# Lab06 for CPT205 Computer Graphics

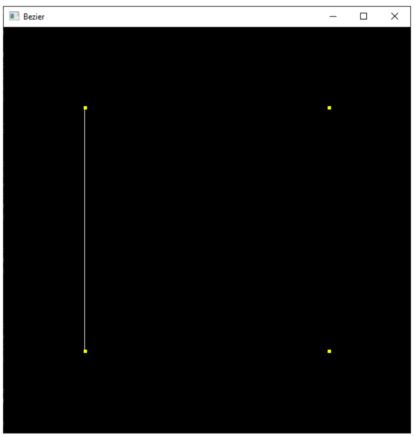
## Part 1. Bezier Curves

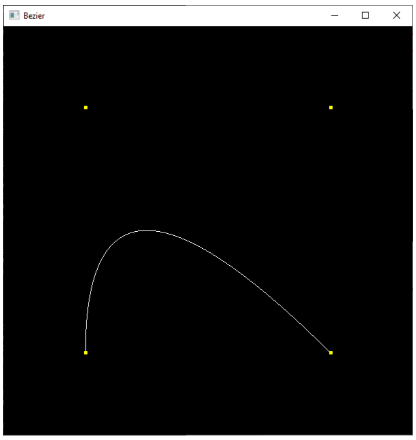
A Bezier curve is defined by a set of control points $P_0$ through $P_n$, where n is called its order (n = 1 for linear, 2 for quadratic and so on). The first and last control points are the end points of the curve; however, the intermediate control points (if any) may not lie on the curve.

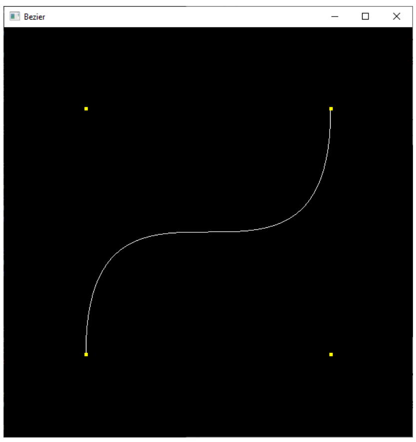
Type in the following code, compile and run it. Get a good understanding of what the program does.

```
// Bezier_Curves.cpp

#define FREEGLUT_STATIC
#include <GL/freeglut.h>

GLfloat step = 2;

GLfloat ctrlpoints[4][3] =  // 4 points to define a cubic Bezier curve
{
   { -2.0, -2.0, 0.0 },      // Point 0
   { -2.0,  2.0, 0.0 },      // Point 1
   {  2.0, -2.0, 0.0 },      // Point 2
   {  2.0,  2.0, 0.0 } };    // Point 3

void when_in_mainloop()  // idle callback function
{
   glutPostRedisplay(); // force OpenGL to redraw the current window
}

void keyboard_input(unsigned char key, int x, int y)
{
   if (key == 'q' || key == 'Q')
      exit(0);
   else if (key == 'i' || key == 'I') {
      step++;
      if (step > 4)
         step = 4;
   }
   else if (key == 'd' || key == 'D') {
      step--;
      if (step < 2)
         step = 2;
   }
}

void myinit(void)
{
   glClearColor(0.0, 0.0, 0.0, 1.0);

   /* glMap1f(GLenum target, GLfloat u1, GLfloat u2, GLint stride, GLint order, const GLfloat *points)
   glMap1f defines a one-dimensional evaluator.
   target specifies what the control points represent. It also specifies the type of points, e.g.
   GL_MAP1_VERTEX_3: Each control point has three floating-point values representing x, y, and z.
   GL_MAP1_VERTEX_4: Each control point has four floating-point values representing x, y, z and w.
   GL_MAP1_TEXTURE_COORD_3: Each control point has three floating-point values representing the s, t, and
   r texture coordinates.
   GL_MAP1_TEXTURE_COORD_4: Each control point is four floating-point values representing the s, t, r and
   q texture coordinates.
   u1 and u2 specify a linear mapping of u.
   stride specifies the number of floats or doubles between the beginning of a control point and the
   beginning of the next in the data structure referenced in points.
   order specifies the number of control points. It must be positive.
   points specifies a pointer to the array of control points */
   glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, step, &ctrlpoints[0][0]);

   glEnable(GL_MAP1_VERTEX_3); // Enable the evaluator.
}

void myDisplay(void)
{
   myinit();

   glClear(GL_COLOR_BUFFER_BIT);
```

```
    //Plot Bezier Curve between Point 1 and Point 4
    glColor3f(1.0, 1.0, 1.0);
    int number = 50;
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i <= number; i++)
    {
       /* glEvalCoord1f evaluates the one-dimensional maps that are enabled.
       void glEvalCoord1f(GLfloat u);
       u specifies a value that is the domain coordinate u to the basis function defined in a previous
       glMap1 function. */
       glEvalCoord1f((GLfloat)i / number);
    }
    glEnd();

    //Plot 4 Points
    glPointSize(5.0);
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_POINTS);
    for (int i = 0; i < 4; i++)
    {
       glVertex3fv(&ctrlpoints[i][0]);  // Draw each control point
    }
    glEnd();

    glFlush();
}

void myReshape(GLsizei w, GLsizei h)
{
    glViewport(0, 0, w, h);  // define the viewport

    glMatrixMode(GL_PROJECTION);  // the projection transformation
    glLoadIdentity();  // clear the matrix
    glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);

    glMatrixMode(GL_MODELVIEW);  // back to model-view matrix
    glLoadIdentity();  // clear the matrix
    gluLookAt(0, 0, 5, 0, 0, 0, 0, 1, 0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Bezier");

    glutReshapeFunc(myReshape);
    glutDisplayFunc(myDisplay);

    glutIdleFunc(when_in_mainloop);

    glutKeyboardFunc(keyboard_input);

    glutMainLoop();
}
```







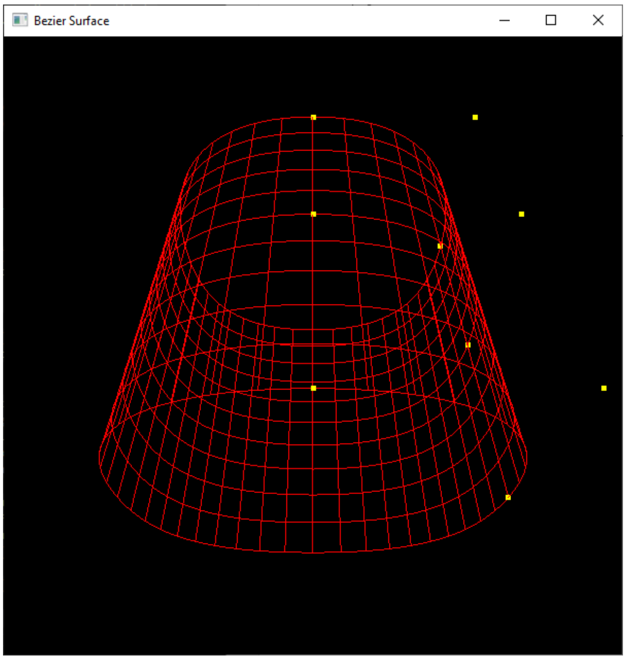
## Part 2. Bezier Surfaces

Type in the following code, compile and run it. You will get a Bezier Surface.

```
// Bezier Surfaces

#define FREEGLUT_STATIC
#include <GL/freeglut.h>

GLfloat r = 0; // incremental step for angle of rotation

//Control points
GLfloat ctrlpoints[3][3][3] =
{
   {{0.0, 2.0, 2.0},  {2.0, 2.0, 2.0},  {2.0, 0.0, 2.0}},
   {{0.0, 2.0, 0.0},  {2.0, 2.0, 0.0},  {2.0, 0.0, 0.0}},
   {{0.0, 2.0, -2.0}, {2.0, 2.0, -2.0}, {2.0, 0.0, -2.0}},
};

void keyboard_input(unsigned char key, int x, int y)
{
   if (key == 'q' || key == 'Q')
      exit(0);
   else if (key == 'i' || key == 'I')
   {
      r += 1;
      if (r >= 360)
         r = 0;
   }
   else if (key == 'd' || key == 'D') {
      r -= 1;
      if (r <= 0)
         r = 360;
   }
}

void when_in_mainloop()  // idle callback function
{
   glutPostRedisplay(); // force OpenGL to redraw the current window
}

void myinit(void)
{
   glClearColor(0.0, 0.0, 0.0, 1.0);

   /* glMap2f defines a two-dimensional evaluator.
   void glMap2f(GLenum target, GLfloat u1, GLfloat u2, GLint ustride, GLint uorder, GLfloat v1, GLfloat v2,
   GLint vstride, GLint vorder, const GLfloat * points);
   Compared with glMap1f, glMap2f adds v1 and v2 to specify a linear mapping of v. The definitions of
   vstride and vorder are the same as ustride and uorder. */
   glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 3, 0, 1, 9, 3, &ctrlpoints[0][0][0]);
   glEnable(GL_MAP2_VERTEX_3);

   /* glMapGrid2f defines a two-dimensional mesh.
   void glMapGrid2d(GLint un, GLdouble u1, GLdouble u2, GLint vn, GLdouble v1, GLdouble v2);
   un specifies the number of partitions in the grid range interval [u1, u2]. Must be positive.
   u1 and u2 specify the mappings for integer grid domain values i = 0 and i = un.
   vn specifies the number of partitions in the grid range interval [v1, v2].
   v1 and v2 specify the mappings for integer grid domain values j = 0 and j = vn. */
   glMapGrid2f(20, 0.0f, 1.0f, 20, 0.0f, 1.0f);

   //Do depth comparisons and update the depth buffer.
   glEnable(GL_DEPTH_TEST);
}

void myDisplay(void)
{
   myinit();

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   //Plot Bezier surface
   int num1 = 10;
   int num2 = 30;

   glPushMatrix();
```

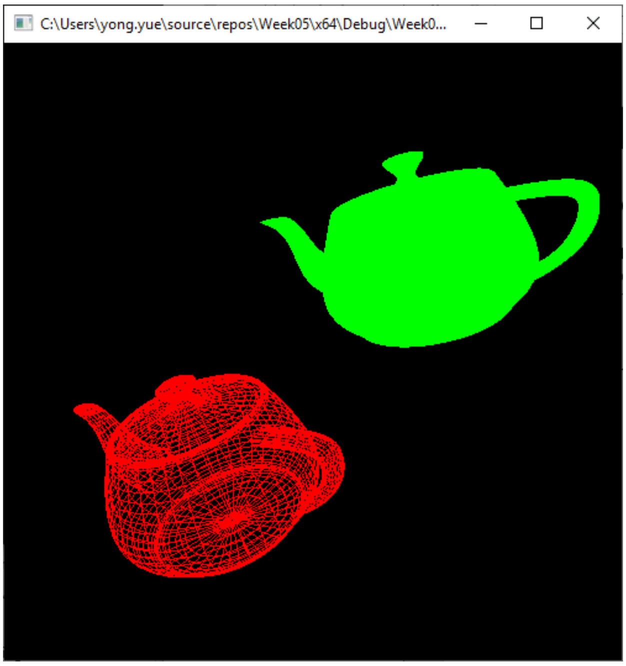
```
    glRotatef(r, 1, 0, 0);

    glPushMatrix();

    for (int m = 0; m <= 3; m++)  //Draw a quarter circle at a time
    {
       glRotatef(90, 0.0f, 0.0f, 1.0f);
       glColor3f(1.0, 0.0, 0.0);
       for (int j = 0; j <= num1; j++)
       {
          glBegin(GL_LINE_STRIP);
          for (int i = 0; i <= num2; i++)
          {
             /* glEvalCoord2f evaluates the one-dimensional maps that are enabled.
             void glEvalCoord2f(GLfloat u, GLfloat v);
             u specifies a value that is the domain coordinate u to the basis function defined in a
             previous glMap2 function.
             v specifies a value that is the domain coordinate v to the basis function defined in a
             previous glMap2 function. */
             glEvalCoord2f((GLfloat)i / num2, (GLfloat)j / num1);
          }
          glEnd();

          glBegin(GL_LINE_STRIP);
          for (int i = 0; i <= num2; i++) {
             glEvalCoord2f((GLfloat)j / num1, (GLfloat)i / num2);
          }
          glEnd();
       }
    }

    //Plot 9 points
    int m = 0;
    glPointSize(5.0);
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_POINTS);
    for (int j = 0; j < 3; j++)
    {
       for (int i = 0; i < 3; i++)
       {
          glVertex3fv(&ctrlpoints[j][i][m]);
       }
    }
    glEnd();

    glPopMatrix();
    glPopMatrix();

    glFlush();
}

void myReshape(GLsizei w, GLsizei h)
{
    glViewport(0, 0, w, h);                 // define the viewport

    glMatrixMode(GL_PROJECTION);            // projection transformation
    glLoadIdentity();                       // clear the matrix
    glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
    gluLookAt(0, 0, 6, 0, 0, 0, 0, 1, 0);

    glMatrixMode(GL_MODELVIEW);             // back to modelview matrix
    glLoadIdentity();                       // clear the matrix
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Bezier Surface");

    glutDisplayFunc(myDisplay);
    glutKeyboardFunc(keyboard_input);
    glutReshapeFunc(myReshape);
    glutIdleFunc(when_in_mainloop);
```

```
    glutMainLoop();
}
```




## Part 3. Teapots

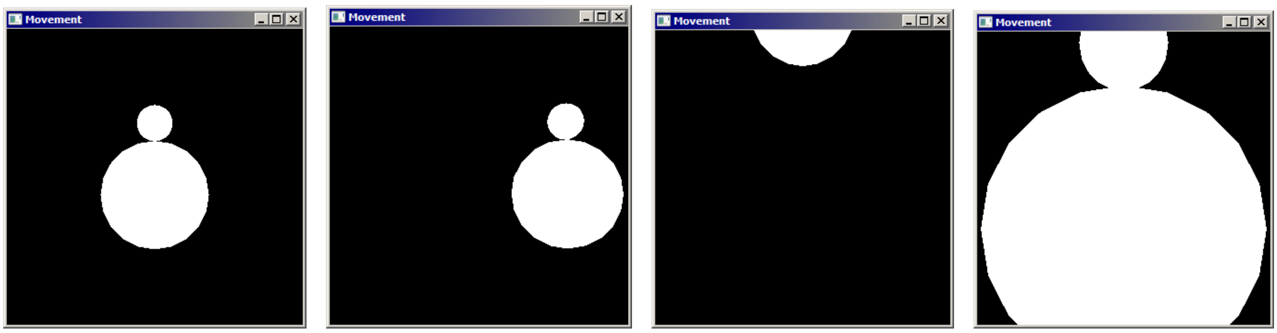
This part consists of a header file (**Teapot.h**) and a source code file (**Teapot.cpp**) for creating a wireframe teapot and a solid teapot. Read and load the two files into a new project, compile and run it. You will see the two teapots which you can rotate around the three axes.



## Part 4. Zoom and Pan (Further Sample Code for Viewing and Projection – Lab 05)

```
// Zoom and Pan

#define FREEGLUT_STATIC
#include <GL/freeglut.h>

//Hide the terminal windows
#pragma comment (linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"")

// XYZ position of the camera
float x = 0.0f, y = 1.0f, z = 5.0f;

void changeSize(int w, int h)
{
   // Prevent a divide by zero, when window is too short
   if (h == 0)
      h = 1;
   float ratio = w * 1.0 / h;

   // Use the Projection Matrix
   glMatrixMode(GL_PROJECTION);

   // Reset Matrix
   glLoadIdentity();

   // Set the viewport to be the entire window
   glViewport(0, 0, w, h);

   // Set the correct perspective.
   gluPerspective(45.0f, ratio, 0.1f, 100.0f);

   // Get Back to the Modelview
   glMatrixMode(GL_MODELVIEW);
}

// Draw a simple man body
void drawMan()
{
   glColor3f(1.0f, 1.0f, 1.0f);
   // Draw Body
   glTranslatef(0.0f, 0.75f, 0.0f);
   glutSolidSphere(0.75f, 20, 20); // radius, lines of longitude and lines of latitude

   // Draw Head
   glTranslatef(0.0f, 1.0f, 0.0f);
   glutSolidSphere(0.25f, 20, 20);
}

void renderScene(void)
{
   // Clear Color and Depth Buffers
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   // Reset transformations
   glLoadIdentity();
   // Set the camera
   gluLookAt(x, y, z, x, y, z - 1.0, 0.0f, 1.0f, 0.0f);

   glPushMatrix();
   glTranslatef(0.0, 0.0, 0.0);
   drawMan();
   glPopMatrix();

   glutSwapBuffers();
}

void processNormalKeys(unsigned char key, int x, int y)
{

   //Press ESC quit
   if (key == 27)
      exit(0);
}

void processSpecialKeys(int key, int xx, int yy)
{
   float fraction = 0.2f;
```

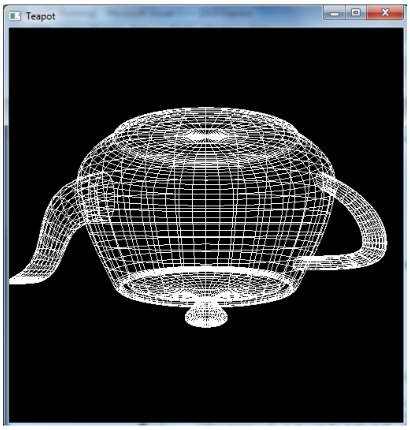
```
    switch (key)
    {
       //Camera move left
    case GLUT_KEY_LEFT:
       x -= fraction;
       break;
       //Camera move right
    case GLUT_KEY_RIGHT:
       x += fraction;
       break;
       // Camera move up
    case GLUT_KEY_UP:
       y += fraction;
       break;
       // Camera move down
    case GLUT_KEY_DOWN:
       y -= fraction;
       break;
       // Zoom out
    case GLUT_KEY_PAGE_UP:
       z += fraction;
       break;
       // Zoom in
    case GLUT_KEY_PAGE_DOWN:
       z -= fraction;
       break;
    }
}

int main(int argc, char** argv)
{
    // init GLUT and create window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(320, 320);
    glutCreateWindow("Movement");

    // register callbacks
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutIdleFunc(renderScene);  // draw content when there is no window event
    glutKeyboardFunc(processNormalKeys);
    glutSpecialFunc(processSpecialKeys);

    // OpenGL init
    glEnable(GL_DEPTH_TEST);

    // enter GLUT event processing cycle
    glutMainLoop();

    return 1;
}
```



## Part 5. Sample Solutions to Tasks 2-6 of Part 1.5 for Lab05 (Viewing and Projection)

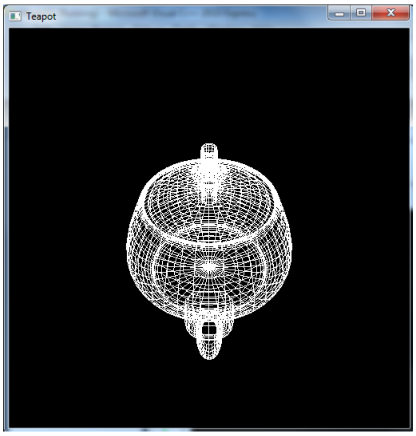
**Task2:** Specify different values for the **gluLookAt** function to see the following image.

```
gluLookAt(0,0,5,0,0,0,0,-1,0);
```



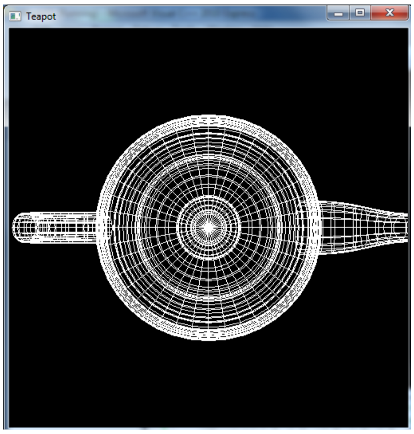

**Task3:** Specify different values for the **gluLookAt** function to see the following image.

```
gluLookAt(5,5,0,0,0,0,0,1,0);
```



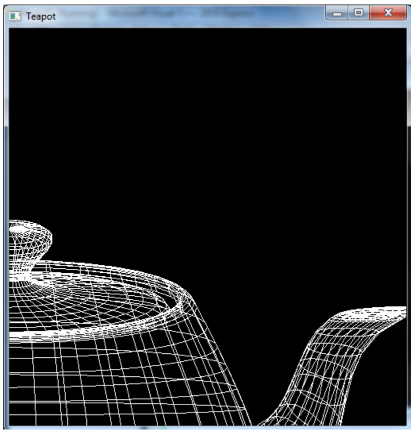

**Task4:** Specify different values for the **gluLookAt** function to see the following image.

```
gluLookAt(0,5,0,0,0,0,0,0,-1);
```




**Task5:** Specify different values for the **glFrustum** function to see the following image.

```
glFrustum(0.0, 1.0, 0.0, 1.0, 1.5, 20.0);
gluLookAt(0, 0, 5, 0, 0, 0, 0, 1, 0);
```




**Task6:** Add the **glRotatef** function to see the following image.

```
glRotatef(90, 1, 0, 0);
```