TU WIEN Informatics

Erstellung einer interaktiven Web-App für Computergrafik-Themen

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Samo Kolter
Matrikelnummer 11810909

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr. techn. Eduard Gröller

Wien, 28. September 2021

Samo Kolter	Eduard Gröller

TU WIEN Informatics

Creating an Interactive Web App for Computer Graphics Topics

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Samo Kolter
Registration Number 11810909

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr. techn. Eduard Gröller

Vienna, 28th September, 2021

Samo Kolter   Eduard Gröller

Erklärung zur Verfassung der Arbeit

Samo Kolter

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. September 2021

____

Samo Kolter

Kurzfassung

Die Computergrafik ist ein Teilgebiet der Informatik mit, wie die Bezeichnung schon nahelegt, starkem visuellen Bezug. Oft bedarf es vieler komplexer mathematischer Berechnungen, um einen Computer dazu zu bringen, eine visuelle Repräsentation eines Objekts auf dem Bildschirm darzustellen. Doch glücklicherweise lassen sich viele Algorithmen, die in der Computergrafik verwendet werden, auch schön veranschaulichen. Das Ziel dieser Bachelorarbeit war, es Interessierten zu erleichtern, ausgewählte Themen der Computergrafik mit intuitiver, visueller Unterstützung besser zu verstehen. Der Fokus lag dabei in erster Linie auf Bézier-Kurven und Verallgemeinerungen davon (konkret B-Spline- und NURBS-Kurven). Um das Ziel zu erreichen, wurde eine Web-App mit interaktiven Demos im Zusammenhang mit den genannten Themenbereichen entwickelt. Diese Demos können mit jedem beliebigen modernen Browser ausgeführt werden.

Existierende Werke in Zusammenhang mit dem Thema der Bachelorarbeit (Publikationen zur Computergrafik-Lehre und dafür verwendete interaktive Tools sowie bereits existierende interaktive Online-Lehrmaterialien/Demos) werden präsentiert. Ebenso wird beschrieben, wie das gesammelte Wissen bei der Implementierung berücksichtigt wurde, und welche Entscheidungen bei der Implementierung getroffen wurden. Details der technischen Umsetzung werden ebenfalls besprochen. Abschließend werden die im Rahmen der Bachelorarbeit gesammelten Erfahrungen reflektiert.

Abstract

Computer Graphics is, as the name suggests, a subdomain of Computer Science with strong relation to visuals. Often a lot of complex math is necessary to make a computer render a visual representation of something onto the screen. However, in many cases the algorithms used can also be explained nicely in a very visual manner. The goal of this Bachelor's Thesis was to find novel ways to introduce people interested in Computer Graphics to selected topics, mainly focusing on Bézier Curves and their generalizations (B-Spline and NURBS curves). To reach this goal, interactive web-based demos that can be viewed with any state-of-the-art browser were created.

Related existing work is presented (publications on approaches to teaching Computer Graphics and existing teaching material, as well as learning resources/demos that were found online). The ways in which the collected knowledge was used when implementing the demos are described as well as key decisions that had to be made for the concrete implementation of the web app. Important implementation details are discussed, too. Finally, an overview of the lessons learnt over the course of the whole project is given.

Contents

Kurzfassung vii

Abstract ix

Contents xi

1 Introduction 1
1.1 Motivation 1
1.2 Problem Statement 1

2 Existing Work 3
2.1 Literature Related to Computer Graphics Teaching 3
2.2 Web-based Learning Material Covering Computer Graphics Topics . . 5

3 Demos Created 11
3.1 Introduction to Bézier Curves 11
3.2 Bézier Curves and Bernstein Polynomials 12
3.3 B-Spline Curves 13
3.4 NURBS Curves 14
3.5 Barycentric Coordinates 15

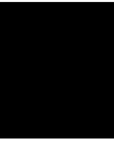
4 Tech Stack and Tools Used 17
4.1 Node.js and Node Package Manager (npm) 17
4.2 Programming Language: TypeScript 18
4.3 Website Stylesheets with SCSS 19
4.4 Using Materialize for Website Styles 19
4.5 Bundling Required Resources: Webpack 19
4.6 Rendering the Demos Onto the Screen: p5.js 20
4.7 Mathematical Notation in the Browser: MathJax 21
4.8 Deployment: GitHub Pages 21

5 Implementation Details 23
5.1 Project Configuration Details 23
5.2 Typescript Code Written for the Project 26

6 Reflection **31**
6.1 Technical Aspects 31
6.2 General Aspects 35

7 Conclusion **37**
7.1 Summary 37
7.2 Use Cases for the Web App 37
7.3 Open Issues/Possible Further Work 38

Bibliography **39**

CHAPTER 1 

Introduction

1.1 Motivation

During my bachelor studies of Media Informatics and Visual Computing at TU Wien I learned many algorithms or abstract mathematical concepts that are commonly applied in the field of Computer Graphics, Computer Vision, or Computer Science in general. In many cases, having a look at formal definitions was not sufficient for me to really understand those concepts. Quite often, some concrete example, or - even better - an interactive demo (that allowed me to tweak parameters and see the consequences of the changes in real time) helped solidify my understanding. Since I enjoyed the Computer Graphics related classes a lot, the idea of creating some demos for certain Computer Graphics topics came into my mind. However, at this stage the exact goals (topics to cover, type of interaction, implementation details etc.) were not yet defined.

1.2 Problem Statement

As a search in the academic realm and on the web showed, a significant amount of interactive, digital teaching material and tools for Computer Graphics already exists (for details please see Chapter 2). However, an issue especially with older material is that it is either not available anymore or it does not run on recent hardware.

The main goal of this Bachelor's thesis was to create learning material explaining Computer Graphics topics in a novel way. Key requirements were:

1. **Interactivity:** Encourage active engagement with the topic at hand with interactive features, e.g. allowing consumers of the learning material to change parameters and see the effects immediately instead of just passively reading about a topic

2. **Ease of Access:** The material should be available to anyone interested in the specific topics and not require any additional installation or setup.

3. **Gradual Learning Curve:** Initially, a high-level overview of the topic at hand should be provided. Ideally, people completely unfamiliar with the topic should also be able to grasp the core concepts. Introduce more and more details later.

4. **Making the Math Make Sense:** Computer Graphics concepts can be expressed with mathematical definitions and formulas. However, formal definitions are not always intuitive to people who have never heard about a topic before. If possible, the learning material should explain *why the math makes sense* after all.

Web browsers were chosen as the platform to create the learning material. They fulfill the first two requirements (interactivity and ease of access) very well - a more thorough discussion can be found in Chapter 6. Interactive demos were created and published on a publicly available website. Due to time constraints only a few selected topics could be covered. The demos focus mostly on Bézier curves and generalizations thereof. This topic was chosen because of personal interest and because it is a sub-topic of geometric modeling, commonly taught in introductory Computer Graphics classes [1]. A detailed discussion of the demos follows in Chapter 3.

CHAPTER 2

Existing Work

Before the interactive demos of this Bachelor's thesis were created, it was necessary to get an overview of "what is already there". To create new learning tools helping as many people as possible, two main questions had to be answered:

1. What topics are generally most relevant when studying Computer Graphics?
2. What topics were already explained in great detail with existing material?

In this chapter, existing literature related to Computer Graphics education and learning material is discussed. Recent publications describing new technologies and tools that may assist learning Computer Graphics are also presented. However, not every useful learning resource is necessarily directly connected to academia and described in a publication. Therefore an extensive web search for additional Computer Graphics learning resources was carried out too.

2.1 Literature Related to Computer Graphics Teaching

2.1.1 Overview of Computer Graphics Education

A look at what is taught at universities in (introductory) Computer Graphics is useful to understand what topics might be most relevant. Balreira et al. [1] collected information on Computer Graphics courses of 20 universities. They gathered data based on the keywords used in the course descriptions. Keywords were repeatedly put into related clusters, those clusters were then further grouped together in clusters that became more and more "high-level". In the final set of topic clusters, Rendering was by far the most common with 75% of the keywords used in university course descriptions relating to it, followed by modeling (14%), and animation (7%). Interestingly, in the lower-level clusters

that the high-level clusters were based on, the topic cluster of geometric modeling ranked very high with 85% of the examined university course descriptions mentioning it - despite modeling in general ranking very low among the high-level clusters.

2.1.2 What makes Computer Graphics difficult to teach?

Suselo et al. [2] carried out a systematic review of literature describing tools for teaching Computer Graphics topics and tried to extract the common key issues that were identified in it. They found four key factors commonly giving students trouble while trying to understand Computer Graphics related topics:

1. Insufficient knowledge of mathematics and basic programming
2. Problems understanding transformations, projections, and 3D geometric modeling
3. Problems making the connection between theory, programming, and application
4. Passive learning (lack of interaction with peers and teaching staff)

The authors also mention concrete examples of existing solutions that were used to mitigate those problems, e.g. e-learning platforms used in classes. They identify three main approaches to teaching Computer Graphics related topics: bottom-up (starting with the very basics and building on that), top-down (using a more complex example and splitting it into smaller, easier to understand modules, often abstracting away low-level details), and hybrid (a combination of bottom-up and top-down).

2.1.3 Technologies and Tools for Teaching and Learning Computer Graphics over the Years

A follow-up paper by Suselo et al. [3] looked at publications describing technologies and tools that are (or were) used to support teaching of Computer Graphics topics. The researchers state that resources with interactive features (e.g. ability to change parameters on the fly) provide additional insight compared to text books, static images, or slides. Teaching and learning material spanning almost three decades was analyzed. Topics covered by the mentioned learning materials include the graphics pipeline, shading, and 3D transformations.

Many tools used the OpenGL Graphics API. Often a simpler and more beginner-friendly graphics API was developed, trying to make graphics programming more accessible to newcomers. Some tools tried to completely remove the programming aspect and focus on high-level understanding of algorithms and concepts. For the implementation of most tools some variant of C or Java was used. Some tools were also implemented as Java applets that could be run in the browser. The authors also included two tools they developed themselves (with other colleagues): a mobile app with AR features helping users to understand 3D transformations [4] and an extension of the CodeRunner plugin

for the Moodle e-learning platform, facilitating the creation of interactive exam questions and practice examples using OpenGL [5].

Unfortunately, not a single one of the 20 teaching tools mentioned in this literature review was available to be tested. Either no link to the software or website was provided or the resources simply did not run on state-of-the-art machinery as the technologies used were outdated. For example, tools developed as Java Applets cannot be used in any current versions of web browsers as Java Applets are no longer supported [6]. Lack of compatibility was generally a recurring issue with learning material for Computer Graphics, both in the tools described in less recent publications and in older interactive learning material found on the web (described in the following section).

2.1.4 WebGL as the Modern Approach to Teaching Computer Graphics

Ten years ago version 1.0 of the WebGL standard was released by the Kronos Group [7]. WebGL essentially brought OpenGL to browsers, as it conforms closely to the OpenGL ES 2.0 specification. The WebGL Graphics API is supported by all major browsers, allowing the rendering of 2D and 3D graphics without any additional libraries [8]. Recently, a few publications appeared that discuss Computer Graphics learning material using WebGL. On the basis of an in-depth comparison of WebGL and OpenGL, Angel [9] concludes that WebGL appears to be more suitable for teaching introductory Computer Graphics. The main arguments for WebGL are its wide-spread support (it runs on practically any computer, tablet, or smartphone with a web browser), more modern and easier API, and the wide-spread availability of additional tools and resources on the web. Angel suggests that the benefits outweigh the drawbacks of WebGL like the different nature of JavaScript as the programming language (compared to C, C++ or Java that are traditionally used with OpenGL) or the need for writing some additional HTML code (that is very similar for most applications anyway).

WebGL was also mentioned as the technology of choice in recent publications discussing Computer Graphics learning tools. Pattanaik and Benamira developed a set of interactive exercises and demos for the web, some of them using WebGL (paper: [10], web link: [11]). Topics covered include basics of vector algebra and transformations, the WebGL rendering pipeline, light/reflection models, shading, texture mapping, and post-shader operations. Rocha et al. also developed a web application utilising WebGL for interactive demos discussing basics of 3D Computer Graphics (paper: [12], web link: [13]) covering similar topics.

2.2 Web-based Learning Material Covering Computer Graphics Topics

Complementary to the search among publications described above, online learning material for Computer Graphics topics was collected too. To limit the scope of the search

so that it produces as much useful “inspiration” for creating interactive demos as possible, a lot of learning material had to be excluded.

The material had to comply with the requirements “interactivity” and “ease of access” outlined in Chapter 1. For learning material to be considered “interactive”, it should for example allow consumers to change parameters of a particular demonstration themselves and see the consequences of those changes in real-time. This means that resources with only static content (text, formulas, images, and videos) are not included. Resources hosted on the web that can be accessed via browsers also cater best to the second requirement (ease of access), therefore only web-based learning material and tools are part of this list.

Several combinations of Computer Graphics related keywords were used during the search for resources with Google’s search engine. The collected online resources were then also tagged with keywords describing what topics they cover. Many different approaches to teaching could be observed, also the topics are covered in varying degrees of detail. Some resources only cover a very specific topic with some demo, providing no additional information, while others are full-fledged tutorial websites trying to teach several related concepts in a systematic manner using additional interactive elements. This section covers resources with promising, novel approaches to teaching. A list of all material collected is also included.

2.2.1 Resources Teaching Basics of Linear Algebra

As the field of linear algebra is particularly important in Computer Graphics, a solid understanding of its core concepts is essential to anyone interested in Computer Graphics. Ström et al. created “Immmersive Math” [14], a website teaching the basics of linear algebra by means of an online textbook with fully interactive figures. Over the course of 10 chapters, topics including vectors, vector operations, matrices, and matrix operations/properties are covered. Examples of concrete use cases for the topics described are also provided. For example, in the chapter discussing vectors, a clone of the classic game “Breakout” is used to demonstrate vectors in use. Additionally, definitions for terms used can be viewed quickly by hovering over the words. The website “Interactive Linear Algebra” created by Margalit and Rabinoff [15] uses a very similar approach to teaching linear algebra.

2.2.2 Hands-on Approach: Learning Computer Graphics via Graphics Programming

A more hands-on approach to learning about fundamentals of Computer Graphics is learning graphics programming, as it applies Computer Graphics concepts. Resources were found on the web that aim to teach graphics programming in a very systematic manner, over several chapters. Users following along learn about some new concept and get the chance to see theory put into practice with live demos implementing those concepts using a graphics API. Either the demos have some kind of UI that allows

changing parameters (sliders, input fields etc.) or the source code is available and can be modified directly. The linear algebra foundations are provided as needed, but they are not discussed as thoroughly as in dedicated linear algebra textbooks.

Some websites following this approach were found. An example is "Learn Computer Graphics using WebGL" created by Wayne Brown [16]. In addition to interactive demos and source code in many modules, it offers a learning experience that resembles that of e-learning systems, with multiple choice questions at the end of each module. The open-source project "WebGL fundamentals"[17] does not explicitly claim to be a course teaching Computer Graphics, however many topics introduced there are also commonly taught in introductory Computer Graphics curricula [1]. "Introduction to Computer Graphics" by David Eck [18] uses OpenGL and Java2D in addition to discussing WebGL and related technologies (JavaScript and its APIs, HTML Canvas, SVG etc.). The materials of the "Graphics Programming" lecture at the University of Marburg are also publicly available (including lecture slides, code examples and interactive demonstrations) [19]. Examples using a wide variety of tools for graphics programming are provided (namely OpenGL, WebGL, and the visual programming environment GSN Composer [20]), however the main focus lies on teaching using OpenGL.

2.2.3 Resources explaining Parametric Curves

Parametric curves are a sub-topic of geometric modeling, a topic that is commonly taught in introductory Computer Graphics courses [1]. Parametric curves are also discussed in the Computer Graphics course at TU Wien that I completed during my Bachelor's studies. In the lectures, interactive demos [21] implemented as Java Applets are used to showcase parametric curves and common related algorithms. Unfortunately, the version of the interactive demos hosted online already does not run on recent browsers. Because of this fact (and due to my own interest in the topic) the search results for learning materials include a disproportionate number of demos related to parametric curves. Among this material, the focus lied mainly on demos covering Bézier curves and their generalizations (B-Splines, NURBS curves).

Bézier Curves and De Casteljau's Algorithm

A demo created by Price [22] allows users to understand the workings of De Casteljau's algorithm for evaluating Bézier curves in an interactive way. Control points can be defined at arbitrary positions. If users then move the mouse cursor from one side of the screen to the other, they can see how the line is drawn. The recursive nature of the algorithm is shown by drawing additional "mid-point lines and control points" for each iteration of the algorithm.

Bostock created an Observable notebook covering De Casteljau's algorithm and its mathematical definition in greater detail [23]. First, an animated cubic Bézier curve is displayed, with a visual representation similar to that of the two demos mentioned

above. Later, however, additional animations and formulas are shown, covering the mathematical background, too, like a chapter in an interactive textbook.

Madsen created Bézier curve demos in the scope of his “Programming Design Systems” Course [24]. While the website itself only covers Bézier curves from an end-user/designer perspective and does not go into the details on how they are rendered by computers, the demos were still very interesting, as the code used for the demos (created using the p5.js [25] library) is publicly available on GitHub [26]. The code of those demos was used as the initial starting point for the Bézier curve demos created in this project. Yet another Bézier curve demo [27] was also found on the web (utilizing the Desmos Graphing calculator), however it did not provide additional educational value compared to the other demos.

B-Splines and NURBS

Fuhr’s interactive web application [28] shows the differences between Bézier curves and B-Spline curves. The pre-defined control points of a provided cubic Bézier curve and a B-Spline curve can be moved. The curve parameter, commonly called t or u, can be modified using a slider. The point on the curve then moves accordingly. Additionally, the basis functions of each control point are presented. The current influence of each control point (the value of its *blending function*) is shown by graph segments and the radius of a circle that gets bigger the more influence a control point has. An animation can also be started to increase the parameter at a consistent speed and make it move along the curve by itself.

Only two web-based demos covering NURBS curves were found. Benton created a demo [29] showcasing NURBS curves and the basis functions of their control points, as well as the influence of the knot vector and the control point weights. The parameter p of the NURBS curve can also be modified. Unfortunately no explanation for all those concepts is given and the number of control points cannot be changed. The “NURBS Calculator” developed by Gami [30], however, tries to explain the NURBS concepts. This is done by giving users control over the number of control points, the control point weights and positions, the value of the curve parameter u, the curve degree, and the knot vector. The curve type can also be changed from NURBS to B-Spline or to Bézier. Some information on the differences between those curve types is also given. Furthermore, explanatory text is displayed when hovering over question mark buttons placed near the available controls. A FAQ section covering further mathematical background on NURBS is also included. The tool also supports the import and export of curve configurations as text files using syntax similar to JSON [31].

Other related topics

Bostock created a Spline editor [32] showcasing many different types of curves used in geometric modeling, however, unfortunately, no explanations on the way those curves are constructed were provided. Two demos covering parametric surfaces were also retrieved

via web search. Kovacs created an interactive demo for Bézier surfaces [33]. A demo covering NURBS surfaces [34] was found on the web, too.

2.2.4 Demos for Other Topics Related to Computer Graphics

3D Graphics and Rendering are also significant sub-topics of Computer Graphics. The collections of demos created by Rocha et al. [12] and Pattanaik and Benamira [11] that were already mentioned in the previous section cover a slew of topics in those fields.

Vector math and matrix operations are essential concepts that are used in practically any application involving Computer Graphics. However, a few concepts might not be intuitive to people learning the basics. One example are vector/matrix transformations. As finding the right order in which matrix transformations have to be applied on some 2D/3D object to achieve the desired result was no trivial task for me personally, additional emphasis was put on this specific sub-topic when searching for learning material. Several interactive demos covering vector and matrix operations were found [35], [36], [37], [38], [39].

Barycentric coordinates were another specific topic that was confusing to me personally in my journey learning the basics of Computer Graphics. In addition to one of the demos created by Pattanaik and Benamira [11] this topic is covered by other demos that were found online [40], [41], [42], [43], [44].

CHAPTER 3

Demos Created

This chapter gives a short overview of the demos that were created for this Bachelor's thesis. In total, there are five demos. Four of them cover Bézier curves and their generalizations. Those four demos build on each other and should be explored in order of appearance, especially if website visitors are new to the topics discussed. Finally, a demo for barycentric coordinates is also presented.

For the curve demos, some mathematical background was provided to users interacting with them. The main source of information on the mathematical and algorithmic details behind the parametric curves presented were the course notes of the "Introduction to Computing with Geometry" course at Michigan Technological University [45]. The Wikipedia pages on Bézier, B-Spline, and NURBS curves [46] and video lectures on Bézier and B-Spline curves found on YouTube [47] were also helpful. However, all those sources used slightly different mathematical notations. For the demos, custom notation combining all the sources' notation styles that also stays consistent between the demos was created in an effort to reduce user confusion.

3.1 Introduction to Bézier Curves

This demo should give users an intuitive understanding for what Bézier curves are and how they are evaluated by the computer in order to be drawn onto the screen. Users are made familiar with Bézier curves step-by-step. A demo guide accompanies the user and tries to convey more of the general idea and concepts behind Bézier curves with each step.

3.1.1 Interactive Features and Visualization Techniques Used

Users may define an arbitrary number of control points for a Bézier curve. Each control point can be moved. At any time control points may be added after a particular control

point by clicking the plus icon that is shown while the user hovers over a control point or drags it. This means that control points may also be added between two existing control points. Upon dragging/hovering a control point, a minus icon is also shown, allowing users to delete the control point.

A slider is provided, allowing users to change the value of the curve parameter t. If the value of t changes, a red point also moves along the curve accordingly. The movement of the point can also be animated by using controls similar to those of media players. Clicking the play button makes the animation start. The value of t then automatically increases continuously (and resets back to 0 once the value 1 is reached) - as a consequence the point also moves along the curve. The speed and the direction of the animation can be changed, too, using two more buttons.

The De Casteljau algorithm that is used to find the position of the point on the curve for a particular value of t is also showcased, drawing inspiration from [22] and [28]. The interpolated temporary control points and lines that are created in order to find the point on the curve are also drawn, visually demonstrating the recursive nature of De Casteljau's algorithm.

3.1.2 Bonus Feature: Demo Guide

Depending on how many control points the user has added to the Bézier curve demo, a demo guide shows different useful contextual information. Initially, while there are no control points on the canvas, the user is told to add a single point. Then, the demo guide asks the user to add another point. Once the second point is added, the concept of linear interpolation between two points is introduced, followed by the prompt to add yet another point. If the user has done that, he/she has constructed a quadratic Bézier. Apart from making the user aware of this fact, an introduction to the concept behind De Casteljau's algorithm is provided. The demo guide then asks the user to further explore the demo by adding yet another control point, creating a cubic Bézier curve. Finally, users are informed that they can add as many control points as they want and are made aware of the fact that each additional control point makes the rendering of the Bézier curve computationally more expensive The problem with global control of each control point is also illustrated.

3.2 Bézier Curves and Bernstein Polynomials

The second demo tries to take a look at the relationship between Bézier curves and Bernstein polynomials. To provide some context, users are first presented with the formula for the Bézier curve

$$C(t) = \sum_{i=0}^{n} b_{i,n}(t) \cdot P_i$$

and the formula for the Bernstein polynomials

$$b_{i,n} = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i}$$

The meaning behind the variables (except the curve parameter t that was also discussed in the first demo) is described.

In the interactive part of the demo, users can use a canvas to add control points for the Bézier curve and edit the value of the parameter t, like in the first demo. The curve drawing visualization can be toggled on/off, if desired by the user. The main focus, however, lies on the additional features introduced.

The graphs for the $b_{i,n}$ of each P_i are displayed next to the canvas. When hovering or dragging a P_i, its corresponding $b_{i,n}$ is highlighted in the graph plot. The graphs update whenever new control points are added or control points are removed. Depending on the current value of t, a vertical red line is also plotted on the graph. All those visual cues should help users make the connection between the graph plot and the curve that is drawn.

The $b_{i,n}$ are also written out explicitly below the graph, displaying their current values, too. Users can see the current Bézier curve formula (sum of $b_{i,n}$ multiplied by the currently used P_i) that changes in real-time.

A much more visual demonstration of the meaning of the Bernstein polynomials is also given in the form of *control-point influence-bars* that can be toggled on/off by the user. Those bars are displayed right next to each P_i (they can also be moved) and show the current value of each $b_{i,n}$. If a bar is full, this means that the $b_{i,n}$ is equal to its maximum value 1. Vice versa, a value of 0 is displayed as an empty bar, supported by an additional `no influence` info text.

Yet another visualization technique in this demo was the visualizer for the currently active (dragged or hovered) P_i. This visualizer maps the values for the P_i's corresponding $b_{i,n}$ across the range of t onto the line that is drawn. Instead of the usual black line, a line is drawn using the P_i's associated color. The thicker the line, the greater the value of the P_i's $b_{i,n}$.

3.3 B-Spline Curves

The third demo of this project focuses on B-Spline curves and their differences compared to Bézier curves. Again, initially, a lot of related mathematical terminology is introduced, mainly the knot vector T and the B-Spline basis functions $N_{i,p}$. The introductory text describes how the knots t_i in T split the range of t into segments that make it possible for each $N_{i,p}$ to only have local control. Remarks concerning the curve domain are also made (e.g. t not necessarily being $\in [0,1]$, or differences between open and clamped B-Spline curves).

The interactive demo keeps many of the concepts from the second demo. Again, the user can edit control points as he/she wishes. However, now a clamped second-degree B-Spline curve is rendered instead of the Bézier curve. The control point influence visualizations described previously also work in this demo. They now use the $N_{i,p}$ (p being the curve's degree) of each P_i. A plot of the $N_{i,p}$ is drawn, too, with the same interactive features as in the previous demo.

The first addition are buttons with plus and minus icons, allowing users to change the curve degree p. Of course, the curve changes in real-time, just like the plots for the $N_{i,p}$.

As the knot vector is one of the core features of B-Spline curves, it is also shown to users. The length of the knot vector is determined automatically on every change of the number of the P_i and it is pre-filled with values, depending on the chosen curve type (see next paragraph). Those values can be edited at any time by the user (invalid inputs are sanitized). Again, the curve and the plot update after every change made.

This demo also allows the user to switch the curve type. They can choose between open B-Spline curve, clamped B-Spline curve and emulated Bézier curve. Depending on the selected curve type, the knot vector changes immediately and is subsequently calculated differently every time a control point is added or removed. For open B-Splines, the knot vector values are simply set to be equidistant (the first value being 0 and the final one being 1). If the curve type is set to "closed B-Spline", the first $p + 1$ knot vector values are 0 and the last $p + 1$ are 1. Finally, for the emulated Bézier curve, the degree of the curve of the demo is always equal to the number of control points - 1 (the user cannot change the degree then), the first half of the knot vector consists of zeros, the second half of ones. This allows users to see immediately, what the differences between the curve types are and how the knot vector changes when switching curve types.

The visualization for the curve evaluation process is also updated after every relevant change to the curve (knot vector or curve degree). It also draws markers for the knots onto the curve. As the user changes the value of t, the visualization also adapts as it moves past a particular knot in the knot vector: the user can see that those $N_{i,p}$ whose value is guaranteed to be 0 are not considered at all when evaluating the curve (or, worded differently: only the non-zero $N_{i,p}$ are used for evaluating the curve).

3.4 NURBS Curves

The final curve demo discusses NURBS curves, a generalization of B-Spline curves. This demo tries to showcase the main difference compared to B-Spline curves: the added control point weights. Practically all features from the NURBS demo were transferred to this demo. Only the curve evaluation visualization is not shown to users as it uses three dimensions and it could not be visualized in a meaningful way.

The control point weights can be edited by the user. If this is done, the curve and plots update accordingly too. The user can see how the control point with higher weight "pulls" the curve closer, as if it had a magnetic force. This is further illustrated with the

same visualization methods outlined above (control point influence bars and visualizer for influence of active control point). In the, plot the original basis functions without weights (or with weights of 1, to be exact) are drawn in the same plot as the weighted basis functions, with dotted lines to allow comparisons.

3.5 Barycentric Coordinates

This demo was the first one that was created in the scope of this project, allowing users to understand how barycentric coordinates work. The color of the point that is initially placed in the middle of the triangle changes as the user moves the point around, showing that the coefficients of the control points (the barycentric coordinates) can be used for vertex color interpolation. Interactive aspects from existing demos (discussed in Chapter 2) were adapted. While no ground-breaking new visualization technologies were used, this demo was still interesting from an implementation standpoint as the draggable control points used here were the first interactive UI elements that were implemented. The underlying implementation of draggable vertices (`DragVertex`) is used throughout the whole application. Also, basics of rendering text onto the canvas with p5.js were tried out here first.

CHAPTER 4

Tech Stack and Tools Used

As the project that was created in the course of this Bachelor's thesis is a web application, the most obvious choice is using the well-established combination of JavaScript, HTML, and CSS that is used in all common modern browsers to display websites with interactive features on the client-side. This combination of technologies has become the de-facto standard on the web and there are no real alternatives. However, there are still many different technologies that can be used to enhance the web developer experience (e.g. code libraries provided as JavaScript modules and languages like TypeScript or SCSS). Bundling all the code and other static assets required for a website is also not a trivial task. The final bundled, production-ready project then also has to be made publicly available via the web to allow users to use the website or web app in their own browsers. Tooling for those purposes exists, too. This section gives an overview of all the technologies used for this project.

4.1 Node.js and Node Package Manager (npm)

As stated on the Node.js website, "Node.js is a JavaScript runtime built on Chrome's V8 Engine"[48]. It facilitates the execution of JavaScript outside of browsers. With Node, JavaScript can also be used directly on Windows, Mac, or Linux, with full access to the file system and the networking capabilities of the local machine (e.g. HTTP, TCP). Initially, Node.js was mostly associated with web servers and the backend. In his talk at JSConf in 2009 Ryan Dahl, the inventor of Node.js, also focused mainly on the performance and scalability benefits of using Node.js for building web servers [49]. However, over the course of the years, Node.js and its use cases evolved further. It can now be used to build complex web applications, microservices, command-line tools, or even full-fledged desktop applications (with the help of frameworks like Electron)[50][51].

Another Node feature that is very popular and relevant (also for frontend projects, like this Bachelor's thesis) is its module system that allows programmers to simply import

modules offering functionality that is needed for particular projects. It was introduced before modules were supported natively by browsers and uses the CommonJS syntax, which differs from the more recent, native JavaScript module syntax [52][53]. Every node installation comes with some built-in modules for functionality like HTTP, file system interaction, etc. [54]. However, Node also supports the installation of third-party-modules for various purposes. With every installation of Node, the Node package manager (npm) is also included. npm is the name of Node's package manager as well as its registry of publicly available packages. Such so-called *npm packages* in turn include node modules. Everyone can publish his/her own packages to npm to share them with other developers around the world [55]. npm was also used to install and manage the dependencies of this particular project (Side note: Webpack, the module bundler used for this project, and its development server (webpack-dev-server) also both run in Node [56]).

4.2 Programming Language: TypeScript

JavaScript is the scripting language that has become the de-facto standard on the web. However, the dynamic nature of JavaScript has some disadvantages. More generally speaking, an unfortunate implication from the design of dynamically typed languages is that code completion in IDEs is significantly less extensive and powerful than in statically typed languages. Also, dynamically typed languages are commonly thought to be more error-prone as type errors cannot be discovered statically and instead can only occur later, at run-time. But still, dynamically typed languages like JavaScript are used a lot in practice, mainly because of their flexibility compared to statically typed languages where generally more code is needed to achieve the same result [57].

TypeScript is a programming language developed by Microsoft. It essentially extends JavaScript's syntax with additional features known from statically typed languages. This means that any JavaScript code is also valid TypeScript code [58]. The most central feature of the language is that type annotations can be added to variables. Those type annotations are used by TypeScript's compiler to infer the desired type of the variable when this same variable is used later and warn developers if they use the variable in a "wrong" way, e.g. by assigning it a value of a type other than the type defined in the type annotation. Building on this idea, e.g. the desired type of input arguments of functions can then also be added via a type annotation. TypeScript code is transpiled to JavaScript code using the TypeScript compiler. This JavaScript code can then run in any browser [59]. TypeScript also allows the use of features from newer ECMAScript standards (Note: ECMAScript is the standard JavaScript is based on) in older browsers that do not support them yet [60].

Another great advantage of TypeScript is that it offers so-called *declaration files* for external libraries. Those files are essentially type annotations for the libraries' public API. Declaration files can even be created for native JavaScript libraries, allowing developers to use them almost as if they were written entirely in TypeScript. Many such declaration files for native JavaScript libraries are available via npm in the `@types` scope. In fact,

for this project the declaration files from `@types/p5` were used for the p5.js library that is also written in vanilla JavaScript.

4.3 Website Stylesheets with SCSS

Sass (short for "Syntactically awesome stylesheets") is a language used for styling content on the web, just like CSS. It offers more concise syntax compared to CSS. For example, selectors can be nested, which is not possible in CSS [61]. The relation between Sass and CSS is similar to the relation between TypeScript and JavaScript: The former is an extension of the latter, but browsers cannot understand that extended syntax natively. So, a compiler that converts the Sass code into CSS code (that browsers understand) is needed [62]. Sass syntax is quite different from CSS, but there is also a different kind of syntax for Sass called SCSS (Sassy CSS) [63] that is very similar to CSS. SCSS is also the language used for styles in this project.

4.4 Using Materialize for Website Styles

As designing the website was not a primary focus of this project, a library for CSS styles was used. For this project, Materialize [64], a library implementing Google's Material Design guidelines [65], was chosen. A SCSS version of the library is also available [66]. If using the SCSS version, variables defining the colors used in the styles provided by the library can be overridden, allowing developers to quickly change the color theme of the website. The SCSS version of Materialize with custom colors was also used in this project.

4.5 Bundling Required Resources: Webpack

A website may consist of many different files that are all required to offer users visiting the website the experience envisioned by the site's developers. In the most trivial case developers write a single HTML and reference all the required content manually. So, they would add HTML tags for the overall structure and content of the page, and add stylesheets written in CSS that define the site's layout and design on different end-user devices and viewports. If additional user interaction with the site leading to dynamic changes of the page content is required, the logic for this is written in a JavaScript file that is referenced in the HTML document via a `<script>` tag. Most sites also use static assets like images or videos. They have to be referenced appropriately, too. If there are other pages on the website, each new page might get its own HTML with all the necessary CSS, JavaScript, and other assets, possibly fetched from some other sites.

While this approach sounds simple at first glance, it quickly becomes cumbersome and tedious. In general, the bigger a project gets, the more difficult it becomes to manage all the resources it requires. For example, this is the case for websites with several subsites, or even more so with modern Single-Page-Applications that load the content to

be displayed on user interaction dynamically. Furthermore, larger projects with lots of JavaScript generally use code split into various modules that are imported from several files. If many modules are imported, it is also very likely that they depend on each other. So, the imports have to be done in the correct order. This quickly becomes a nightmare if one uses many different external libraries where even naming conflicts and other problems might occur. Another issue arises if TypeScript or Sass are used, which both require additional pre-processing steps - one cannot simply send TypeScript or Sass files to browsers, as they cannot be interpreted by the browsers.

Module bundlers help solve all these problems, and more. They take care of resolving the dependencies of a project for the web correctly. In the traditional sense, module bundlers are merely used to resolve dependencies between JavaScript code modules and create a bundled JavaScript file with all the modules resolved in the correct order. This bundled file can then be consumed by the browser. It does not have to fetch the required libraries/modules from several sources itself, instead it gets all required code in a single bundle. But many module bundlers offer additional, more general bundling features for projects on the web. They also simplify the management of additional resources a website might need apart from JavaScript (e.g. HTML, CSS, static assets like images, and more). Moreover, they can be configured to also take care of additional preprocessing that converts code from languages browsers do not understand (like TypeScript or Sass) to the browser's native languages (like JavaScript or CSS). The output JavaScript/CSS is then provided to client browsers by the server. Often commodities that simplify the development process (like sharing HTML templates or JavaScript code between subsites) are also provided. Additionally, often convenient development configurations for projects are supported, like for example so-called *hot-reloading* of only the parts of the code that actually changed when the most recent save occured. For example, the application may reload, if one of the SCSS stylesheets changes, and the result becomes visible immediately. Another common feature of module bundlers is code minification: the bundled code gets condensed furthermore and all unnecessary characters are removed. This results in a smaller overall *bundle size* which in turn improves loading times of websites [67].

Webpack [68] is a popular module bundler that was also used for this project. It offers all of the features mentioned above. Moreover, several plugins can be installed. For example, the webpack-bundle-analyzer[69] plugin gives developers an overview over what modules contribute how much to the overall bundle size.

4.6 Rendering the Demos Onto the Screen: p5.js

When planning to render graphics in the browser, the HTML Canvas Element has to be used. Browsers implement two JavaScript APIs that can be used to render graphics onto the HTML Canvas. The first option is WebGL. WebGL is essentially OpenGL for the browser (as has already been discussed in Chapter 2) and can be used to render both 2D and 3D graphics onto the Canvas. The alternative is the Canvas API [70] that is limited to 2D only, but still quite powerful. Many drawing/graphics libraries exist that

can be used to render elements onto the HTML Canvas. Some are wrappers for WebGL (like Three.js [71]) and meant to be used with 3D Graphics in mind, others use only the Canvas API (like fabric.js [72]) and focus more on other features like for example simplifying interactive features. p5.js [25] was the library chosen for the demos created in the scope of this work. It represents a "hybrid solution" as rendering modes using the Canvas API and WebGL are both supported. The library aims to be very accessible also to people that do not have as much experience with JavaScript, graphics programming, or coding in general. Additional libraries abstracting built-in browser APIs are also included with p5.js, including a simplified DOM API and a sound API.

4.7 Mathematical Notation in the Browser: MathJax

MathJax [73] is a JavaScript library that facilitates the use and display of mathematical formulas written in TEX, MathML, or AsciiMath in the browser. The input formulas can be transformed to SVG, HTML and CSS, or MathML. As mathematical formulas are an integral part of this project, MathJax was also used in this particular project.

4.8 Deployment: GitHub Pages

Every website needs server infrastructure were it is hosted from. GitHub provides a fairly straightforward solution for website deployment called GitHub Pages [74]. GitHub Pages can be used with any website project that has a repository on GitHub. A corresponding npm package called gh-pages [75] is also available. This package provides a command-line utility. Its most important command, `gh-pages`, takes care of all the work required to deploy the website. For example, `gh-pages -d dist` automatically publishes the content of the `dist` folder of the current project to GitHub Pages, making it publicly available. Many resources available online explain the details of this process. A YouTube video by the channel "Traversy Media"[76] was used as a starting point for deployment of this project via GitHub Pages.

CHAPTER 5

Implementation Details

The technologies used in this project and the reasons for using them were already outlined above. This chapter gives an overview of how components were put together in practice in the codebase for this project. An overview of the file and folder structure and the reasoning behind it is given. For interested readers, this project's code is also available publicly on GitHub. Some parts of the codebase might be subject to change in the future. This chapter describes the codebase as it was on 15th of September, 2021.

5.1 Project Configuration Details

To make it possible to run the project in development mode or build the project and deploy it to a web server, some configuration is necessary. The core configuration files can all be found in the project root directory.

5.1.1 Node Configuration

This project uses Node.js. Every Node.js project also needs some configuration. The `package.json` file keeps track of the project's dependencies. It stores all the npm packages the project needs in order to become executable. If the project is cloned from GitHub, running `npm install` on any machine that has Node.js installed is enough to get the project up and running. Scripts can also be defined in this file. A script can trigger a pre-defined command or series of commands. For example, in this project, typing `npm run deploy` causes the whole project to be built, storing all files in a folder called `dist`, and then deploying them to GitHub Pages.

5.1.2 tsconfig.json

`tsconfig.json` stores the TypeScript compiler configuration that defines what JavaScript version the TypeScript code is transpiled to and how strict the compiler's type checking

should be, among other things.

5.1.3 Webpack Configuration

The Webpack configuration files are especially important as they define the whole build process of the project. In many projects using Webpack, a `webpack.config.js` file is included. This file defines how Webpack should do its job of bundling all the necessary data (HTML, JavaScript, CSS, and static assets) for the website. The most essential parts of the Webpack setup for this project are outlined here.

Config Files

In this project, Webpack can be run with two separate configurations: one for production and one for development. Shared configuration parameters between these two configurations are stored in `webpack.common.js`. The webpack-merge npm package is then used to merge the common configuration parameters with those specific to the production (`webpack.prod.js`) and development (`webpack.dev.js`) configurations.

With the production configuration, Webpack puts all the bundled data in a folder whose content can then be deployed to public web servers as-is. This folder is commonly called `dist`, this name was also chosen in this project. On the other hand, with the development configuration, the bundled data is merely loaded into memory (and not stored on the hard disk). This data is then served on a local development server (provided by the webpack-dev-server [77] npm package) running on the machine that is listening on a pre-configured port. In the case of this project the development server runs on `localhost:5500`.

Loaders

For all files that are not plain `.js` files, dedicated loaders have to be used that define what Webpack should do with them. They can be used for various tasks, such as simply putting static assets into a desired folder, allowing developers to use the `import` syntax for files that are not JavaScript modules or pre-processing/transpiling files. Some concrete examples: In this project, the file-loader is used to load all images and store them in the `imgs` folder. The scss-loader, css-loader and style-loader combined allow one to import styles defined in an `.scss` file using the `import` syntax. The awesome-typescript-loader takes care of properly loading all TypeScript files in the project by compiling the TypeScript code files, converting them to JavaScript files and then adding those to the final bundle. All these loaders are defined in the `module.loaders` property of `webpack.common.js`.

HTMLWebpackPlugin

HTMLWebpackPlugin [78] is the only Webpack plugin that is used in this project. However, it has a very significant purpose: It creates the final HTML files for each site

in the demo by processing the EJS templates, including all the relevant code/styles, and inserting the data provided to the templates. Each site is created by a separate instance of HTMLWebpackPlugin that is added to the `plugins` array property of the Webpack config exported by `webpack.common.js`. The HTMLWebpackPlugin instances accept a configuration object that is then used to define the template to be used for the output HTML page, the name/path of the output file, and the chunk (`.ts` file that imports all the page code and styles) to add in the `<script>` tag of the output `.html` file. Additional parameters can also be passed to the template and be consumed by it using EJS [79] syntax. In this project the text for the `<title>` and the main heading of each demo's site were added this way.

5.1.4 Webpack In Action: How The Sites Get Their Content

To demonstrate how all the Webpack-related concepts outlined above come together to produce a working HTML page that browsers can show to users, we discuss how the main page of this project, `index.html` is created. The core parts of the process are essentially the same for the production and development configurations. All the required content for the main page is stored in `src/index`. The `index.ejs` is a template file that is converted to `index.html` by the HTMLWebpackPlugin. Images that should be used in the main page are stored in `imgs` (the file-loader takes care of putting them in a specific folder that is referenced in the EJS template). As outlined above, the HTMLWebpackPlugin also accepts one or more chunks that are inserted into the output HTML in the `<script>` tag. While the main page does not have any significant JavaScript logic, it still has a chunk, too. This is because with the Webpack setup in this project, styles defined in `.scss` files are also imported via JavaScript. The `index.ts` file is handed to HTMLWebpackPlugin as the chunk it should use. `index.ts` contains the following line:

```
import './index.scss';
```

This single line tells Webpack to take the content of the `index.scss` file, convert it to CSS styles and then translate them into JavaScript code manipulating the DOM that is inserted into the chunk file. As soon as this chunk is then loaded and executed by the browser, all the styles defined in `index.scss` are applied. `index.scss` itself imports global CSS styles from `src/global-styles/styles.scss` using the SCSS `@import` syntax and defines some additional styles specific to the main page. Still, the cascade of imports does not stop here. In the `styles.scss` file, styles related to the Materialize SCSS library that is used in this project are also imported.

For creating the demo sites the process is very similar. Each demo site uses the same EJS file (`demo.ejs`) for creating the HTML file. Furthermore, it has its own subfolder containing a `.ts` file and a `.scss` file. The `.scss` file is always imported in the `.ts` file, just like with the main page. However, additionally each of those files also imports the p5 library (as it is used for rendering the demos onto the canvas) as well as any needed TypeScript classes, interfaces, functions etc. that were written to make the demos

work. Explanatory text is also added to the each demo page in its `.ts` file, optionally using MathJax for typesetting of any mathematical formulas.

5.2 Typescript Code Written for the Project

All the TypeScript code that was written for the implementation of the demos (and is not tied to a particular demo page) can be found in `src/demos/ts`. This folder contains two subfolders: `demo-material` and `utils`. `demo-material` contains all the TypeScript code directly related to the interactive demo material that is presented on the website (all the curve demos with their visualizations, and also the barycentric triangle demo). On the other hand, `utils` contains various utility functions and classes that are used throughout the whole application and not tied to a specific demo.

5.2.1 Utility Functions And Classes

The files inside `utils` are split and organized by their purpose.

Interactivity Utilities

Interactivity was a key requirement in this project. Code related to general interactive features of the application can be found in the `interactivity` subfolder within `utils`.

A design goal for the implementation of each demo was that there should be a central class that manages all relevant demo state and also carries out computations that are of interest for several instances of other classes. This concept was especially important for the curve demos. For example, the current state of properties like the curve control points, basis functions etc. should only be stored in the `CurveDemo` class (more details on the class hierarchy etc. will be discussed in the following subsection) and not be duplicated. `observer-pattern.ts` contains interfaces for the implementation of the observer pattern (one for the subject and one for the observers that subscribe to the subject). A discussion of the pattern can be found in [80]. For example, these interfaces were used to inform all classes using curve demo data of relevant changes in the curve demo (more on that later).

Custom interfaces for adding functionality for modeling is-part-of relationships between between a container class and elements it contains were created in `container.ts`. The expected use of these interfaces is documented in the codebase. A use case of these interfaces was the relationship between the control points of a curve and the curve demo: If the user clicks a control points' "add control point" button, a new point should be added to the control points of the demo after that point. If its "remove" button is clicked, the point should be removed from the curve demo control points. To achieve this, the curve control points needed a way to tell their container "what they want it to do". For this purpose, the `Container` and `ContainerElement` interfaces were used.

Finally, `checkbox.ts` contains a utility class for adding a checkbox to the UI. This checkbox can toggle an arbitrary property and have other side-effects defined by the user,

a label and tooltip text that is displayed on hover can also be added. It was used in this project to toggle visualizations for the curve demos.

p5 Utilities

To improve the usability of the p5 library for this project, some utility code was written, too. It is spread across multiple files in `utils/p5`.

One of the biggest and most complicated tasks related to p5 was finding a scalable way to deal with canvas events (clicks, canvas resize, etc.). p5 wraps native canvas DOM events in custom event-handling callbacks. The native canvas element is not meant to be accessed directly. For example, instead of adding a listener for the `click` event to the HTML canvas element, a function has to be provided to p5's `mousePressed()` function. Each subsequent call to this function would overwrite the previous event handling function. So, adding multiple event handlers (e.g. one for each object added to the canvas) is not possible with p5 out of the box. A possible workaround would have been using the native canvas HTML element, which would have probably worked against p5's event system. An alternative approach was chosen and implemented in (`sketch.ts` and `sketch-content.ts`) in the `sketch` subfolder.

`sketch.ts` contains a custom `Sketch` class while `sketch-content.ts` contains interfaces and type guards for content that can be added to a `Sketch` instance. `Sketch` is a wrapper around p5's `createCanvas()` method that allows the user to add drawable objects (instances of classes that implement the custom `Drawable` interface) to the canvas. If these drawables also handle events (e.g. their classes also implement custom event handling interfaces that include event handler functions), their event handler functions are also called each time the respective canvas event occurs.

Another goal was to reduce redundancies in the code that uses the p5 library. For this purpose, lower-level tasks (such as drawing a line of a certain color and width between two points) that would have resulted in a lot of very similar, repeated code were put into utility functions in `misc.ts`. This file also contains other small p5 utilities.

Finally, some reusable classes implementing p5 canvas objects for the `Sketch` that are not tied to a specific demo were also created. `vertex.ts` contains the implementation of vertices (points) that can be rendered onto the screen. A draggable variant was also written, with optional add and remove buttons. This was used for the control points of every curve demo. `polygon.ts` contains a polygon that can be initialized with a set of point positions. If desired, a polygon whose control points can be dragged can also be created. A draggable polygon with three vertices was used for the barycentric coordinate demo.

Other Utility Functions

Color was a rather surprising issue that had to be dealt with when creating the demos (for more details on that see Section 6.1.8). Therefore, many utility functions related to

color had to be created. They can be found in `color.ts`.

Some mathematical concepts were also needed for the creation of the demos. Related utility functions can be found in `math.ts`. Helper functions for the DOM are stored in `dom.ts`. Finally, `misc.ts` contains several utility functions that do not fit any specific category.

5.2.2 Demo Material Code

For the interactive elements of the demos, especially those of the curve demos, many classes were written. The overall code structure will be discussed briefly in the following section.

Classes Related To Curve Demos

The curve demos are all building on each other, so with a rather "quick and dirty" approach to programming them, also a lot of the code for each demo would have been very similar and there would have been a lot of code duplication, making changes to the demos very tedious and difficult to implement. Most probably, the codebase would also have been more difficult to understand. In an effort to make the codebase maintainable, a basic framework for a curve demo that each of the specific curve demos builds upon was created. There are mainly two concepts that were used to reduce code duplication and improve reusability. The first are abstract base classes that implement core functionality while leaving the specifics open to be implemented for each particular type of curve that is presented. The second are reusable classes that do not depend on specifics of the curve - they just work with *any* type of curve.

All abstract base implementations/classes that are used for the curve demos can be found in the `abstract-base` folder. The most important is the abstract `CurveDemo` class. For each type of curve discussed in the demos, the concrete implementation of `CurveDemo` is the central class that manages all the state of the curve. Those parts of the state management that work the same for every type of curve are already implemented. An example is the management of the curve's control points (including the assignment of fitting colors to each vertex or logic that allows users of the class to define how much additional curve information should be shown). Functions and properties that are specific to each type of curve are defined in the concrete implementing subclasses. For example, they specify how a point on the curve is found, how each blending function that defines the influence of a control point for the current value of the curve parameter t is calculated, what the curve domain is, or what conditions have to be met for the curve to be considered "valid". All the other curve-related classes in the codebase rely on the `CurveDemo`'s data, change its properties, and/or react to any relevant changes that occur (defined in a dedicated `DemoChange` interface).

In general, every `CurveDemo` uses four additional core components, defined in separate classes:

1. An instance of Curve that uses the current values of the curve properties stored in the CurveDemo to render the actual curve onto the screen.

2. An instance of ControlsForParameterT that allows users to manipulate the value of the curve parameter t with interactive elements (a slider and buttons).

3. An instance of CurveDrawingVisualization that conveys to users visually how the point on the curve for the current value of the curve parameter t is found (e.g. for Bézier curves, this is a visualization of De Casteljau's algorithm).

4. An instance of InfluenceVisualizerForActiveControlPoint that "projects" the influence of a particular active (hovered/dragged) control point on the overall curve (for each value of t). The curve is then drawn in the color of the respective control point, but it becomes thicker or thinner depending on the value of the *blending function* of a control point for the current t. Concrete example of blending functions are the Bernstein polynomials of Bézier curve control points or the basis functions of B-Spline curve control points.

Next to CurveDemo, two more curve-related classes are abstract. CurveDrawingVisualization provides some basic configuration parameters for the visualization of the evaluation of a curve (as already mentioned above), leaving the exact implementation open. GraphPlotter is extended for each concrete curve type and plots the blending functions of the control points (also already discussed above) as graphs, each colored in the respective control point's color.

The shared folder contains all classes that do not depend on specifics of the type of curve that is discussed in a demo. This includes the Curve, ControlsForParameterT, and InfluenceVisualizerForActiveControlPoint classes already mentioned above. The control point influence bars that are used in three of the four curve demos are also "curve-type-agnostic", just like the class that is responsible for rendering a line at the current value of t on top of the curve graphs plotted by concrete GraphPlotter instances.

For each type of curve, the abstract classes of abstract-base were implemented in the subfolder for the corresponding curve type. Additional classes for specifics of each curve type and demos were also created. For the Bernstein polynomial demo, a class that shows the current formulas for the Bézier curve formula and the Bernstein polynomials it consists of was written. Dedicated classes for the B-Spline demos allow the user to change the knot vector and the curve degree interactively, as well as the type of B-Spline curve that should be visualized. Those classes were reused for the NURBS demos too. Furthermore, the NURBSCurve class extends the BSplineCurve as NURBS curves share most of their properties with B-Spline curves. The only addition are the control point weights that turn the basis functions into weighted basis functions. The NURBS demo also allows the user to change those control point weights, so for this yet another class was created.

Barycentric Triangle

The interactive parts of the barycentric triangle demo were implemented in `barycentric-triangle.ts`. Two classes are included there, one for the triangle used in the barycentric coordinate demo, and one for the point that can be moved across that triangle surface and beyond (it also computes the barycentric coordinates).

CHAPTER 6

Reflection

Over the course of the project, many decisions had to be made, from the overall concept for the web app to its actual implementation. There is always a multitude of ways to solve the same problem. Each time a decision for a particular solution is made, other options are ruled out. This may of course have significant impact on the path the project takes as time passes and the number of new problems that may arise. This chapter discusses the major decisions that had to be made and what lessons were drawn from them.

6.1 Technical Aspects

6.1.1 Choice of Platform and Programming Languages

Luckily, over the course of the project, the decision to build a web application proved to be a good one. Every device that could be used for viewing the demos that were planned to be implemented (PC, tablet, smartphone) has a web browser. All currently popular browsers provide users a very similar experience, often only differing in small details. Every browser “speaks” JavaScript, HTML and CSS. My previous experience with developing for the web allowed me to create the demos without any huge problems related to those languages themselves.

Creating this project as a desktop application or native mobile application would not have been as practical as writing a web app. None of the other possibilities would have provided the same cross-platform compatibility without additional work. Looking back, choosing TypeScript instead of plain JavaScript for writing the code was also the right decision. The added type safety and better code completion suggestions in the IDE were huge advantages. However, there were some issues with missing type declarations for libraries (discussed in detail later).

6.1.2 Experiences Made By Not Using a Frontend Framework

Early on in the project, a decision had to be made whether the web app should be a single-page application (SPA) or a more traditional website with separate HTML files. When going the "SPA-route", using a frontend framework or library such as Angular, React, or Vue would have benn the natural choice. However, this would still have introduced a significant number of additional third-party dependencies and may also have added complexity to the project. As the project was considered "simple enough" to not need concepts like component-based architecture and advanced routing capabilities that are common characteristics of frameworks and libraries for SPAs, the alternative route of creating a "simple" website with links between the different pages - without any frontend framework - was chosen.

An implicit consequence of the choice for building a more traditional website instead of using frontend frameworks or libraries was that many smaller tedious tasks such as creating input elements for controlling the demos had to be implemented in an imperative way using the browser's native DOM API. All the required HTML elements had to be created or selected explicitly, assigning event listeners and CSS classes (or inline styles) manually, all by calling the respective DOM API functions. This would have been less tedious in a frontend framework.

In the project, also some advanced, nested HTML structures were built with DOM API modifications in TypeScript code. Then CSS classes were added and styles were applied in TypeScript code, too. Also, the same HTML template was used for all demos. However, some elements still needed to be different between demos, so again the DOM API was used in the code to make the layout work. This made the project more difficult to understand, as reading through all the code is just not as quick as looking at HTML markup directly. Probably using a frontend framework that offers encapsulated components with strictly separated HTML markup, JavaScript/TypeScript logic, and CSS styles, would have been useful. Such components could also have been inserted into HTML markup in a simpler fashion, and experimenting with the layout would have been quicker. Their more declarative syntax would also have made it easier to keep an overview.

More generally speaking, unfortunately, the codebase turned out to not be as modular and easily extendable as initially desired. For example, the base demo class has many different responsibilities, making it difficult to understand.

6.1.3 p5.js vs. Alternatives

Another important decision in the beginning phase of the project was choosing a library for rendering the demos onto the HTML canvas (or even using no external libraries at all). As already outlined above, p5.js was the library chosen for the demos created in this project. Unfortunately, using p5.js sometimes felt tedious or even "hacky". More unexpected problems are discussed below. Unfortunately, due to lack of experience with other libraries like Three.js or the plain WebGL browser API, reflecting on this decision is not really possible.

6.1.4 Difficulties with Responsive Design

Initially, one of the goals of the web app project was that it would be comfortably usable on any of the popular devices used to access the web, ranging from desktops and laptops to mobile phones and tablets. While all the demos do work on mobile (touch support is also given), most are not in fact comfortable to use. Over the course of the project providing responsive design and a consistent user experience across all end-user devices unfortunately proved too time-consuming. Responsiveness considerations had to be cancelled.

6.1.5 Difficulties Setting Up Webpack

Refusing to use any frontend framework also implied that the management of all the resources related to the project had to be configured from scratch. Configuring Webpack to correctly transpile the TypeScript code to JavaScript, SCSS to CSS, and correctly including all the dependencies for each demo (HTML, JavaScript, images etc.) was more difficult then expected. Making the HTML templates for the demos work, including only the JavaScript code, CSS styles and static assets that were required for the subsite was also quite difficult. Configuring Webpack's dev-server with hot reloading of only the parts of the codebase that changed also took up a significant amount of time. However, all those struggles also lead to a much better understanding of all the useful concepts that modern frontend frameworks offer developers "out of the box".

6.1.6 Using p5.js vs. Using Native Browser APIs

As p5.js is a JavaScript library focussing on "making coding accessible and inclusive for artists, designers, educators, beginners, and anyone else" [25] it also abstracts away some of the details of the native HTML Canvas API and DOM API. This abstraction was useful at times. But at the same time, an unfortunate consequence of this is that the implementation of many advanced features needed for the project created in the scope of this Bachelor's thesis proved more difficult. For some advanced tasks, using the native DOM API was actually more simple as some specific functionalities were not provided by the "DOM API Wrapper" of p5. Attaching event listeners to the canvas element created by p5 also felt awkward and "hacky". The lack of certain features in p5 meant that a combination of p5's DOM API wrapper and the native DOM API was used. This probably impaired the overall readability and quality of the codebase.

6.1.7 Performance Problems

The code used for the interactive demos is probably not written in the most efficient way. With some quick fixes performance could be kept at an acceptable level. However, unfortunately due to time constraints and lack of experience, the programs could not be optimized to the desired degree. p5's 3D renderer that utilizes WebGL under the hood was also not used (the reasoning behind this being that the demos were only meant to be

implemented in 2D anyway). Maybe with WebGL enabled (or when using other WebGL libraries like Three.js), performance would have been better.

6.1.8 Coloring Vertices: Not As Easy As Expected

The control points of the parametric curves in the created demos (Bézier curve, B-Spline curve, NURBS curve) can be edited as desired by users. New vertices can be added, also in between two existing vertices. Of course at any time, vertices may also be removed. This created an interesting and surprisingly difficult problem in the implementation of the demos because a requirement was that each vertex had to have its own color that was as distinguishable from its direct neighbors as possible. This was important as graphs directly related to each control point were also shown (i.e. the Bernstein polynomials of Bézier curves or the basis functions of NURBS curves). Each control point's color had to be clearly distinguishable from the background, too. Those requirements led to an unexpected digression into the realms of color (color spaces, luminance of colors etc.). A considerable amount of additional time was spent coming up with logic for finding fitting colors that are "not too similar" to others, among other things.

6.1.9 Issues with Library Type Declarations and Documentation

In a few occasions the "official" type declaration files for p5.js provided via `@types` on npm were incomplete or erroneous. This led to some unnecessary confusion. Also, even after spending a lot of time googling and searching the official documentation, no solution on how the TypeScript version of the MathJax library worked and how it could be set up to make it usable in the project was found. The workaround for this was to simply include a JavaScript version of a MathJax configuration directly via the HTML `<script>` tag, not utilizing the advantages of TypeScript.

6.1.10 Issues With State Management

A well known issue in frontend web applications is state management. Every click, keyboard input or other event may change the state of some part of the application in some way. Quite often several different parts of the application have to "know" about the state of particular parts of the application or might even want to update it. Furthermore, certain classes/components have to store some states locally. Keeping all those states in sync can quickly become difficult, and it also was in this project. There were some factors that made things more complicated.

To inform classes using curve demo data of changes, the commonly used observer pattern was implemented. The curve demo then takes over the role of a so-called *subject*. Several *observers* can *subscribe* to that subject. The subject pushes updates to all subscribed observers. The way this pattern was implemented in this project led to an unnecessary increase in complexity. Unfortunately, when changes occurred, the observers were only notified that *a certain type of* change occurred and not *what exactly* that change was. This meant that additionally every observer also had to store a reference to the curve

demo and check what the current values of its properties were. Maybe the use of a dedicated state management library would have helped. At the same time, yet another factor made the software more complicated: the render loop.

Per default, p5.js renders onto the canvas in a loop, 60 times per second. This fact was used (and maybe abused) in this application, too. Instead of responding to changes or events, some classes just used the render loop to get the latest state of parts of the demo. It was therefore unfortunately not always clear, where each application part got its data from and who modified it when.

6.2 General Aspects

6.2.1 Differences in Mathematical Notation

Many different sources were used to gain understanding of the mathematical foundations for the topics discussed in the demos. Unfortunately, there were often minor differences in notation that made learning a lot more difficult. In the codebase and on the website, custom notation had to be used to stay consistent between different demos and not confuse users unnecessarily.

6.2.2 Issues Finding Literature/Relevant Existing Work

It was initially very difficult to pinpoint *what* kind of literature and existing material should be searched for. Even more difficult was then to find material that could be relevant. In fact, many interesting resources were discovered rather late in the project.

6.2.3 Structuring The Written Part of The Bachelor's Thesis

The implementation part provided a very interesting, but still quite demanding challenge, as expected. However, it was much more difficult than initially estimated to define the overall structure of the written part of this Bachelor's thesis and put the work that was done into words. Many sections were rewritten many times, chapters and sections were rearranged frequently. Filtering out the important findings and not getting lost in details was a significant challenge for me. Hopefully this work still provides value in some way.

6.2.4 Planning Issues

The amount of time and effort spent for this Bachelor's thesis was greater than expected. At several times, seemingly simple tasks took much longer than initially planned. But still, overall the whole project was a tremendous learning experience. Despite occasional moments of frustration there were also lots of joyful moments.

CHAPTER 7

Conclusion

7.1 Summary

The rather vague initial goal of creating a website with some (hopefully useful) interactive demos for Computer Graphics topics spawned several other related artifacts.

First, a quick overview of teaching methods, approaches and related tools in the field of Computer Graphics was provided based on existing academic publications. Related existing interactive online teaching material was discussed later. This collection of learning material could provide good starting points for people interested in learning more about certain Computer Graphics topics and may save some time otherwise spent searching around the web by oneself. To me, it also served as inspiration for the creation of the interactive demos of this project.

The actual website that was created and its core implementation details were also discussed, explaining some of the reasoning behind the way the demos were created. The discussion of implementation details also included a detour into web development technologies and tools that were used. Writing this section helped me personally understand the complex world of web development and all the new things that keep emerging in it a bit better. Hopefully the content discussed is also valuable to others.

7.2 Use Cases for the Web App

The primary reason for creating the demos of this project was to help with learning. So obvious use cases are in the field of Computer Graphics education. Hopefully the demos created are useful to some people learning the topics covered. The demos could potentially also be used by teachers and educators, if they consider them intuitive and insightful.

7.3 Open Issues/Possible Further Work

7.3.1 Search Engine Optimization

In theory, the website of this project and its demos are available to anyone as the website is available publicly. However, some search engine optimization would be necessary to make sure this website is included in the top search results when people want to learn about the topics covered by its demos.

7.3.2 Evaluation of Usability

Hopefully, this website will be useful to at least a few people trying to learn any of the topics covered by the website's demos. Due to time constraints, the demos could unfortunately not be created with feedback of a larger group of potential users.

It would be very interesting to see if the requirements for the demos outlined in Chapter 1 (especially the "gradual learning curve" and the ability of the demos to "make the math make sense") were actually met. Most probably, a user study would be able to at least partially answer those questions. A well-designed user study may pinpoint aspects of the topics covered in the demos that were explained poorly and indicate how they can be presented to users in a more effective way.

7.3.3 Feedback, Additional Demos

This project could potentially evolve further. Feedback on the whole project would be very helpful. The existing demos could be expanded, new interactive demos could be added, also by other contributors. However, most probably a rewrite of parts of the application would be necessary to make the code more reusable, modular, and thus make the whole codebase easier to extend. Documentation should be improved, too.

The website featuring the interactive demos is available for anyone to explore:
`https://sejmou.github.io/interactive-computer-graphics/`

Feature requests, bug reports or pull requests can be submitted anytime to the public code repository on GitHub:
`https://github.com/Sejmou/interactive-computer-graphics`

Feel free to contribute!

Bibliography

[1] Dennis Balreira, Marcelo Walter, and Dieter Fellner. What we are teaching in introduction to computer graphics. In *Proceedings of the European Association for Computer Graphics: Education Papers*, EG '17, page 1–7, Goslar, DEU, 2017. Eurographics Association.

[2] Thomas Suselo, Burkhard C. Wünsche, and Andrew Luxton-Reilly. The journey to improve teaching computer graphics: A systematic review. 12 2017.

[3] Thomas Suselo, Burkhard C. Wünsche, and Andrew Luxton-Reilly. Technologies and tools to support teaching and learning computer graphics: A literature review. In *Proceedings of the Twenty-First Australasian Computing Education Conference*, ACE '19, page 96–105, New York, NY, USA, 2019. Association for Computing Machinery.

[4] Thomas Suselo, Burkhard C. Wünsche, and Andrew Luxton-Reilly. Mobile augmented reality as a teaching medium in an introductory computer graphics course. In *2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*, pages 72–76, 2018.

[5] Burkhard C. Wünsche, Edward Huang, Lindsay Shaw, Thomas Suselo, Kai-Cheung Leung, Davis Dimalen, Wannes van der Mark, Andrew Luxton-Reilly, and Richard Lobb. Coderunnergl - an interactive web-based tool for computer graphics teaching and assessment. In *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, pages 1–7, 2019.

[6] The end of Applets. `https://www.infoq.com/news/2021/03/end-of-applets/`. Accessed: 2021-08-22.

[7] Release statement for WebGL 1.0 (Khronos Group). `https://www.khronos.org/news/press/khronos-releases-final-webgl-1.0-specification`. Accessed: 2021-08-27.

[8] WebGL (Mozilla Developer Network). `https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API`. Accessed: 2021-08-27.

[9] Ed Angel. The case for teaching computer graphics with webgl: A 25-year perspective. *IEEE Computer Graphics and Applications*, 37(2):106–112, 2017.

[10] Sumanta N. Pattanaik and Alexis Benamira. Teaching Computer Graphics During Pandemic using Observable Notebook. In Beatriz Sousa Santos and Gitta Domik, editors, *Eurographics 2021 - Education Papers*. The Eurographics Association, 2021.

[11] Sumant Pattanaik. Computer graphics fundamentals (Observable). https://observablehq.com/collection/@spattana/class-4720. Accessed: 2021-08-22.

[12] David Rocha, Daniel Exposito, Juan Ruiz de Miras, and María Dolores Robles Ortega. A web application to support teaching of computer graphics to engineering students. *The International journal of engineering education*, 34(1):34–44, 2018.

[13] David Rocha, Daniel Exposito, Juan Ruiz de Miras, and María Dolores Robles Ortega. Interactive web platform to support the teaching of computer graphics. http://www4.ujaen.es/~demiras/cgex/. Accessed: 2021-08-25.

[14] Jakob Ström, Karl Åström, and Tomas Akenine-Möller. Immersive math. http://immersivemath.com/ila/index.html. Accessed: 2021-03-10.

[15] Dan Margalit and Joseph Rabinoff. Interactive linear algebra. https://textbooks.math.gatech.edu/ila/matrix-transformations.html. Accessed: 2021-03-10.

[16] Wayne Brown. Runestone academy: Learn computer graphics using webgl (interactive textbook). https://runestone.academy/runestone/books/published/learnwebgl2/index.html. Accessed: 2021-08-25.

[17] WebGL fundamentals. https://webglfundamentals.org/. Accessed: 2021-08-25.

[18] David Eck. Source and demos (for introduction to computer graphics). https://math.hws.edu/graphicsbook/source/index.html. Accessed: 2021-08-25.

[19] Thorsten Thormählen. Graphics programming course of Phillips-University Marburg. https://www.uni-marburg.de/en/fb12/research-groups/grafikmultimedia/lectures/graphics. Accessed: 2021-08-25.

[20] GSN Composer documentation. https://www.gsn-lib.org/docs/index.php. Accessed: 2021-08-29.

[21] Martin Kilian, Torsten Mohs, Raphael Straub, Claudia Bangert, and Hartmut Prautzsch. CAGD-Applets - an interactive tutorial on geometric modeling. https://cagd-applets.webarchiv.kit.edu/mocca/html/noplugin/inhalt.html. Accessed: 2021-07-27.

[22] Chris Price. Bézier curve demo (inspired by animations on Wikipedia (https://en.wikipedia.org/wiki/B%C3%A9zier_curve#Constructing_B%C3%A9zier_curves)). http://bezierdemo.appspot.com/. Accessed: 2021-08-29.

[23] Mike Bostock. Demo of De Casteljau's algorithm (Observable). https://observablehq.com/@mbostock/de-casteljaus-algorithm. Accessed: 2021-08-30.

[24] Rune Madsen. Programming Design Systems: Custom shapes. https://programmingdesignsystems.com/shape/custom-shapes/index.html. Accessed: 2021-03-10.

[25] Official p5.js website. https://p5js.org/. Accessed: 2021-08-05.

[26] Rune Madsen. Code for Quadratic Bézier Demo of Programming Design Systems (GitHub). https://github.com/runemadsen/programmingdesignsystems.com/blob/master/examples/shape/custom-shapes/quad-animation.js. Accessed: 2021-03-10.

[27] Bézier curve demo (Desmos). https://www.desmos.com/calculator/cahqdxeshd. Accessed: 2021-08-29.

[28] Richard Fuhr. Exploring Bézier and Spline curves. demo: https://richardfuhr.neocities.org/BusyBCurves.html, accompanying YouTube video: https://youtu.be/-aiErrvLRfE, transcript with further information: https://richardfuhr.neocities.org/BusyBCurvesTranscript.html, article on medium.com: https://medium.com/@rdfuhr/exploring-bezier-and-spline-curves-a8261b3c7a8b. All accessed: 2021-07-27.

[29] Alex Benton. NURBS demo. http://bentonian.com/teaching/AdvGraph0809/demos/Nurbs2d/index.html. Accessed: 2021-08-30.

[30] Pawan Gami. NURBS calculator. http://nurbscalculator.in/. Accessed: 2021-08-05.

[31] Introducing JSON (json.org). https://www.json.org/json-en.html. Accessed: 2021-08-31.

[32] Mike Bostock. Spline editor. https://observablehq.com/@d3/spline-editor. Accessed: 2021-08-05.

[33] Viktor Kovacs. Bézier surface demo. http://kovacsv.github.io/JSModeler/documentation/examples/bezier.html. Accessed: 2021-08-29.

[34] Peter Polgar. NURBS surface demo. https://peterpolgar.github.io/NURBS-surface-demo/. Accessed: 2021-08-30.

[35] Thomas van den Berge. A tool for understanding 3d matrix transformations. article: http://thomasmountainborn.com/transforms-2/, demo:http://thomasmountainborn.com/TransformsPlayer/index.html. Accessed: 2021-03-10.

[36] Yuri Sulyma. Matrix visualizer (2d/3d). https://epiplexis.xyz/a/fxh/matrix_visualizer. Accessed: 2021-03-10.

[37] H. Miller. MIT Mathlet: Matrix vector (+eigenvalues/vectors). https://mathlets.org/mathlets/matrix-vector/. Accessed: 2021-03-10.

[38] Desmos: Matrix transformations tool. https://www.desmos.com/calculator/vkws7hybxc. Accessed: 2021-03-10.

[39] Wolfram Alpha: Examples for geometric transformations. https://www.wolframalpha.com/examples/mathematics/geometry/geometric-transformations. Accessed: 2021-03-10.

[40] T. J. Jankun-Kelly. Barycentric coordinates (Oservable). https://observablehq.com/@infowantstobeseen/barycentric-coordinates. Accessed: 2021-08-29.

[41] Interpolating in a triangle - Code Plea. https://codeplea.com/triangular-interpolation. Accessed: 2021-09-04.

[42] Barycentric coordinates (GeoGebra demo). https://www.geogebra.org/m/ZuvmPjmy. Accessed: 2021-08-29.

[43] Seung Joon Choi. CodePen: Barycentric coordinates. https://codepen.io/erucipe/pen/gpBgpR. Accessed: 2021-03-10.

[44] CutTheKnot: Tool for barycentric coordinates. https://www.cut-the-knot.org/Curriculum/Geometry/Barycentric.shtml. Accessed: 2021-07-27.

[45] Ching-Kuang Shene. Introduction to computing with geometry - Michigan Technological University (course notes). https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/. Accessed: 2021-08-05.

[46] Wikipedia pages on Bézier, B-Spline and NURBS curves. Bézier curve: https://en.wikipedia.org/wiki/B%C3%A9zier_curve, B-Spline curve: https://en.wikipedia.org/wiki/B-spline, NURBS curve: https://en.wikipedia.org/wiki/Non-uniform_rational_B-spline. Accessed: 2021-09-06.

[47] Ian Shiach. Video lectures about Bézier and B-Spline curves (YouTube). Bézier curves: https://youtu.be/2HvH9cmHbG4, B-Splines: https://youtu.be/qhQrRCJ-mVg. Accessed: 2021-09-06.

[48] Official Node.js website. https://nodejs.org/en/. Accessed: 2021-08-05.

[49] JSConf 2009: Talk by Ryan Dahl about his Node.js project. https://youtu.be/ztspvPYybIY. Accessed: 2021-08-05.

[50] Mateusz Gajda. Why use Node.js for web development? scalability, performance and other benefits of Node based on famous web applications. https://tsh.io/blog/why-use-nodejs/. Accessed: 2021-08-10.

[51] Official Electron website. https://www.electronjs.org/. Accessed: 2021-08-10.

[52] Craig Buckler. Understanding ES6 modules (sitepoint.com). https://www.sitepoint.com/understanding-es6-modules/. Accessed: 2021-08-10.

[53] JavaScript modules (Mozilla Developers Network). https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules. Accessed: 2021-08-10.

[54] Node.js modules tutorial. https://www.tutorialsteacher.com/nodejs/nodejs-modules. Accessed: 2021-08-10.

[55] Node package manager (npm) docs. https://docs.npmjs.com/about-npm. Accessed: 2021-08-13.

[56] Why webpack (official website). https://webpack.js.org/concepts/why-webpack/#birth-of-javascript-modules-happened-thanks-to-nodejs. Accessed: 2021-08-29.

[57] Eamonn Boyle. Static types vs dynamic types. https://instil.co/blog/static-vs-dynamic-types/. Accessed: 2021-08-17.

[58] Angular docs (about TypeScript). https://angular.io/guide/upgrade#migrating-to-typescript. Accessed: 2021-08-17.

[59] Official TypeScript website. https://www.typescriptlang.org/. Accessed: 2021-08-05.

[60] New TypeScript features that improve the developer experience). https://www.sitepen.com/blog/new-typescript-features-that-improve-the-developer-experience. Accessed: 2021-08-17.

[61] Sass style rules (official docs). https://sass-lang.com/documentation/style-rules. Accessed: 2021-08-17.

[62] Sass basics. https://sass-lang.com/guide. Accessed: 2021-08-17.

[63] Sass vs. SCSS. https://thesassway.com/sass-vs-scss-which-syntax-is-better/. Accessed: 2021-08-17.

[64] Materialize CSS website. https://materializecss.com/. Accessed: 2021-08-30.

[65] Material Design website. https://material.io/design. Accessed: 2021-08-30.

[66] Materialize CSS: Using the SCSS version. https://materializecss.com/sass.html. Accessed: 2021-08-30.

[67] Nabil Nalakath. Module bundlers in 5 minutes — the what, the why, and the which (BetterProgramming). https://betterprogramming.pub/javascript-module-bundlers-2a1e9307d057. Accessed: 2021-08-29.

[68] Official webpack website. https://webpack.js.org/. Accessed: 2021-08-05.

[69] Webpack Bundle Analyzer (npm). https://www.npmjs.com/package/webpack-bundle-analyzer. Accessed: 2021-09-07.

[70] Canvas API (Mozilla Developer Network). https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API. Accessed: 2021-08-31.

[71] Three.js website. https://threejs.org/. Accessed: 2021-08-31.

[72] fabric.js website. http://fabricjs.com/. Accessed: 2021-08-31.

[73] MathJax website. https://www.mathjax.org/. Accessed: 2021-08-17.

[74] About GitHub Pages (official website). https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages#publishing-sources-for-github-pages-sites. Accessed: 2021-08-30.

[75] gh-pages Node package on npm. https://www.npmjs.com/package/gh-pages. Accessed: 2021-08-30.

[76] GitHub Pages Deploy & Domain (tutorial on YouTube by Traversy Media). https://youtu.be/SKXkC4SqtRk. Accessed: 2021-08-30.

[77] webpack-dev-server (webpack website). https://webpack.js.org/configuration/dev-server/. Accessed: 2021-09-07.

[78] HtmlWebpackPlugin (webpack documentation). https://webpack.js.org/plugins/html-webpack-plugin/. Accessed: 2021-09-07.

[79] EJS website. https://ejs.co/. Accessed: 2021-09-07.

[80] Dominik Gruntz. Java design: On the observer pattern. *Java Report*, 2002.