COSC363 Computer Graphics
Lab01: OpenGL Basics, Mesh Models

Aim:

In this lab, you will familiarize yourself with the structure of basic OpenGL programs, and also implement camera motion in a simple scene consisting of GLUT objects. You will also learn to create and display mesh models in OFF format.

I. Teapot.cpp:

1. The program **`Teapot.cpp`** has a structure similar to that given in lecture notes ([1]:7-9). It draws a teapot (using the function `glutSolidTeapot`), and additionally a floor plane (using the function "drawFloor"). The teapot is drawn at the origin of the reference frame. The floor is drawn using a set of lines parallel to the *x* and *z* axes. The program produces the output shown in Fig. (1).

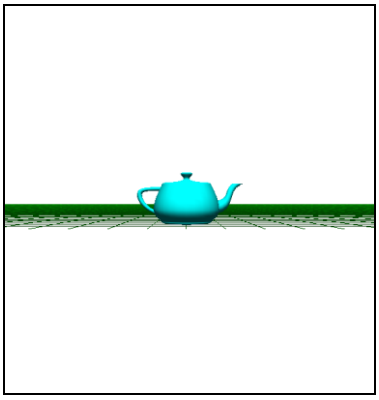

Fig. (1)

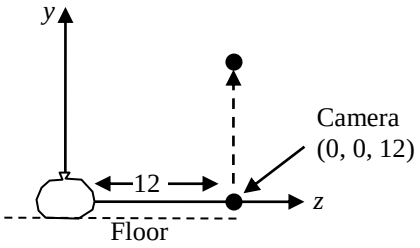

Fig. (2)

2. **Camera:** The output you just obtained is the view of the teapot from a point along the *z*-axis, at a distance 12 units from the origin (see Fig. (2)). The first three parameters (0, 0, 12) of the `gluLookAt()` function define this position of the camera. Try changing the camera position to (0, 10, 12), lifting the camera up by 10 units along the y-direction. The display should change to that shown in Fig. (3). Rotate the teapot by 60 degs about the x-axis (see slide [1]:22). The output of the program is shown in Fig. (4).

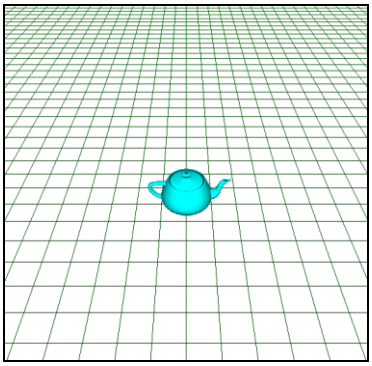

Fig. (3)

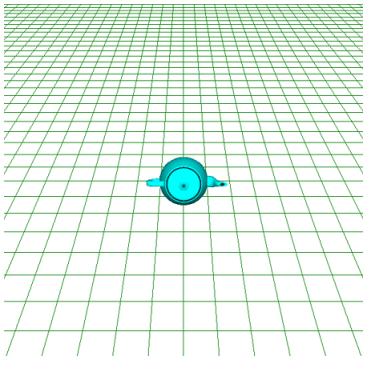

Fig. (4)

3. **Callback functions:** We will now implement a special keyboard callback function to move the camera up and down using the arrow keys. Define a global `int` variable `cam_hgt` to represent the height of the camera and initialize it with the current value 10. Modify the corresponding parameter in the function `gluLookAt()` to use this variable instead of a constant value 10. Define a callback function `special()` as shown below (see slides [1]:37, 38):

```
//--Special keyboard event callback function ---------
void special(int key, int x, int y)
{
    if(key == GLUT_KEY_UP) cam_hgt++;
    else if(key == GLUT_KEY_DOWN) cam_hgt--;
    glutPostRedisplay();
}
```

You will need to register the above function as a callback, by adding the following statement in main() (See slide [1]-38).

```
glutSpecialFunc(special);
```

Please note that the above statement should be added *before* the call to `glutMainLoop()`. Run your program to test the correctness of your implementation. Limit the movement of the camera to the range [2 – 20].

4. Our next task is to continuously move the camera along a circular path around the teapot, at the current height from the floor given by `cam_hgt` (Fig. (5)). The radius of the camera's orbit is 12 units. Define a global `float` variable `theta` to represent the positional angle of the camera (in degs), and initialize it to 0.

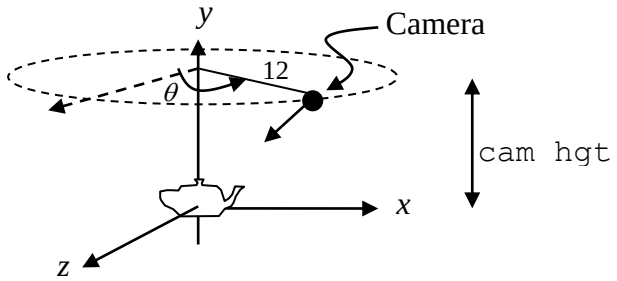

Fig. (5)

Specify the position of the camera in `gluLookAt()` as (12sinθ, cam_hgt, 12cosθ). [Warning: `cos()`, `sin()` functions assume that the parameter is defined in radians, not degrees! So please convert theta to radians]. Define a timer callback function (see slides [1]: 47, 48) to continuously increment `theta` at regular intervals of 50 msecs as follows:

```
void myTimer(int value)
{
      theta ++;
      glutPostRedisplay();
      glutTimerFunc(50, myTimer, 0);
}
```

Also add the statement `glutTimerFunc(50, myTimer, 0);` inside main(). If implemented correctly, the camera should continuously hover around the teapot. The height can still be changed using arrow keys.

Exercises:

Replace the teapot with other GLUT objects. Please refer to the file "GLUT-GLU-Objects.pdf" for a brief description of the built-in objects.

Reduce the field of view of the camera (say, from 50 degs to 30 degs) and observe the effect this change produces on the display.

II. Mesh Models:

The Object File Format (OFF) is a convenient ASCII format for storing 3D model definitions (meshes). It uses simple vertex-list and polygon-list structures for specifying a polygonal model. Unlike other 3D mesh formats, the OFF format does not intersperse commands with values on every line, and therefore can be easily parsed to extract vertex coordinates and triangle indices. The basic structure of a mesh file in OFF format is shown in Fig. (6).

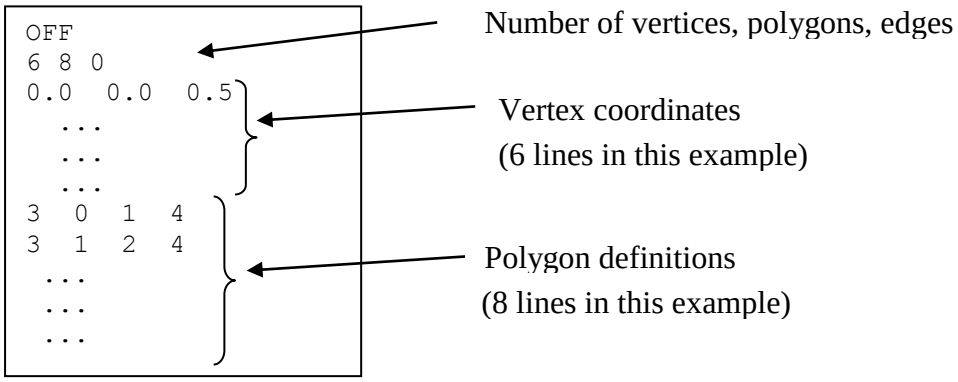

Fig. (6)

The first line contains the header keyword OFF. The second line contains the total number of vertices (n_v), total number of polygons (n_p) and edges (n_e) in the model. The number of edges is always set to 0. The second line will be followed by n_v lines containing three-dimensional coordinates of the vertices. The file will then contain n_p lines defining each polygon in terms of its vertex indices. The first number in a polygon definition indicates the number of vertices of that polygon. For a triangular mesh, this number will always be 3. The following three numbers on the same line give the vertex indices of that polygon. We will consider a simple example below.

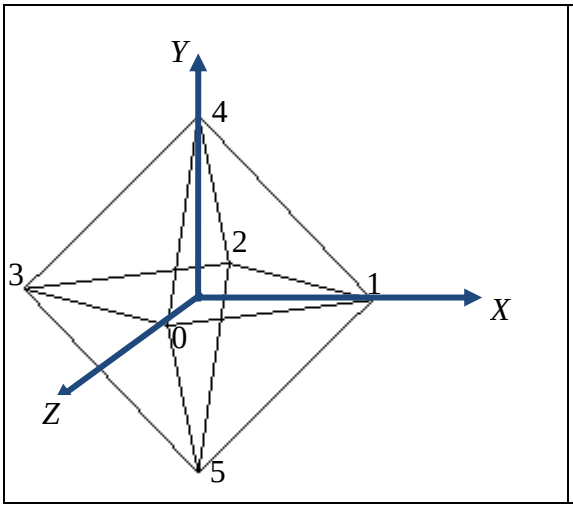

Fig. (7)

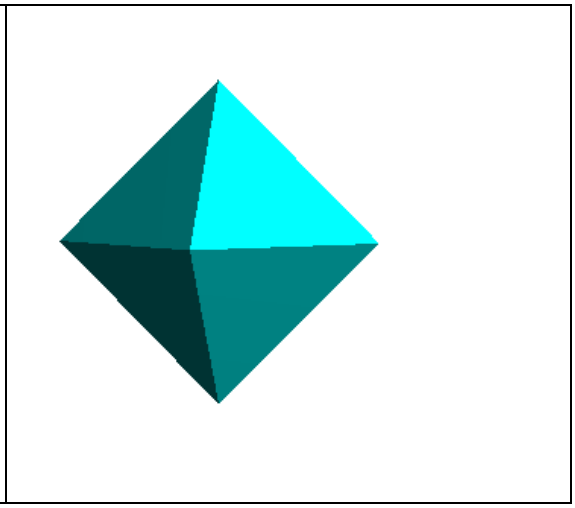

Fig. (8)

An octahedron contains 6 vertices and 8 triangles. The octahedron in Fig. (7) intersects each of the principal axes at a distance 0.5 units from the origin. Thus the first vertex (with index 0) has coordinates (0, 0, 0.5), the second vertex has index 1 and coordinates (0.5, 0, 0) and so on. Now refer to Fig. (6), which gives the model definition for this shape. The first triangle has vertex indices 0, 1, 4. Triangles can be defined in any order, but note that the vertices must be oriented in an anti-clockwise sense when viewed from outside the model. This will ensure that the surface normal vectors are directed outwards from each triangle, as required by illumination models.

1. Create a plain text file with name "Octahedron.off" and create the model definition for the octahedron (by completing the missing entries in Fig. (6)).

2. The program **`Model3D.cpp`** includes the function and variable declarations for reading data from an OFF file. The vertex coordinates are stored in arrays `x, y, z`; and the triangle indices in arrays `t1, t2, t3`. The variable `nvert` stores the total number of vertices, and `nface` the total number of polygonal faces (triangles or quads). The `display` function contains a `glBegin()..glEnd()` block to render the triangles. If your model definition is correct, the output should look similar to that given in Fig. (8). The program also allows you to rotate the object about the x, y axes using the arrow keys.

3. Load the mesh file "Sphere.off" by specifying the file name on the second line of the initialize() function. This file contains the model of an icosphere, and the output of the program is shown in Fig. (9). One of the triangular faces of the sphere appears to be wrongly rendered/lit. This triangle is the third triangle in the face list of the OFF file, and has 4, 2, 5 as the vertex indices. Please correct this entry in the face list to get a proper rendering of the sphere.

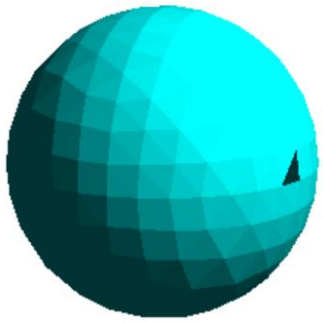

Fig. (9)

4. Three models "Cone.off", "Cow.off", "Oni.off" are provided in the file Models_OFF.zip. Change the mesh file name in the program to get the displays of these models (Fig. (10)).

5. You can get a wireframe display of the models by changing the polygon display mode from `GL_FILL` to `GL_LINE` in the function call `glPolygonMode(…)`. Please disable lighting in the wireframe display mode, by commenting out the line `glEnable(GL_LIGHTING)` or changing it to `glDisable(GL_LIGHTING)`.

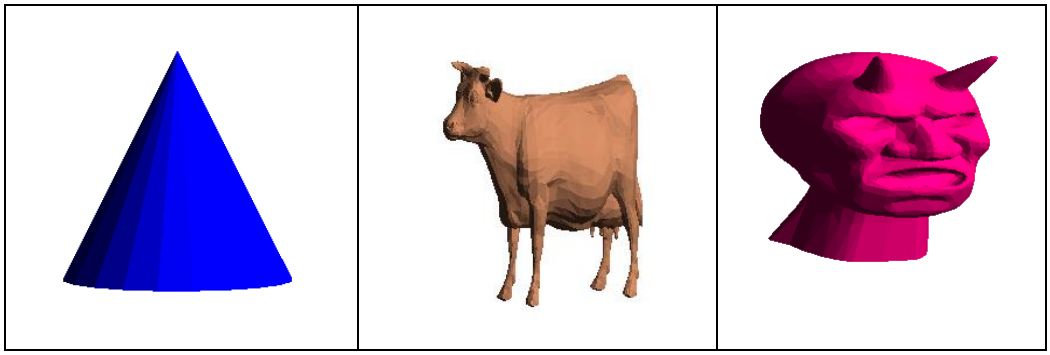

Fig. (10)

III. Quiz-01

Please complete the quiz in the 'Lab Material' section on Learn. The quiz will remain open until **5pm, 10-March-2023**.

A question within a quiz may be attempted multiple times. However, a fraction of the marks (25%) will be deducted for each incorrect answer. Only the first submission of the quiz is considered for grading.