# COSC363 Computer Graphics
# **Lab04: Object Modelling**

## Aim:

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

## I. Boat.cpp:

The program `Boat.cpp` displays only the grid of the floor plane. It however contains the data required for generating a simple model of a country boat as shown in Fig.1. We will construct the surfaces of this model using polygonal strips.

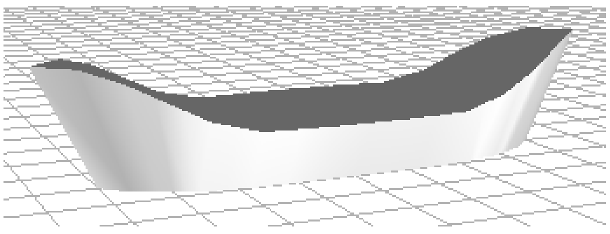
Fig. 1.

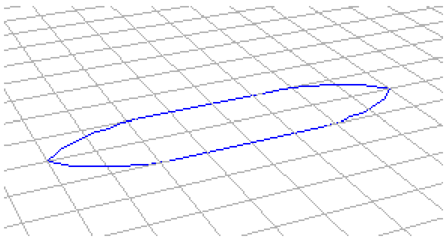
Fig. 2.

1. The function `drawBase()` contains the vertex coordinates of a polygon shown in Fig. 2. This polygon represents the base panel of the boat. The polygon has a convex shape, and therefore can be rendered using the primitive type GL_POLYGON (see slide [5]-3). Please include the code for drawing this polygon in the `drawBase()` function. You should get the output in Fig. 2.

2. The function `drawSide()` contains the vertex coordinates of points $V_i$ and $W_i$ on two polygonal lines forming the bottom and top edges of the side of the boat. Such polygonal shapes can be easily designed by plotting the points on a graph paper (Fig. 3). Both polygonal lines have the same number of points. In the `drawSide()` function, include the code for drawing a quad strip (GL_QUAD_STRIP) connecting pairs of points on the polygonal lines (see slide [5]-8). You should get the output in Fig. 4. The scene can be rotated using left/right arrow keys.

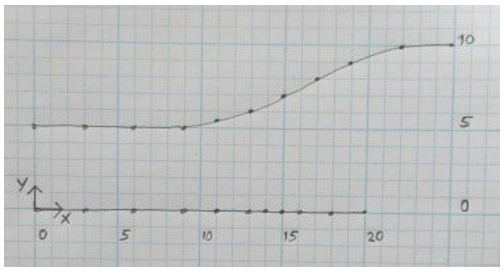

Fig. 3.

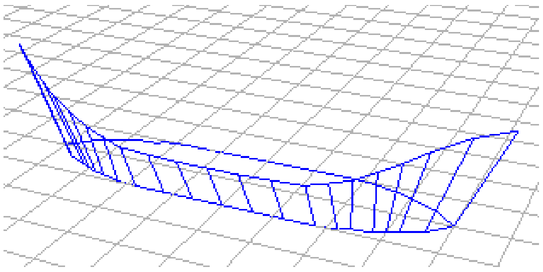
Fig. 4.

3. In the `display()` function, include a second call to the `drawSide()` function to generate a copy of this quad strip, and scale it by factors (1, 1, -1) to create a reflection about the xy-plane (Fig. 5.)

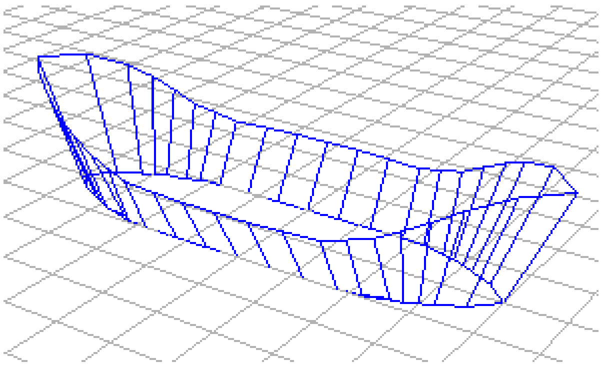

Fig. 5.

4. Lighting calculations require the definition of surface normal vectors for each quad in the polygon strips. Include the call to the `normal()` function in the code for generating the quad strips (see Slide [5]-12). In the `display()` function, change the polygon mode from GL_LINE to GL_FILL, and enable lighting. You should now get the output shown in Fig. 1.

5. Assign texture coordinates to the vertices of the quad strip as shown on Slide [5]-13. Enable texturing in the `display()` function. You should get an output similar to that in Fig. 6.

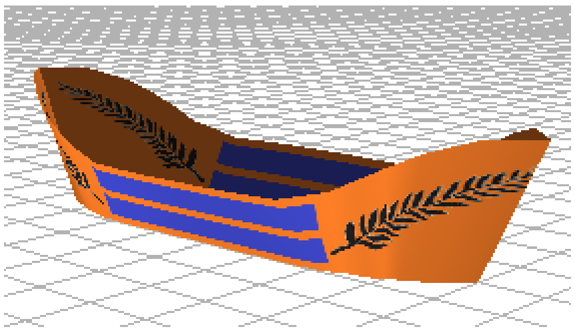

Fig. 6.

## 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".

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

These points $W_i$ define the rotated base curve. Construct a triangle strip between the two polygonal curves using the method given on slide [5]-8. 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
         wy[i] = ...
         wz[i] = ...
     }

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

     //Copy W array to V for next iteration
}
```

3. Now add functions to compute normal vectors at each vertex of the triangle strip as shown on Slide [5]-11. 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).

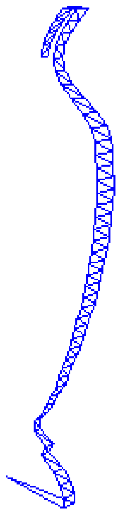
Fig. (a)

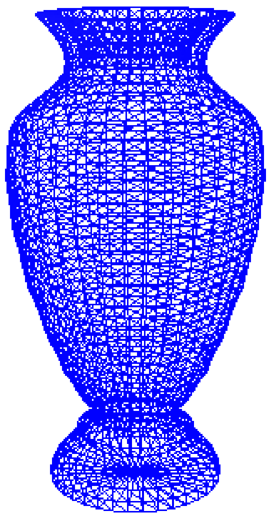
Fig. (b)

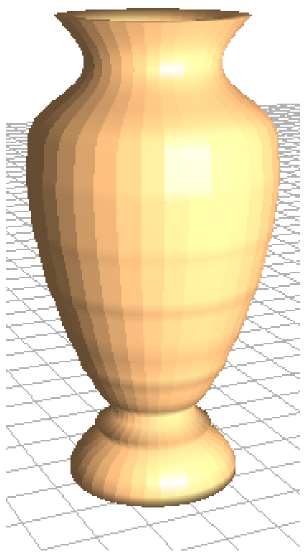
Fig. (c)

4. The program includes the function to load a bitmap texture "VaseTexture.bmp". We will texture map the image to the whole surface generated above. Assign texture coordinates to the points on the triangle strip, assuming that the points are nearly equally spaced. In the `display()` function, enable texture mapping. Please refer to Slide [5]-18 for more information on computing texture coordinates for the vertices of polygonal strips. The textured output is shown in Fig (d).

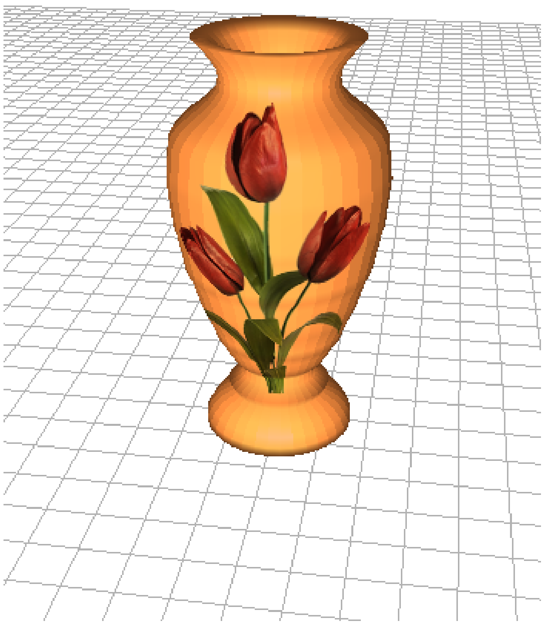

Fig (d)

Ref:

[5] Lec05_ObjectModelling.pdf (COSC363 Lecture notes)

## III. DevIL_Test.cpp

The Developers Image Library (a.k.a Open Image Library) is a useful library with OpenGL interoperability features, supporting image operations. It can be used to load textures in PNG, JPG formats. A test file showing the implementation of texture loading functions and a sample image "Colors.png" are provided. The program loads the image and displays a quad, texture mapped with the image.

## IV. Quiz-04

The quiz will remain open until **5pm, 27 March, 2020**.