ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Μελέτη και εφαρμογή αλγορίθμων για τη σχεδίαση splines και καμπυλών Bezier

Study and implementation of algorithms for drawing splines and Bezier curves

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΑΤΡΙΒΗ

Της

Ζαμπακίκα Κλεοπάτρας

Βόλος, Ιούνιος 2019

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Μελέτη και εφαρμογή αλγορίθμων για τη σχεδίαση splines και καμπυλών Bezier

Study and implementation of algorithms for drawing splines and Bezier curves

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΑΤΡΙΒΗ

Της

Ζαμπακίκα Κλεοπάτρας

Επιβλέποντες :

Τσομπανοπούλου Παναγιώτα
Αναπληρώτρια Καθηγήτρια Π.Θ.

Βασιλακόπουλος Μιχαήλ
Αναπληρωτής Καθηγητής Π.Θ.

Ευμορφόπουλος Νέστωρ
Επίκουρος Καθηγητής Π.Θ.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την

….....

Τσομπανοπούλου Παναγιώτα
Αναπληρώτρια Καθηγήτρια Π.Θ.

….....

Βασιλακόπουλος Μιχαήλ
Αναπληρωτής Καθηγητής Π.Θ.

….....

Ευμορφόπουλος Νέστωρ
Επίκουρος Καθηγητής Π.Θ.

Βόλος, Ιούνιος 2019

.....

Κλεοπάτρα Ζαμπακίκα

Διπλωματική Εργασια για την απόκτηση Μεταπτυχιακού Διπλώματος Ειδίκευσης στην «Επιστήμη και Τεχνολογία Υπολογιστών, Τηλεπικοινωνιών και Δικτύων», στα Πλαίσια του Προγράμματος Μεταπτυχιακών Σπουδών του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας

Declaration of Authorship

I, Kleopatra Zampakika, confirm that this thesis is my own work. All direct or in-direct sources used are acknowledged as references. This thesis was not previously presented to another examination board and has not been published.

Dedicated to my family and my friends. . .

Abstract

Computer Graphics is one of the most exciting and rapidly growing fields. In order to create realistic-looking images we have to accurately compute the outer surface of objects. To achieve this goal we first start with an understanding of curves, and once we have an algorithm to calculate and visualize them, we can extend it to a surface.

The purpose of this thesis is to study and design flexible curves and implement their generation algorithms in a Java application. Specifically, interactive software was developed for calculation and visualization of Bezier curves and surfaces.

The interface and content of this application were designed with an educational nature, providing straight forward and easy to understand examples of all the basic concepts. Through a series of stand-alone tasks students will come across a detailed demonstration of Bezier curve and surface creation, witness their value for practical applications like vector drawing or 3D computer graphics and compare them with the corresponding piecewise splines.

The core algorithms on the field were implemented and even more are left as future extensions, with the ambition to create a practical educational tool for Bezier curve studying.

Περίληψη

Τα γραφικά υπολογιστών είναι από τους πιο ενδιαφέροντες και ραγδαία αναπτυσσόμενους τομείς. Προκειμένου να δημιουργηθούν εικόνες που να πλησιάζουν την πραγματικότητα πρέπει να υπολογιστεί με ακρίβεια η εξωτερική επιφάνεια των αντικειμένων. Για να επιτευχθεί αυτός ο στόχος αρχικά θα πρέπει να κατανοηθούν οι καμπύλες και μόλις υλοποιηθεί ένας αλγόριθμος για αυτές, μπορεί να γίνει επέκταση στις επιφάνειες.

Σκοπός αυτής της διπλωματικής είναι να μελετηθούν και να σχεδιαστούν οι εύκαμπτες καμπύλες καθώς και να υλοποιηθούν οι αλγόριθμοι κατασκευής τους σε μία εφαρμογή Java. Συγκεκριμένα αναπτύχθηκε ένα δια δραστικό λογισμικό για τον υπολογισμό και την απεικόνιση των καμπυλών και επιφανειών Bezier και splines.

Η εμφάνιση και το περιεχόμενο της εφαρμογής ακολουθούν ένα μοτίβο εκπαιδευτικής εφαρμογής. Παρέχει ξεκάθαρη και εύκολη κατανόηση όλων των βασικών εννοιών επί του θέματος με παραδείγματα. Με μία σειρά από αυτόνομα τμήματα οι χρήστες μπορούν να παρακολουθήσουν μία λεπτομερή επίδειξη δημιουργίας καμπυλών Bezier, να επιβεβαιώσουν την χρησιμότητα τους για τη διανυσματική σχεδίαση σε 3D γραφικά και να τις συγκρίνουν με αντίστοιχα τμηματικά splines.

Υλοποιήθηκαν οι κυριότεροι αλγόριθμοι πάνω στον τομέα και υπάρχουν αρκετοί που μπορούν να προστεθούν σαν μελλοντικές επεκτάσεις, με τη φιλοδοξία να δημιουργηθεί ένα πρακτικό εκπαιδευτικό εργαλείο για την εκμάθηση των καμπυλών Bezier και splines.

For the fulfillment of this Thesis, I would like to thank my professor Panagiota Tsompanopoulou for her advice and guidance and my colleague Parnassos Ioannis for his support, collaboration and ideas.

Also I would like to thank my family for their support and patience...

Contents

Table of Figures

1. Introduction

This thesis was made with a training purpose to create a user-friendly tool for students. The use of the tool by the students themselves can help in the smoother introduction of mathematical concepts in the classroom, the recognition of mathematical properties and structures and improve their mathematical intuition. The tool gives students the ability to create their own designs with the help of Bezier curves and adjust what they make based on their attributes. This process and the fact that visually realizing the use of curves helps to understand mathematical concepts. The creation of the tool was done in such a way that students gradually understand the meaning of the curves and make applications on them and later to see how to make a whole surface based on the Bezier curves. Then they can see differences with the splines and do their own tests.

It is difficult for an educator to have the utmost attention of students and to incorporate them into such mathematical concepts. However, when students get into the process of creating something on their own or even collaborating, then learning becomes more interesting. Such an app can offer students a slight dose of enthusiasm and keep them focuses. The whole process can be done in the form of a game. Students can store their designs, and everyone should try to explain to their classmates how this was created based on the control point positions they chose.

2. Curves

A curve is a generalization of a line with not necessarily zero curvature. It consists of an infinitely large set of points, each one having two neighbors, except for the two endpoints. When those endpoints are the same the curve is closed. The definition may be simple but their utilization expands not only in mathematical domains but also in everyday practice.

2.1. Curve Representation

The mathematical representation of a curve can be classified into three categories, implicit, explicit, and parametric.

Explicit curves have the form y = f(x). This type of curves cannot represent vertical lines and is single-valued, since for each value of x the function calculate only one value of y. As a result by using implicit functions it is impossible to model every curve in two dimension space like for example every closed curve. Also there is no passage to 3D graphics because every curve is confined to a single plane.

Implicit curve representation has the form f(x, y) = 0. In contrary to the explicit form it can represent multivalued curves. A common example is the circle $x^2 + y^2 - R^2 = 0$. For computer graphics this form is often unintuitive, more difficult to render and a impractical for 3D curves, but useful for modeling and medical imaging.

The types of curves described above are axis dependent and can be used when the function is known. For computer graphics we prefer a far less limiting approach, the parametric curves. The form of this type is P (t) = (x (t); y (t)) where the parameter t takes values in the range of [a, b], often normalized in [0, 1].

It is clear that parametric form is less comprehensible and analytical because we do not have all the information to understand the shape of curve and geometry. On the other hand the parametric representation is ideal for computer graphics since it is dimension independent, unaffected by infinite slope problems, directly transformable and easy to express in vector and matrix form.

2.2. Splines

In mechanics, spline is a flexible metal or plastic strip, which is curved in order to pass through several predefined points. In computer graphics it refers to a curve that is defined by two or more points or that connects a series of points. Another reference for this term is the mathematical equation that defines such a curve, a special type of piecewise polynomial. Every piece is defined by a polynomial function and at the connection point between two pieces there is no discontinuity. Those points connecting the pieces are called control points or knots.

2.2.1. Interpolation Definition

As mentioned before a curve consists of an infinitely large set of points. If only a few of them are known a technique is needed to estimate the rest and fill the curve. This process is called interpolation and is used to missing data filling, data smoothing, prediction making, and several other applications.

Complicated curves can be approximated with polynomials. Polynomial interpolation on a given dataset will find the polynomial of the lowest possible degree that passes through all the points of the dataset.

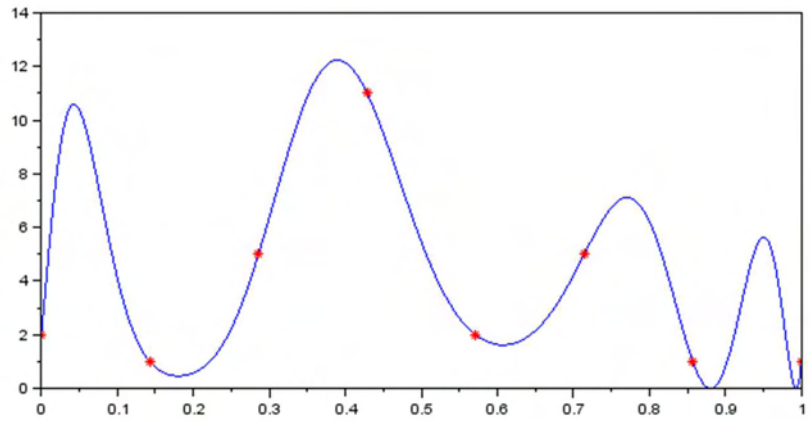

Figure 1 : Example of interpolated curve passing through given points

2.2.2. *Spline Interpolation*

The polynomial provided with interpolation will have a degree that equals to the number of known points minus one. In general, for n data points, there exists one polynomial of degree up to n−1 going through every data point. By using polynomial interpolation with polynomials of high degree we come across Runge's phenomenon, a problem of oscillation at the edges. To avoid these undesirable oscillations we can use the method of splines, introduced by I. J. Schoenberg in 1946. Spline interpolation can approximate the curve piece by piece with lower degree polynomials, thus avoiding this issue.

The most famous type of splines is cubic. A cubic spline pass thought m control points and is constructed by third degree piecewise polynomials. This method is used a lot in every day practice. It also initiated the growth of modern CAD (computer aided design)

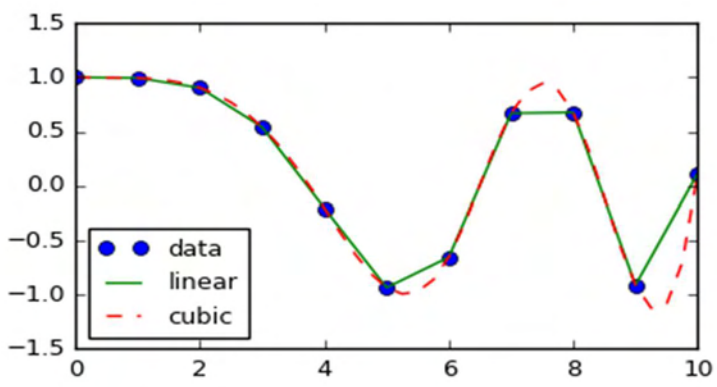

Figure 2: Comparing data interpolation of 1st and 2nd degree

To construct smooth curves in computer graphics the most commonly used polynomial is the cubic. In this type of splines the first and second derivatives at the points matches for the neighboring pieces. At the endpoints the second derivative is commonly set to zero since this provides a boundary condition that completes the system of equations. This is the natural cubic spline which makes a symmetric tridiagonial system. This system can be solved and gives the coefficients of the polynomials.

Consider n+1 points (p_0, p_1, …, p_n)

Where each point p_i consists of the coordinate pair (x_i, y_i).

The i_{th} piece of the spline will be represented by:

$$S_i(t) = a_i + b_i t + c_i t^2 + d_i t^3 \quad (1)$$

Where parameter $t \in [0, 1]$ and i=0, …., n-1.

Then:

$$S_i(0) = p_i = a_i \quad (2)$$

$$S_i(1) = p_{i+1} = a_i + b_i + c_i + d_i \quad (3)$$

Taking the derivative of S_i (t) for each interval gives:

$$S_i'(0) = b_i = X_i \quad (4)$$

$$S_i'(1) = b_i + 2c_i + 3d_i = X_{i+1} \quad (5)$$

Solving (2) – (5) gives:

$$a_i = p_i \quad (6)$$

$$b_i = X_i \quad (7)$$

$$c_i = 3(p_{i+1} - p_i) - 2X_i - X_{i+1} \quad (8)$$

$$d_i = 2(p_i - p_{i+1}) + X_i + X_{i+1} \quad (9)$$

Equations for interior points:

$$S_{i-1}(1) = p_i \quad (10)$$

$$S_{i-1}'(1) = S_i'(0) \quad (11)$$

$$S_i(0) = p_i \quad (12)$$

$$S_i''(1) = S_{i+1}''(0) \quad (13)$$

The endpoints already satisfy two more equations:

$S_0(0) = p_0$ (14)

$S_{n-1}(1) = p_n$ (15)

And the selected boundary condition complete the system:

$S_0''(0) = 0$ (16)

$S_{n-1}''(1) = 0$ (17)

Rearranging all these equations leads to the following symmetric tridiagonial system:

$$\begin{bmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ & & : & : & : & : & : \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ : \\ X_{n-1} \\ X_n \end{bmatrix} = \begin{bmatrix} 3\,(p_1 - p_0) \\ 3\,(p_2 - p_0) \\ 3\,(p_3 - p_1) \\ 3\,(p_4 - p_2) \\ : \\ 3\,(p_n - p_{n-2}) \\ 3\,(p_n - p_{n-1}) \end{bmatrix}$$

By solving this linear equation system and with (1), (6), (7), (8), (9), we can recursively evaluate a_i, b_i, c_i, d_i and $S_i(t)$ $\forall\, i \in [0, n-1]$.

2.3. Bezier Curves

Bezier curve is a useful tool in creation of vector graphics. They are widely used in computer graphics to model smooth curves. They were first described by Paul de Casteljau in 1959 but they became known in 1970's when Pierre Bezier introduced an industrial application of Bezier curves and surfaces for designing automobile bodies.

A Bezier parametric curve is defined by n control points P_0, P_1,... , P_n, n>1. Quadratic (parabolic) and cubic Bezier curves are the most common, although Bezier curves can be defined for any degree n. For n=2 we get a straight line. The edges of the curve are the first and the final point respectively. The curve passes only from the edges, while the other points between them are called control points and help in the curve's creation. By moving the control points the curve is changing in a way that can be intuitively obvious. If all the control points are collinear the curve is a straight line.

There are two ways of drawing a Bezier curve. One is an analytical expression and the other a geometric algorithm. They are different ways to reach an identical result. The first by computation through Berstein polynomial, a linear combination of Berstein basis polynomials. The second way is to evaluate the curve with de Casteljau's algorithm, a recursive and numerically stable method. Through the algorithm it is possible to understand the concepts of Bezier curves. For this reason also we will start directly with the second method that also has an educational aspect.

2.3.1. De Casteljau algorithm

Named after Paul de Casteljau, this algorithm is a geometric approach in drawing Bezier curves, without using any complicated calculus. With this algorithm it is possible to construct a Bezier curve or just find a particular points on it for specific parametric values. It can also be used to split a single Bezier curve into two at a random parameter value. In terms of computing speed when compared to the direct approach the algorithm is usually slower but is more numerically stable.

To construct a Bezier curve we must find several points through which the curve will pass. These points depend on a parameter t in [0, 1]. After we have calculated enough points drawing the Bezier curve is as easy as connecting those dots. It is obvious that the more points we have, the better is the approximation of the Bezier curve. For example, if the algorithm is used to draw by hand we could first look for the center of the curve and

afterwards look for the quarter points on the curve and then connect the five points we have, including the end points.

A cubic Bezier curve example is the following. We have two end points P0, P3 and two control points P1, P2, as seen in figure 3. We first look for the center (t=1/2) of the curve. For each segment the points corresponding to t=0.5 are calculated and consecutive ones are connected to each other. The newly created segments, serve as assisting control lines, a temporary but necessary step in the process and can be seen in figure 4.

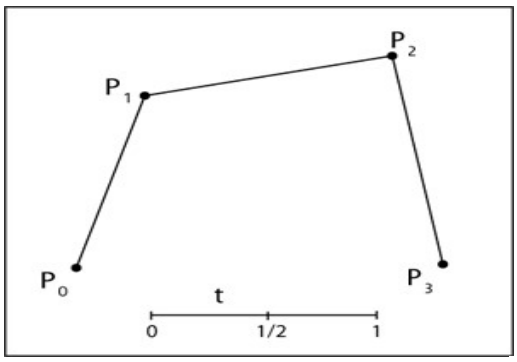

Figure 4 : Step One

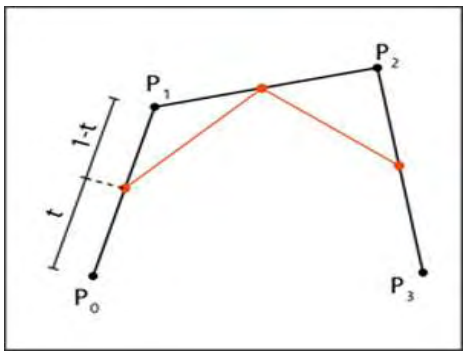

Figure 3 : Step Two

In the next step the middle point of these new segments have to be determined. That step is repeated as many times as the degree of the Bezier curve is. For a cubic Bezier curve the degree is 3 so that step is executed three times.

The second execution will create the yellow segment as shown in figure 5 and one last execution of the same step for our example's cubic curve, in order to find the middle point of the last segment as shown in figure 6. The final output is the center of the Bezier curve (t=0,5).

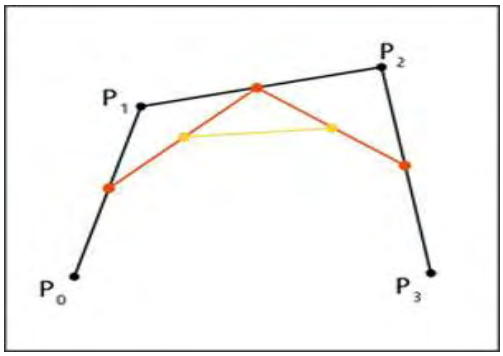

Figure 5 : Step Three

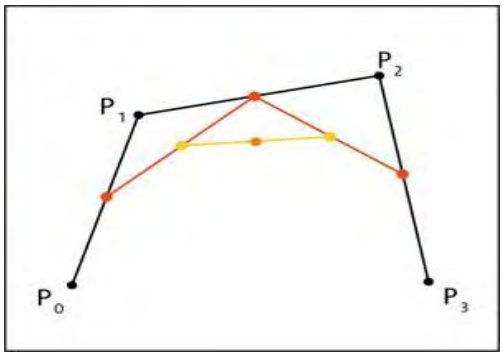

Figure 6 : Step Four

If this algorithm is repeated for many values of t, for example t from 0 to 1 with step 0.01 the middle point of the last segment will be drawing the actual Bezier curve (figure 7).

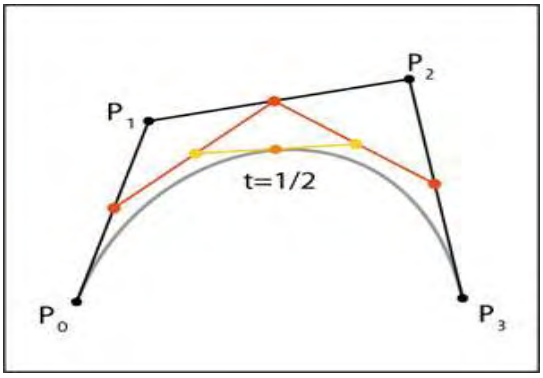

Figure 7 : Final Curve

Evaluating the equation of the curve for every values of t in [0 , 1] is like walking along the curve. The result is a position in 2D or 3D space. So, if we want to visualize a parametric curve the process begins by evaluating the equation for several values of t. Afterwards by connecting the resulting generated points in space we create a polygonal path as illustrated in next figure.

Figure 8: polygonal path

Even with four segments we begin to see what the final shape of the curve may look like. For smooth result we need to run this algorithm for a large number of segments. This is, how we can calculate and draw Bézier curves. With sampling in regular intervals and by connecting the points to create several sequential connected line segments.

The pseudocode implementation for a bicubic Bezier curves is the following:

```
dot de_casteljau(point A1, point A2, point A3, point A4, float t){
// compute first 3 points and segments A1A2, A2A3 and A3A4
    dot A12 = (1 - t) * A1 + (t * A2);
    dot A23 = (1 - t) * A2 + (t * A3);
    dot A34 = (1 - t) * A3 + (t * A4);
// new segments A1A2A2A3 and A2A3A3A4
    dot A1223 = (1 - t) * A12 + (t * A23);
    dot A2334 = (1 - t) * A23 + (t * A34);
// finally compute last point on A1A2A2A3A2A3A3A4
    return (1 - t) * A1223 + (t * A2334);
}
```

The De Casteljau method is computationally more expensive than the calculating the analytical expression (evaluating the Bernstein polynomials directly)) but is numerically more stable.

2.3.2. Bernstein Polynomials

For a cubic Bezier curve with control points P_1, P_2, P_3 and P_4 the final curve is a result of combining the four control points weighted by some value.
For example:

$$P_{curve}(t) = P_1*k_1+P_2*k_2+P_3*k_3+P_4*k_4$$

Where k_1, k_2, k_3, k_4 are scalar coefficients with a weighted contribution on control points. Intuitively we can realize that for $t = 0$, the first point matches with control point P_1 and similarly for $t = 1$ the last point with P_4.

$$P_{curve}(0)=P_1*1+P_2*0+P_3*0+P_4*0$$

$P_{curve}(1)=P_1*0+P_2*0+P_3*0+P_4*1$

For all other values of t the coefficients ki are computed with the following equations

$k_1(t) = (1-t)^3$

$k_2(t) = 3(1-t)^2*t$

$k_3(t) = 3(1-t)*t^2$

$k_4(t) = t^3$

In order to evaluate a point on the Bezier curve for a specific value of t we have to replace t on the equations above. Computations will provide all coefficients k_i which are then multiplied with the corresponding control points. For example, in order to create a Bezier curve as a polygon path with 5 then we must evaluate 6 points with t being incremented by 1/5. The following pseudocode demonstrates how to compute these 6 points along the Bezier curve:

```
pieces = 5;
for (i = 0; i <= pieces; ++i) {
    t = i / pieces;
    K1 = (1 - t) * (1 - t) * (1 - t);
    K2 = 3 * (1 - t) * (1 - t) * t;
    K3 = 3 * (1 - t) * t * t;
    K4 = t * t * t;
    Pt = (P1 * K1) + (P2 * K2) + (P3 * K3) + (P4 * K4);
}
```

For any change on the control points the code must be executed again.

A general and formal way to represent this method is a sum:

$$P_{curve}(t) = \sum_{i=0}^{n} b_{i,n}(t)P_i,\ t \in [0,1]$$

The coefficients $B_{i,n}$ are known as the Bernstein polynomials. They were first defined by Sergei Bernstein, a Russian mathematician, in 1910. Polynomials are expressions of finite length. They consist of variables, constants and non-negative integer exponents. Also they only use operations of addition, subtraction and, multiplication.

Bernstein polynomials can be computed with the following formula:

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \; i = 0,\ldots,n.$$

where the terms $\binom{n}{i}$ are known as binomial coefficients and can be easily computed using factorials:

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!}$$

For n = 3, the binomial coefficients are 1, 3, 3, 1.

The sum of all n+1 Bernstein polynomials of degree n is one.

2.4. Properties of Bezier curves

A Bezier curve is surrounded by the convex hull of control points. This specific property isused in computer graphics to optimize intersection tests. If convex hulls do not intersect, then the corresponding Bezier curves wont either. Checking for the convex hulls intersection first can give a faster result because they are rectangles or triangles and in general much simpler figures than the curve.

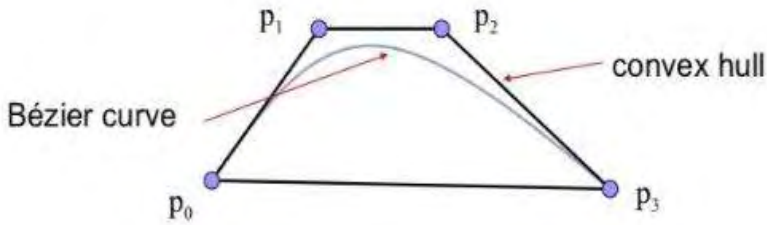

Figure 9 : Bezier curve and convex hull

In Bezier curves there is no local control. Moving a control point requires recalculation that affect the aspect of the entire curve. In addition high degree curves are computationally expensive to evaluate. The way to overcome those limitations is similar to splines mention above. By using a series of piecewise Bezier curves that are at least continuous we can easily model complex shapes with low degree curves. The computational cost to change shape is also drastically decreased due to recalculation of only the modified piece and in some occasions its neighbors. The piecewise curve is called composite Bezier curve, composite Bezier spline or polybezier. A closed path composed of Bezier curves is called beziergon or bezigon. A well-known application of composite Bezier curves is TrueType fonts, an outline font standard used by Apple and Microsoft.

Bezier curves have one more interesting property. As mentioned the first and last point of the curve are matching with the first and last control point. For the example of the cubic Bezier curve with control points P1, P2, P3, P4 the lines P1-P2 and P3-P4 are tangent to the first and last point on the curve. As a result, a transition between two Bézier curves is invisible if the line P3-P4 of first curve and P1-P2 from the second are collinear. This property can be used for splitting an existing Bezier curve in two or to concatenate existing Bezier curves together in one unified smooth curve.

This property is very useful when we want to draw a complex smooth curve consisting from several low degree piecewise Bezier curves. By ensuring that the consequent tangent are collinear we secure 1st order continuity of the combined curve. 2nd order continuity is also possible but it is more complex to implement and requires higher degree of piecewise curves.

2.5. *Bezier surfaces*

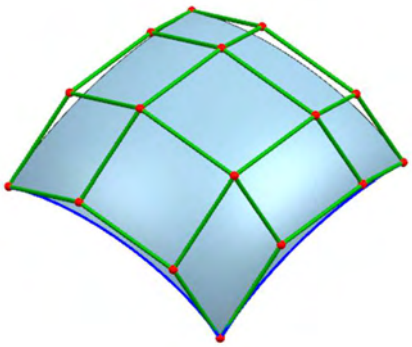

Figure 10 : Bezier Surface

Bezier surfaces are a type of mathematical spline widely used in computer graphics. Their utility was developed by French engineer Pierre Bezier in 1962 while working for Renault, for use in designing automobiles. Bezier surfaces don't have any degree limitation, but bicubic Bezier surfaces are simple, easy to compute and generally enough for most applications.

As with the Bezier curve, a Bezier surface is defined by a set of control points. Once we have a clear view on the principles of Bezier curve it is straightforward to extend the same technique to Bezier surfaces. The most commonly used Bezier surfaces are bicubic and similar to bicubic curves where we had four control points we will define the surface with sixteen points as a grid of 4x4.

Just like Bezier curves the surface does not necessarily pass through the central control points but is stretched toward them. They act as an attractive force and are visually intuitive.

While describing curves we used one parameter t to move along the curve. In the case of surfaces, we will need two. Actually there is one parameter u which draw the curve and other one v which moves the curve on 3D space, both contained within the range [0, 1]. In general a Bezier surface is a sum of many Bezier curves in parallel either looking in the u or v direction.

2.5.1. Bezier surface equation & properties

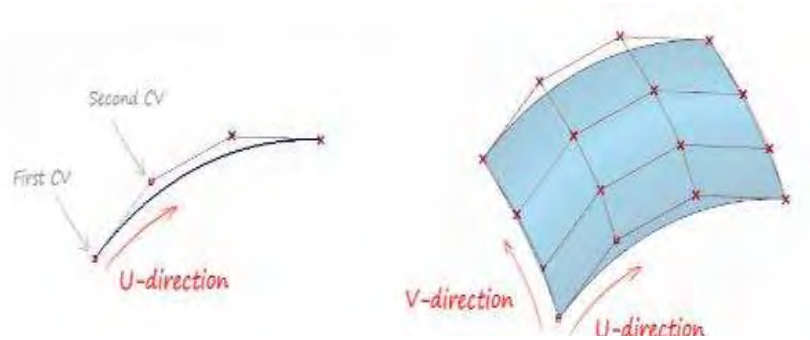

Figure 11: Bezier surface parameters

A Bezier surface (or patch) is constructed as the tensor product of two Bezier curves. The isoperimetrical curves for u = 0, u = 1 and v = 0, v = 1 are called border curves of surface. We can think the Bezier surface as the tensor product of two Bezier curves.

A Bézier surface with 2 dimensions is defined as a parametric surface. Position of any point P(u,v) on the surface is:

$$\mathbf{p}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u)\ B_j^m(v)\ \mathbf{k}_{i,j}$$

Where once more

$$B_i^n(u) = \binom{n}{i}\ u^i (1-u)^{n-i}$$

is a Bernstein polynomial, and $\binom{n}{i} = \frac{n!}{i!(n-i)!}$

$$k_1(u, t) = k_1(v, t) = (1-t)^3$$

$$k_2(u, t) = k_2(v, t) = 3(1-t)^2 * t$$

$$k_3(u, t) = k_3(v, t) = 3(1-t) * t^2$$

$$k_4(u, t) = k_4(v, t) = t^3$$

But again using this method to evaluate points on Bezier surface isn't always robust and can be slow. An easier solution is to treat each row on the control point grid as an individual Bezier curve. This way is similar to De Casteljau algorithm and is actually its adaptation for Bezier surfaces. First step is to create the n Bezier curves using only parameter u. Then for every v in [0, 1] the corresponding points on the previously generated curves will be used as control points to generate a new set of curves in v direction.

The pseudo code for this algorithm is the following:

```
point Bezier_Surface(P[16], u, v)
{
    for (i = 0; i < 4; ++i) {
       C [0] = P[i * 4];
       C [1] = P[i * 4 + 1];
       C [2] = P[i * 4 + 2];
       C [3] = P[i * 4 + 3];
       D[i] = Bezier_Curve(C, u);
    }
    return Bezier_Curve(D, v);
}
```

Properties of Bezier surfaces:

- All lines for constant u, v are Bezier curves.
- Bezier surface transforms just like its control points for all linear transformations or translations.
- Bezier surfaces are inside the convex hull of control points.
- A Bezier surface pass only through the corner control points.

In computer graphics Bezier Surfaces are used in meshes of bicubic patches where m = n = 3. They are superior to triangle meshes in representing smooth surfaces since they are easier to manipulate and they require less control points thus less memory.

Figure 12: Teapot made from Bezier patches

3. Tools

Third chapter enumerates the resources used for application development. The programming language, toolkits, development platform and required libraries will be described in order to provide a complete view of the technical details.

3.1. Programming Language-JAVA

Java is a concurrent computer programming language, object-oriented and allows creation of reusable code and. Java is easy to learn, general purpose, modular and was designed to be platform independent. It can run on any computer with no specific hardware requirements or any software dependency. Despite being relatively slower and more memory consuming than C++ the ease of use and the huge database of available libraries make Java an efficient tool in the hands of beginner and professional programmers.

3.2. JavaFX

JavaFX is a toolkit for Java used to create Graphical User Interfaces. With it we can create visually pleasing applications. It is very important in those applications to have a good structure. With a proper structure it is easier to maintain and support the application, ensure stability or add new features, very important factors in enterprise applications. JavaFX has replaced Swing, the previously recommended Java GUI toolkit, as it is more consistent and has more features. It is also more modern too and allows designing using XML layout files and CSS styling, like web applications.

JavaFX also includes built in graphics, audio, video, charts and embedded web applications and can also reference APIs from any Java library. With JavaFX any kind of application can be developed. They can be deployed across multiple platforms and display information in a high performance and GUI interface that features graphics, audio, video or even animation.

3.2.1. JavaFX Hierarchy

In this section we will see the various features in JavaFX. This step is essential in order to understand how JavaFX is used in an efficient way.

A JavaFX application can have one or more stages Each stage is a separate window and has a scene attached. Each scene can have attached to it, called the scene graph.

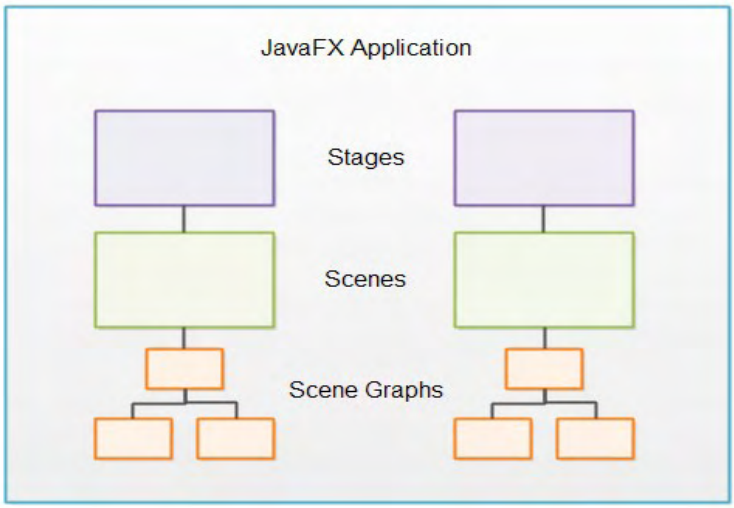

Figure 13: JavaFX application structure

- *Stage*

The outer frame and in general the application window. A multistage application means that we can have several windows open at the same time. One of the stages is the primary and first window to be opened. If additional windows are needed we can create additional Stage objects.

- Scene

A stage needs a scene in order to display something. Only one scene can be shown in the stage but different scenes can be exchanged and shown one at a time. To understand why mane scenes are needed we can think of a simple computer game. First scene would be the menu, another for the game screen and another for the high scores. Every scene is represented by a Scene object. JavaFX applications must instantiate all Scene objects they need.

- Scene Graph

The collection of visible objects attached to a scene is called scene Graph. It consists of several nodes.

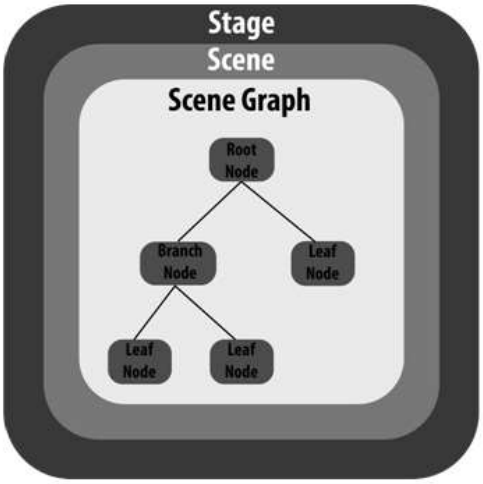

Figure 14: Scene graph

3.3. Tools used in project

This section will provide all necessary information to reproduce the implemented project from scratch. The versions of the above tools that were used will be listed and the already existing mathematical and 3D libraries will be provided.

The whole project is based on Java 8 with latest Java Development Kit verified the JDK 8u181. Further updates on Java 8 should function properly but for the exact same project recreation the update 181 is recommended

The IDE used for development is Netbeans with last version used the 8.2. Using a newer version will require an update on project files that could cause unknown incompatibilities.

JavaFX is included in the JDK so ensuring the JDK version is right will also provide the correct JavaFX bindings.

Apart from the core installation a mathematical library needs to be included since linear algebra tools are required on almost every algorithm demonstrated. For the specific case the common-math-2.2 was used and can be acquired from Apache commons repository.

Third party JavaFx classes for handling 3D graphics are directly inserted in source files and can be found in appendix or through project's online repository.

If all the above tool versions are correct or compatible importing the existing project or creating a new and copying the source files will lead to an exact regeneration of the developed application.

4. Implementation

This chapter presents the implemented JavaFX application. Every part of the design and algorithms used for every scene will be described in depth. To provide a better viewpoint on the hierarchical structure chapter four concludes with a class diagram tagged by JavaFX element definition.

By navigating through the application users have the opportunity to witness the basic concepts of Bezier curves and surfaces. By presenting a set of different approaches it provides not only an understanding of the mechanics and algorithms, but also the importance of their usage for vector graphics and animation. In the current version those approaches are split into five independent subtasks.

4.1. Primary stage - Menu

As described in Section 2.2.1 every JavaFX application follows a hierarchy based on stages and scenes. The application runs in a single window frame thus only one stage is created. This primary stage is actually a combination of two elements, a tab selection menu and an area where the actual scene is enclosed.

JavaFX provides easy access to some of the more frequently used menu structures and the tab menu came in handy for organizing all the tasks in a single window frame. Every tab contains a separate scene and a unique control bar which offers the ability to parameterize the inputs, save the output or interact at runtime.

The standalone tasks are represented as different scenes with only one displayed at a time. Despite the fact that only one scene is visible, all scenes are executed concurrently, meaning that switching tasks retains all user activity by keeping them active in background.

4.2. Scene One – Bezier Curves

First and also default tab contains the Bezier Curve sub-application. Primary objective is to demonstrate a step by step creation of a Bezier curve. The number and position of control points are defined by user and can be added, removed or modified at runtime. Apart from curve visualization, every layer of inner control points and lines are also generated and can be optionally displayed. The scene itself consists of a multilayer 2D drawing canvas and a settings bar for user interaction.

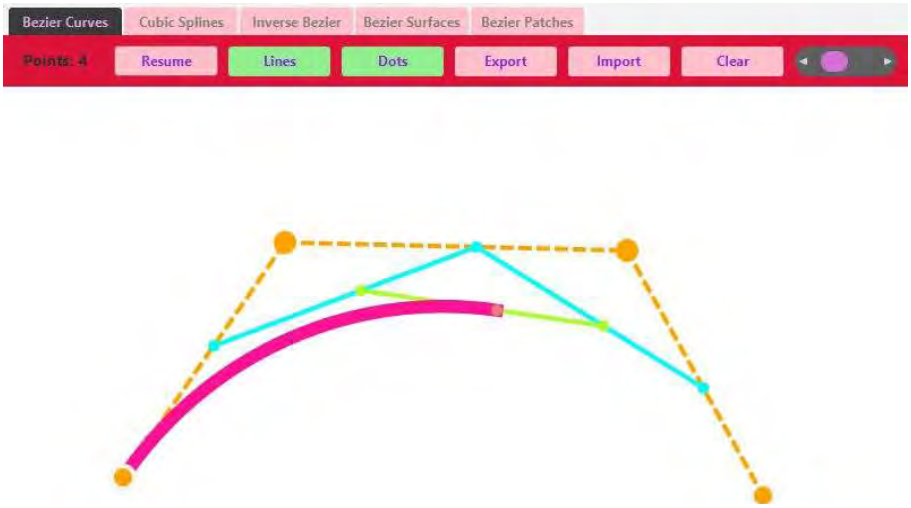

Figure 15: Scene 1 - Bezier Curves

4.2.1. Description

The Bezier Curve scene starts with a blank 2D space. In order for any process to start user must first select at least 2 points. Input is given with mouse in an interactive way where points can be added removed or relocated. Selected points are displayed as connected dots, with an extra bold stroke on first and last one to highlight the Bezier curve's endings. An AnimationTimer, a JavaFX tool for real time animation, is used to provide a smooth frame by frame display of the curve rendering. Each frame causes a slight increase on the parametric variable t and de Casteljau algorithm calculates the next curve's point. By default assisting lines and dots based on the geometric representation are

visible, aiming for the better understanding of the curves creation. Also for the same cause animation can be paused and its speed can be adjusted. Deleting control points will instantly recalculate the curve's correct path, while dragging one will create the illusion of a smooth curve movement following the control point.

An extended usage of action listeners, simplified by Java 8's lamda expressions provided all the necessary user interaction. User input source is mainly mouse movement and buttons, thus providing a compatible input method for touch screens and hoping for a smooth future porting into android application.

4.2.2. Development

Development of the Bezier Curve Scene was split in three phases. General appearance, algorithmic calculations and user interaction. This also separated the JavaFX routines and objects used from the mathematical implementation of the de Casteljau algorithm.

First phase begun with a static analysis of every possibly needed shape and data structure. The selected data set and their format was the result of numerous approaches and tryouts. Since the main purpose was not the optimal fastest implementation but a user friendly smooth experience lot of attention was given in accurate visualization and css styling. What is seen in this scene is actually a collection of JavaFX shapes, mainly circles and lines, added in a Pane called desktop.

Shapes used where split into groups in order to properly separate visible layers. This layer separation was not only a visual improvement, but also necessary for proper user interactions, since objects selected by mouse are always the ones on top.

Next development phase focused on the mathematical calculations beneath curve generation. JavaFX can automatically create Bezier curves of three or four control points but in our case this was insufficient. In order to support any number of control points the de Casteljau algorithm was implemented. As mentioned above a JavaFX AnimationTimer gives the pace for the animation's frame rate, and for every frame a point on the curve is rendered. There is no upper limit for the number of points since all data is stored in ArrayLists, Java's dynamically allocated arrays. In total three sets of two dimension

arraylists are allocated. First one and also most important is dedicated to the control points, including all the secondary assisting points that emerge while executing de Casteljau algorithm. Those correspond to the center of the previous level of control points and are relocated at run time. All the computations are just moving those assisting control points. The other two arraylists have a similar structure and are used for the optional visualization of dots and lines. They contain their position and are also relocated for each value of t according to the current positions of assisting control points.

Finally, apart from control point handling, user can via a set of buttons administer the visual layers, clear screen, halt the animation and adjust drawing speed. Users can also save the current collection of control points on computers' file system.

4.3. Scene Two – Cubic Splines

The second scene generates at the same time a piecewise Bezier curve and cubic splines as reference. Through this process users can demonstrate the impact control points have in curve's continuity and its application in vector graphics.

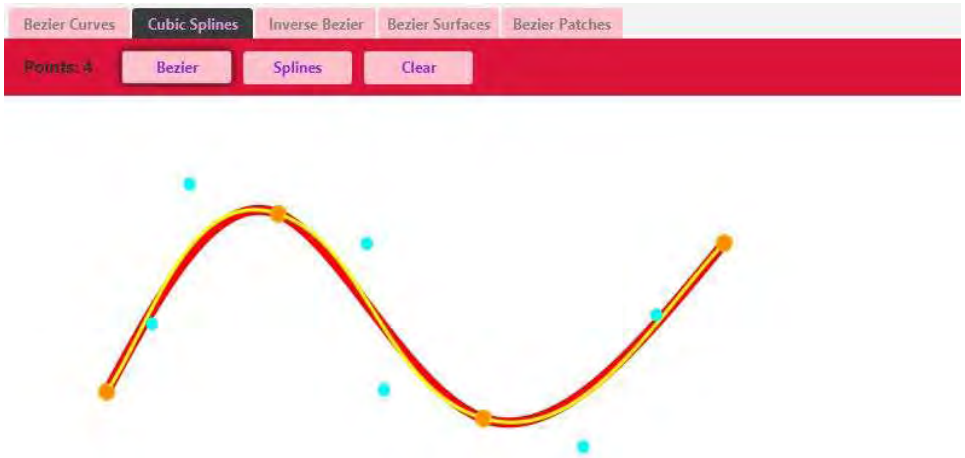

Figure 16: Scene 2 - Cubic Splines

Furthermore we come across an extra benefit of using Bezier curves while control points are not constant. With spline interpolation any change on control points would require a complete curve recalculation from end to end. On the other hand he corresponding piecewise Bezier curve has to redraw only the neighbor pieces of the affected control point.

4.3.1. Description

Once more the scene begins with a blank 2D canvas. Mouse interaction is needed to assign the locations of control points. The points selected will be drawn as orange dots. Between every selection two additional equidistant points will be automatically drawn as smaller blue dots. Three buttons are available for users and can be used with any amount of control points.

Splines: Runs a cubic spline parametric algorithm to generate a curve passing through all the orange control points.

Bezier: Draws a piecewise curve consisting of several cubic Bezier curves. The final curve is smooth and will be passing from all orange control points. Blue points will be relocated automatically so that all pieces fit in the equivalent part of cubic spline.

Clear: Resets the canvas, clears allocated space and awaits new user input.

Even after the curves are drawn new points can be added. Control points can be relocated with mouse and new curves can be redrawn based on the updated information. The generated Bezier curve will always be the best match of the reference cubic spline. By moving any control points users can modify the curve and demonstrate how it changes according to their position. Moving the blue control points trigger a smoothing function that ensures continuity.

Within this scene Bezier curves show their advantages in comparison to spline interpolation.

1. Splines are calculated according to the points the pass through without any way for local adjustments on the final curve. On the other hand Bezier pieces have the internal assisting control points as blue dots that only serve as corrections without affecting the curve base route.

2. Moving a control point require a complete recalculation of the whole curve with spline interpolation where piecewise Bezier pieces only redraw the 2 pieces that are directly affected by the change.
3. Scalability not an issue for Bezier curves where the increase in control points creates a linear increase in computation cost.
4. Piecewise Bezier offers a designer friendly way to modify the curve, with the option to either retain a smooth path by ensuring continuity, or create local edges on demand.

4.3.2. Development

Unlike the previous scene where the focus was on a geometrical representation and real time movement this time we focus in mathematic algorithm adaptation.

For cubic splines a library for linear algebra is used to perform LU decomposition. The algorithm for parametric spline interpolation is described in Section 1.2.2. Separately for coordinates x and y the tridiagonial system is populated and solved. The generated cubic spline is drawn as a bold red curve. Both user input and generated data are stored in arrays till reset with the clear button. The storage this time is not dynamic and theoretically infinite as in the first scene with Arraylists. But on the other hand the calculations are faster and high precision arithmetic can be used.

Piecewise Bezier follows a more complex approach. The process makes use of data generated from spline interpolation. For every piece inverse Bezier algorithm is run to generate a local fit on reference curve. The inverse algorithm is also the main focus of the next scene so it will be described on next section. The final curve illustrated as a yellow line passes from all user given control points.

The output of Bezier fitting algorithm creates the complete path as with every piece being a cubic Bezier that perfectly matches the reference spline part but there is a missing and very important step. By only doing this we cannot guarantee 1st degree continuity at the connecting points of two consecutive pieces. In order to fix continuity but preserve a good fit an extra algorithm was necessary.

As known from theory the imaginary straight line connecting the end point of Bezier curve and the previous control point is also the tangent at that endpoint. So in order to have 1st degree continuity between two consecutive Bezier pieces the connecting point, previous and next control point must be all on the same imaginary straight line.

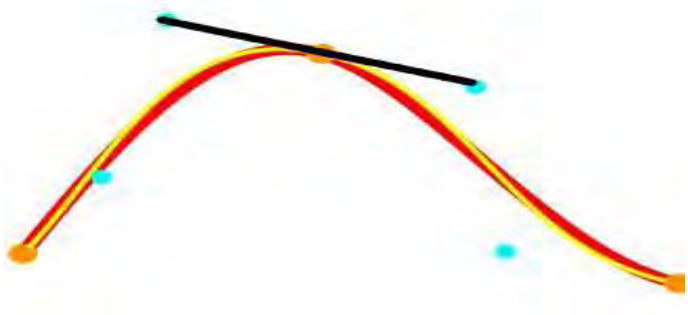

Figure 17: continuity between bezier pieces

In case that the three points are not aligned the blue control points have to be relocated. To achieve minimum impact on the curve the relocation is split evenly for the two blue dots. It would be as if both are rotating with same angular speed with orange point as center and on opposite direction till they are aligned. They also adjust their radius on their average. After this algorithm is run on all user defined control points the final piecewise Bezier curve is ready, smooth and a close match to the cubic splines used as reference.

This extra algorithm is executed for all points as a last step after piecewise Bezier initial creation. After that any modification on control points require a re-execution only on the affected point. The computation cost is very low and again linear to the affected points.

To demonstrate both the ability to retain 1st degree continuity but also the creation of local edges on demand the trigger of the smoothing function described above is not always enabled. By default it is executed on initial Bezier creation for all points, and upon relocation of control points it is only triggered by the first blue point per cubic Bezier piece.

4.4. Scene Three – Inverse Bezier

In the first scene we witnessed how given any number of control point we can generate a Bezier curve. But what if we need the exact opposite. Suppose we have a known curve path, and we want to calculate which control points could create a Bezier curve as close to the given one as possible. This task is demonstrated in the third scene, where users are called to provide the curve path dynamically and the curve's control points are automatically calculated.

4.4.1. Description

Following the same approach with previous scenes we start again with a blank 2D canvas. A dynamic path can be drawn with mouse interaction by clicking and dragging inside the drawing area. Upon mouse button release the calculation process begins, in order to create a Bezier curve in the specified path.

The path drawn by user is a bold green line. End points and two extra assisting points on path are also highlighted. On top of the original path the generated Bezier curve will be drawn with purple color and two extra red dots as the estimated control points. Upon new mouse click everything is reset and a new path can be redrawn instantly.

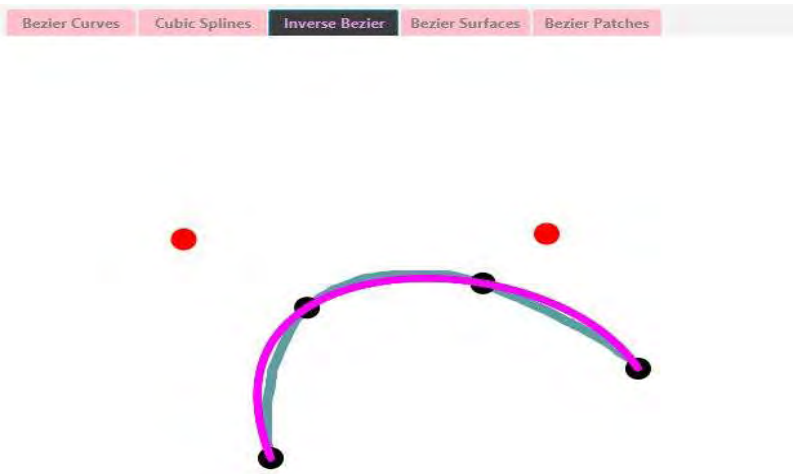

Figure 18: Scene 4 - Inverse Bezier

Default number of control points is set to be four (cubic Bezier curve) thus the generated curve can have up to one turning point. More complex curves could be calculated with either higher number of control points, or by dividing the path into smaller simpler curves.

4.4.2. Development

When user is called to draw a curve every point is stored but only four will be used for the actual calculations. Those four are the ones highlighted with dark blue color. Using the parametric equation to calculate Bezier curves we know that the first point is the result for t = 0, second for t=0.33, third for t=0.66 and the last for t=1.

With the four equations we form a matrix and solve for the coordinates of the red control points. Again java libraries for linear algebra are used for LU decomposition.

Now that we have all four points an algorithm similar to first scene's generates the cubic Bezier curve on top of the given path. When a new path is redrawn all previous data are cleared.

4.5. Scene Four – Bezier Surfaces

The forth Scene is a demonstration of the most commonly used Bezier surface, bicubic patch, in 3D space. It is defined by sixteen control points and can be freely modified and regenerated in real time. Users can freely rotate, zoom, and manipulate the surface making full use of JavaFX 3D capabilities.

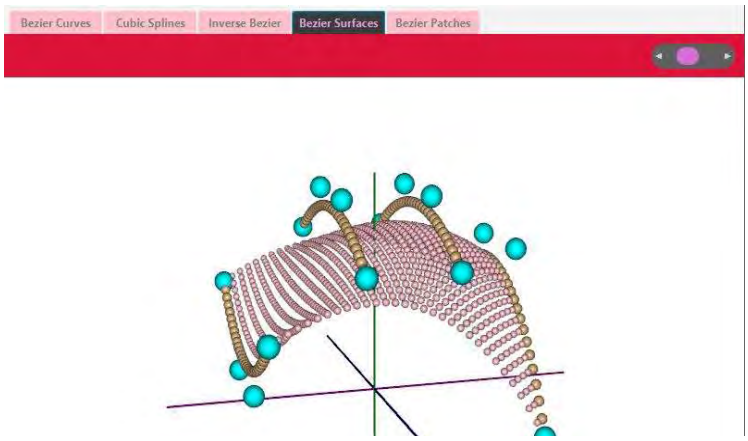

Figure 19: Scene 4 - Bezier Surfaces

4.5.1. Description

Starting in a 3D space with predefined control points a surface is built with adjustable density. Users have freedom over camera angle and distance and can freely modify the surface by moving the control points. Every change triggers a total redraw of the surface thus giving the feeling of surface following the control points at real time. The density is not only for visual interpretation but also defines the steps of parameter t while executing.

De Casteljau algorithm in both directions u and v. Increasing density will actually fill the surface but could be slow on a weak machine. The control points appear in space as light blue spheres, and the surface also consists of many smaller pink spheres.

4.5.2. Development

For representation in 3D space JavaFX's camera and 3D libraries are used. As described in chapter 2.2.2 camera perspective was added in order to replicate a 3D world. Afterwards the sixteen control points where added in predefined coordinates and were drawn as blues spheres in the scene. The next step follows is to calculate the actual surface with the surface version of de Casteljau algorithm as described in Section 1.4.1. The algorithm is running for two parametric values u and v. The first pass creates the pseudo control points that are illustrated as spheres with light brown color. Based on those control points the second pass will create the actual surface consisting of many small pink spheres. The Density is adjustable but greatly affect computation speed.

Apart from control points that are stored in a dedicated array all other object are dynamically created and drawn when instantiated in the 3D canvas. The grid of 4x4 control points has predefined values but any point can be moved with mouse interaction. Mouse over a point highlights and enlarge it in order to assist users. While dragging a control point real time calculations will redesign the surface.

4.6. Scene Five – Bezier Patches

Following the same pattern of previous section this final scene demonstrates an actual application of Bezier surfaces in computer graphics. Several bicubic surfaces (4x4) are combined together to form a bigger surface. This Scene demonstrates both the manipulation of 3D surface and application of an algorithm to maintain continuity between neighboring patches. Any change on the internal control points will automatically relocate the internal control points of neighboring patches. This technique can be used in tool for designing smooth 3D objects where small local changes will not break the overall polished surface.

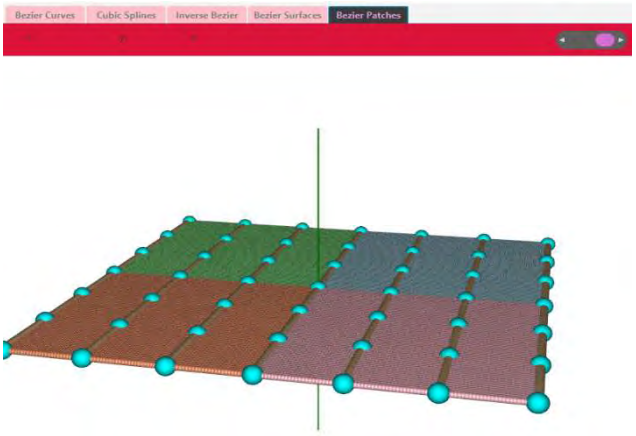

Figure 20: Scene 5 - Bezier Patches

4.6.1. Description

Starting in 3D space a Bezier surface is predefined as combination of four smaller bicubic (4x4) surfaces. From control perspective everything is almost same as in previous scene. All light blue spheres are the control points and can be moved with mouse. The different bicubic surfaces are illustrated with different colors.

Control points are split in two categories. The outer ten points on the edges of a patch, which are directly shared with neighbor patches, and the four internal ones. On all those sixteen control points apart from their job to generate the Bezier patch also a simple algorithm applied in order to maintain 1st degree continuity between neighbors.

4.6.2. Development

The mechanism behind this scene is very similar to Surface scene. This time all calculations are run for each 4x4 grid separately. A subtask that ensures 1st degree continuity on consecutive surfaces will force their connecting edge to be smooth.

5. Conclusion

In the present thesis an educational tool for the learning of the Bezier and splines curves was developed. By means of this tool it can be achieved the full understanding of the curves based on their characteristics and their properties. The user, even knows the meaning of the curves, or tries to understand them by using this application, has the possibility to create his own examples. The interactive way which is used in this application can make the learning and the understanding of the mathematical concepts more pleasant. The Bezier curves, splines and the surfaces form an important factor in the design and display of graphics.

The basic concepts of Bezier curves and surfaces with focus on their geometrical representation could be accessed through a series of stand-alone tasks. The interactive nature of the application creates a gamified version of an educational tool. It offers knowledge through fun and hides complicated algorithms in easy to understand animations. A first step for a useful and practical educational tool for Bezier curve studying.

The tool is important because it offers the possibility to the user to interfere in the curve that has created and change it even by transferring the control points or adding and eliminating other control points. With this feature the user understands instantly the way that curves are created and designed.

In the future this tool could be enriched with other functionalities based on the existing curves or even to exist the possibility to design more formats of curves and surfaces. Furthermore, they could be added some tabs where the user can test if has understood correctly the properties of the curves. Meaning to recognize what type of curve is the one that he created or even to press a button to design a random curve and find its type that was used for the design.

6. Bibliography

1. https://en.wikipedia.org/wiki/B%C3%A9zier_curve
2. https://en.wikipedia.org/wiki/B%C3%A9zier_surface
3. http://www.e-artouche.ch
4. https://pomax.github.io/bezierinfo/
5. http://mathworld.wolfram.com/CubicSpline.html
6. http://fourier.eng.hmc.edu/e176/lectures/ch7/node6.html
7. Computer-Aided Design Volume 21, Issue 4, May 1989, Pages 194-200
   GC1 continuity conditions between adjacent rectangular and triangular Bézier surface patches
8. Curves and Surfaces for Computer-Aided Geometric Design A Practical Guide Book 3rd Edition 1993 , Gerald Farin
9. Graphical Models and Image Processing Volume 58, Issue 3, May 1996, Pages 223-232
   Regular Article : Curve Fitting with Bézier Cubics

Appendix

UML