**Computer Graphics: Parametric Curves Website**

**By Shani Bar-Gera**

**Supervisors: Boaz Sternfeld**

Summer 2021

**Table of Contents**

Introduction………………………..………………….………...……………………..…...3

Website Overview……………………………………….....……....……...………...........4

Technologies and Platforms…………………………………...………………….….….15

Development Process…………………………………………….……...……............…17

Future Work………………………………………………………………..…………....….32

Conclusion……………………………………………………………....……………….....34

References……………………………………………………………...……………...…..35

**Introduction**

The goal of the project was to create a website that enables to portray different parametric curves learned in the computer graphics course easily and in a way that makes it easier to learn them.

The user should be able show 6 different curves learned in computer graphics class:

* Monomial Basis
* Lagrange
* Cubic Hermite Spline
* Cubic Spline
* Bezier
* B-Spline

The user shuld be able to:

* Choose which curve to show and play with it’s control points by moving, adding and deleting them
* See the equation of the chosen curve and change it’s paramteres
* Split the screen in order to show multiple curves at once and allow the user to compare between them

I decided to achieve this goal by creating a website using JavaScript that is based on a former website and adjusted it to the new demands.

**Website Overview**

1. Opening Page

Graphical user interface, text

Description automatically generated

1. A screenshot of a computer

   Description automatically generated with medium confidenceSelect curve - the user can choose a curve from the curve menu and click it for it to drag and drop it to the main box.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

1. Play with control points – move control points by dragging them, add control point by clicking on the desired playce on screen it should be added, delete a control point by clicking on it

Graphical user interface, chart

Description automatically generated

1. Control curve’s equation variables:
   * A screenshot of a computer

     Description automatically generated with medium confidencen: number of control points
   * k: orderGraphical user interface

     Description automatically generated
   * steps: number of points to draw to make curve

Graphical user interface, chart

Description automatically generated

* + t: show a specific point

A screenshot of a computer

Description automatically generated with medium confidence

* + Parametrization – change the number of points that build the curve between every two control points.

Initialy the points are distributed equaly as seen in the example, between P0 and P1 there are 4/8 points and between P1 and P2 there are 4/8 points. In the parametrization section t1 is exactly in the middle between t0 and t2 and there are 4 points on each side therefore will be 4 points between P0 and P1 and 4 points between P1 and P2.

Graphical user interface, chart

Description automatically generatedGraphical user interface

Description automatically generated

After moving t1 in the parametrization section the points are distributed differently. In the example t1 was moved closer to t0 so there will be only 2 points between t1 and t0 and 6 points between t1 and t2 so therefore there will be 2 points between P0 and P1 and 6 points between P1 and P0 .

Graphical user interface

Description automatically generated

1. Graphical user interface, chart

   Description automatically generatedLearn more information about the curve’s equation and parametrization by clicking the information button
2. Split the screen to show multiple curves at once and then drag and drop a curve from the curve menu to the desired section of the screen

Change the number of rows and cloumns to split the screen.

Text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

drag and drop a curve from the curve menu to the desired section of the screen at the example we are trying to put monomial basis curve to the top left scrtion of the screen

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

All the screen divides filled with curves:

Graphical user interface

Description automatically generated  
  
It is also possible to drop a new curve on a square that already has a curve in it and by that replacing the old content. In the example Bezier curve was replaced with cubic spline.

Graphical user interface, chart

Description automatically generated

1. Use “Clear All” button to clear all curves from screen and go back to starting point.

Graphical user interface, text

Description automatically generated

**Technologies and Platforms**

A picture containing text, first-aid kit, sign

Description automatically generated

* **HTML**

HTML is short for HyperText Markup Language which is the standard markup language for documents designed to be displayed in a web browser and describes the structure of a web page. The HTML elements are the building blocks of HTML pages. The elements are delineated by tags, written using angle brackets for example <img></img> provides an image element. The curves are displayed in a <canvas> element. Browsers do not display the HTML tags, but use them to interpret the content of the page.

The main page of the project’s website is written in HTML. It is the page that provides the website’s layout and embeds the rest of code section in CSS and JavaScript.

Icon

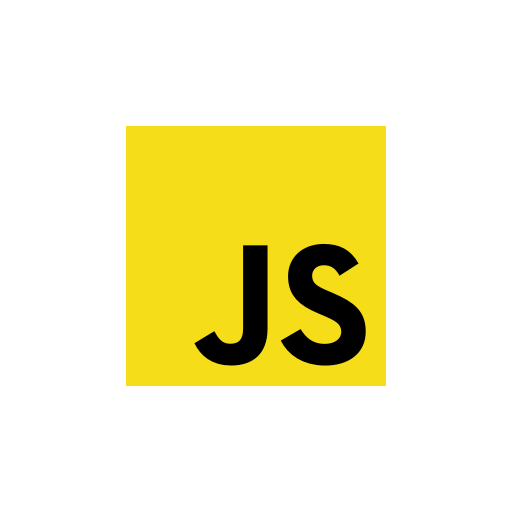
Description automatically generated

* **CSS**

CSS is short for Cascading Style Sheets and is a style sheet language used for describing the presentation of a document written in a markup language such as HTML and other markup languages. It is also a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colours, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, which reduces complexity and repetition in the structural content and enable the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods.

* **JavaScript**

JavaScript (often shortened to JS) is a lightweight, interpreted, object-oriented language with first-class functions, and is best known as the scripting language for Web pages, but it's used in many non-browser environments as well. It is a prototype-based, multi-paradigm scripting language that is dynamic, and supports object-oriented, imperative, and functional programming styles.

JavaScript runs on the client side of the web, which can be used to design / program how the web pages behave on the occurrence of an event.

It can function as both a procedural and an object oriented language. Objects are created programmatically, by attaching methods and properties to otherwise empty objects at run time, as opposed to the syntactic class definitions common in compiled languages like C++ and Java. Once an object has been constructed it can be used as a blueprint (or prototype) for creating similar objects.

JavaScript's dynamic capabilities include runtime object construction, variable parameter lists, function variables, dynamic script creation, object introspection, and source code recovery (JavaScript programs can decompile function bodies back into their source text).

Most of the projects code was written in JavaScript. Most important functions that were done in JavaScript are the dynmaic splitting of the screen and the drawing of the curves on the screen.

Icon

Description automatically generated

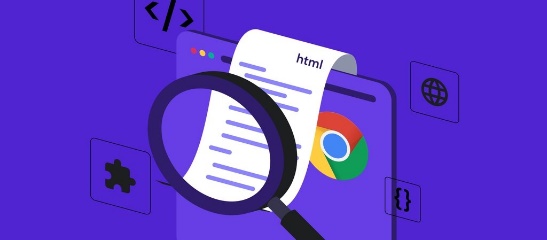
* **Visual Studio Code**

Visual Studio Code is a source-code editor made by Microsoft for Windows, Linux and macOS. It has feature that support debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. It was the working envoirment that I chose to write the code of the project.

* **Github**

GitHub is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management (SCM) functionality of Git, plus its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and wikis for every project. I used git for my project to create a backup to the layers of changes and in order to see clearly what were the changes that I made each time.

* **Chrome Inspect Tool**



A built in tool in Google Chrome web browser that is designed to help web developers check and edit the HTML, CSS and JS of the page displayed in the browser. Also allows to use a console that can get logging from the code and show errors. Similar tools are availabe in other browsers. This tool was very beneficial to test and debug my project.

**The Development Process**

In order to create the website, we separated our development process into 5 main parts:

1. Designing the website’s page layout
2. Choosing existing code to base the website on
3. Adjusting the existing code to fit the page layout
4. Adding the different curves algorithms to the code
5. Improving UI

I will now talk about each stage, my thought process, challenges and resolutions from each part.

**Designing the Layout**

Firstly I had to learn how to build a website. I used w3school as a tutorial and various YouTube toturials. Then I wanted to decide how the website will look like. I was considering different possible layouts.

The first offer was this:

Text

Description automatically generated with low confidence

In this option I planed to have dropdown windows that each will have a curve to play with. Making this design made me learn a lot about using HTML and CSS.

But after discussing it with my supervisor we decided to go for something that looked like this:

Diagram

Description automatically generated

The new design was intended to look more like a graphing tool and less like a slideshow, maximized the browser’s space better to display the curves on most of it and allowed to split the screen and compare different curves with one another.

I began learning how to create a basic layout using HTML, creating the header, a side navigation pane, a footer, and the main section. Later I added buttons to the navigation panes for the different curves and a button to split the screen. I also experimented with the CSS files trying to decide what will be the formatting and color scheme of my website. It looked something like this:

Graphical user interface, text

Description automatically generated

After creating a basic layout, I tried to understand how to add a split screen option. At first, I thought to dynamically create <div>s in my main section and set them with a specific size and location. For that I had to start learning some JavaScript as well. After searching for examples online I learned about table element that I fulfilled my requirements.

I created my first experiment with it.

Initial layout:

Graphical user interface, text

Description automatically generated

After Clicking the button:

Text

Description automatically generated

After it worked I tried to add it to the main div of my layout and change the styling.

This is how I set the initial table to look like:

A screenshot of a computer

Description automatically generated with medium confidence

Calendar

Description automatically generated with medium confidence

After I succeeded to add the table, I wanted to be able to drag and drop an item from the curve menu to the desired table cell. In order to do that I had to learn how to make items draggable and how to listen to events in JavaScript. After many attempts I finally succeeded to make the create the dragging and dropping option and transferring the data from the button text to the table cell I got something like this:

A screenshot of a computer

Description automatically generated with medium confidence

This was acceptable for the meantime, but I was unsatisfied with how the table’s cells fit their size to the content and not the other way around. After spending a lot of time trying to understand how to stop this at the end, I dropped this and moved on to other things only to comeback to it later. In the end what that fixed this issue was adding an empty canvas element to the cells by default and replace it with a new one when a curve is being dropped.

At this point I marked making the layout a success, learning in the process a lot about HTML, CSS and the JavaScript table element and how to dynamically change it’s grid using JavaScript. I now decided to try and understand how to use the canvas element and how to display the curve in each table cell.

**Designing the Curve’s Display**

I wanted to get a uniform template on how I would like to display all the different curves that I could drop in my table cells. For that I reviewed many different websites the show parametric curves in different ways.

The first website I reviewed:

[Interpolating Lagrange curve (ibiblio.org)](https://www.ibiblio.org/e-notes/Splines/lagrange.html)

Where it looked like this:

A picture containing chart

Description automatically generated

I was not very pleased with this website since its code was not very organized and very hard to understand. In addition, there were many glitches while playing with the control points, the experience was not very smooth. Also, I did not like how the information was organized. I was very hard to understand the color coding, what information refers to which curve, etc.

I ended up not using this websites code as a skeleton but I did get some inspiration from it on how to display the parametrization curves in my website.

The second website I reviewed:

[General Cubic Hermite Spline Demo (codepen.io)](https://codepen.io/liorda/pen/KMvBwM)

Which looked like this:

Graphical user interface

Description automatically generated

I had some issues with running the code of this website outside of the demo and in addition also had a hard time to understand how the tangents affect the curve from moving them. I figured that if I had a hard time probably other students would as well, so I decided not to go with this display. But I did get the inspiration from this website to use labels for my curves control points as well.

The third website I reviewed:

[GitHub - ryanmid/bezier-curves: A simple Bezier curve Javascript (ES5) Demo](https://github.com/ryanmid/bezier-curves)

That looked like this:

Chart, line chart

Description automatically generated

I was very pleased with this website, the git code was running smoothly, the code was very organized and easy to understand with a lot of documentation and generic functions. Also, it already had access to button and control points. In addition, I liked that it drew the dots showing me how the curve was drawn. I felt like this was very helpful in understanding how the curve was built. I decided to try and merge this code with my initial layout in hope that if this succeed, I could easily add more algorithms of the other curves.

**Merging Layout with GitHub Code**

Firstly, I wanted to add input to my layout that the Bezier GitHub code could get input from. After learning how HTML input works and how to create it in JavaScript (I wanted to create in JavaScript since I wanted it created dynamically depends on the given curve).

After learning about inputs and attempting to implement them my website looked like this:

Graphical user interface, application

Description automatically generated

I felt like this was good enough for the mean time and will format it later to look better. My next step was to try and add to each table cell a canvas with a unique ID so the Bezier app I got from GitHub could write into it. In the original website the canvas was created through the HTML file and was with a fixed width and height. For my needs this did not fit since I wanted screens that change their size by input, so I wanted canvases that fit their size to the table cells (the “screens”). This was an issue I had for a very long time and had a hard time to fit the canvas to the table cells that change sizes. The problem was the cells changed their size dynamically to fit to the content. I wanted the other way around; I wanted the content to fit to the table’s cells. In addition, the I made the input bar and header of the curve fit to content as well which also caused issues in understanding the canvas’ size. After a lot of issues, I decided to calculate manually the size I desire for the cells, set a fixed height for the curve’s header and input bar and with that I was able to calculate the size I wanted for the canvas. This still caused a small issue when trying to split the table dynamically when there are still content in it. I decided to solve this by removing from the user this option and created a screen control that clears the screen between table grid changes. The new menu looked like this:

Text

Description automatically generated

After being able to fit the canvas’ to the table cell I began to try and connect the Bezier app to my canvas and inputs.

At the end of a long learning process on how the GitHub code worked I succeeded to portray it in my table cell:

A screenshot of a computer

Description automatically generated with medium confidence

After successfully showing the Bezier curve I wanted to see if can add different curves to this GitHub code. I implemented an easy linear line curve that simply connect the first and last control point and tried by a click of a button from the curves menu to switch between the Bezier curve and the basic linear curve.

After succeeding this is what it looked like:

A screenshot of a computer

Description automatically generated with medium confidence

An issue I had with the inputs was to connect the correct input the the correct canvas. I solved this by using Ids for all my elements where the ID startes with the location of the cell its is for example “t\_0\_0\_canvas” or “t\_0\_1\_header”.

**Creating All the Curves**

My next step was to go thorough each of the curves in the geometric modeling lecture understand it better and implement its algorithm.

The code iterates on the steps and for each step it calls the relevant curve to interpolate the coordinates of the point in the specific step. After that we simply draw a straight line between all the points. The more steps the more accurate the curve will be.

I created a base class called Curve that all the other curves with inherit from it. In each step a Curve class is called to interpolate its coordinates. I wanted to save a member called point for the coordinates and a member called base the save the coordinates of all the sub curves coordinates of the current t (current step).

Lagrange Curve

I started with the easiest algorithm which was the Lagrange using this equation:

C

The base member remembered for t all the and .

Chart

Description automatically generated

In the picture above we can see for t=0.16 ( 8/50), n=4 in the main curve the white point marked and in the parametrization section 4 points marked on 4 different curves, each curve represents a specific . It’s important to mention that the order of the Lagrange curve is set by n so for n=4 this is cubic Lagrange.

I also colored the control points of the main curve with the color of the matching sub curve in the parametrization section. In other words and are in the same color.

Cubic Spline

The equation for cubic spline is:

I got the base functions for the lecture and created with them the sub curve for the parametrization:

**A screenshot of a computer

Description automatically generated with low confidence**

In order to calculate the Ks I first attempted to use a code snippet example from the internet 3:

CSPL.**getNaturalKs** = **function(**xs, ys, ks**)** // in x values, in y values, out k values

**{**

**var** n = xs.length-1;

**var** A = CSPL.\_gaussJ.zerosMat**(**n+1, n+2**)**;

**for(var** i=1; i<n; i++**)** // rows

**{**

A**[**i**][**i-1**]** = 1/**(**xs**[**i**]** - xs**[**i-1**])**;

A**[**i**][**i **]** = 2 \* **(**1/**(**xs**[**i**]** - xs**[**i-1**])** + 1/**(**xs**[**i+1**]** - xs**[**i**]))** ;

A**[**i**][**i+1**]** = 1/**(**xs**[**i+1**]** - xs**[**i**])**;

A**[**i**][**n+1**]** = 3\* **(** **(**ys**[**i**]**-ys**[**i-1**])**/ **((**xs**[**i**]** - xs**[**i-1**])**\***(**xs**[**i**]** - xs**[**i-1**]))**

+ **(**ys**[**i+1**]**-ys**[**i**])**/ **((**xs**[**i+1**]** - xs**[**i**])**\***(**xs**[**i+1**]** - xs**[**i**]))** **)**;

**}**

A**[**0**][**0 **]** = 2/**(**xs**[**1**]** - xs**[**0**])**;

A**[**0**][**1 **]** = 1/**(**xs**[**1**]** - xs**[**0**])**;

A**[**0**][**n+1**]** = 3 \* **(**ys**[**1**]** - ys**[**0**])** / **((**xs**[**1**]**-xs**[**0**])**\***(**xs**[**1**]**-xs**[**0**]))**;

A**[**n**][**n-1**]** = 1/**(**xs**[**n**]** - xs**[**n-1**])**;

A**[**n**][**n **]** = 2/**(**xs**[**n**]** - xs**[**n-1**])**;

A**[**n**][**n+1**]** = 3 \* **(**ys**[**n**]** - ys**[**n-1**])** / **((**xs**[**n**]**-xs**[**n-1**])**\***(**xs**[**n**]**-xs**[**n-1**]))**;

CSPL.\_gaussJ.solve**(**A, ks**)**;

**}**

The problem with this code was that it relays on the xs and the ys to be ordered. I was not interested in that, so I found a new calculation for them:

**,**

And got a curve looking like this:

Shape

Description automatically generated

Cubic Hermite Spline

After creating the Cubic Spline, I wanted to create the Cubic Hermite Spline. I Could have easily used the same functions I used for Cubic Spline but instead of calculating the Ks I wanted to get them as input. I viewed how it was done in the Hermite spline demo I mentioned before 12. I didn’t really like how it was portrayed so I decided instead to create control lines for each control point and had to add to the GitHub code a way to control them. I learned a lot about accessing the mouse movement and controlling the points through the mouse in the process.

Chart

Description automatically generated

In the picture above we can see that the control point has a light blue control line connected to it that can be control either by point a or point b.

B-Spline

I took the equation for B-Spline from the lecture slide:

C

I had a few issues with the size of the control points array of the parametrization. It was to small for the recursion to occur so I had to add buffers from both ends in the size of k so the extended parametrization control points array looked like this:

And then got and output looking like this:

Chart, line chart

Description automatically generated

Similarly to the Lagrange in this curve I also color coded the control points to match the matching sub curve ( only that here there is a buffer so the first k sub-curves are in the same color as and the last k sub-curves are in the same color as .

Monomial Basis

A curve with a monomial basis gets an equation that looks like this:

C

In order that C(t) will be a parametric curve we need to have a continuity of There are many options to decide how to get the coefficients , I chose to use least squares. I got the coefficients with the following calculations:

U - coefficients matrix of the polynomial

- vector of row #i of U

P – control points matrix

row #i of P is the control point

I got a curve that looks like this:

Graphical user interface

Description automatically generated

**Improving UI**

After finishing to add all the curves I added some finishing improvements to make it easier for the user:

* + Graphical user interface, chart

    Description automatically generatedInformation button to all curves equations where I specified my calculations for the specific curve:
  + Adding a curve the table cell not only by dragging the curve name and dropping in the desired cell but also by simply clicking on the curve name in the menu and then adding it to the top left table cell.
  + Adding a control point by clicking the desired position for it on the screen. At the beginning I only connected the new control points to the edges of the curve, afterwards I improved it to add the new control point Pi to be between the Ps with the closest x coordinates, or:
  + I had many issues with positioning of all the elements that I had to sort out and took a while
  + Deleting a control point by clicking on it
  + Alerts when n < 2, n < k
  + Set 2 as min value for n, k
  + A README page to the GitHub project I created

**Future Work**

**Add More Features**

I have worked a lot to try and make the website as learning friendly as possible but during my work there were a few features a I thought be good to add and didn’t have time to.

Here are the following features:

* An information button that will pop a page with general information about parametric curves
* Add an option to open all curves at once and save common curves displays for future use, maybe add an option to set a few fixed grid displays that are frequently used
* Add option to match all control points from differnt screens in order to better compare the curves. For example the user could play with control points of Lagrange and it will change simultaneously also the control points of Bezier in the other screen. Like that the student could see more clearly the differences between the different curves.
* Add option to change color schemes. I based my current color scheme on Visual Studio Code dark mode which I feel most comftrable working with but this might not be the color scheme that is comftarble to everyone. In the future it could be nice to add a “Change Color Scheme” control.

**Add Tutorial Popups**

Many website these days have popup messages when site is first used as tutorials and kind of walk the user through the different buttons and options of the app. It could be nice to add something like this to the website in the future for student who want to use the website for learning and have never used it before.

**Make Code More Efficient and Generic**

Although I have put a lot of thought to make the code as efficient and generic as possible so other students could improve the website in the future with ease there are still a few ways to make it even better, probably with a bigger use of inheritance. For example currently there is a class called Curve that all the other curves classes inherit from but Cubic Hermite Spline for example is very similar to Cubic Spline. By making it inherit from C-Spline it could save some code duplication. I didn’t do this because it caused many bugs that I felt were not worth to spend my time on but this could be a good improvement in the future.

**Add Surfaces**

Currently the website presents only parametric curves. In the futuer it might improve the informationt he websites offers by allowing also to present the variouse surfaces learned in the computer graphics course.

**Write Code with React**

From reading and learning about web development I leared the most web developers use React or Angular for their web development. Since I wanted to relay on an existing code to build on that did not use those I chose to not do that as well. But a good improvement for the future could be to move the code to be written with React (or Angular). It could make future developement of the website much easier and the code much more elegant and efficient.

**Improve Screen Change**

One of the biggest issues I faced during my project was how to resize the canvas where the curves are drawn every time the screen size changes. The biggest issue with this was that I need to save all the important data from the previous run delete the running curve object and redraw it on the new canvas with the saved data. I had trouble finding out how to save the data and access all of the different occasions the canvas size should change. In the future it could be good to add a “save curve data” inorder to make the screen change a smoother experience for the user.

**Conclusion**

In conclusion the project granted me an opportunity to learn many new and interesting things in the programming world and develop useful skills.

I got my first experience with web development that included mastering the use of HTML, CSS, JavaScript, web inspection tools. I got familiar with useful elements like Table and Canvas.

I got a change to have a deeper understanding of various parametric curves and their algorithms.

I also learned about the challenges integrating pieces of code written by different people together and using existing git projects that were not written by me to serve me as a helpful tool.

I got first hand experience with being a software architect, making technological decisions, front-end testing, web design and implementation.

Overall, this project gave me a wider base of knowledge in diverse fields and a useful experience that could serve me greatly in the future.

**References**

1. Web Development Tutorials - <https://www.w3schools.com/>
2. Bezier demo (used as base code) - https://github.com/ryanmid/bezier-curves
3. C-Spline Demo - <http://blog.ivank.net/interpolation-with-cubic-splines.html>
4. Old Lagrange Website - https://www.ibiblio.org/e-notes/Splines/lagrange.html
5. Old Splines Website - <http://scaledinnovation.com/analytics/splines/aboutSplines.html>
6. Chorme Inspecting Tool Tutorial - https://www.browserstack.com/guide/inspect-element-in-chrome
7. About JavaScript - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript>
8. About HTML - <https://www.investopedia.com/terms/h/html.asp>
9. About CSS - <https://developer.mozilla.org/en-US/docs/Web/CSS>
10. About GitHub - <https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/>
11. About Visual Studio Code - <https://code.visualstudio.com/docs/supporting/FAQ>
12. Cubic Hermite Demo - [General Cubic Hermite Spline Demo (codepen.io)](https://codepen.io/liorda/pen/KMvBwM)
13. Least Squares - https://mathworld.wolfram.com/LeastSquaresFitting.html