Applied Geometric Modeling

MD Kamruzzaman

March 2, 2024

Abstract

This report is my exam materials for the course Dte-3604 "Applied Geometric Modeling", it will mainly focus on the theoretical part of the project that I have been implementing along side the class lectures throughout the course, giving an explanation to the concept behind the different types of curves, before explaining briefly how they were implemented.

1 Introduction

In terms of geometric modelling, it is rather important to know how to define various curves and surfaces. Be that polynomial B-splines, Hermite splines etc. B-Splines are typically defined by degree, order knot vectors and control polygons, with the knot vector and degree defining the basis function. Basis functions can then be used to calculate stuff like points and derivatives. Using this it is possible to Blend the local curves using a given a B Function to generate Blending Splines. Or one can try to use said points in order to "smooth-en" the graph in subdivision. When it comes to the coding we primarily used the GMlib library. An in house open source programming library. And we used C++ to do the actual coding. As such, most of my code was the result of building upon the pre-existing work done in the files downloaded from and working with Qt Creator and the theory to build on and implement everything else.

2 Implementation

2.1 B-Spline

2.1.1 Implementation

The point of implementing it first is not only to learn how to implement things in c++, but also to have a starting point from where we can create the Sub-Division curve, Blending Spline Curve later. When it comes to our B-Spline implementation, I did a third degree B-Spline, For $d = 3$ we have the following formula for a B-spline curve on $[t_i, t_{i+1}]$, [Lak]

$$c(t) = \begin{pmatrix} (1 - w_{1,i}(t)) \\ w_{1,i}(t) \end{pmatrix}^T \begin{pmatrix} (1 - w_{2,i-1}(t)) & w_{2,i-1}(t) \\ 0 & (1 - w_{2,i}(t)) \end{pmatrix} \begin{pmatrix} (1 - w_{3,i-2}(t)) & w_{3,i-2}(t) \\ 0 & (1 - w_{3,i-1}(t)) \\ 0 & 0 \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}$$

To implement this formula, we created one B function and a common W function. We have divided this formula into 2 part. In the First part we have multiplied first and second matrix and used the result in Second part by multiplying with third matrix. We used the total result to determine the control points, however we did not used the control points directly from the formula. Here I have used 7 control points to determine the curve.[Lak]

2.1.2 Result

Figure 1: B-Spline

2.2 Closed Sub-Division Curve

Generally Subdivision Curves focuses on corner cutting in order to smooth out curves. One way to go about doing this is rather simple. First we need a point set that represent a graph, then we replace the old point set with a new bigger point set that is situated on the lines between the old point set. This is essentially the basis behind Lane-Riesenfeld subdivision algorithm, which we have implemented in our code.

2.2.1 Lane-Reisenfield Algorithm Implementation

Most of our implementation is just using the formulas shown in the Book[3] As such most of the challenge laid in trying to implement the LaneReisen- FieldClosed, doublePart, and smoothPart-Closed functions. The reason we only implemented these is that we are hard coded the program to only work on closed curves. 2 3 4 [Lak]

```
vector⟨Point⟩ LaneRiesenfeldClosed( vector⟨Point⟩ P, int k, int d )
    int n = P.size;                          // The number of intervals
    int m = 2^k n + 1;                       // The final number of points
    vector⟨Point⟩ Φ(m);                      // The return vector – m points.
    for ( int i=0; i < n; i++ )
        Φ_i = P_i;                           // Inserting the initial points
    Φ_n = P_0;                               // Closing the curve
    for ( int i=0; i < k; i++ )              // For each level of refinement
        n = doublePart(Φ, n);
        smoothPartClosed(Φ, n, d );
    return Φ;
```

Figure 2: Lane ReisenFieldClosed

```
int doublePart( vector⟨Point⟩& P, int n )
    for ( int i=n-1; i > 0; i-- )
        P_2i = P_i;
        P_2i−1 = 1/2 (P_i + P_i−1);
    return 2n − 1;
```

Figure 3: DoublePart

```
void smoothPartClosed( vector⟨Point⟩& P, int n, int d )
    for ( int j=1; j < d; j++ )
        for ( int i=0; i < n − 1; i++ )
            P_i = 1/2 (P_i + P_i+1);
        P_n−1 = P_0;
```

Figure 4: SmoothPartClosed

2.2.2 Result

After implementing all these three algorithms we get the following output: 5

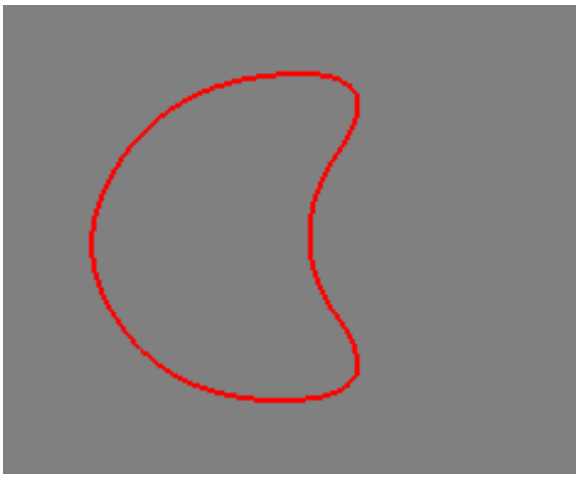

Figure 5: Closed Sub-division Curve

2.3 Model Curve

Any Curve can be converted into a spline curve by adding knot vectors. By doing this we end up with a bunch of sub curves, that are linked to the internal knot values. Thus, a sub-curve is only a restriction of the domain of a curve. I have implemented a closed parametric curve called "Fish Curve".[wol]

2.3.1 Implementation

This was the first type of curve we have implemented as a way to learn how to utilize GMlib. To do this we made sure to find suitable parametric equations whereas we made sure to use the formulation for its domain and derivatives in order draw the curve. I have used the following formula for the implementation:

$$x(t) = a\left(\cos(t) - \frac{\sin^2(t)}{\sqrt{2}}\right)$$

$$y(t) = a\sin(t)\cos(t)$$

Figure 6: Formula for Fish curve

2.3.2 Result

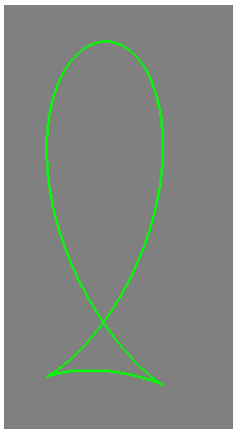

Figure 7: Model Curve

2.4 Blending Spline

When it comes to going from regular B-Splines to Blending splines, the process is rather simple. You need to make sure that you take the B-function: the Polynomial function of first order $B(t) = 3t^2 - 2t^3$ and blend it using the formula: 8

$$c(t) = \begin{pmatrix} 1 - B \circ w_{1,i}(t) & B \circ w_{1,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-1}(t) \\ c_i(t) \end{pmatrix}$$

Figure 8: Blending Spline formula

2.4.1 Implementation

The implementation of Blending Spline is rather similar to B-Spline but there are some key differences. First we make it so that instead of being hard coded for degree 3 its now hard coded for degree 1. Also the fact that our implementation is meant to be based on our model curves, we have to make sure that the program is capable of handling different curves, whether or not they are open or closed. As such we need to check if the curve is closed and do the necessary calculation depending on the answer we get. We also make sure to use our B-function on the knot interval map in order to blend the curves.[Lak]

2.5 Affine transformations

In the context of curves, an affine transformation refers to a geometric transformation that modifies the shape, size, and position of a curve without altering its fundamental properties. Affine transformations include operations such as translation (shifting the curve without changing its orientation), scaling (changing the size of the curve), rotation (turning the curve around a point), and shearing (slanting the curve in a particular direction). In this project I have implemented only translation, rotation and scaling. [Lak]

2.5.1 Implementation

In order to make sure our blending spline curve, we made sure to take advantage of the "localsimulate" function so that the changes that is done do the curve, is updated automatically. There were some challenges with trying to understand what sections and sizes of the unit-vector would make the curve move where and at what speed and what point the rotation would rotate around. But once i had grasped that i went on to design an "animation" using my fish curve as a basis. The way it works now is that the Fish travels in a square using translation, it will change direction whenever it reaches a corner, and will rotate itself to reflect this new direction. And while all this is taking place, the fin section and face of the curve is constantly rotating to simulate movement. I also have it set up so that the fish will constantly change color between red and black.

2.5.2 Result

9

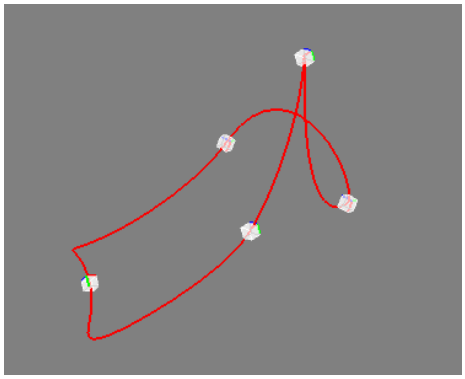

Figure 9: Affine transformation

3 Summary

In this paper I have gone trough the theoretical part about the various Curves, what procedure we followed to implement them. Now for the most part I am happy with the way this turned out. Even though i did have some trouble at the start when it came to implementing stuff, and also at the end while implementing affine transformation, as the course progressed, I felt that I started to get a better grasp at using C++ and GMlib. Now when it comes to the results the graphs do match the theory that has been presented. The only regret I have is to not being able to implement Blending Spline Surface with Special effects.

4 Appendix

In order to set up and run the application a few steps are needed. First you need to go to the course in Canvas [ins]. Then you got Syllabus and scroll all the way down. Then you follow the guide that is highlighted in 10, but instead of downloading the QmlDemo/Gmlib that is referenced in the setup process, you instead use the Demo/Gmlib found in my repository [Kam].

References

[ins] Mangler informasjonskapsel — uit.instructure.com. https://uit.instructure.com/courses/28487/assignments/syllabus. [Accessed 02-03-2024].

[Kam] MD Kamruzzaman. GitHub - mkjim456/Applied-geometry — github.com. https://github.com/mkjim456/Applied-geometry.git. [Accessed 02-03-2024].

[Lak] Arne Lakså. Blending techniques in Curve and Surface constructions. https://www.geofo.no/artikler/BlendBokEng.pdf. [Accessed 02-03-2024].

[wol] closed parametric equation for fish curve - Wolfram—Alpha — wolframalpha.com. https://www.wolframalpha.com/input?i=closed+parametric+equation+for+fish+curve. [Accessed 02-03-2024].


Quick links:
Blend Bok - english
Blend bok - norsk
Geometry representation
Spline representation
Subdivision Surfacees
Subdivision blending splines
Catmull-Clark
Simple sub surf
Qt GMlib guide

Figure 10: