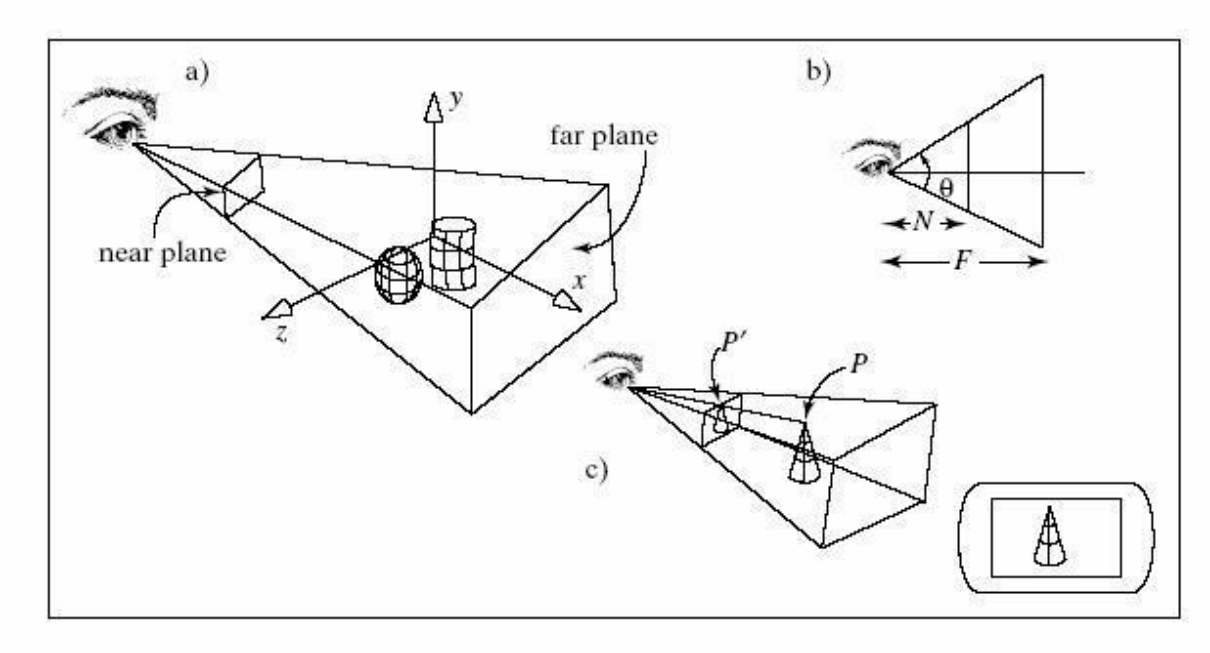
Orthographic: Orthographic projection maps objects directly onto the screen without affecting their relative sizes. This projection is used mainly in architectural and computer-aided design applications, where the actual measurements of the objects are more important than how they might look.

```
void glOrtho(GLdouble left,
             GLdouble right,
             GLdouble bottom,
             GLdouble top,
             GLdouble near,
             GLdouble far)
```



Perspective: We will use the perspective projection to get a more realistic rendering of an object. The object(s) will have the unmistakable characteristic of foreshortening: the further an object is from the camera, the smaller it appears in the final image. This is because the viewing volume of perspective projection is a frustum (a truncated pyramid whose top has been cut off by a plane parallel to its base). Objects that are closer to the apex of the pyramid appear smaller while objects closer to the base appear larger.

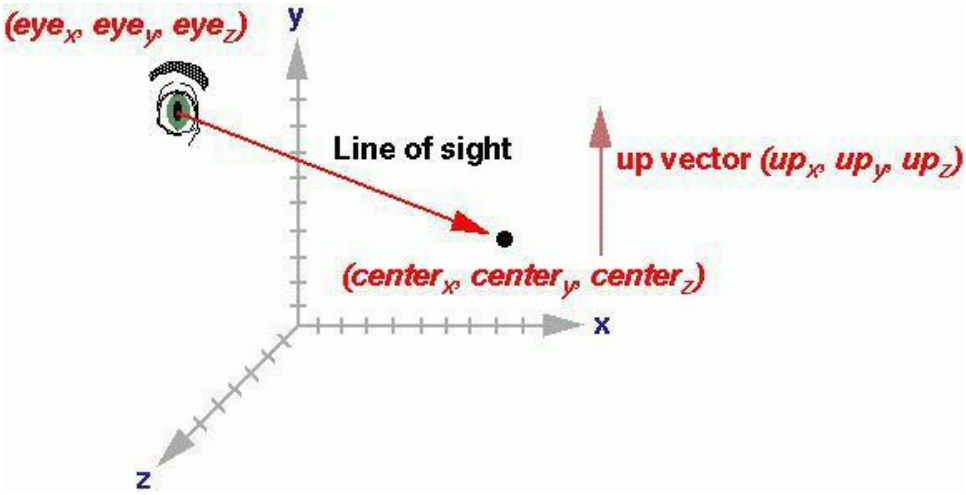
```
void gluPerspective(GLdouble fovy,
                    GLdouble aspect,
                    GLdouble zNear,
                   GLdouble zFar)
```



The camera

Creates a viewing matrix that allows you to look somewhere other than the origin of your scene.

```
GLvoid gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
                 GLdouble centerx, GLdouble centery, GLdouble centerz,
                 GLdouble upx, GLdouble upy, GLdouble upz)
```



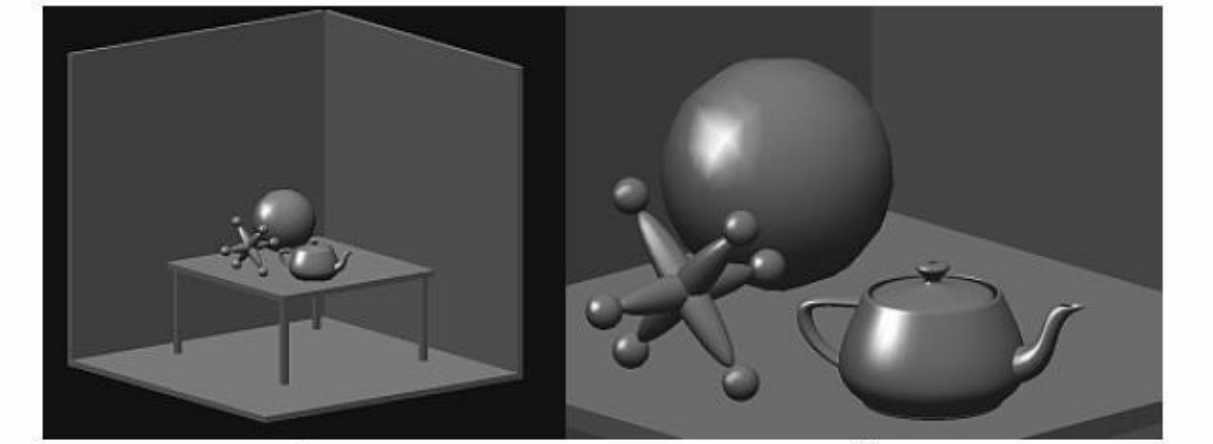
Question 1: (3D Solid Modeling)

The following view scenes use a camera set by
gluLookAt(2.3,1.3,2,0,0.25,0,0.0,1.0,0.0).

Part(a) uses a large view volume that encompasses the whole scene;
Part(b) uses a small view volume that encompasses only a small portion of the scene, thereby providing a close-up view.

The scene contains three objects resting on a table in the corner if a “room”.

- Each of the three walls is made of flattening a cube into a thin sheet and moving it into position.
- The jack is composed of three stretched spheres oriented at right angles to each other, plus six small spheres at their ends.
- The table consists of a tabletop and four legs. Each of the table’s five pieces is a cube that has been scaled to the desired size and shape.
- The layout for the table is based on four parameters that characterize the size of its parts: topWidth, topThick, legLen, and legThick. A routine tabelLeg() draws each leg and is called four times within the routine table() to draw the legs in the four different locations. The different parameters used produce different modeling transformations within tableLeg().



a) b)

Use these to start the task:

```
void wall(double thickness) {
	glPushMatrix();
	glTranslated(0.5, 0.5 * thickness, 0.5);
	glScaled(1.0, thickness, 1.0);
	glutSolidCube(1);
	glPopMatrix();
}
void tableLeg(double thick, double len)
{
	glPushMatrix();
	glTranslated(0, len / 2, 0);
	glScaled(thick, len, thick);
	glutSolidCube(1.0);
	glPopMatrix();
}
void jackpart() {
	glPushMatrix();
	glScaled(0.2, 0.2, 1.0);
	glutSolidSphere(1, 15, 15);
	glPopMatrix();
	glPushMatrix();
	glTranslated(0, 0, 1.2);
	glutSolidSphere(0.2, 15, 15);
	glTranslated(0, 0, -2.4);
	glutSolidSphere(0.2, 15, 15);
	glPopMatrix();
}
void jack() {
	glPushMatrix();
	jackpart();
	glRotated(90.0, 0, 1, 0);
	jackpart();
	glRotated(90.0, 1, 0, 0);
	jackpart();
	glPopMatrix();
}
void table(double topWid, double topThick, double legThick, double legLen) {
	//draw a table - atop and four legs glPushMatrix();
	glTranslated(0, legLen, 0);
	glScaled(topWid, topThick, topWid);
	glutSolidCube(1.0);
	glPopMatrix();
	double dist = 0.95 * topWid / 2.0 - legThick / 2.0;
	glPushMatrix();
	glTranslated(dist, 0, dist);
	tableLeg(legThick, legLen);
	glTranslated(0, 0, -2 * dist);
	tableLeg(legThick, legLen);
	glTranslated(-2 * dist, 0, 2 * dist);
	tableLeg(legThick, legLen);
	glTranslated(0, 0, -2 * dist);
	tableLeg(legThick, legLen);
	glPopMatrix();
}
```

```
void SetupLights()
{
	GLfloat mat_ambient[] = { 0.7f, 0.7f, 0.7, 1.0f };
	GLfloat mat_diffuse[] = { 0.6f, 0.6f, 0.6, 1.0f };
	GLfloat mat_specular[] = { 1.0f, 1.0f, 1.0, 1.0f };
	GLfloat mat_shininess[] = { 50 };
	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mat_ambient);
	glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
	glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
	glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
	//set the light source properties
	GLfloat lightIntensity[] = { 0.7f, 0.7f, 1, 1.0f };
	GLfloat light_position[] = { -7.0f, 6.0f, 3.0f, 0.0f };
	glLightfv(GL_LIGHT0, GL_POSITION, lightIntensity);
	glLightfv(GL_LIGHT0, GL_DIFFUSE, lightIntensity);
}
void displayWire(void) {
	// Setup light
	SetupLights();
	//set the camera
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	double winHt = 1.0;
	glOrtho(-0.5, 0.5, -0.5, 0.5, -1, 1);
	//gluPerspective(60, 640 / 480, 0.001, 100);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	gluLookAt(0.1, 0.1, 0.1, 0, 0, 0, 0.0, 1.0, 0.0);
	//start drawing
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	---- - > Your Code !!
}
void main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(640, 480)
	glutInitWindowPosition(200, 150);
	glutCreateWindow("Colored Dots");
	glutDisplayFunc(displayWire);
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	glShadeModel(GL_SMOOTH);
	glEnable(GL_DEPTH_TEST);
	glEnable(GL_NORMALIZE); glClearColor(1.0, 1.0, 1.0, 0.0);
	glutMainLoop();
}
```

Question 2:

Create an OpenGL program using the above settings which are used for shading texture and lighting, along with certain properties of the objects' surfaces.

The program should draw the house shown in the following figure.

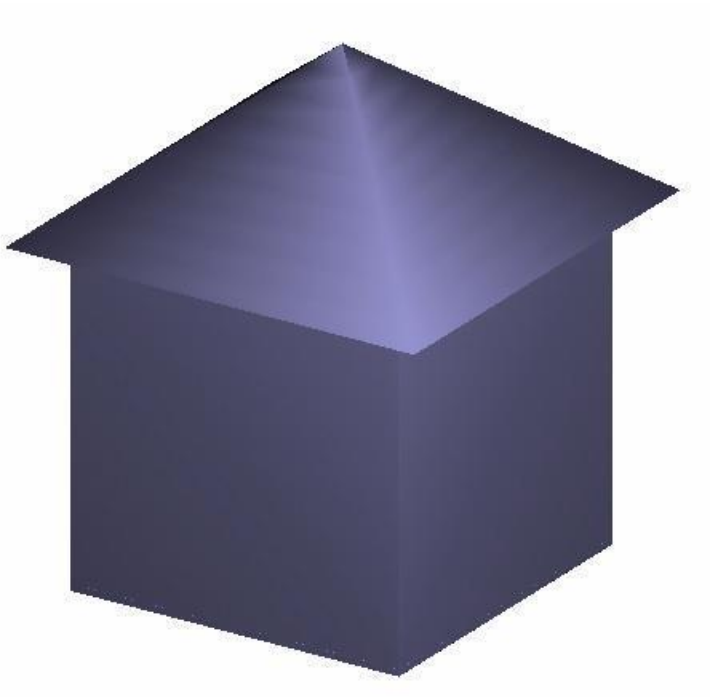
This figure consists of a cube made of four "walls", using the function used above and a solid cone which is placed exactly at the top of the cube.

The settings of the camera are as follows:

```
gluLookAt(1.3, 1.3, 2.0, 0.0, 0.25,0.0,0.0,1.0,0.0);
```

and the light intensity can be:

```
GLfloat lightIntensity[]={0.7f,0.7f,1.0f,1.0f};
```



Hint:

Use the following code:

```
#include "glut.h"
void wall(double thickness)
{
	glPushMatrix();
	glTranslated(0.5, 0.5 * thickness, 0.5);
	glScaled(1.0, thickness, 1.0);
	glutSolidCube(1);
	glPopMatrix();
}
void displayWire(void)
{
	//set the light source properties
GLfloat lightIntensity[] = { 0.7f,0.7f,1,1.0f }; GLfloat light_position[] =
{ 7.0f,6.0f,3.0f,0.0f };
glLightfv(GL_LIGHT0, GL_POSITION, lightIntensity);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightIntensity);
	--->Your Code !!
}
void main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(640, 480);
	glutInitWindowPosition(200, 150);
	glutCreateWindow("Colored Dots");
	glutDisplayFunc(displayWire);
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	glShadeModel(GL_SMOOTH);
	glEnable(GL_DEPTH_TEST);
	glEnable(GL_NORMALIZE);
	glClearColor(1.0, 1.0, 1.0, 0.0);
	glutMainLoop();
}
```

Question 3:

Using the previous program that draws a house, draw a village as shown below.

- It consists of 2 rows of house which are parallel of the x-axis.
- Each row consists of 6 houses.
- The distance between each two houses is 1.5 units.
- The first house on the first row is constructed at (0.6,0,0).
- The distance between the two rows of houses= 5 units.

Adjust your Glut camera get the required snapshot.