

Graphics Report Rhys Lewis

by Rhys Thomas Lewis

FILE	12025_RHYS_THOMAS_LEWIS_GRAPHICS_REPORT_RHYS_LEWIS_311302_533593988.DOCX (544.52K)		
TIME SUBMITTED	07-DEC-2020 01:32AM (UTC+0000)	WORD COUNT	1013
SUBMISSION ID	138516478	CHARACTER COUNT	5070

2020

Graphics and Games Programming

REPORT

RHYS LEWIS

Table of Contents

1. Algorithms/Methods..... 2

 Base functions..... 2

 Placing Points 2

 Calculating and Displaying Points 2

2. Calculating a single point 3

 Step 1: Discover “t” and “y” 4

 Step 2: Multiply Base Functions..... 4

 Step 3: Reduce to single point 4

3. Screenshots..... 5

1. Algorithms/Methods

This program uses a few simplistic algorithms that, in combination, accomplish the task. The mathematics behind calculating and drawing the curve is all done in a class known as “cBSpline”. In it’s constructor, it defines the current and maximum resolution, calculates base functions and sets the location of all the points off screen.

Base functions

To calculate a uniform non rational B spline curve (henceforth referred to as UNRBSC), you must perform a simple matrix multiplication using 4 base functions and the 4 points in your curve and dividing the total by 1/6 for each step along your curve using a slowly increasing variable known as “t” to represent the current step along the curve, with “t” eventually reaching 1 to represent the final point. To simplify this, at the beginning of the program, the program calculates the base functions a number of times equal to a variable known as “maximumResolution” (1000 by default), these values are then saved to a lookup table known as “LookupMatrix” for ease of use and to save processing time on each frame.

```
inline void cBSpline::CreateAndFillLookupTable()
{
    LookupMatrix = new float*[maximumResolution+1];
    for (int i = 0; i < maximumResolution+1; i++)
    {
        LookupMatrix[i] = new float[4];
    }

    float t;
    for (int i = 0; i < maximumResolution+1; i++)
    {
        t = i;
        t /= maximumResolution;

        LookupMatrix[i][0] = pow((1-t),3);
        LookupMatrix[i][1] = (3 * pow((t),3)) - (6 * pow((t),2)) + 4;
        LookupMatrix[i][2] = (-3 * pow((t), 3)) + (3 * pow((t),2)) + (3 * t) + 1;
        LookupMatrix[i][3] = pow((t),3);
    }
}
```

Figure 1: Base Functions being calculated and placed into a table

Placing Points

To calculate a curve, points need to be present. Upon booting up the program, the screen is completely blank, the user must use their mouse to place and save points using left click. Using right click will cause the currently selected point to be reset to its original position off screen. In order to select which point the user is placing, the system scans every frame for any keyboard input, pressing numbers 1-4 on the keyboard will cause the system to select that point, using a switch statement and a variable known as “PointNumberController”. The first and last points are coloured green and red respectively in order to make it simple to understand why the curve moves in the way it does. This is achieved by the update function looking for the 1st and 4th point and setting their colour differently while every other point is coloured white by default.

Calculating and Displaying Curve

Using the theory discussed earlier, we know that to calculate a UNRBSC, you need both base functions and points, now that both are available to the user, it is a simple matter of performing the equation as explained. The code uses a new array, known as “basePoints” to temporarily save the function result. We use “currentResolution” to determine how many steps are between t=0 and t=1, determining how detailed the curve is. We begin by initiating a loop that will

Deriving the Uniform Non-Rational B-Spline

- This gives 16 equations in 16 unknowns. The solution for which gives:
$$\text{point on curve} = \frac{1}{6} \begin{pmatrix} 1-t \\ t \end{pmatrix} \begin{pmatrix} 1-t \\ t \end{pmatrix} \begin{pmatrix} 1-t \\ t \end{pmatrix} \begin{pmatrix} 1-t \\ t \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$
- This is the curve for the parametric range of t from 0 to 1. Each segment in series has a parameter range increasing uniformly – from 0 to 1, then from 1 to 2, then 2 to 3 and so on.

Figure 2: UNRBSC tutorial. Source: PowerPoint presentation supplied by University of Wales Trinity Saint David, 2020

repeat equal to the current resolution, as that is the limit to how many points we should draw. We find how many steps we need by setting “t” to our current loop increment and then dividing by (currentResolution -1) and multiplying it by our maximumResolution to get the relative position of where t would be if the loop were between 0 and 1 and then the multiplication gives us that position relative to the Lookup Table containing the base functions we created earlier. Now that we have this value, we can simply calculate where the point should be by running the algorithm using our arrays of control points and the LookupMatrix Table we established earlier, using t, now converted to an int known as “y”. Once we have multiplied the base formulae by the points, we add the results together and divide the result by 1/6 to get our point on the curve. The program then saves this point as the “currentPoint”. If it is not the first point, it then draws a line using a line drawing algorithm from a previous assignment between the current point and the previous one. It then sets the currentPoint to the previousPoint and repeats. The eventual result of this is a full UNRBSC.

```

inline void CBspline::CalculateAndRenderCurves(RenderClass graphics)
{
    float t;
    int y;

    float basePoints[4][2];
    float currentPoint[2] = {0,0};
    float previousPoint[2] = {0,0};
    for (int i = 0; i < currentResolution; i++)
    {
        t = i;
        t /= (currentResolution - 1);
        t *= maximumResolution;

        y = static_cast<int>(t);

        basePoints[0][0] = LookupMatrix[y][0]*controlPoints[0][0];
        basePoints[0][1] = LookupMatrix[y][0]*controlPoints[0][1];

        basePoints[1][0] = LookupMatrix[y][1]*controlPoints[1][0];
        basePoints[1][1] = LookupMatrix[y][1]*controlPoints[1][1];

        basePoints[2][0] = LookupMatrix[y][2]*controlPoints[2][0];
        basePoints[2][1] = LookupMatrix[y][2]*controlPoints[2][1];

        basePoints[3][0] = LookupMatrix[y][3]*controlPoints[3][0];
        basePoints[3][1] = LookupMatrix[y][3]*controlPoints[3][1];

        currentPoint[0] = basePoints[0][0] + basePoints[1][0] + basePoints[2][0] + basePoints[3][0];
        currentPoint[1] = basePoints[0][1] + basePoints[1][1] + basePoints[2][1] + basePoints[3][1];

        currentPoint[0] *= 0.166;
        currentPoint[1] *= 0.166;

        if (i!=0)
        {
            HamLine(previousPoint[0], currentPoint[0], previousPoint[1], currentPoint[1], graphics);

            previousPoint[0] = currentPoint[0];
            previousPoint[1] = currentPoint[1];
        }
    }
}

```

Figure 3: UNRBSC being calculated and displayed

The resolution of the curve can be changed while the program is running by pressing either the A or the D key, the switch statement mentioned earlier that looks for keyboard inputs is also checking for these keys, and if they are pressed, they increase and decrease the resolution relative to the current resolution, with the lower resolutions increasing and decreasing by less in order to appreciate the major changes on the lower end and the higher resolution doing the opposite as the higher the resolution becomes, the harder it becomes to tell the difference.

2. Calculating a single point

To demonstrate how a point is calculated, this section will briefly run through a random number during a single frame of the program running, here the crucial information needed to calculate the curve.

Point positions:

Point 1: 34, 445

Point 2: 255, 384

Point 3: 172, 113

Point 4: 426, 93

Resolutions:

Current: 6

Maximum: 1000

Loop instance:

$i = 2$

Step 1: Discover "t" and "y"

$t = i = 2$

$t / (\text{currentResolution} - 1) = 2/5 = 0.4$

$t * \text{maximumResolution} = 0.4 * 1000 = 400$

$y = t \text{ as int} = 400$

Step 2: Multiply Base Functions

$\text{basePoints}[0][0] = \text{LookupMatrix}[y][0] * \text{controlPoints}[0][0] = 0.216 * 34 = 7.344$

$\text{basePoints}[0][1] = \text{LookupMatrix}[y][0] * \text{controlPoints}[0][1] = 0.216 * 445 = 96.12$

$\text{basePoints}[1][0] = \text{LookupMatrix}[y][1] * \text{controlPoints}[1][0] = 3.323 * 255 = 824.16$

$\text{basePoints}[1][1] = \text{LookupMatrix}[y][1] * \text{controlPoints}[1][1] = 3.232 * 384 = 1241.08$

$\text{basePoints}[2][0] = \text{LookupMatrix}[y][2] * \text{controlPoints}[2][0] = 2.488 * 172 = 427.94$

$\text{basePoints}[2][1] = \text{LookupMatrix}[y][2] * \text{controlPoints}[2][1] = 2.488 * 113 = 281.14$

$\text{basePoints}[3][0] = \text{LookupMatrix}[y][3] * \text{controlPoints}[3][0] = 0.064 * 426 = 27.3$

$\text{basePoints}[3][1] = \text{LookupMatrix}[y][3] * \text{controlPoints}[3][1] = 0.064 * 93 = 6$

Step 3: Reduce to single point

$\text{currentPoint}[0] = \text{basePoints}[0][0] + \text{basePoints}[1][0] + \text{basePoints}[2][0] + \text{basePoints}[3][0] = 7.344 + 824.16 + 427.94 + 27.3 = 1286.7$

$\text{currentPoint}[1] = \text{basePoints}[0][1] + \text{basePoints}[1][1] + \text{basePoints}[2][1] + \text{basePoints}[3][1] = 96.12 + 1241.08 + 281.14 + 6 = 1624.3$

$\text{currentPoint}[0] *= 0.166 = 1286.7 * 1/6 = 213.6$

$\text{currentPoint}[1] *= 0.166 = 1624.3 * 1/6 = 269.63$

Point = 213.6, 269.63

3. Screenshots

```
class cBSpline
{
public:
    cBSpline();
    ~cBSpline();

    int controlPoints[4][2];
    void CalculateAndRenderCurves(cRenderClass graphics);
    int currentResolution;
    int maximumResolution;

private:
    void CreateAndFillLookUpTable();
    void HamLine(int x1, int x2, int y1, int y2, cRenderClass graphics);

    float** LookupMatrix;

cBSpline::cBSpline()
{
    currentResolution = 100;
    maximumResolution = 1000;
    CreateAndFillLookUpTable();

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            controlPoints[i][j] = -50;
        }
    }
}
```

Figure 10:cBSpline and its constructor

```
void MouseFunc(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON)
    {
        if (state == GLUT_DOWN)
        {
            bspline.controlPoints[PointNumberController][0] = x;
            bspline.controlPoints[PointNumberController][1] = SCREEN_HEIGHT - y;
        }
    }

    if (button == GLUT_RIGHT_BUTTON)
    {
        if (state == GLUT_DOWN)
        {
            bspline.controlPoints[PointNumberController][0] = -50;
            bspline.controlPoints[PointNumberController][1] = -50;
        }
    }
}
```

Figure 7:Mouse function scanning for clicks



Figure 6: Curve with a resolution of 1000

```
case 49 /*1*/:
    PointNumberController = 0;
    break;
case 50 /*2*/:
    PointNumberController = 1;
    break;
case 51 /*3*/:
    PointNumberController = 2;
    break;
case 52 /*4*/:
    PointNumberController = 3;
    break;
```

Figure 9:Keyboard input switch statement setting currently selected point

```
for (int i = 0; i < 4; i++)
{
    switch (i)
    {
        case 0:
            graphics.setColour(0, 1, 0); // Colour start point in Green
            break;
        case 3:
            graphics.setColour(1, 0, 0); // Colour End point in Red
            break;
        default:
            graphics.setColour(1, 1, 1); // Colour other points in White
            break;
    }
    graphics.drawPixel(bspline.controlPoints[i][0], bspline.controlPoints[i][1]);
}
```

Figure 8:Update function setting colour

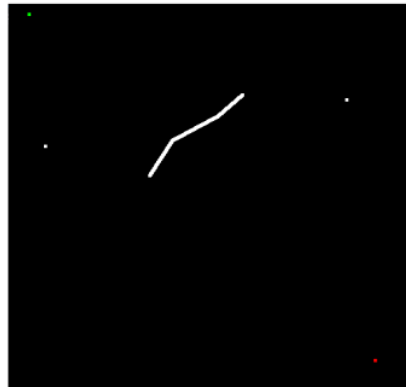


Figure 5: Curve with a resolution of 4

```
switch (key)
{
    case 92 /*A*/:
    {
        if (bspline.currentResolution > 0.5 && bspline.currentResolution > 1 && bspline.currentResolution < 2)
        {
            if (bspline.currentResolution < 10)
            {
                bspline.currentResolution += 1;
            }
            else if (bspline.currentResolution < 100)
            {
                bspline.currentResolution += 10;
            }
            else
            {
                bspline.currentResolution += 100;
            }
        }
        break;
    }
    case 100 /*D*/:
    {
        if (bspline.currentResolution > bspline.maximumResolution)
        {
            if (bspline.currentResolution < 10)
            {
                bspline.currentResolution -= 1;
            }
            else if (bspline.currentResolution < 100)
            {
                bspline.currentResolution -= 10;
            }
            else
            {
                bspline.currentResolution -= 100;
            }
        }
        break;
    }
}
```

Figure 4: Resolution Statement demonstrating how relative increase and decrease is determined

Graphics Report Rhys Lewis

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

74/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

GRADING FORM: SMU GENERIC_GJ

RHYS THOMAS LEWIS

74

RESEARCH-INFORMED LITERATURE



Essential topics identified and reviewed here. Whilst you reference material in your report you also need to include a formal References section at the end of your report..

6

KNOWLEDGE AND UNDERSTANDING



A good understanding of the method required for curve rendering in this assignment demonstrated. Code works and shows a very good understanding of the graphics methods involved.

16

ANALYSIS



A good interpretation of the problem and synthesis of a working solution. Additional control points would have been nice to see . Code works and shows a very good understanding of the graphics methods involved. Use of mouse control is nice to see.

16

PRACTICAL APPLICATION AND DEP.



Some very, very good work here. Additional control points would have been nice to see but nevertheless a good demonstration of graphics skills in OpenGL.. Good SE principles demonstrated - eg code re-use (line drawing) and commenting.

30

SKILLS FOR PROFESSIONAL PRAC.



A good report. See comments earlier regarding references. You need to include a reference section. Report structured.

6