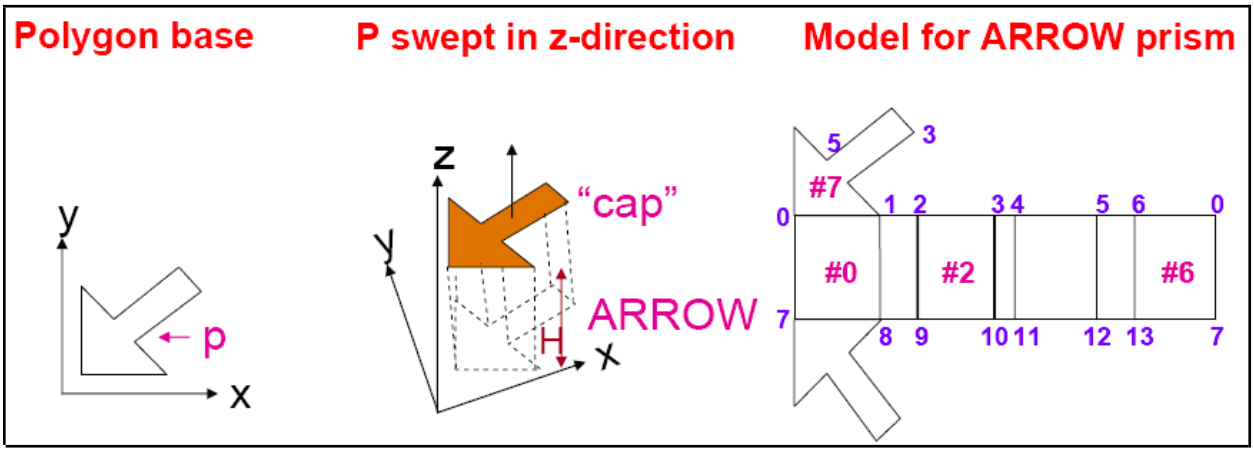
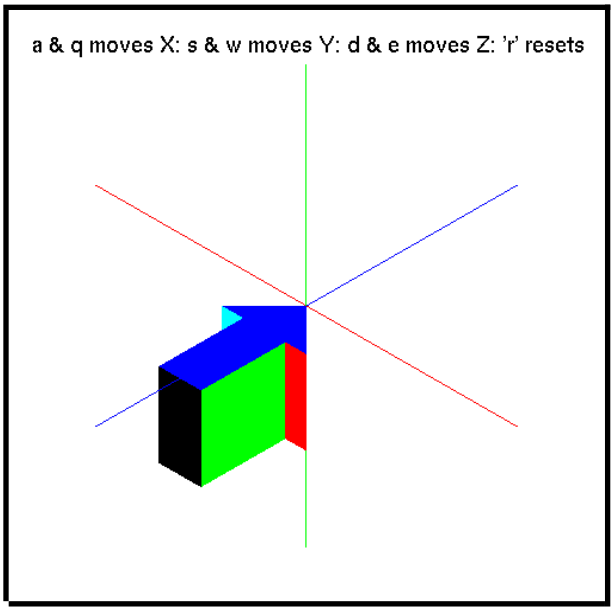
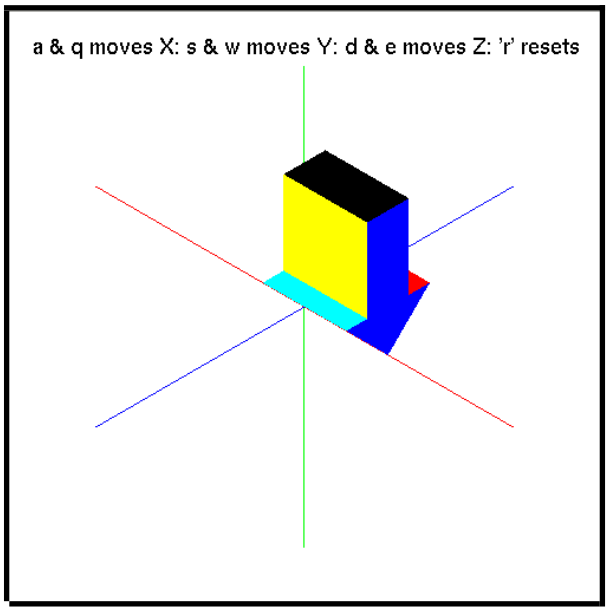
Duc Nguyen
CSE-520

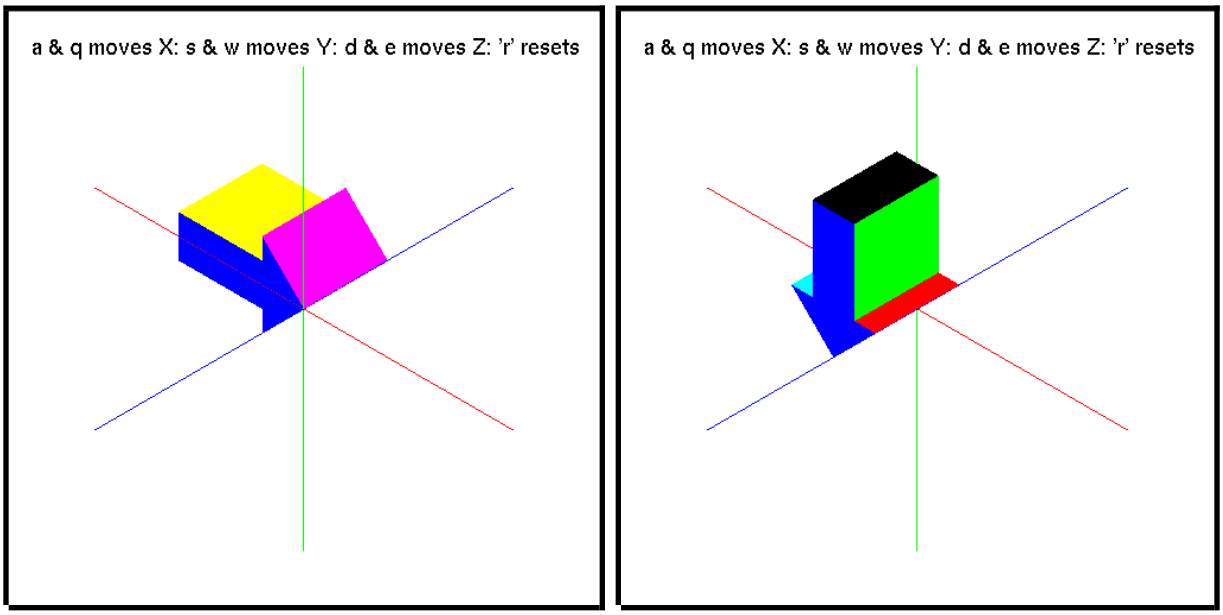
Lab 7: Extrusion

1. As shown in the figure below, draw an ARROW prism by following the below steps:
    1. Create the vertice list for the base arrow and draw the base arrow.
    2. Create the vertice list for the cap arrow and draw the cap arrow.
    3. Create the face list for the cap, base and walls.
    4. Draw the whole prism.









Report:

The intention behind this lab exercise is to teach us about the extrusion technique utilized to convert a 2-Dimensional polygon into a 3-Dimensional prism. This is an important technique in making 3D shapes with simple 2D images which can help reduce the amount of data required to have, say a complex looking shape with multitudes of 3D shapes, so for example the arrow we made could represent a house, which would take a cube and a triangular prism, however we could represent it with just a 2D polygon then extrude it to achieve the same result. This will make computation much more accessible and easier to render by not utilizing as much resources as necessary. I hope by displaying this, I earned a lab score of 20 for my efforts.

Source Code:

1. Makefile
2. display.cpp
3. util3D.cpp
4. util3D.h
5. Mesh.cpp
6. Mesh.h
7. prism.txt

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

char  DATA_FILE[100];

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

// Text Func.
void text(const char* string) {
   const char *p;
   for(p = string; *p; p++)
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *p);
}

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;
   }

//The light components that are reflected
   float mat_specular[] = { 1, 0.8, 0.4, 1.0 };
   float mat_ambient[] = { 0.8, 0.8, .2, 1.0 };
   float mat_diffuse[] = { 1, 0.8, 0.4, 1.0 };
   float mat_shininess[] = { 500.0 };
   float light_position[] = { 1.0, 1.0, 1.0, 0.0 };
   float light[] = { 1, 1, 1 };
   float lmodel_ambient[] = { 1, 1, 1, 1.0 };
   float light1[] = {1, 0, 0 };
   float light_position1[] = { -1.0, -1.0, -1.0, 0.0 };
   glClearColor (1.0, 1.0, 1.0, 0.0);
   glShadeModel (GL_SMOOTH);

   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_ambient);

   glLightfv(GL_LIGHT0, GL_POSITION, light_position);

   glLightfv(GL_LIGHT0, GL_DIFFUSE, light );
   glLightfv(GL_LIGHT0, GL_AMBIENT, light );
   glLightfv(GL_LIGHT0, GL_SPECULAR, light );


   if ( lighton ) {    //turn lightling on
      glEnable(GL_LIGHTING);
      glEnable(GL_LIGHT0);
   }
```

```
   glEnable(GL_DEPTH_TEST);
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
   glOrtho(-5.0, 5.0, -5.0, 5.0, 0.1, 100 );
   glMatrixMode(GL_MODELVIEW); // position and aim the camera
   glLoadIdentity();
   gluLookAt( 5, 5, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
   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();

        glBegin(GL_LINES);
// x axis
   glColor3f(1.0, 0.0, 0.0);
        glVertex3f(-5, 0, 0);
        glVertex3f( 5, 0, 0);
// y axis
        glColor3f(0.0, 1.0, 0.0);
        glVertex3f(0, -5, 0);
        glVertex3f(0, 5, 0);
// z axis
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(0, 0, -5);
        glVertex3f(0, 0, 5);
        glEnd();

   glColor3f(0, 0, 0);
   glRasterPos3f(-5.5, 3, 1);
        text("a & q moves X: s & w moves Y: d & e moves Z: 'r' resets");

        glFlush();
}

void keyboard ( unsigned char key, int x, int y ) {
   switch ( key ) {
   case 27:
   glutDestroyWindow(window);
   exit ( 0 );
   case 'a':
      anglex = ( anglex + 3 ) % 360;
      break;
   case 'q':
      anglex = ( anglex - 3 ) % 360;
      break;
   case 's':
      angley = ( angley + 3 ) % 360;
      break;
   case 'w':
      angley = ( angley - 3 ) % 360;
```

```
      break;
   case 'd':
      anglez = ( anglez + 3 ) % 360;
      break;
   case 'e':
      anglez = ( anglez - 3 ) % 360;
      break;
   case 'r':
      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("CSE-520: Lab 7, Extrusion");
   glutDisplayFunc(display);
   glutKeyboardFunc( keyboard );
   glClearColor( 1, 1, 1, 0 ); //white background
   glViewport ( 0, 0, 500, 500 );
   init ();
   glutMainLoop();

   return 0;
}
```

```
#include "util3D.h"
#include <math.h>

XYZ::XYZ() {
  x = y = z = 0;
}

XYZ::XYZ( double x0, double y0, double z0 ) {
  x = x0;  y = y0;  z = z0;
}

void XYZ::set( double x0, double y0, double z0 ) {
  x = x0;  y = y0;  z = z0;
}

XYZ XYZ::getXYZ()
{
  XYZ a( x, y, z );

  return a;
}

void XYZ::getXYZ( double a[] )
{
  a[0] = x;
  a[1] = y;
  a[2] = z;
}

void XYZ::print()
{
  cout << "(" << x <<", " << y << ", " << z << ")" << endl;
}


Point3::Point3():XYZ()
{
}

Point3::Point3( double x0, double y0, double z0 ): XYZ( x0, y0, z0 )
{
}

Point3::Point3( const Point3 &p )
{
  x = p.x;
  y = p.y;
  z = p.z;
}

Vector3 Point3::operator - ( const Point3 &p )
{
  Vector3 v1;

  v1.x = x - p.x;
  v1.y = y - p.y;
  v1.z = z - p.z;
```

```
  return v1;
}

Point3 Point3::operator + ( const Vector3 &v )
{
  Point3 p1;

  p1.x = x + v.x;
  p1.y = y + v.y;
  p1.z = z + v.z;

  return p1;
}


//Vector3.cpp : Vector3 class functions and external functions operating on Vector3

Vector3::Vector3():XYZ()
{
}

Vector3::Vector3( double x0, double y0, double z0 ): XYZ( x0, y0, z0 )
{
}

Vector3::Vector3( const Vector3 &v )
{
  x = v.x;
  y = v.y;
  z = v.z;
}

Vector3 Vector3::operator + ( const Vector3 &v )
{
  Vector3 v1;
  v1.x = x + v.x;
  v1.y = y + v.y;
  v1.z = z + v.z;

  return v1;
}

Vector3 Vector3::operator - ( const Vector3 &v )
{
  Vector3 v1;
  v1.x = x - v.x;
  v1.y = y - v.y;
  v1.z = z - v.z;

  return v1;
}

//cross product
Vector3 Vector3::operator ^ ( const Vector3 &v )
{
  Vector3 v1;
  v1.x = y * v.z - z * v.y;
  v1.y = z * v.x - x * v.z;
  v1.z = x * v.y - y * v.x;
```

```
  return v1;
}

//dot product
double Vector3::operator * ( const Vector3 &v )
{
  double d;
  d = x * v.x + y * v.y + z * v.z;

  return d;
}

//vector + point --> point
Point3 Vector3::operator + ( const Point3 &p )
{
  Point3 p1;

  p1.x = x + p.x;
  p1.y = y + p.y;
  p1.z = z + p.z;

  return p1;
}

double Vector3::magnitude()
{
  return sqrt(x * x + y * y + z * z );
}

void Vector3::normalize()
{
  double d = x*x + y*y + z*z;

  if ( d > 0 ) {
    d = sqrt ( d );
    x /= d;
    y /= d;
    z /= d;
  }
}

//------------------- external functions ---------------

//scalar times vector
Vector3 operator * ( double a, const Vector3 &v )
{
  Vector3 v1;
  v1.x = a * v.x;
  v1.y = a * v.y;
  v1.z = a * v.z;

  return v1;
}

Vector3 operator * ( const Vector3 &v, double a )
{
  return a * v;
}
```

util3D.h

```
#ifndef UTIL3D_H
#define UTIL3D_H
#include <vector>
#include <iostream>

using namespace std;

class XYZ {
  public:
    double x;
    double y;
    double z;
    XYZ ();
    XYZ ( double x0, double y0, double z0 );
    void set ( double x0, double y0, double z0 );
//    XYZ &getXYZ( XYZ &xyz );       //return by reference, this works also
    XYZ getXYZ();
    void getXYZ( double a[]  );
    void print();
};

class Vector3;        //forward declaration

class Point3: public XYZ
{
  public:
    Point3();
    Point3( double x0, double y0, double z0 );
    Point3( const Point3 &p);
    Vector3 operator - ( const Point3 &p );    //point - point --> vector
    Point3  operator + ( const Vector3 &v ); //point + vector --> point
};

class Vector3: public XYZ
{
  public:
    Vector3();
    Vector3( double x0, double y0, double z0 );
    Vector3( const Vector3 &v );
    Vector3 operator + ( const Vector3 &v ); //vector + vector --> vector
    Vector3 operator - ( const Vector3 &v );    //vector - vector --> vector
    Vector3 operator ^ ( const Vector3 &v ); //cross product
    double  operator * ( const Vector3 &v ); //dot product
    Point3  operator + ( const Point3 &p );  //vector + point --> point
    double magnitude();
    void  normalize();                      //make it a unit vector
};

Vector3 operator * ( double a, const Vector3 &v );
Vector3 operator * ( const Vector3 &v, double a );

#endif
```

Mesh.cpp

```
/*
 * Lab 7: Extrusion
 * Modified Code of Dr. Tong Yu
*/

#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()
{
        glPolygonMode( GL_FRONT, GL_FILL );
        glPolygonMode( GL_BACK, GL_FILL );

       //draw one polygon at a time
        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
}

void Mesh::setColor( int n )
{
        if (n == 0)
           glColor3f(0.5, 0.5, 0.5);
        else if ( n == 1 )
           glColor3f(1, 0, 0 );
        else if ( n == 2 )
           glColor3f(0, 1, 0 );
        else if ( n == 3 )
           glColor3f(0, 0, 0 );
        else if ( n == 4 )
           glColor3f(1, 1, 0 );
        else if ( n == 5 )
           glColor3f(0, 1, 1 );
        else if ( n == 6 )
           glColor3f(1, 0, 1 );
        else
           glColor3f(0, 0, 1 );
}
```

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
```

```
14 9 9
0 0 0   1 1 0   0.5 1 0   0.5 3 0   -0.5 3 0  -0.5 1 0  -1 1 0
0 0 2   1 1 2   0.5 1 2   0.5 3 2   -0.5 3 2  -0.5 1 2  -1 1 2

0.707 -0.707 0    0 1 0   1 0 0   0 1 0   0 -1 0   0 1 0  -0.707 -0.707 0
0 0 1   0 0 -1

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 6
7   0 1 2 3 4 5 6    7 7 7 7 7 7 7
7   7 8 9 10 11 12 13   8 8 8 8 8 8 8
```