Applied Geometry and Special Effects
Exam Submission

Kristoffer Berg Wilhelmsen

December 10, 2022

Abstract

Splines are a set of piecewise polynomials used to create smooth curves, and are central to applied geometry. This report takes a brief look at the theory behind different types of splines, and investigates programmatic implementations in C++ using the geometric modelling library GMlib. By applying theory to implementations, we gain a better understanding of splines and how the theory behind them translates to practical use cases.

1 Introduction

During the 1950s, we saw an increased demand for development within applied geometry [Lakså, 2022a, p. 1-4]. This was largely due to the increase in automated processes, as well as the introduction of the computer. The goal was to develop methods, algorithms and calculations for curve and surface representations, and apply these in computer graphics and simulations.

Splines are an essential part of applied geometry, and a recurring theme throughout this report. A spline, which is a set of piecewise polynomials, is a type of parametric geometry that has several subordinate types. Each type has various properties, which make for different use cases. We will take a closer look at b-splines, subdivision curves, blending splines and blending spline surfaces.

This report serves as an exam submission in the course *Applied Geometry and Special Effects*. The purpose of the report is to present relevant theory that has been discussed throughout the course, as well as programmatic implementations of "different geometric structures and other dynamic objects that continuously change shape" [Lakså, 2022b]. This is vital not only due to the importance of geometry in computer graphics as previously mentioned, but it can also be emphasized by a famous quote by the German mathematician and astronomer Johannes Kepler [Kepler, 1600]:

> Where there is matter, there is geometry.
> — Johannes Kepler

2 Methods

The work conducted in this report can be divided into six different subsections. Each subsection briefly describes the essence of each topic, which can then be connected to the different implementations in Section 3. The subsections have been listed in the chronological order of which the work has been done, and are as follows:

1. B-spline
2. Closed subdivision curve
3. Model curve
4. Blending spline curve
5. Special effects
6. Blending spline surface

The workflow has largely consisted of theory followed by implementations. Lectures, discussions and Arne Laksås book [Laksa, 2022a] have been a central part of the course. Figure 1 is an example of how theory was typically discussed in combination with lectures. All implementations have been written in the C++ programming language. This is due to the task requiring the geometric modelling library GMlib [Bratlie et al., 2019] to be used, which is a lso written in C++. Furthermore, most of the programming has been done under the supervision of professor Arne Laksa. Although none of the code has been directly copied, it is important to note that Laksa has made a significant verbal contribution to the implementations, and that most implementations are based on existing GMlib code. The verbal contribution was necessary due to the complexity of the task, combined with the short amount of time in which the task had to be completed.

2.1 B-spline

B-splines are the industry standard for splines as of today [Laksa, 2022a, p. 75]. Roughly speaking, they consist of three main components. These are control points, a set of local basis functions and the knot vector.

A set of control points define a control polygon that is used to create interpolation points. The number of interpolation points is equal to the number of control points minus the start and the end point [Laksa, 2022a, p. 102-104]. We can either define a set of control points, or in the case that we have a larger number of interpolation points than degrees of freedom, we can approximate the control points. This is necessary in order to obtain a square matrix, which is used in the final calculations. Approximation can be done by means of the least squares method.

The set of basis functions and the knot vector are essential to b-splines, as well as being closely related [Laksa, 2022a, p. 82]. The basis functions dictate

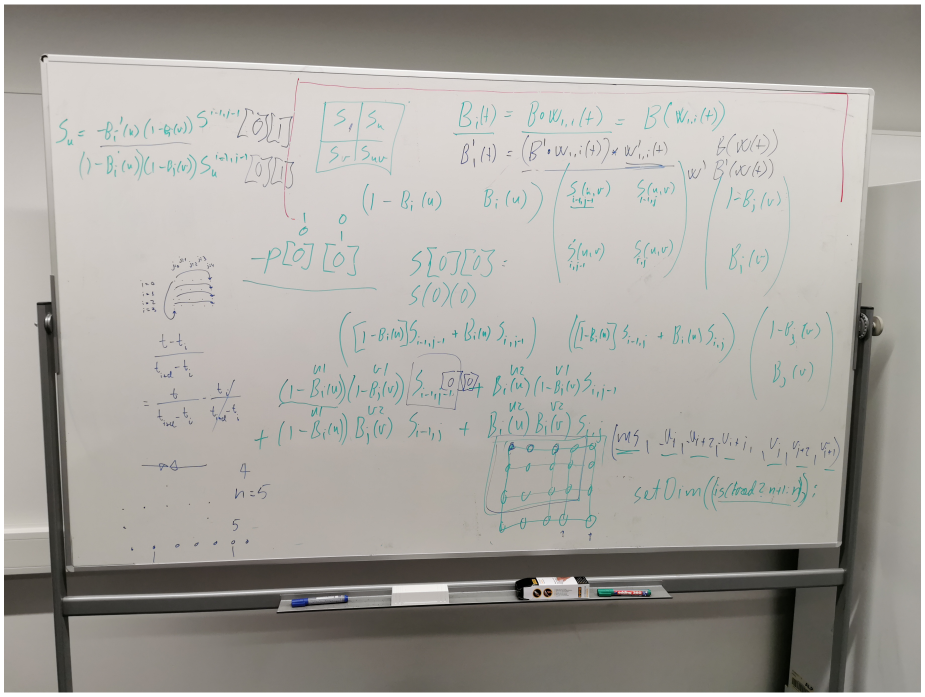

Figure 1: Example of how theory and implementation has been discussed.

the behavior of what we call local curves, while the knot vector consists of values that describe their domain. The curves are local in the sense that only the local curves are affected by changes relative to their position, as opposed to the global curve (e.g. Bézier curve). When the local curves are put together, they form the b-spline.

2.2 Closed subdivision curve

Subdivision is a method in relation to curves which is used to smoothen a curve, usually by cutting corners [Lakså, 2022a, p. 106]. Given a set of initial points, the method replaces the original set with a new, larger set. There are two types of subdivision. One where new points are added between the original points, and another which uses interpolation. We will focus on the former, while also limiting the scope to a closed curve.

Lane-Riesenfeld is a subdivision algorithm which is performed in two steps. The first step consists of doubling the number of points from the initial set. After the number of points has been doubled, they must be smoothened out. This is done by iterating over the new set of points, setting the value of each individual point equal to the middle of two old points.

2.3 Model curve

The model curve is neither a complex implementation nor very interesting by itself. It is simply an implementation of a given parametric curve, which we will use in the next part of the report concerning blending splines. The blending spline will copy the model curve, and in the case that one of the model curves fail to provide sufficient results (e.g. model curve is too easy to copy, not enough complexity) it can be useful to have several implementations to have some options to choose from.

2.4 Blending spline curve

The blending spline curve is similar to the b-spline, in the sense that some calculations have similarities, as well as the presence of the knot vector. The real difference here is that instead of using control points to create local curves, we use an implementation of a model curve as a parameter. The model curve is then split into a number of local curves that we define, which are blended into a single curve using a blending function that we specify.

2.5 Special effects

The main part of the project is to implement an affine transformation of local curves that cause dynamic changes to the local curves of the blending spline. Examples of possible transformations include, but are not limited to: rotation, tilting, turning and rolling. The idea behind this is to see how the blending spline behaves when local curves are modified.

2.6 Blending spline surface

The blending spline surface is built upon the same concept as the blending spline curve, with the distinction that it uses local surfaces as opposed to local curves [Lakså, 2022a, p. 217]. We first specify a global surface which we use as a parameter, and the number of local surfaces to divide the global surface into in each direction (u, v). Knot vectors must be created for each direction, followed by the local surfaces themselves. The local surfaces are then blended into a single surface by a defined blending function in the same fashion as the blending spline curve.

3 Results

Similarly to Section 2, the results have been divided into six different subsections. The ordering of the subsections is equivalent to the ordering in the aforementioned section.

3.1 B-spline

We have implemented two methods of creating a b-spline. Figure 2 shows the first implementation, where the b-spline is plotted by specifying a set of control points. The control points used for this particular b-spline are (0, 0), (0, 3), (1, 2), (2, 3), (2, 0), (1, 1). All points have a z-value of 0, and the actual values are floats. The points are visually represented in Figure 3.

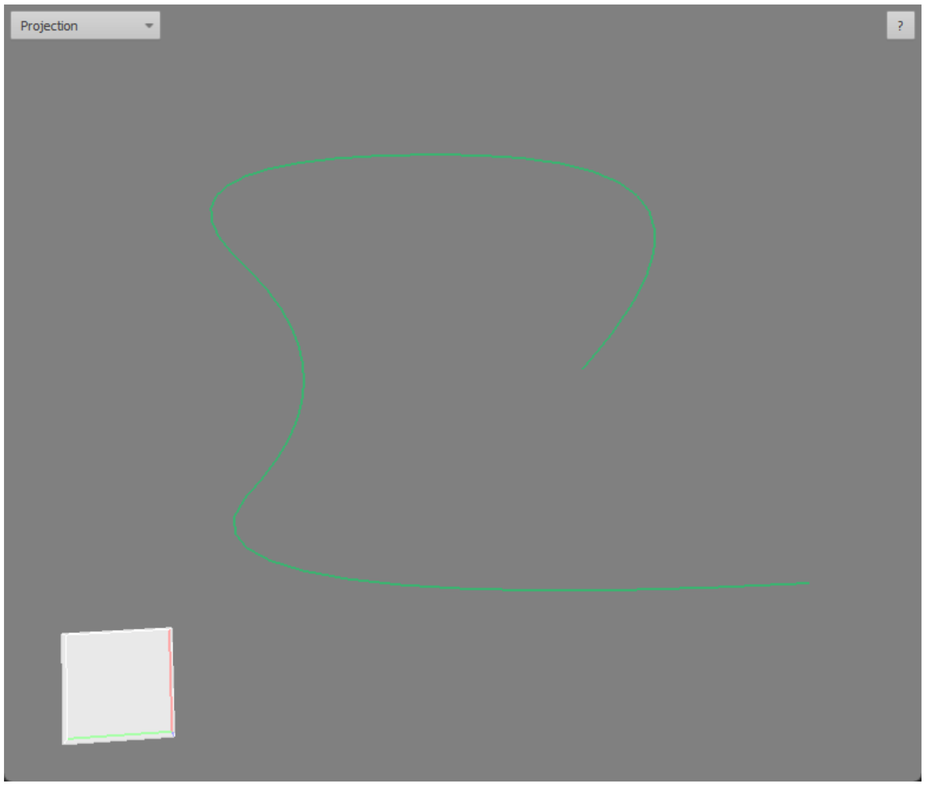

Figure 2: B-spline plotted with control points specified.

The second implementation which uses least squares method to approximate interpolation points can be seen in Figure 4, where two visualizations have been plotted with a different number of interpolation points. The green shapes are circles inserted into the scene for comparison, and the red shapes are b-splines. The left-hand side of the picture marked as *a*) shows a b-spline using 20 interpolation points, and the right-hand side marked as *b*) only uses 4. This emphasizes the importance of using a sufficient amount of interpolation points in order to get a good representation. Note that the curves in *a*) have been moved slightly apart to easily make a distinction.

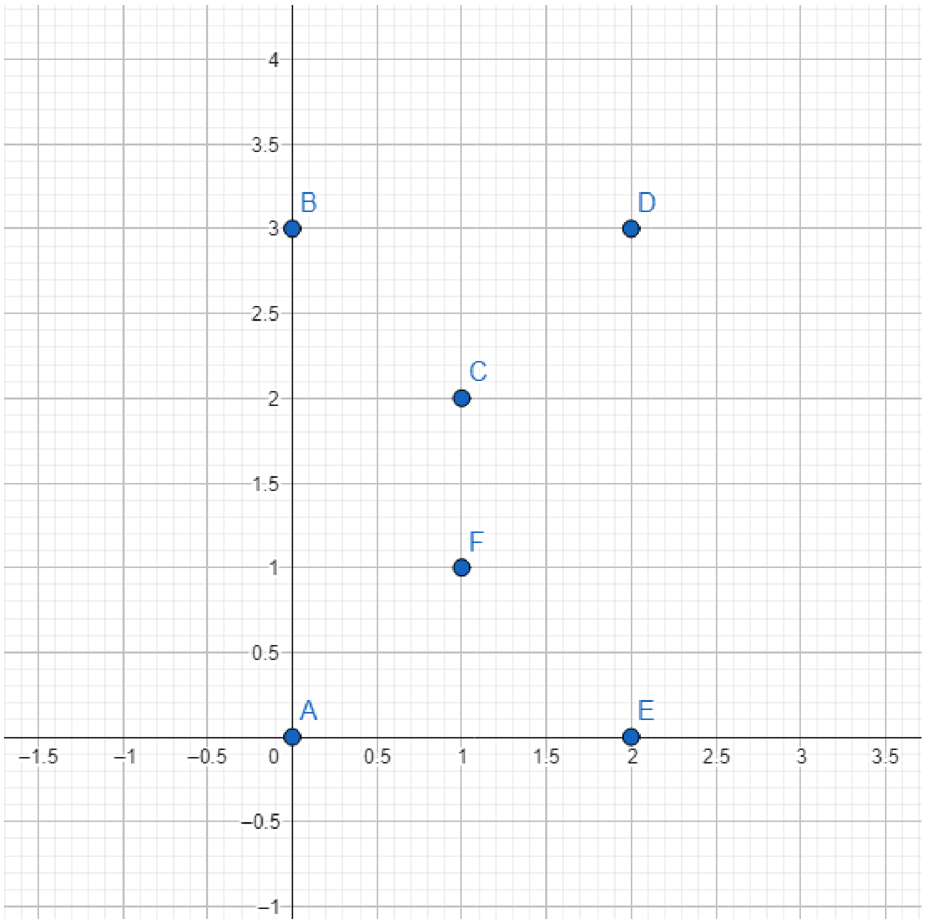

Figure 3: Visual representation of points used in first b-spline.

3.2 Closed subdivision curve

A closed subdivision curve with two levels of refinement, which is the number of times the points in the curve is doubled, can be seen in Figure 5. The same points as used in the b-spline have been used here. Curve a) has 4 levels of refinement, while curve b) has 2 levels of refinement. The figure shows that curve a), which has a higher level of refinement, is smoother than curve b).

3.3 Model curve

Three different model curves have been implemented. Figure 6 shows the first curve, which is an astroid curve [Kokoska, 2013, p. 2]. Figure 7 shows the second curve, which is an epitrochoid curve [Kokoska, 2013, p. 18]. The third and final

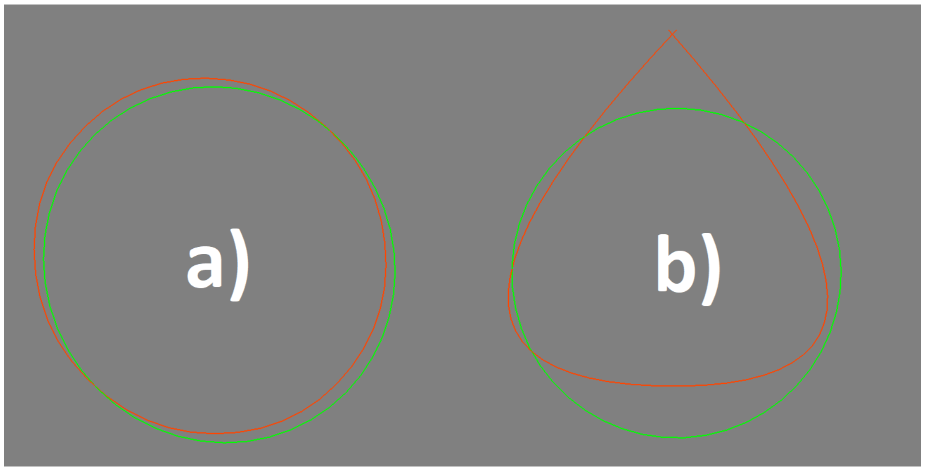

Figure 4: B-spline plotted by approximating interpolation points with least squares method.

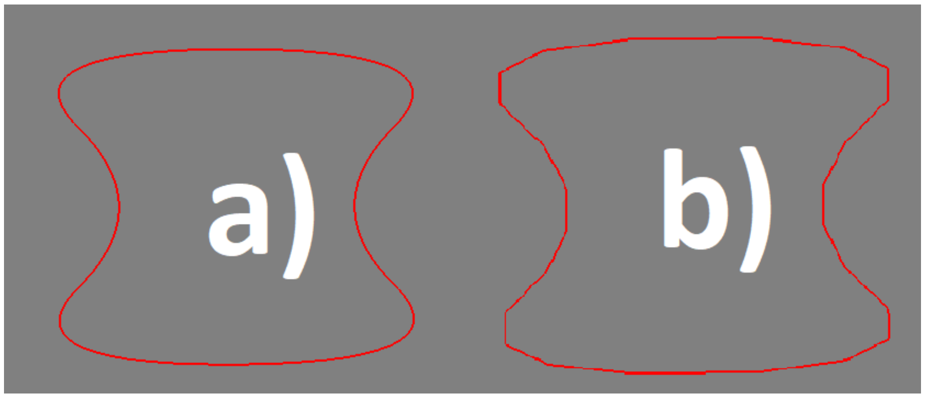

Figure 5: Closed subdivision curve with different levels of refinement.

model curve can be seen in Figure 8, and is a nephroid curve [Kokoska, 2013, p. 33].

3.4 Blending spline curve

The epitrochoid curve was chosen as the model curve for the blending spline to copy. The epitrochoid curve has symmetric features and a lot of variation in its curves. This should make for a more interesting animation, due to the fact that there will be a larger amount of local curves to observe change in. Figure

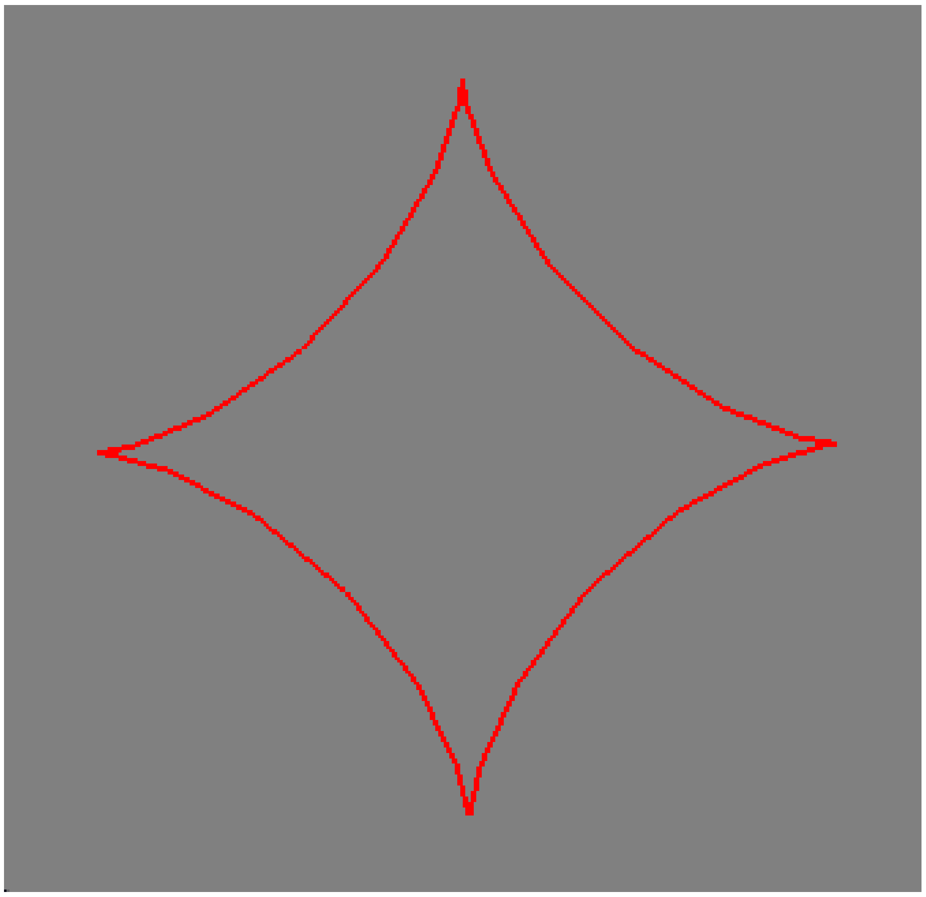

Figure 6: Astroid curve.

9 shows the blending spline plotted with 14 local curves, where each local curve is indicated by a box. The implemented b-function is a 3rd degree polynomial function.

3.5 Special effects

To apply special effects to our blending spline curve, we can implement the virtual method *localSimulate* which is inherited from *SceneObject*. This method is continuously called during runtime, which makes applying special effects easy. As requested in the task description [Laksâ, 2022b], we have implemented effects that change the shape of the curve by applying a *roll*- and *translation*-effect to the curve. Figure 10 illustrates how the special effects affect the curve at a certain point during runtime. Note that the curve is continuously going back

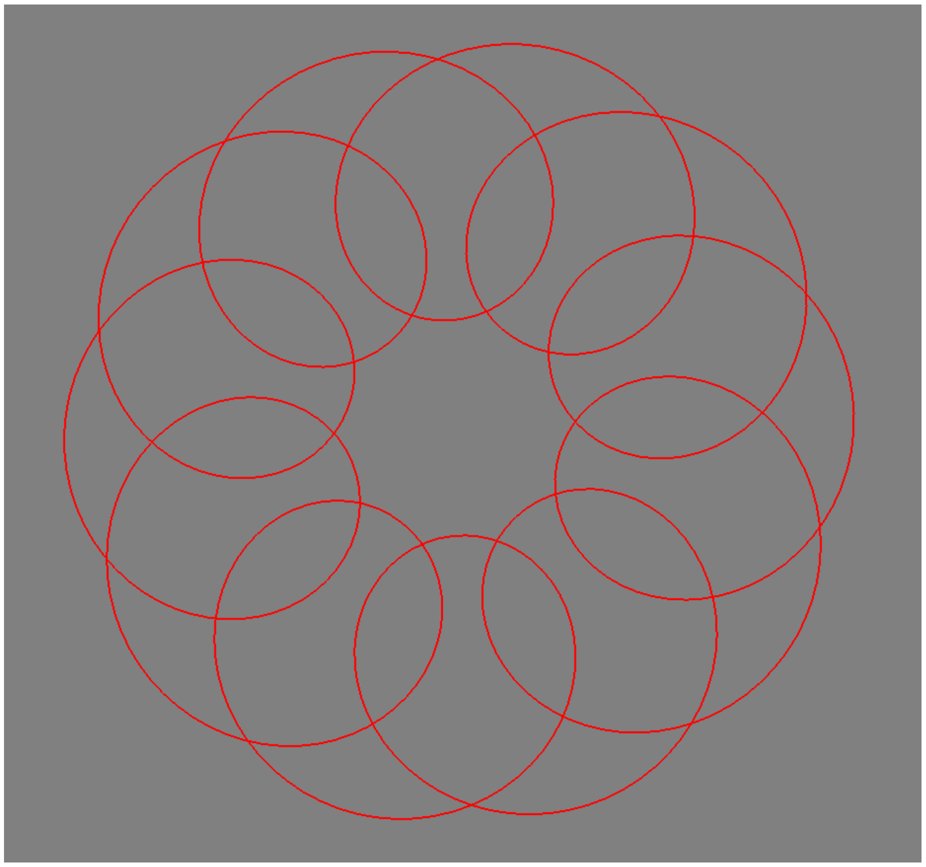

Figure 7: Epitrochoid curve.

and forth between its original and modified shape, although this is not obvious by looking at the figure.

3.6 Blending spline surface

A good way of illustrating a blending spline surface is to modify the position of one or more local surfaces, and observing the change in the global surface. On Figure 11 we see a torus that has been used to approximate a blending spline surface consisting of 8 local surfaces in each direction. The torus has two states, *a*)and *b*). In state *a*), no local surfaces have been modified. In state *b*), we can clearly see how three local surfaces have been modified. The result is a change in the blending spline surface at positions that correspond to the modified local surfaces.

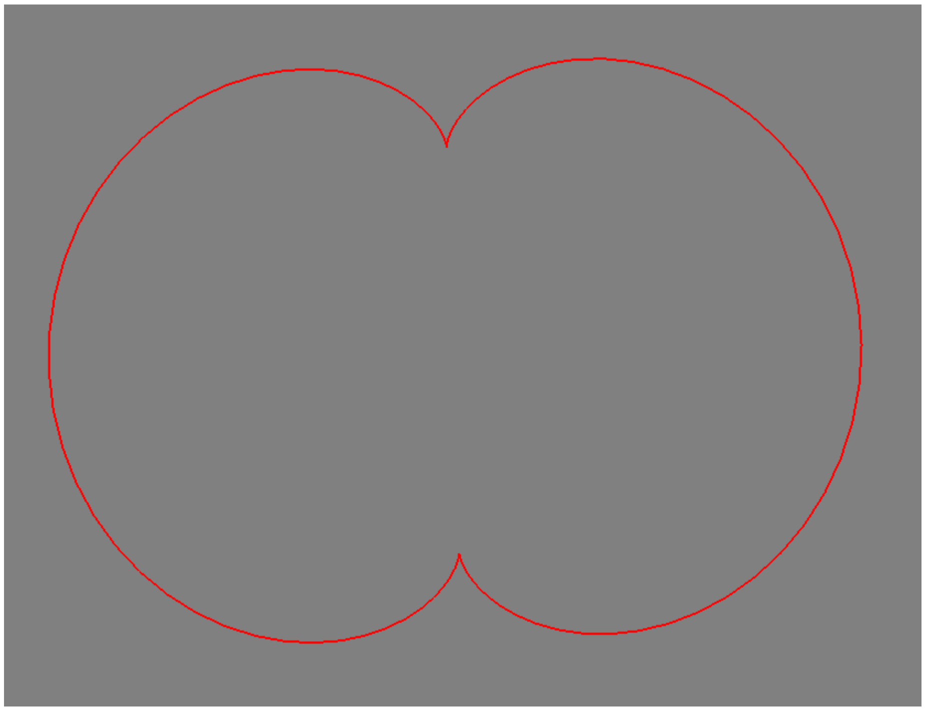

Figure 8: Nephroid curve.

4 Discussion

Programming the implementations was a relatively painless process, although there were a few challenges.

The most notable challenge was an inconsistency in intelligent code completion (IntelliSense). Most of the time when editing H- or CPP-files, IntelliSense would be present. However, when editing C-files, IntelliSense would have limited or no presence at all. When writing code in a modern IDE, you expect IntelliSense to be present. It makes writing code much faster and easier. Both Visual Studio 2022 and multiple versions of Qt Creator were tested, however the issue persisted.

Compilation and syntax errors would not appear visually in the editor in the same manner as the IntelliSense issue. This meant that the only way to find out where errors were present was to read output from the compiler.

Although neither of these issues prevent development entirely, they certainly slowed it down. For instance, when attempting to call a function from GMlib, it was necessary to open and read the corresponding header or C++ files when the syntax was not memorized. This added a lot of overhead to the implementations.

A minor issue worth mentioning is various bugs in the code that were en-

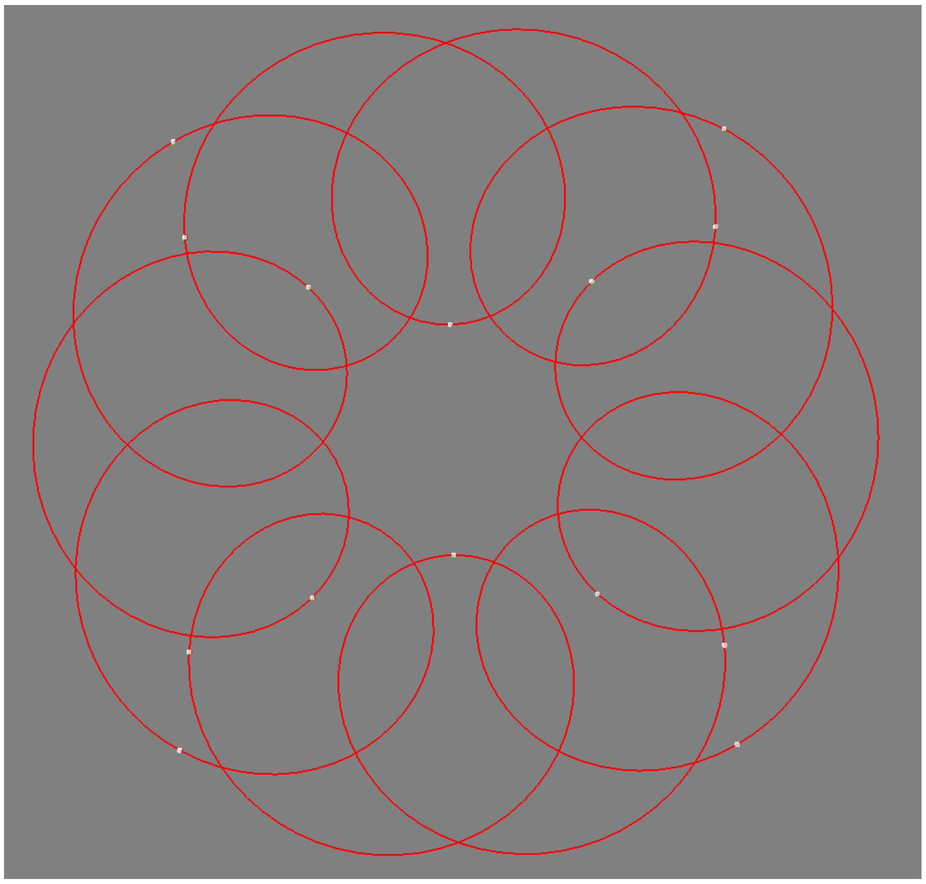

Figure 9: Blending spline curve using epitrochoid curve as the model curve.

countered during development. There were times where as little as a missing bracket could cause the program's behavior to be entirely different from what was expected. This would result in a lot of debugging, but luckily professor Arne Laksă was present at campus a lot of the time to assist in detecting logical errors.

Lastly, an unexpected issue that occurred during the end of the project work was implementing the blending spline surface. This boiled down to the provided literature not translating as easily to a practical implementation as one could hope. However, with the help of fellow student Jan Einar Kristiansen, the blending spline surface was implemented. As such, it is important to acknowledge and thank him for his contribution.

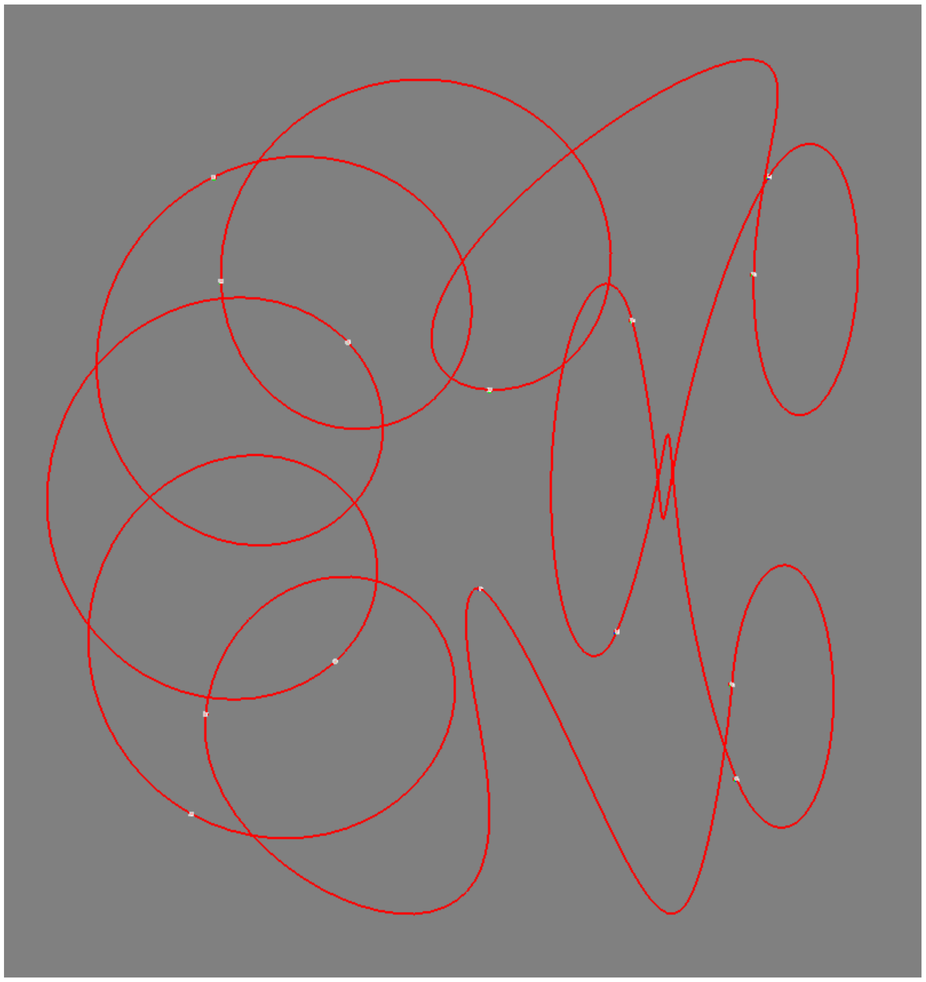

Figure 10: Special effects on the blending spline curve.

5 Conclusion

During the work done in this report, we have taken a closer look at splines, which are a central part of applied geometry. By diving deeper into relevant theory and connecting it to practical implementations, we are more or less forced to absorb some amount of knowledge relevant to the topic have worked with. With everything taken into consideration, we can conclude that although no groundbreaking research was done, the project work was both highly educational and meaningful.

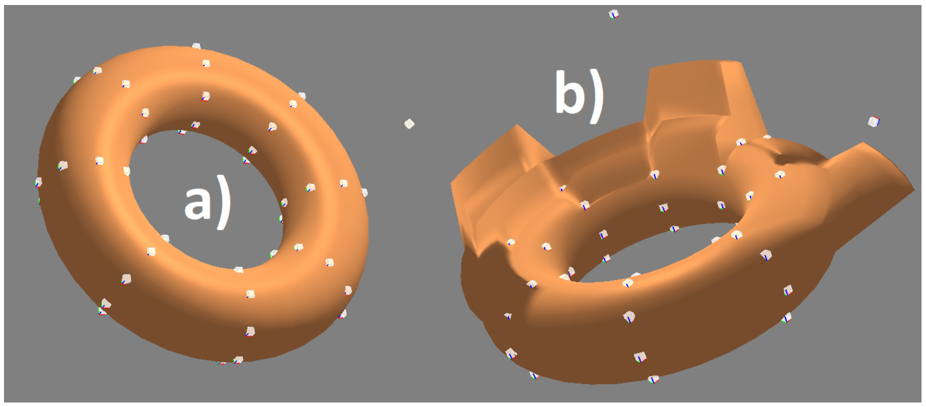

Figure 11: Torus approximation by a blending spline surface.

References

[Bratlie et al., 2019] Bratlie, J., Nordeng, C., Dalmo, R., Laksaa, A., Tran, T., Brustad, T., and Pedersen, A. (2019). GMlib. `https://source.coderefinery.org/gmlib/gmlib1`. Accessed: 05.12.2022.

[Kepler, 1600] Kepler, J. (1600). Quote of Johannes Kepler. `https://www.buboquote.com/en/quote/5060-kepler-where-there-is-matter-there-is-geometry`. Accessed: 05.12.2022.

[Kokoska, 2013] Kokoska, S. (2013). Fifty Famous Curves, Lots of Calculus Questions, And a Few Answers. `https://elepa.files.wordpress.com/2013/11/fifty-famous-curves.pdf`. Accessed: 28.11.2022.

[Lakså, 2022a] Lakså, A. (2022a). *Blending techniques in Curve and Surface constructions*. The geometry publishing house.

[Lakså, 2022b] Lakså, A. (2022b). Course syllabus. `https://uit.instructure.com/courses/27058/assignments/syllabus`. Accessed: 05.12.2022.

A Installation and setup

The project repository is available online and can be cloned from: `https://github.com/kribw/applied-geometry`. The relevant files to the project can be found in `/applied_geometry/demo/appliedgeometry`.

Compiling GMlib is a prerequisite to compiling and running the project. Instructions on how to compile GMlib can be found here `https://source.coderefinery.org/gmlib/gmlib1/qmldemo/-/wikis/getting_started`.

Once GMlib is compiled, the "cmake" directory in the build files must be specified in `/applied_geometry/demo/CMakeLists.txt`. This can be set by changing the *gmlib_DIR* variable to the corresponding directory on line 27.

It may be necessary to set the path for the vcpkg toolchain-file, which is used for other potential dependencies. This can be done either through the IDE or CMake. The user will be alerted of any other missing dependencies during compilation, which can easily be installed through vcpkg.

The main project which needs to be compiled is located in `/applied_geometry/demo/`. It is recommended to compile with CMake. Once the user has confirmed that the project compiles, the features (e.g. b-spline, blending spline surface) that the user wants to run must be uncommented at the bottom of *initializeScenario* in `/applied_geometry/demo/scenario.cpp`. Now the program can be compiled, and the process can be repeated to test other functionality.