COSC363 Computer Graphics
Lab03: Object Modelling

Aim:

This lab introduces object modeling techniques using triangle fans and triangle strips. Such methods are commonly used for triangulating polygonal surfaces and also for generating models of sweep surfaces.

I. MapleLeaf.cpp:

The program `MapleLeaf.cpp` displays a closed polygonal line as shown in Fig. 1. The two-dimensional vertex coordinates of the polygon are stored in arrays `vx[]` and `vy[]`. The polygon is drawn inside the function `drawBase()` with GL_POLYGON as the primitive type. The drawn object can be rotated about the y-axis using the arrow keys.

Fig. 1.

Fig. 2.

1. In the `display` function, change the polygon mode (3rd line) from GL_LINE to GL_FILL, and note that the chosen primitive type GL_POLYGON in this case does not produce the correct output (Fig. 2). Change the polygon mode back to GL_LINE.

2. The polygon can be triangulated using a triangle fan with (0, 6, 0) as the central point (Fig. 3). Create a function `drawTriangulated()` that generates this triangle fan:

```
glBegin(GL_TRIANGLE_FAN);
   glVertex3f(0, 6, 0);
   for (int i = 0; i < N; i++)
   {
      glVertex3f(vx[i], vy[i], 0);
   }
   glVertex3f(vx[0], vy[0], 0);     //Closed fan
glEnd();
```

Inside the display function, replace the statement `drawBase();` with `drawTriangulated();`. The program should produce the output shown in Fig. 3. Now, if you change the polygon mode to GL_FILL, you should get the correct display of the polygon as given in Fig. 4.

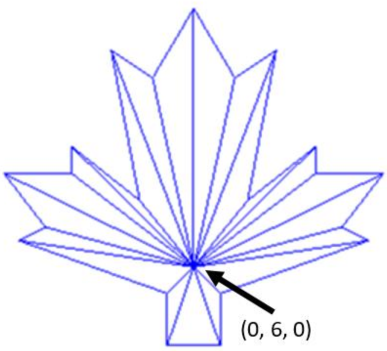

Fig. 3.


Fig. 4.

3. Write another function `drawModel()` that creates two copies of the triangulated polygon. Translate the first copy of the polygon to (0, 0, -1), and the second to (0, 0, 1), as shown in Fig. 5. Remember to use `glPushMatrix()`… `glPopMatrix()` blocks. Inside the display function, replace the statement `drawTriangulated();` with `drawModel();` to produce an output as given in Fig.5.

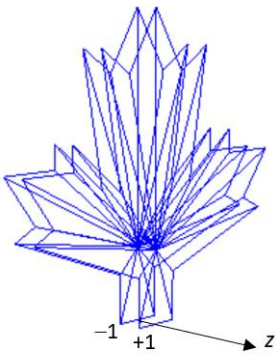

Fig. 5.

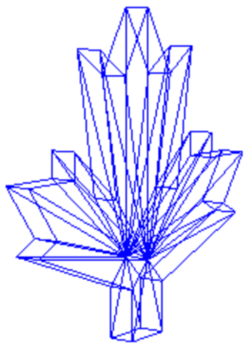
Fig. 6.

4. We will now fill the space between the two polygonal surfaces using a quad strip, to create a solid 3D model. Modify the function `drawModel()` to include a quad strip as given below:

```
glBegin(GL_QUAD_STRIP);
for (int i = 0; i < N; i++)
{
   glVertex3f(vx[i], vy[i], -1);  //Vertices on the first polygon
   glVertex3f(vx[i], vy[i],  1);  //Vertices on the second polygon
 }
glEnd();
```

The program should produce an output similar to that given in Fig. 6.

5. We are now ready to render the solid model under a light source. Enable lighting by uncommenting the two `glEnable()` statements in the `initialize` function, and change the polygon mode to GL_FILL. For lighting to work properly, every polygonal face should have a normal vector assigned to it. In the `drawModel()` function assign the normal vector `glNormal3f(0, 0, -1)` to the whole triangulated polygon at (0, 0, -1). Similarly, assign the vector `glNormal3f(0, 0, 1)` to the triangulated polygon at (0, 0, 1). The normal vector for the polygonal elements of the quad strip must be computed as given in Slide [4]-12. Include the floor plane in the scene by adding `drawFloor();` to the display function. Change the colour of the model if required. A sample output of the program is shown in Fig. 7.

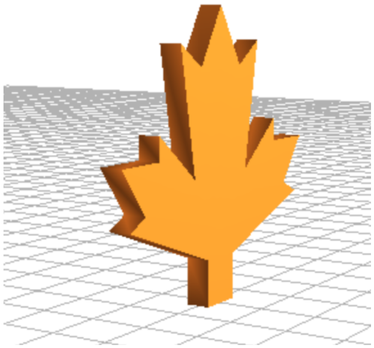

Fig. 7.

6. A method to generate planar shadows is given in Slide [3]-24. If implemented correctly, this method should produce an output shown in Fig. 8. Note that the shadow region is not rendered correctly, since parts of the floor are visible through the shadow. This is a common artefact known as “depth fighting” or “z-fighting”, present in displays when two planes overlap at the same distance from the camera. In this case, the shadow object and the floor plane are both at ground level (y = 0). We can get a proper rendering of the shadow as shown in Fig. 9, by moving the floor plane below ground level by a small distance; for example, by translating the floor to (0, -0.1, 0).

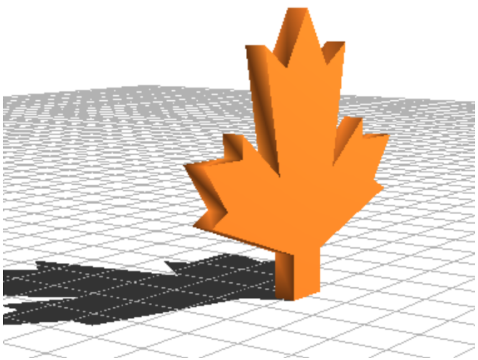

Fig. 8.

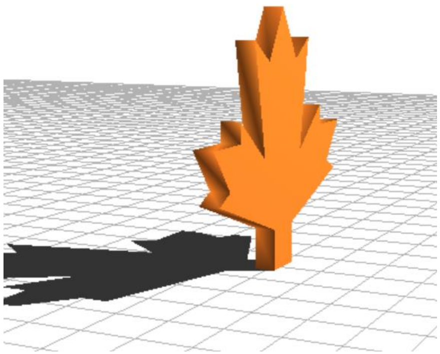

Fig. 9.

II. Vase.cpp:

The program `Vase.cpp` contains the vertex data for a base curve in arrays vx[i], vy[i], vz[i], i = 0 ... N-1, N=50. It displays only the floor grid of the scene. The scene can be rotated using left/right arrow keys. The up/down arrow keys change the height of the camera.

1. In the `display()` function, transform each point (vx[i], vy[i], vz[i]) about the y-axis by 10 degrees to form a new set of points (wx[i], wy[i], wz[i]), i=0...N-1. The function already includes the declaration of these array variables. Use the following equations, and include your code after the statement marked "Start here".

   $w_x[i] = v_x[i]\cos\theta + v_z[i]\sin\theta$, θ = 10 Degs (Convert to radians!)
   $w_y[i] = v_y[i]$
   $w_z[i] = -v_x[i]\sin\theta + v_z[i]\cos\theta$

   Please also see the equations given on slide [4]-14. The points W_i define the rotated base curve. Construct a triangle strip between the two polygonal curves using the method given on slide [4]-11. The required output is shown in Fig. (a).

2. Copy the coordinates W_i to V_i for the next iteration, and repeat the steps 36 times to get a 360 degree revolution of the base curve (Fig. (b)). Steps 1 and 2 can be implemented as nested loops:

```
for(int j = 0; j < 36; j++)       //36 slices in 10 deg steps
{
     for(int i = 0; i < N; i++)   //N vertices along each slice
     {
         wx[i] = ...     //Get transformed points W using 10 deg rotn
         wy[i] = ...
         wz[i] = ...
     }

     glBegin((GL_TRIANGLE_STRIP);  //Create triangle strip using V, W
      ...
     glEnd();

     //Copy W array to V for next iteration
     for(int i = 0; i < N; i++)
     {
          vx[i] = wx[i];
          vy[i] = wy[i];
          vz[i] = wz[i];
     }
}
```

3. In the `display()` function, change the polygon mode from GL_LINE to GL_FILL, and enable lighting. The output with lighting is given in Fig. (c).

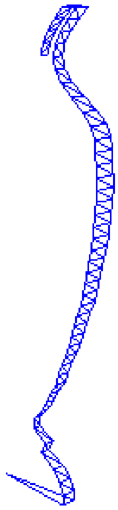

Fig. (a)

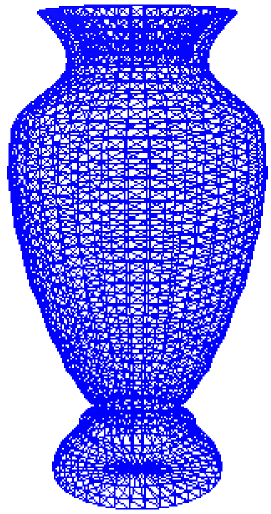

Fig. (b)

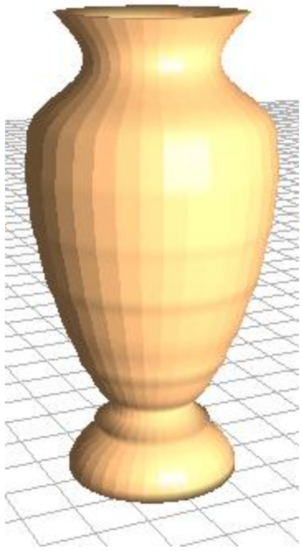

Fig. (c)

Please save your work. We will use the above models in next week's lab for generating texture mapped displays.

Ref:

[3] Lec03_Illumination.pdf (COSC363 Lecture slides)
[4] Lec04_ObjectModelling.pdf (COSC363 Lecture slides)

III. Quiz-03

The quiz will remain open until **5pm, 26 March, 2021**.