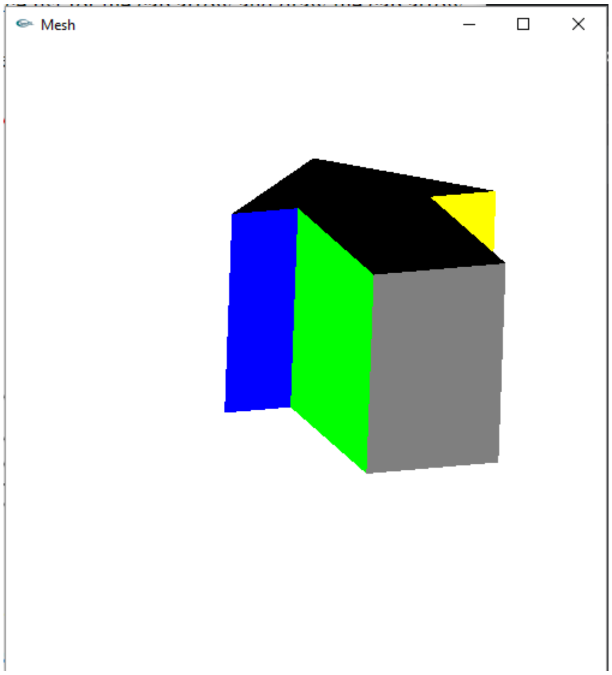
Daniel Meyer

CSE 520

Tong Yu

Lab 7 Report



Data1.txt

```
14 8 8

0 0 0   2 0 0   1.5 0.5 0   3 2 0   2 3 0   0.5 1.5 0   0 2 0

0 0 2   2 0 2   1.5 0.5 2   3 2 2   2 3 2   0.5 1.5 2   0 2 2

-1 0 0   -0.707 0.707 0   0.707 0.707 0

1 0 0   0 -1 0   0 0 1   0 0 -1 -1 0 0

4   0 7 8 1   0 0 0 0

4   1 8 9 2   1 1 1 1

4   2 9 10 3  2 2 2 2

4   3 10 11 4  3 3 3 3

4   4 11 12 5  4 4 4 4

4   5 12 13 6  5 5 5 5
```

4 6 13 7 0 6 6 6

4 6 13 7 0 6 6 6

Template.cpp

```
//display.cpp
#include "Mesh.h"
#include <stdlib.h>
#include <GL/glut.h>
//#include <SDL/SDL.h>

char  DATA_FILE[100];

int anglex = 0, angley = 0, anglez = 0;          //rotation angles
int window;
Mesh msh;
int lighton = 0;

void init(void)
{
     glClearColor(1.0, 1.0, 1.0, 1.0);
     if (!msh.readData(DATA_FILE)) {
          cout << "Error opening file " << DATA_FILE << endl;
          return;
     }

     glClearColor(1, 1.0, 1.0, 0.0);
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     glShadeModel(GL_FLAT);
     glEnable(GL_DEPTH_TEST);

     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     glOrtho(-3.0, 3.0, -3.0, 3.0, 0.1, 100);
     glMatrixMode(GL_MODELVIEW); // position and aim the camera
     glLoadIdentity();
     gluLookAt(5, 5, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0);
     glColor3f(0, 0, 0);

}

void display(void)
{
     /*
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glPushMatrix();
    glRotatef(anglex, 1.0, 0.0, 0.0);   //rotate the object about x-
axis
    glRotatef(angley, 0.0, 1.0, 0.0);   //rotate about y-axis
    glRotatef(anglez, 0.0, 0.0, 1.0);   //rotate about z-axis

    msh.renderMesh();
    glPopMatrix();
    glFlush();
}

void keyboard(unsigned char key, int x, int y)
{
    switch (key) {
    case 27:
        glutDestroyWindow(window);
        exit(0);
    case 'x':
        anglex = (anglex + 3) % 360;
        break;
    case 'X':
        anglex = (anglex - 3) % 360;
        break;
    case 'y':
        angley = (angley + 3) % 360;
        break;
    case 'Y':
        angley = (angley - 3) % 360;
        break;
    case 'z':
        anglez = (anglez + 3) % 360;
        break;
    case 'Z':
        anglez = (anglez - 3) % 360;
        break;
    case 'r':                                       //reset
        anglez = angley = anglex = 0;
        break;
    }
    glutPostRedisplay();
}


int main(int argc, char *argv[])
```

```
{
    if (argc < 2) {
        cout << "\nUsage: " << argv[0] << " data file name [0/1]
(e.g. " << argv[0] << " data.txt 1)" << endl;
        return 1;
    }
    strcpy(DATA_FILE, argv[1]);
    if (argc > 2)
        lighton = atoi(argv[2]);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    window = glutCreateWindow("Mesh ");
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glClearColor(1.0f, 1.0f, 1.0f, 0.0f); //white background
    glViewport(0, 0, 500, 500);
    init();

    glutMainLoop();

    return 0;
}
```

Mesh.cpp

```
//Mesh.cpp : Mesh class member functions

#include "Mesh.h"

using namespace std;

Mesh::Mesh()
{
    nVertices = nNormals = nFaces = 0;
}

//Read mesh data from file
bool Mesh::readData(char fName[])
{
    fstream ins;
    ins.open(fName, ios::in);
    cout << "opening file " << fName << endl;
```

```
      if (ins.fail()) return false;                // error - can't
open file
      if (ins.eof())  return false;       // error - empty file
      ins >> nVertices >> nNormals >> nFaces;     // read in number of
vertices, normals, and faces
      for (int i = 0; i < nVertices; i++) { //read vertices
            Point3 p;
            ins >> p.x >> p.y >> p.z;
            vertexList.push_back(p);
      }
      for (int i = 0; i < nNormals; i++) {  //read normals
            Vector3 v;
            ins >> v.x >> v.y >> v.z;
            normalList.push_back(v);
      }
      cout << endl;
      for (int i = 0; i < nFaces; i++) {
            Polygon p;

            ins >> p.n;
            for (int j = 0; j < p.n; j++) {
                  int vertexIndex;
                  ins >> vertexIndex;
                  p.vertices.push_back(vertexIndex);
            }
            for (int j = 0; j < p.n; j++) {
                  int normalIndex;
                  ins >> normalIndex;
                  p.normals.push_back(normalIndex);
            }
            faceList.push_back(p);
      }

      return true;
}

//render the mesh
void Mesh::renderMesh()
{
      //Draw each polygon of the mesh

      glEnable(GL_CULL_FACE);
      glFrontFace(GL_CW);
      glCullFace(GL_BACK);  //do not render back faces

      //draw base
```

```
    glBegin(GL_POLYGON);
    for (int i = 0; i < nVertices / 2; i++)
    {
        glVertex3f(vertexList[i].x, vertexList[i].y,
vertexList[i].z);
    }
    glEnd();

    //draw walls
    for (int i = 0; i < nFaces; i++) {
        setColor(i);
        glBegin(GL_POLYGON);
        //specifying vertices of the polygon
        for (int j = 0; j < faceList[i].n; j++) {
            int vi = faceList[i].vertices[j];     //vertex index
            int ni = faceList[i].normals[j]; //normal index
            //glNormal3f(normalList[ni].x, normalList[ni].y,
normalList[ni].z);
            glVertex3f(vertexList[vi].x, vertexList[vi].y,
vertexList[vi].z);
        } //for j
        glEnd();
    }  //for i

    glFrontFace(GL_CCW);
    glCullFace(GL_BACK);  //do not render back faces
    //draw cap
    glBegin(GL_POLYGON);
    for (int i = nVertices / 2; i < nVertices; i++)
    {
        glVertex3f(vertexList[i].x, vertexList[i].y,
vertexList[i].z);
    }
    glEnd();
}

void Mesh::setColor(int n)
{
    if (n == 1 || n == 8)
        glColor3f(1, 0, 0);
    else if (n == 2 || n == 9)
        glColor3f(0, 1, 0);
    else if (n == 3 || n == 10)
        glColor3f(0, 0, 1);
    else if (n == 4 || n == 11)
        glColor3f(1, 1, 0);
```

```
      else if (n == 5 || n == 12)
           glColor3f(1, 0, 1);
      else if (n == 6 || n == 13)
           glColor3f(0, 1, 1);
      else if (n == 7 || n == 14)
           glColor3f(0, 0, 0);
      else
           glColor3f(0.5, 0.5, 0.5);
}
```

Mesh.h

```
#ifndef MESH_H
#define MESH_H

#include <vector>
#include <fstream>
#include <GL/glut.h>
#include "util3D.h"

using namespace std;

class Polygon {
public:
      int n;                     //n sides
      vector <int> vertices;     //vertex indices of vertexList;
      vector <int> normals;      //indices of normals at vertices
};

class Mesh {
public:
      int nVertices;             //number of vertices
      int nNormals;                   //number of normals
      int nFaces;                       //number of polygons
      vector<Point3> vertexList;
      vector<Vector3> normalList;
      vector <Polygon> faceList; //each face is a polygon
      Mesh();
      bool readData(char fileName[]);
      void renderMesh();              //render the mesh
      void setColor(int n);
};

#endif
```

Summary:

For this assignment we had to create a prism using sweeping. This involved creating a data file containing all points and vertices for each face. This data file is then read by the program and then is parsed into mesh.cpp where it is then r3endered out as polygons. First, the base arrow is rendered followed by each of the wall faces, and finally the cap is rendered. Overall, the assignment was a success with the program compiling and running successfully. As such I believe I have earned the full 20 points for the assignment.