

Richard Hankey p151406 parametric curves

by Rich Harris

FILE	6496_RICH_HANKEY_RICHARD_HANKEY_P151406_PARAMETRIC_CURVES_3 11302_1539832776.DOCX (624.65K)		
TIME SUBMITTED	04-DEC-2020 05:00PM (UTC+0000)	WORD COUNT	3011
SUBMISSION ID	138362675	CHARACTER COUNT	21686

ACCA6012

Graphics and Games Programming

**Assignment 1:
Parametric Curves**

Richard Hankey

P151406

Table of Contents

Introduction.....	3
User Guide.....	3
Algorithms.....	3
Output.....	6
References and Text Consulted for Visual Studio Solution.....	9
Appendix.....	9
globals.h.....	9
cRenderClass.h.....	10
cRenderClass.cpp.....	10
MouseControl.h.....	16
MouseControl.cpp.....	17
PixelData.h.....	19
PixelData.cpp.....	20
stdafx.h.....	22
GraphicsTemplate.cpp.....	22

Introduction.

This report will detail the classes and algorithms, including a breakdown of one iteration of the algorithm, used in the Visual Studio Project Titled: "Parametric Curves". (Unfortunately, the file folder is actually called "Graphics Template", due to renaming this file causes corruption).

Further to this, the report will contain screenshots and code snippets of the solution to showcase what has been implemented. The Appendix will have the Visual Studio Project Solution in its entirety if application is required.

User Guide.

Upon running the program solution a black window will open. User will then click on the window to draw a single pixel. Further clicking on the window will result in more pixels. This not exhaustive as the drawing of pixels onto to this screen is dynamically created. When user has reached four pixels onto the screen a Uniform Non-Rational B-Spline will be drawn onto the screen. One more click onto the window will output a second B-spline connecting to the previous B-spline.

User can then move the pixels by moving the mouse over each pixel and slowly moving the cursor in the desired direction and the pixel will follow. To stop the moving of the pixels, simply make a sudden movement of the cursor in any direction. If user draws extra pixels nothing will happen, this is to allow further B-splines to be drawn on the screen when this solution is further implemented.

Algorithms.

The B-spline has been derived by using the polynomial equation:

$$\frac{1}{6} ((1-t)^3 \quad 3t^3 - 6t^2 + 4 \quad -3t^3 + 3t^2 + 3t + 1 \quad t^3) = \begin{pmatrix} C1 \\ C2 \\ C3 \\ C4 \end{pmatrix}$$

Equation a

Where t is the incremental value of the curve segment.

The polynomial equation has been implemented as a class member function like so:

```
float * cRenderClass::basisCalc(float t)
{

    float basisFuncs[4]{ (1 - t)*(1 - t)*(1 - t) , (3 * t*t*t - 6 * t*t + 4) , (-3 * t*t*t + 3 * t*t + 3 * t + 1) , (t*t*t) };

    return basisFuncs;

}
```

Figure 1

```
float *p0 = pixel[0].getPixelData();
float *p1 = pixel[1].getPixelData();
float *p2 = pixel[2].getPixelData();
float *p3 = pixel[3].getPixelData();
float *p4 = pixel[4].getPixelData();
float *p5 = pixel[5].getPixelData();
float point[2];

if (i >= 3)
{
    for (float t = 0; t < 1; t += 0.001)
    {
        if (i != 4)
        {
            float *base = graphics.basisCalc(t);

            point[0] = (p0[0] * base[0] + p1[0] * base[1] + p2[0] * base[2] + p3[0] * base[3]) * 1 / 6; //x values
            point[1] = (p0[1] * base[0] + p1[1] * base[1] + p2[1] * base[2] + p3[1] * base[3]) * 1 / 6; //y values
            graphics.drawPixel(point[0], point[1]);
        }
        else
        {
            float *base = graphics.basisCalc(t);

            point[0] = (p1[0] * base[0] + p2[0] * base[1] + p3[0] * base[2] + p4[0] * base[3]) * 1 / 6;
            point[1] = (p1[1] * base[0] + p2[1] * base[1] + p3[1] * base[2] + p4[1] * base[3]) * 1 / 6;
            graphics.drawPixel(point[0], point[1]);
        }
    }
}
```

Figure 2

By returning the values as a pointer to an array of floats, this allows the calculations to be separated for each position of the control point.

For example in Figure 2, p0[0](which is the x position), is multiplied by the base[0], which in fact is the C1 of equation a, and p1 is multiplied by base[1] and so on.

For calculating the first spline the algorithm uses pixels p0 to p3 and for the additional spline it uses pixels p1 to p4.
To dig deeper into this algorithm: The t value is set at 0 and is incremented by 0.001 every iteration. Equation b details the second iteration of the iteration process.

The 4 control points are:

P0x: 100	P1x: 150	P2x: 200	P3x: 250
P0y: 100	P1y: 50	P2y: 100	P3y: 50

And the basis calculations are :

$$\begin{array}{l}
 C1 \\
 C2 \\
 C3 \\
 C4
 \end{array}
 \begin{pmatrix}
 (1-t)^3 \\
 3t^3 - 6t^2 + 4 \\
 -3t^3 + 3t^2 + 3t + 1 \\
 t^3
 \end{pmatrix}
 \frac{1}{6}$$

The calculation would be as thus:

$$\begin{aligned}
 P'_x &= \begin{pmatrix} P0x: 100 * C1 \\ P1x: 350 * C2 \\ P2x: 500 * C3 \\ P3x: 850 * C4 \end{pmatrix} \frac{1}{6} \\
 P'_y &= \begin{pmatrix} P0y: 600 * C1 \\ P1y: 300 * C2 \\ P2y: 600 * C3 \\ P3y: 300 * C4 \end{pmatrix} \frac{1}{6} \\
 &= \begin{pmatrix} P0x: 100 * (1-t)^3 + P1x: 350 * 3t^3 - 6t^2 + 4 + P2x: 500 * -3t^3 + 3t^2 + 3t + 1 + P3x: 850 * t^3 \\ P0y: 600 * (1-t)^3 + P1y: 300 * 3t^3 - 6t^2 + 4 + P2y: 600 * -3t^3 + 3t^2 + 3t + 1 + P3y: 300 * t^3 \end{pmatrix} \frac{1}{6}
 \end{aligned}$$

Where t = 0.001;

$$\begin{aligned}
 &= \begin{pmatrix} P0x: 100 * 0.997003078 + P1x: 350 * 3.999999404 + P2x: 500 * 1.00300300 + P3x: 850 * 1.00000019e - 09 \\ P0y: 600 * 0.997003078 + P1y: 300 * 3.999999404 + P2y: 600 * 1.00300300 + P3y: 300 * 1.00000019e - 09 \end{pmatrix} \frac{1}{6} \\
 &= \begin{pmatrix} P0x: 99.7003078 + P1x: 1.399.997914 + P2x: 501.5015 + P3x: 0.0000008500001615 \\ P0y: 598.2018468 + P1y: 1.199.998212 + P2y: 601.8018 + P3y: 0.00025500004845 \end{pmatrix} \frac{1}{6} \\
 P'_x &= \begin{pmatrix} X : 2,001.1997226500001615 \\ Y : 2,400.00211380004845 \end{pmatrix} \frac{1}{6} \\
 P'_y &= \begin{pmatrix} X : 2,001.1997226500001615 \\ Y : 2,400.00211380004845 \end{pmatrix} \frac{1}{6} \\
 P'_x &= 333.53328710833336025 \\
 P'_y &= 400.000352300008075
 \end{aligned}$$

Equation b

(It shall be noted that these calculations slightly differ in visual studio than those calculated here but the whole numbers remain the same throughout.)

Output

Upon entering the application a window will open with a black canvas. When the user clicks with the cursor on the black canvas in the window a pixel shall appear at the location:

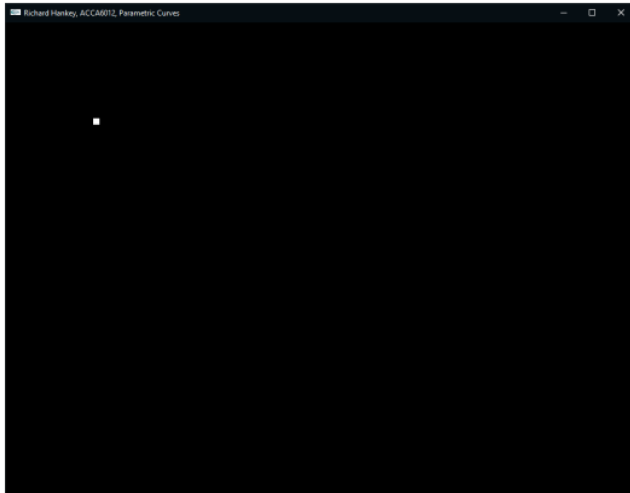


Figure 3

And further subsequent clicks on the black canvas screen will result in more pixels:



Figure 4

When the pixel total reaches 4 (position 3 in the array) a parametric curve or B-Spline curve shall appear on the canvas.

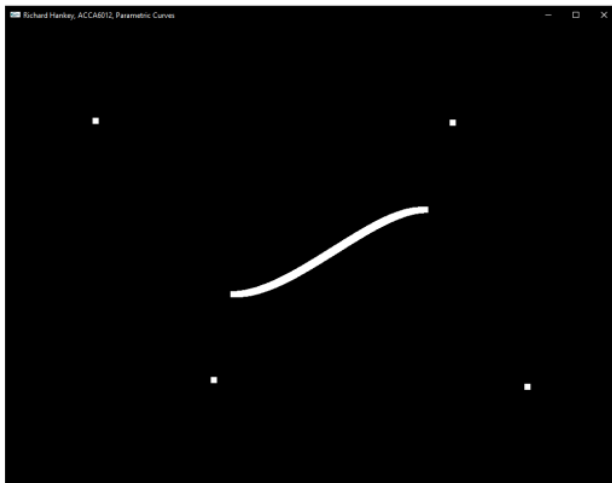


Figure 5

One additional click in the window of the black canvas will result in a second curve will appearing:

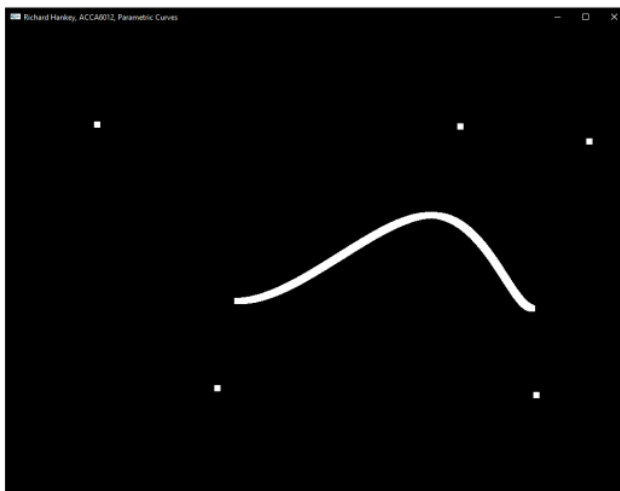


Figure 6

When a pixel is highlighted it will turn red, due to the functionality of detecting a mouse click per pixel being unachievable, a workaround has been developed; if and when the cursor is highlighting a pixel, which is determined by this algorithm in *figure 7*, slowly moving the cursor in a desired direction will move the pixel and recalculate the curve in real time.

```
float *pixpos = pixel[i].getPixelData();

//printf("pixel pos : %i, %i \n", pixpos[0], pixpos[1]);
if ((x > pixpos[0]-graphics.pixelSize()/2 && x < pixpos[0] +graphics.pixelSize()/2)&&
    getMouseY() > pixpos[1] - (graphics.pixelSize() / 2) && getMouseY() < pixpos[1] + (graphics.pixelSize() / 2))
{
    printf("contact mouse class\n");
    mouse_contact = true;
    pixel[i].contactUpdate(mouse_contact);
    if (mouse_contact == true)
    {
        dragPixel(i);
    }
}
else
{
    printf("off contact mouse class\n");
    mouse_contact = false;
    pixel[i].contactUpdate(mouse_contact);
}
```

Figure 7

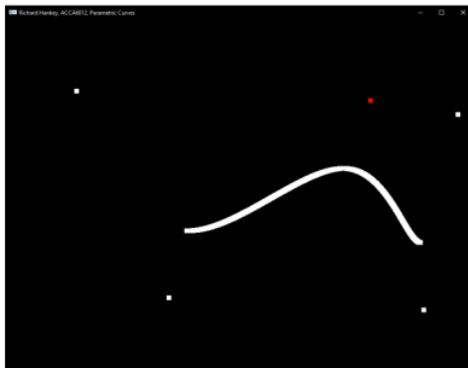


Figure 8



Figure 9

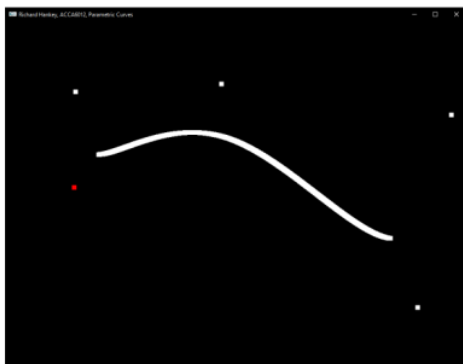


Figure 10

Word Count: 725

References and Text Consulted for Visual Studio Solution

- [1] “Learn OpenGL, extensive tutorial resource for learning Modern OpenGL.” [Online]. Available: <https://learnopengl.com/>. [Accessed: 03-Dec-2020].
- [2] Scratchapixel, “Computing the Pixel Coordinates of a 3D Point (Mathematics of Computing the 2D Coordinates of a 3D Point).” [Online]. Available: <https://www.scratchapixel.com/lessons/3d-basic-rendering/computing-pixel-coordinates-of-3d-point/mathematics-computing-2d-coordinates-of-3d-points>. [Accessed: 03-Dec-2020].
- [3] J. Kessenich, G. Sellers, and D. Shreiner, *OpenGL Programming Guide*. .
- [4] E. Lengyel, *Mathematics for 3D Game Programming and Computer Graphics Third Edition*. .

Appendix

globals.h

```
/////////////////////////////////////////////////////////////////
///
// externals
/////////////////////////////////////////////////////////////////
///
extern void winReshapeFunc(GLint w, GLint h);
extern void renderScene();
extern void update();

/////////////////////////////////////////////////////////////////
///
// constants
/////////////////////////////////////////////////////////////////
///
const int SCREEN_WIDTH      = 1000;
const int SCREEN_HEIGHT = 750;

extern int controlPoints[4][2];

/////////////////////////////////////////////////////////////////
///
// structures
/////////////////////////////////////////////////////////////////
///
struct sRGB
{
    float r, g, b;
};
```

cRenderClass.h

```
#include "stdafx.h"

class cRenderClass
{
public:

    cRenderClass(const int, const int);
    ~cRenderClass();

    void create(int argc, _TCHAR* argv[]);
    void loop();

    void clear();

    void render();
    void setColour(float, float, float);
    void setPointSize(int);
    void drawPixel(int x, int y);
    void bresenhams(int startX, int startY, int endX, int endY); //not working to
intention
    void drawLine(int x0, int x1, int y0, int y1);
    float* basisCalc(float t);
    int pixelSize();

private:
    int mouseX;
    int mouseY;
    int          m_sw, m_sh;
    int          m_point_size;
    float *m_buffer;
    sRGB        m_colour;
};
```

cRenderClass.cpp

```
// cRenderClass.cpp
//
/////////////////////////////////////////////////////////////////
//
// includes
/////////////////////////////////////////////////////////////////
//
#include "stdafx.h"

/////////////////////////////////////////////////////////////////
//
// Declarations
/////////////////////////////////////////////////////////////////
//
cRenderClass graphics( SCREEN_WIDTH, SCREEN_HEIGHT );
PixelData *pixel = new PixelData;
MouseControl mouse;
```

```

////////////////////////////////////
////
// MouseControls
////////////////////////////////////
////
void mouseControls(int button, int state, int x, int y)
{
    //int dat[2]{ mouse.getMouseX(), mouse.getMouseY() }; test to see if able to
    get mouse position.
    mouse.mouseControls(button, state, x, y);

}
void mousePosition(int x, int y)
{
    mouse.mousePos(x, y);
}
void clickdrag(int x, int y)
{
    mouse.mousePos(x, y);
}
////////////////////////////////////
////
// keyboard Controls
////////////////////////////////////
////
void keyboardControls(unsigned char Key, int a, int b)
{
    if(Key == 'c')
    {
        printf("key pressed");

    }
}

////////////////////////////////////
////
// cRenderClass() - constructor
////////////////////////////////////
////
cRenderClass::cRenderClass(const int w, const int h)
{
    m_sw = w;
    m_sh = h;

    int total_count = w*h*3;

    m_buffer = new float[total_count];

    for( int i=0; i<total_count; i++ )
    {
        m_buffer[i] = 0;
    }

    m_point_size = 1;
}

////////////////////////////////////
////

```

```

// cRenderClass() - destructor
//
//
cRenderClass::~cRenderClass()
{
}

//
//
// loop() - enters game loop
//
//
void cRenderClass::loop()
{
    glutMainLoop();
}

//
//
// clear() - clears out the render buffer
//
//
void cRenderClass::clear()
{
    memset(m_buffer, 0, sizeof(float)*m_sw*m_sh * 3);
}

//
// initialize glut stuff
//
//
void cRenderClass::create(int argc, _TCHAR* argv[])
{
    // initialise the glut library
    glutInit(&argc, argv);

    // set up the initial display mode
    // need both double buffering and z-buffering
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

    // set the initial window position
    glutInitWindowPosition(100, 100);

    // set the initial window size
    glutInitWindowSize(SCREEN_WIDTH, SCREEN_HEIGHT);

    // create and name the window
    glutCreateWindow("Richard Hankey, ACCA6012, Parametric Curves");

    // reshape callback for current window
    glutReshapeFunc(winReshapeFunc);

    glutKeyboardFunc(keyboardControls);
    glutMouseFunc(mouseControls);
    glutPassiveMotionFunc(mousePosition);
    //glutMotionFunc(clickdrag);

    // set display callback for current window
    glutDisplayFunc(renderScene);

    // set up the global idle callback
    glutIdleFunc(update);
}

```

```

}

////////////////////////////////////
////
// setPointSize() - set the pixel size
////////////////////////////////////
////
void cRenderClass::setPointSize(int size)
{
    m_point_size = size;
}

////////////////////////////////////
////
// drawPixel() -
////////////////////////////////////
////
void cRenderClass::drawPixel(int x, int y)
{
    int start, end;
    start = -(m_point_size/2);
    end   = (int)((m_point_size/2.0f) + 0.5);

    for( int i=start; i<end; i++ )
    {

        for( int j=start; j<end; j++ )
        {
            if( (x+j < 0) || (x +j >= m_sw) )
                continue;

            if( (y+i < 0) || (y+i >= m_sh) )
                continue;

            m_buffer[(((y+i)*m_sw+(x +j))*3) + 0] = m_colour.r;
            m_buffer[(((y+i)*m_sw+(x +j))*3) + 1] = m_colour.g;
            m_buffer[(((y+i)*m_sw+(x +j))*3) + 2] = m_colour.b;
        }
    }
}

void cRenderClass::bresenhams(int startX, int startY, int endX, int endY)
{
    if (startX > endX)
    {
        bresenhams(endX, endY, startX, startY);
        return;
    }

    int temp;
    int dX = abs(endX - startX);

    int dY = abs(endY - startY);
    bool steep = dY > dX; //check if the line is steep (dont know how to utilize
this yet)

```

```

int p = (2 * dY) - dX;
int pX = 2 * dY;
int pY = (2 * dY) - (2 * dX);

graphics.drawPixel(startX, startY);

for (int i = 0; i < dX; i++)
{
    if (p < 0)
    {
        startX++;
        p += pX;
    }
    else
    {
        if (steep)
        {
            for (int k = 0; k < dY; k++)
            {
                startY += (startY < endY) ? 1 : -1;
                p += pY;
                graphics.drawPixel(startX, startY); //detects
a steep line, but cannot work out how to draw to position
first,
                //without right angles being drawn. It either draws to the x
                }
                //then draws to the y or vise versa.
            }

            /*startY += (startY < endY) ? 1 : -1;
            p += pY;*/

        }
        graphics.drawPixel(startX, startY);
    }
}

} // bresenhams doesnt work, cant get a straight steep slope.

void cRenderClass::drawLine(int x0, int x1, int y0, int y1)
{
    graphics.drawPixel(x0, y0);
    for (int i = x0; i < x1; i++)
    {
        for (int j = y0; j < y1; j++)
        {
            if (i < x1)
            {
                x0++;
                graphics.drawPixel(x0, y0);
                if (j < y1)
                {
                    y0++;
                    drawPixel(x0, y0);
                }
            }
        }
    }
}

```

```

        }
    }
}

float * cRenderClass::basisCalc(float t)
{
    float basisFuncs[4]{ (1 - t)*(1 - t)*(1 - t) , (3 * t*t*t - 6 * t*t + 4) , (-3
* t*t*t + 3 * t*t + 3 * t + 1) , (t*t*t) };

    return basisFuncs;
}

int cRenderClass::pixelSize()
{
    return m_point_size;
}

////////////////////////////////////
///
// render() - renders this buffer to screen
////////////////////////////////////
///
void cRenderClass::render()
{
    // clear the back buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glDrawPixels( m_sw, m_sh, GL_RGB, GL_FLOAT, m_buffer );

    // swap the buffers of the current window
    glutSwapBuffers();

    // clear out the buffer
    clear();
}

////////////////////////////////////
///
// setColour() - sets the current colour
////////////////////////////////////
///
void cRenderClass::setColour(float r, float g, float b)
{
    m_colour.r = r;
    m_colour.g = g;
    m_colour.b = b;
}

////////////////////////////////////
///
////////////////////////////////////
///

```

```

////////////////////////////////////
////
////////////////////////////////////
////
////////////////////////////////////
////
////////////////////////////////////
////
////////////////////////////////////
////
////////////////////////////////////
////
////////////////////////////////////
////

```

```

////////////////////////////////////
////
// winReshapeFunc() - gets called initially and whenever the window get resized
// resizing has been locked
////////////////////////////////////
////
void winReshapeFunc(GLint w, GLint h)
{
    // specify current matrix
    glMatrixMode(GL_PROJECTION);

    // load an identity matrix
    glLoadIdentity();

    // create a projection matrix... i.e. 2D projection onto xy plane
    glOrtho( -SCREEN_WIDTH, SCREEN_WIDTH, SCREEN_HEIGHT, -SCREEN_HEIGHT, -100,
100);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // set up the viewport
    glViewport
    (
        0, // lower left x position
        0, // lower left y position
        (GLsizei) SCREEN_WIDTH, // viewport width
        (GLsizei) SCREEN_HEIGHT // viewport height
    );
}

```

MouseControl.h

```

#pragma once
class MouseControl
{
public:
    MouseControl();
    ~MouseControl();

    int getMouseX();
    int getMouseY();
    void setMouseX(int x);
    void setMouseY(int y);
    void mousePos(int x, int y);

```

```

        int mouseControls(int button, int state, int x, int y);
        int getPixelCount();
        bool mouseContact();
        void dragPixel(int i);

    private:
        bool mouse_contact = 0;
        bool mouse_drag = 0;
        int mouseX;
        int mouseY;
        int count = 0;
};

```

MouseControl.cpp

```

#include "stdafx.h"
extern PixelData *pixel;
extern cRenderClass graphics;

MouseControl::MouseControl()
{
}

MouseControl::~MouseControl()
{
}

int MouseControl::getMouseX()
{
    return mouseX;
}

int MouseControl::getMouseY()
{
    return mouseY;
}

void MouseControl::setMouseX(int x)
{
    mouseX = x;
}

void MouseControl::setMouseY(int y)
{
    mouseY = SCREEN_HEIGHT - y; // microsoft windows sets the y in opposite
    // direction so this little calculation fixes that.
}

void MouseControl::mousePos( int x, int y) // sets and tracks the mouse position
{
    setMouseX(x);
    setMouseY(y);
    //printf("x n y: %i, %i \n", x, y);
    for (int i = 0; i < getPixelCount(); i++)
    {
        /*float *temp = pixel[i].getPixelData();
        float **pixpos = nullptr;
        pixpos[i] = new float[2];*/ // dynamic allocation not working
    }
}

```

```

float *pixpos = pixel[i].getPixelData();

//printf("pixel pos : %i, %i \n", pixpos[0], pixpos[1]);
if ((x > pixpos[0]-graphics.pixelSize()/2 && x < pixpos[0]
+graphics.pixelSize()/2)&& getMouseY() > pixpos[1] - (graphics.pixelSize() / 2) &&
getMouseY() < pixpos[1] + (graphics.pixelSize() / 2))
{

    printf("contact mouse class\n");
    mouse_contact = true;
    pixel[i].contactUpdate(mouse_contact);
    if (mouse_contact == true)
    {

        dragPixel(i);
    }

}
else
{
    printf("off contact mouse class\n");
    mouse_contact = false;
    pixel[i].contactUpdate(mouse_contact);

}

}

}

int MouseControl::mouseControls(int button, int state, int x, int y)// used for the
callback in the glut initialization
{

    if (state == GLUT_DOWN)
    {
        if (button == GLUT_LEFT_BUTTON);
        {

            //not working
            printf("\n click down \n");
            if (mouse_contact == true)
            {

                mouse_drag = true;
                printf("dragging \n");
                pixel->dragUpdate(mouse_drag);

            }
            else
            {
                printf("dragging off \n");
                mouse_drag = false;
                pixel->dragUpdate(mouse_drag);
            }
        }
    }
}

```

```

    }

    pixel[getPixelCount()].setPixelData(getMouseX(), getMouseY()); //
    this sets the position of the pixel to be drawn on the screen
    if (count == 0) // and get increased with every event of the
mouse click
    {
        count++;
        return 0;
    }
    else
    {
        count++;
        return getPixelCount();
    }
}

}

int MouseControl::getPixelCount() // return the amount of clicks/pixel in the window
{
    return count;
}

bool MouseControl::mouseContact()
{
    return mouse_contact;
}

void MouseControl::dragPixel(int i)
{
    pixel[i].movePixel();
}

```

PixelData.h

```

#pragma once
class PixelData : public MouseControl
{

```

```

private:
    float p_Data[2];
    bool contacter = false;
    bool dragging = false;

public:
    void setPixelData( int x, int y);
    float* getPixelData();
    void drawMultPixels();
    void removepixel(PixelData object); //this doesnt work
    void changePixelColour(float r, float g, float b);
    void contactUpdate(bool point);
    bool returnContact();
    void dragUpdate(bool drag);
    bool returnDrag();
    void movePixel();

    PixelData();
    ~PixelData();
};

```

PixelData.cpp

```

#include "stdafx.h"
#include "PixelData.h"

extern cRenderClass graphics;
extern MouseControl mouse;
void PixelData::setPixelData(int x, int y)
{
    p_Data[0] = x ;
    p_Data[1] = y;

}

float * PixelData::getPixelData()
{
    return p_Data;
}

void PixelData::drawMultPixels()
{
    graphics.drawPixel(p_Data[0], p_Data[1]); // this is used to draw the pixel
object
}

void PixelData::removepixel(PixelData object) // was supposed to delete the object, i.e
the pixel placed in the argument (doesnt work)
{

```

```
        delete(&object);
    }

    void PixelData::changePixelColour(float r, float g, float b)
    {
        graphics.setColour(r,g,b);
    }

    void PixelData::contactUpdate(bool point)
    {
        contacter = point;
    }

    bool PixelData::returnContact()
    {
        return contacter;
    }

    void PixelData::dragUpdate(bool drag)
    {
        dragging = drag;
    }

    bool PixelData::returnDrag()
    {
        return dragging;
    }

    void PixelData::movePixel()
    {
        p_Data[0] = mouse.getMouseX();
        p_Data[1] = mouse.getMouseY();
        graphics.drawPixel(p_Data[0], p_Data[1]);
    }

PixelData::PixelData()
{
}

PixelData::~PixelData()
{
}
```

stdafx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once

#include <iostream>
#include <tchar.h>
#include <windows.h>
#include <math.h>
#include "GL/glut.h"
#include "globals.h"

#include "cRenderClass.h"
#include "MouseControl.h"
#include "PixelData.h"
```

GraphicsTemplate.cpp

```
// GraphicsTemplate.cpp
//
//
//
// includes
//
//
#include "stdafx.h"
//
//
// externals
//
extern cRenderClass graphics;
extern PixelData *pixel;
extern MouseControl mouse;
//
//
// renderScene() - render the scene
//
void renderScene()
{
```

```

// set a colour
//graphics.setColour(1, 1, 1);

// set pixel size..
graphics.setPointSize(10);


// draw to an off screen buffer
for (int i = 0; i < mouse.getPixelCount(); i++)
{
    if (pixel[i].returnContact() == true)
    {
        graphics.setColour(1, 0, 0);

    }
    else
    {
        graphics.setColour(1, 1, 1);
    }
    //printf(" direct mouse pos x: %i,y: %i \n", mouse.getMouseX(),
mouse.getMouseY());

    pixel[i].drawMultPixels();


    float *p0 = pixel[0].getPixelData();
    float *p1 = pixel[1].getPixelData();
    float *p2 = pixel[2].getPixelData();
    float *p3 = pixel[3].getPixelData();
    float *p4 = pixel[4].getPixelData();
    float *p5 = pixel[5].getPixelData();
    float point[2];
    //float l_point_x[5]; // L points (L0, L1, etc.)
    //float l_point_y[5];
    //float q_point_x[5]; // Q points (Q0, Q1, etc.)
    //float q_point_y[5];
    // no need for these;

    if (i >= 3)
    {
        for (float t = 0; t < 1; t += 0.001)
        {
            //L_0 (t)=(1-t) P_0+ tP_1 linear interpolation

            /*l_point_x[0] = (1 - t)*p0[0] + t * p1[0];
            l_point_y[0] = (1 - t)*p0[1] + t * p1[1];
            l_point_x[1] = (1 - t)*p1[0] + t * p2[0];
            l_point_y[1] = (1 - t)*p1[1] + t * p2[1];
            l_point_x[2] = (1 - t)*p2[0] + t * p3[0];
            l_point_y[2] = (1 - t)*p2[1] + t * p3[1];
            l_point_x[3] = (1 - t)*p3[0] + t * p4[0];
            l_point_y[3] = (1 - t)*p3[1] + t * p4[1];
            l_point_x[4] = (1 - t)*p4[0] + t * p5[0];
            l_point_y[4] = (1 - t)*p4[1] + t * p5[1];*/ // no
need for this

```

```

Interpolation                                     //Q_0 (t)=(1-t) L_0 (t)+ tL_1 (t) polynomial
l_point_x[1];                                     /*q_point_x[0] = (1 - t)*l_point_x[0] + t *
l_point_y[1];                                     q_point_y[0] = (1 - t)*l_point_y[0] + t *
l_point_x[2];                                     q_point_x[1] = (1 - t)*l_point_x[1] + t *
l_point_y[2];                                     q_point_y[1] = (1 - t)*l_point_y[1] + t *
l_point_x[3];                                     q_point_x[2] = (1 - t)*l_point_x[2] + t *
l_point_y[3];                                     q_point_y[2] = (1 - t)*l_point_y[2] + t *
l_point_x[4];                                     q_point_x[3] = (1 - t)*l_point_x[3] + t *
l_point_y[4];                                     q_point_y[3] = (1 - t)*l_point_y[3] + t *
*/ // no need for this

if (i != 4)
{
    float *base = graphics.basisCalc(t);

    point[0] = (p0[0] * base[0] + p1[0] * base[1]
+ p2[0] * base[2] + p3[0] * base[3]) * 1 / 6; //x values
    point[1] = (p0[1] * base[0] + p1[1] * base[1]
+ p2[1] * base[2] + p3[1] * base[3]) * 1 / 6; //y values
    graphics.drawPixel(point[0], point[1]);
    /*float *base = graphics.basisCalc(t);
    point[0] = (l_point_x[0] * base[0] +
l_point_x[1] * base[1] + l_point_x[2] * base[2] + l_point_x[3] * base[3]) * 1 / 6;
    point[1] = (l_point_y[0] * base[0] +
l_point_y[1] * base[1] + l_point_y[2] * base[2] + l_point_y[3] * base[3]) * 1 / 6;
    point[0] = (q_point_x[0] * base[0] +
q_point_x[1] * base[1] + q_point_x[2] * base[2] + q_point_x[3] * base[3]) * 1 / 6;
    point[1] = (q_point_y[0] * base[0] +
q_point_y[1] * base[1] + q_point_y[2] * base[2] + q_point_y[3] * base[3]) * 1 / 6;
    graphics.drawPixel(point[0], point[1]);*/
}
else
{
    float *base = graphics.basisCalc(t);

    point[0] = (p1[0] * base[0] + p2[0] * base[1]
+ p3[0] * base[2] + p4[0] * base[3]) * 1 / 6;
    point[1] = (p1[1] * base[0] + p2[1] * base[1]
+ p3[1] * base[2] + p4[1] * base[3]) * 1 / 6;
    graphics.drawPixel(point[0], point[1]);
}

}

}

//this draws a line to the points(for visual representation) (doesnt
work//broken)

```

```

        //if (i > 0)
        //{
        ////
        ////
        //// //step back through the dynamic array of pixel objects to capture
the x and y vectors
        // float*before = pixel[i - 1].getPixelData();
        // float*next = pixel[i].getPixelData();
        //
        // //graphics.drawLine(before[0], before[1], next[0], next[1]);
doesnt work as intended.
        // graphics.bresenhams(before[0], before[1], next[0], next[1]);
        ////
        //}
        float t = 0;

    }

    // render the scene
    graphics.render();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////
// update() - update function
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////
void update()
{
    // add any update code here...

    // always re-render the scene..
    renderScene();
}

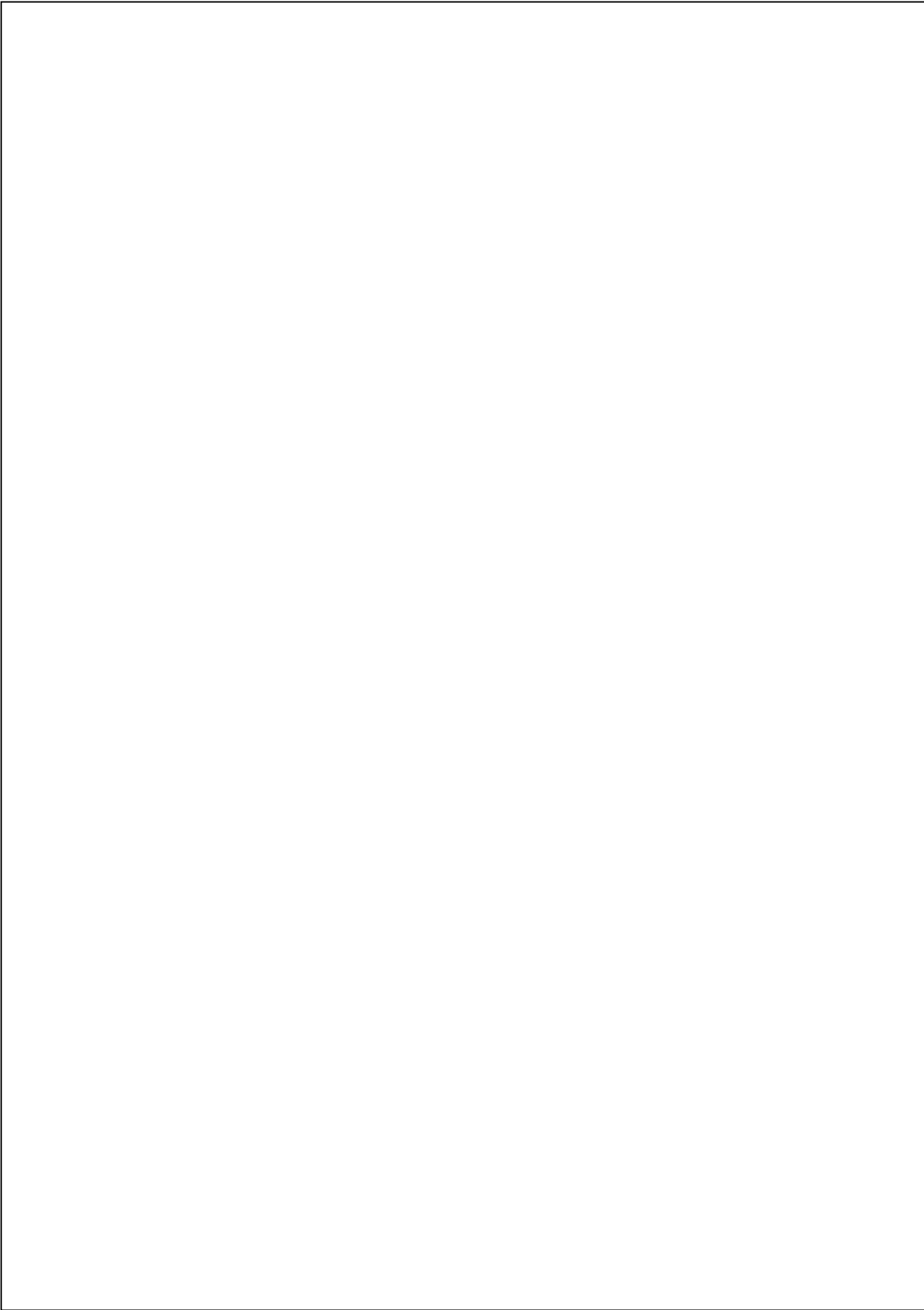
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////
// _tmain() - program entry point
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////
int _tmain(int argc, _TCHAR* argv[])
{
    // init glut stuff..
    graphics.create(argc, argv);

    // good place for one-off initialisations and objects creation..

    // enter game loop..
    graphics.loop();

    return 0;
}

```



FINAL GRADE

GENERAL COMMENTS

Instructor

82/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

GRADING FORM: SMU GENERIC_GJ

RICH HARRIS

82

RESEARCH-INFORMED LITERATURE



Refs included. Relevant and appropriate. Reference made to these in body of report. Essential topics identified and reviewed here. Good.

7

KNOWLEDGE AND UNDERSTANDING



B-spline theory well understood and implemented. Code works and shows a very good understanding of the graphics methods involved.

17

ANALYSIS



A very good interpretation of the problem and synthesis of a working solution. An additional control point and extra spline generated. Use of interactive mouse controls impressive.

17

PRACTICAL APPLICATION AND DEP.



Some very excellent work here. Multiple additional control points would have been even better but nevertheless one additional spline generated. A good demonstration of graphics skills in OpenGL.

33

SKILLS FOR PROFESSIONAL PRAC.



A well structured report. References used are relevant. Some very good SE principles and practice in evidence here.

8