CURSO: CC322 - 2017

Practica 9

1. Introducción

Objetivo general:
Generacion de curvas Bezier.

2 . Recursos Informáticos

https://www.opengl.org
https://www.khronos.org/opengl/wiki/Getting_Started
https://www.khronos.org/opengl/wiki/Category:Core_API_Reference
https://www.python.org/
https://www.jetbrains.com/pycharm/

3. DESARROLLO

1. Verifique si tiene instalado un compilador de codigo en C (cpp, g++)
Sino, proceda a instalarlo en su PC

2. Verifique si tiene un editor de texto adecuado para editar programas en C
(p.ej. Geany, SublimeText, u otro con el que esté familiarizado) si no está – Proceda a Instalarlo.

3.Verifique si tiene instaladas las librerias OPENGL Y GLUT. Si no están proceda a instalarlas.

```
sudo apt-get install freeglut3-dev
```

4. Al final entregara un archivo con el nombre CC322_Lab09_<Nombre_apellido>.zip con los archivos generados en la practica.

4 . Ejemplos Prácticos

Ejemplo 4.1.

Baje del sitio web del curso, el siguiente programa (Lab09_2.cpp).

```
// Bezier Curve Demo
// Uses OpengL evaluator & evaluator grid


#include <vector>
using std::vector;
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;
#include <cstdlib>       // Do this before GL/GLUT includes
using std::exit;
#ifndef __APPLE__
# include <GL/glut.h>    // GLUT stuff, includes OpenGL headers as well
#else
# include <GLUT/glut.h>  // Apple puts glut.h in a different place
#endif


// Global variables
// Keyboard
const int ESCKEY = 27;         // ASCII value of Escape


// Window/viewport
const int startwinsize = 400;  // Start window width & height (pixels)


// Object: constants
```

```
const double vertsize = 20.;   // Size of control pts (pixels)
const double curvewid = 8.;    // Width of drawn curve (pixels)
const double polywid = 2.;     // Width of control polygon (pixels)
const int numcurvesegments = 100;
                               // Number of line segs per curve segment


// Object: variables
vector<vector<GLdouble> > verts;
                               // Vector of 2-D control pts
bool currentlymoving;          // Are we dragging a control pt?
int selectedvert;              // Index of dragged pt;
                               //  valid if currentlymoving
vector<GLdouble> savevert;     // Start coords of dragged control pt
int savemx, savemy;            // Start mouse pos when dragging (pixels)
bool drawpoly;                 // True if control polygon drawn
bool drawverts;                // True if control points drawn


// class BitmapPrinter
// For drawing text using GLUT bitmap text facility.
// Usage:
//     BitmapPrinter p(-0.9, 0.9, 0.1); // Start text x, y, line height
//     p.print("Hello");                // 1st line, start @ (-0.9, 0.9)
//     p.print("there");                // 2nd line, start @ (-0.9, 0.8)
class BitmapPrinter {


// ***** BitmapPrinter: public functions *****
public:


    // Ctor (0, 1, 2, or 3 params)
    // Set cursx, cursy, lineht to the given values.
    BitmapPrinter(double theCursx = 0.,
                  double theCursy = 0.,
                  double theLineht = 0.1)
    { setup(theCursx, theCursy, theLineht); }


    // Compiler-generated copy ctor, copy =, dctor used


    // setup
    // Set cursx, cursy, lineht to the given values.
    void setup(double theCursx,
               double theCursy,
               double theLineht = 0.1)
    { cursx = theCursx; cursy = theCursy; lineht = theLineht; }


    // print
    // Draw the given string, using glutBitmapCharacter, with GLUT's
    // 9x15 font, at cursx, cursy, and then reduce cursy by lineht
    // (i.e., move to the next line).
    //
    // The model/view transformation should probably be the identity
    // (or just translations) when calling this function.
    void print(const std::string & msg)
    {
        glRasterPos2d(cursx, cursy);
        for (std::string::const_iterator ii = msg.begin();
             ii != msg.end();
             ++ii)
        {
            glutBitmapCharacter(GLUT_BITMAP_9_BY_15, *ii);
        }
        cursy -= lineht;
    }


// ***** BitmapPrinter: data members *****
private:


    double cursx, cursy;  // Initial raster pos for text line: x, y
    double lineht;        // How much to reduce cursy each line


};  // End class BitmapPrinter


// drawBezierCurve
// Draws bezier curve, given vector of control points, and number of
//  segments to draw
// Uses OpenGL evaluator
void drawBezierCurve(
    const vector<vector<GLdouble> > & v,  // ctrl pts
    int segs)                             // num segments
{
    // Make sure valid number of points for OpenGL evaluator
    if (v.size() == 0 || v.size() > 8)
        return;
    // Put control points into array
    vector<GLdouble> p(3*v.size(), 0.);
    for (int i = 0; i < v.size(); ++i)
    {
        p[3*i+0] = v[i][0];
        p[3*i+1] = v[i][1];
    }


    // Set up evaluator
    glMap1d(GL_MAP1_VERTEX_3,  // Target
            0., 1.,            // Min/max parameter values
            3,                 // Stride of pts in array
            v.size(),          // Number of control pts
            &p[0]);            // Ptr to data
    glEnable(GL_MAP1_VERTEX_3);


    // And now render using the evaluator


    // Here is how to do the rendering with glEvalCoord*
    /*
    glBegin(GL_LINE_STRIP);
        for (int i = 0; i <= segs; ++i)
        {
            double t = double(i)/segs;  // parameter
            glEvalCoord1d(t);
        }
    glEnd();
    */


    // Here is how to do the rendering with an evaluator grid
    glMapGrid1d(segs, 0., 1.);      // Set up grid
    glEvalMesh1(GL_LINE, 0, segs);  // Do the actual drawing
}


// myDisplay
// The GLUT display function
void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);


    // Draw control points ("verts")
    if (drawverts)
    {
        glPointSize(vertsize);
        glBegin(GL_POINTS);
            for (int i=0; i<verts.size(); ++i)
            {
                if (i==selectedvert)
                    glColor3d(1., 0., 0.);
                else
                    glColor3d(0., 0., 0.);
                glVertex2dv(&verts[i][0]);
            }
        glEnd();
    }


    // Draw control polygon
    if (verts.size()>=2 && drawpoly)
    {
        glLineWidth(polywid);
        glColor3d(0.4, 0.4, 0.4);
        glBegin(GL_LINE_STRIP);
            for (int i=0; i<verts.size(); ++i)
                glVertex2dv(&verts[i][0]);
        glEnd();
    }


    // Draw curve
    glColor3d(0., 0., 1.);
    glPointSize(curvewid);
    glLineWidth(curvewid);
    drawBezierCurve(verts, numcurvesegments);


    // Draw documentation
    glLoadIdentity();
    glMatrixMode(GL_PROJECTION);  // Set up simple ortho projection
    glPushMatrix();
    glLoadIdentity();
    gluOrtho2D(-1., 1., -1., 1.);
    glColor3d(0., 0., 0.);        // Black text
    BitmapPrinter p(-0.9, 0.9, 0.1);
    p.print("Click to create new control point");
    p.print("Click & drag to move point");
    p.print("P       Toggle polygon");
    p.print("C       Toggle control points");
    p.print("Space   Delete last control point");
    p.print("         (NOT selected point)");
    p.print("Esc     Quit");
    glPopMatrix();                // Restore prev projection
    glMatrixMode(GL_MODELVIEW);


    glutSwapBuffers();
}


// myIdle
// The GLUT idle function
void myIdle()
{
    // Print OpenGL errors, if there are any (for debugging)
    if (GLenum err = glGetError())
    {
        cerr << "OpenGL ERROR: " << gluErrorString(err) << endl;
    }
}


// myKeyboard
// The GLUT keyboard function
void myKeyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case ESCKEY:  // Esc: quit
        exit(0);
        break;
    case 'p':     // P: toggle control polygon drawing
    case 'P':
        drawpoly = !drawpoly;
        glutPostRedisplay();
        break;
    case 'c':     // C: toggle control point drawing
    case 'C':
        drawverts = !drawverts;
        glutPostRedisplay();
        break;
    case ' ':     // Space: delete last control pt
        if (verts.size() > 0)
        {
            verts.pop_back();
            selectedvert = -1;
            currentlymoving = false;
            glutPostRedisplay();
        }
        break;
    }
}


// myReshape
// The GLUT reshape function
void myReshape(int w, int h)
{
    // Set viewport
    glViewport(0, 0, w, h);


    // Set up projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.,(double)w,
               (double)startwinsize-h,(double)startwinsize);


    glMatrixMode(GL_MODELVIEW);  // Always go back to model/view mode
}


// myMouse
// The GLUT mouse function
void myMouse(int button, int state, int x, int y)
{
    if (button != GLUT_LEFT_BUTTON)
        return;


    if (state == GLUT_UP)
    {
        currentlymoving = false;
        return;
    }


    int mousex = x;
    int mousey = startwinsize-y;
    // Check if clicked on a vert
    int i;
    for (i = 0; i < verts.size(); ++i)
    {
        int slop = (vertsize/2)+2;
        if (mousex >= verts[i][0]-slop
            && mousex <= verts[i][0]+slop
            && mousey >= verts[i][1]-slop
            && mousey <= verts[i][1]+slop) break;
    }

    // If did not click on a vert, make a new one
    if (i == verts.size())
    {
        verts.push_back(vector<GLdouble>(2));
        verts[i][0] = mousex;
        verts[i][1] = mousey;
    }

    // Select vert & get ready for moving
    selectedvert = i;
    savemx = mousex; savemy = mousey;
    savevert = verts[i];
    currentlymoving = true;

    glutPostRedisplay();
}

// myMotion
// The GLUT motion function
void myMotion(int x, int y)
{
    if (!currentlymoving) return;

    int mousex = x;
    int mousey = startwinsize-y;

    verts[selectedvert][0] = savevert[0]+mousex-savemx;
    verts[selectedvert][1] = savevert[1]+mousey-savemy;

    glutPostRedisplay();
}

// init
// Initializes GL states
// Called by main after window creation
void init()
{
    // Object properties
    currentlymoving = false;
    selectedvert = -1;
    drawpoly = true;
    drawverts = true;

    // OpenGL Stuff
    glClearColor(1.0, 1.0, 1.0, 0.0);  // white background
}

int main(int argc, char ** argv)
{
    // Initialize OpenGL/GLUT
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

    // Make a window
    glutInitWindowSize(startwinsize, startwinsize);
    glutInitWindowPosition(50, 50);
    glutCreateWindow("CC 322 - Bezier Curve");

    // Initialize GL states & register GLUT callbacks
    init();
    glutDisplayFunc(myDisplay);
    glutIdleFunc(myIdle);
    glutKeyboardFunc(myKeyboard);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutMotionFunc(myMotion);

    // Do something
    glutMainLoop();
    return 0;
}
```

a) Modifique el programa **Lab09_2.cpp** para dibujar 2 curvas adicionales, cada una de diferente color, y que puedan ser controladas por separado.

b) Forme una curva continua, que conste de la union de 3, haga un screenshot.