Creative Coding on the Web in p5.js
A Library Where JavaScript Meets Processing

Emil Sandberg

6 June, 2019
This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the bachelor degree in Software Engineering. The thesis is equivalent to 10 weeks of full time studies.

Contact Information:
Author:
Emil Sandberg
E-mail: emilalsandberg@gmail.com

University advisor:
Emil Folino
Department of Computer Science

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
ABSTRACT

Creative coding is the practice of writing code primarily for an expressive purpose rather than a functional one. It is mostly used in creative arts contexts. One of the most popular tools in creative coding is Processing. Processing is a desktop application and in recent years a web-based alternative named p5.js has been developed.

This thesis investigates the p5.js JavaScript library. It looks at what can be accomplished with it and in which cases it might be used. The main focus is on the pros and cons of using p5.js for web graphics. Another point of focus is on how the web can be used as a creative platform with tools like p5.js. The goals are to provide an overview of p5.js and an evaluation of the p5.js library as a tool for creating interactive graphics and animations on the web.

The research focuses on comparing p5.js with plain JavaScript from usability and performance perspectives and making general comparisons with other web-based frameworks for creative coding. The methods are a survey and interviews with members of creative coding communities, as well as performing coding experiments in p5.js and plain JavaScript and comparing the results and the process.

The results from the coding experiments show that compared to plain JavaScript p5.js is easier to get started with, it is more intuitive, and code created in p5.js is easier to read. On the other hand, p5.js performs worse, especially when continuously drawing large amounts of elements to the screen. This is further supported by the survey and the interviews, which show that p5.js is liked for its usability, but that its performance issues and lack of advanced features mean that it is usually not considered for professional projects. The primary use case for p5.js is creating quick, visual prototypes. At the same time, the interviews show that p5.js has been used in a variety of contexts, both creative and practical.

p5.js is a good library for getting started with coding creatively in the browser and is an excellent choice for experimenting and creating prototypes quickly. Should project requirements be much more advanced than that, there might be other options that will work better.

Keywords: creative coding, p5.js, interactive graphics, web frameworks
# CONTENTS

1 INTRODUCTION.............................................................................................................................5
   1.1 BACKGROUND............................................................................................................................5
   1.2 GOALS.........................................................................................................................................5
   1.3 MOTIVATIONS AND VALUES.....................................................................................................6
   1.4 SCOPE.........................................................................................................................................6

2 RESEARCH QUESTIONS...................................................................................................................8

3 RESEARCH METHODS......................................................................................................................9
   3.1 CODING EXPERIMENTS............................................................................................................9
       3.1.1 Setup..................................................................................................................................10
   3.2 INTERVIEWS................................................................................................................................10
   3.3 LITERATURE REVIEW...............................................................................................................11

4 LITERATURE REVIEW....................................................................................................................12
   4.1 CREATIVE CODING.................................................................................................................12
   4.2 FRAMEWORKS FOR CREATIVE CODING.............................................................................14
       4.2.1 Web-Based Frameworks....................................................................................................15
   4.3 INTRODUCTION TO P5.JS.........................................................................................................17
   4.4 SUMMARY..................................................................................................................................18

5 RESULTS...........................................................................................................................................19
   5.1 CODING EXPERIMENTS............................................................................................................19
       5.1.1 Selected Sketches................................................................................................................19
       5.1.2 Development Diary............................................................................................................21
       5.1.3 Static Code Analysis............................................................................................................22
       5.1.4 Performance Analysis........................................................................................................25
   5.2 SURVEY.....................................................................................................................................28
       5.2.1 “How much have you used p5.js?”..................................................................................28
       5.2.2 “What have you used p5.js for?”.....................................................................................28
       5.2.3 “What other similar creative coding tools for the web have you used?”..........................29
       5.2.4 “How would you compare p5.js to other similar tools that you have used?”...............29
   5.3 INTERVIEWS............................................................................................................................30
       5.3.1 Background.......................................................................................................................30
       5.3.2 How p5.js Can Be Used.....................................................................................................31
       5.3.3 Why Choose p5.js?.............................................................................................................32
       5.3.4 Working With p5.js............................................................................................................33
       5.3.5 Comparing p5.js With Other Tools for Creative Coding..................................................34
       5.3.6 Final Thoughts..................................................................................................................35

6 ANALYSIS.........................................................................................................................................37
   6.1 MAIN FINDINGS.......................................................................................................................37
       6.1.1 Usability............................................................................................................................37
       6.1.2 Performance......................................................................................................................37
       6.1.3 Use Cases..........................................................................................................................38
   6.2 ANSWERS TO THE RESEARCH QUESTIONS..........................................................................39
   6.3 VALIDITY THREATS..................................................................................................................40

7 CONCLUSION....................................................................................................................................42

8 FUTURE WORK..................................................................................................................................43

9 REFERENCES.....................................................................................................................................44

10 ANNEXES........................................................................................................................................47
TABLE OF FIGURES

Figure 1: Simple p5.js code example. Source: [33].................................................................17
Figure 2: Static example of Sketch 1......................................................................................19
Figure 3: Static example of Sketch 2......................................................................................20
Figure 4: Static example of Sketch 3......................................................................................20
Figure 5: Static example of Sketch 4......................................................................................21
Figure 6: Code snippets from Sketch 2, illustrating the difference in how objects are drawn
to the canvas..........................................................................................................................23
Figure 7: Code snippets from Sketch 1, illustrating the difference in how the draw() functiona
lity is implemented...................................................................................................................23
Figure 8: Code snippet from Sketch 1, showing the additional setup needed in the plain
JavaScript version to manage the mouse location location. Variable c holds the canvas
element........................................................................................................................................24
Figure 9: Answers to survey question 2. Some answers are too long to fit in the graph,
instead the full answers can be found in Appendix C............................................................29

TABLE OF TABLES

Table 1: Creative coding frameworks for the desktop environment........................................14
Table 2: Creative coding frameworks for web browsers..........................................................16
Table 3: Comparison of the amount of code in the p5.js and plain JavaScript versions of the
coding experiments...................................................................................................................25
Table 4: Runtime performance metrics from the coding experiments, measured in the
Chrome web browser................................................................................................................26
Table 5: Memory usage results from the coding experiments..................................................27
GLOSSARY AND ABBREVIATIONS

API
Application Programming Interface

Canvas
An HTML5 element that allows for drawing graphics in the browser [1]

DOM
Document Object Model

FPS
Frames per second

IDE
Integrated Development Environment

Sketch
In creative coding tools, such as p5.js, a sketch refers to a small coding program

Visual programming language (VPL)
A tool where programs are created through a graphical interface rather than writing instructions in code [2]

WebGL
JavaScript API for 3D and 2D graphics rendering [3]
1 INTRODUCTION

1.1 Background

Creative coding is the practice of writing code primarily for an expressive purpose rather than a functional one and combines the skills of programming with creative arts. It is commonly associated with writing code for art installations, generative artwork, music and data visualizations.

The concept of using computers for creative purposes has been around since the 1960s [4], but the term creative coding has only started being used in the last 15 years, as there has been an increased interest in using programming in art projects [5]. A definition of the term that covers a wide range of activities is “programming for personal expression” [6], that is, expressing oneself creatively through code. As opposed to standard software engineering practice, creative coding is more about exploring code and processes than solving tasks. The term covers both the use of visual programming languages, such as Max/MSP, as well as traditional programming languages, such as Processing [4].

Processing¹ is one of the most popular languages used in creative coding and one of the earliest languages created specifically for this purpose. Casey Reas and Ben Fry were doing research at the Aesthetics and Computation group at MIT Media Lab, led by John Maeda [7], which led to the creation of Processing in 2001 [8]. Processing is intended both as a tool for designers and artists, as well as a suitable environment for anyone wanting to learn to code. The language and its IDE are based around the concept of sketches, which in visual arts are used to quickly test out a variety of ideas and through that develop the project. In the same way, Processing allows for quick iterations and for the practitioner to see the results immediately, without needing to create much overhead [9].

However, as it is built with Java, Processing projects cannot directly be put on the web. Previously this has been solved through the use of Java applets, small programs that could be integrated on websites, but these are practically defunct on the web nowadays [10]. Instead, a modern solution is using a JavaScript library called p5.js. It was created by Lauren McCarthy and the first beta version was released in 2014. It is now being developed as an open source project and is an official part of the Processing Foundation.

There are other Web alternatives for creative coding, a list of which is available at [11]. None of them are related to Processing, except for a library called Processing.js, which is no longer being supported [12].

1.2 Goals

This thesis investigates the p5.js JavaScript library. It looks at what can be accomplished with it and in which cases it might be used. The main focus is on the pros and cons of using p5.js in web graphics tasks. Another point of focus is on how the web can be used as a creative platform with tools like p5.js.

The goals are to provide an overview of p5.js and an evaluation of the p5.js library as a tool for creating interactive graphics and animations on the web.

¹https://processing.org/
1.3 Motivations and Values

Modern websites need to fulfill various and sometimes contradicting requirements. They need to have an immediate visual appeal to keep visitors on them. Aesthetics is an important factor in providing credibility for the website. Eye-catching visuals, such as rich graphics and animations, can be a good way of achieving this appeal and credibility [13]. On the other hand, websites need to perform well and keep the size of resources sent over the network low, especially as many visitors today use mobile devices. Processing and by extension p5.js creates graphics and animations mostly through nothing but code, which might help meet both the performance and visual requirements of websites.

Had Processing been invented today it probably would have been created in JavaScript, as the Web has become such a dominant platform and JavaScript the main language of the web browser [10]. However, there is a lack of academic studies on p5.js as the main focus, while there do exist studies on other similar tools, such as C4 [14] and Gibber [15]. This might be because p5.js is relatively new. This raises the question if the library is noteworthy enough to be studied. One measure of popularity is to look at the number of stars a project has on GitHub (admittedly not very scientific but still indicative), and p5.js is at the top of the list when searching for “creative coding” (as of Jun. 3rd, 2019) [16]. This means that there is no doubt that there is interest in this library. A study focused on it can thus be considered of merit.

Still, the lack of academic material on the p5.js library might be a challenge in completing this thesis. To counter this, sources that cover Processing are used as well, as p5.js is modeled after Processing and, therefore, very similar.

Processing and p5.js are tools primarily aimed at non-programmers such as artists and students in beginner computer science classes [17]. p5.js might, therefore, be overlooked as a professional-grade tool for real-world web design, which would increase the value of a study on it. On the other hand, a tool that is aimed strongly at beginners might be limited in its capabilities or efficiency. The analysis of p5.js needs to include a thorough and critical look at this potential weakness.

It is possible that even though using a framework has more advantages than using plain JavaScript, the latter is still chosen for a project, for reasons such as popularity and maturity. This was noted in [18] and is a point worth considering in the comparative study done here.

By answering the research questions laid out above, this thesis can help web developers evaluate whether a creative coding library like p5.js is a good fit for them and their project.

1.4 Scope

The main method of investigating p5.js will be to compare it with other similar frameworks as well as plain JavaScript. The focus will be on the end-user, i.e. the developer who decides on what technology to use for a project. The thesis will not examine the source code of the library. Comparisons will be made regarding usability, readability, and performance. Aspects like stability and longtime sustainability, which can be relevant to consider in regards to a library, are not included.
When investigating p5.js, this thesis will focus on its graphical functionalities in a web browser context. Interaction with the DOM or audio capabilities require add-on libraries to p5.js and is not included in the study.

Although games could be considered a form of creative coding and p5.js has an add-on library that adds features for aiding it, game development is not considered in this thesis.
2 Research Questions

RQ1: How does p5.js compare to plain JavaScript as a tool for web graphics, from the following perspectives:
   a) Which alternative is easier to develop with, for a proficient web developer with no previous experience of the p5.js library?
   b) Which alternative leads to more readable code?
   c) Which alternative leads to a smaller amount of code?
   d) Which alternative loads faster in the browser?

This question wants to figure out whether using p5.js gives a developer any advantages over using plain JavaScript. This might be the first question a developer asks themselves when looking into using p5.js in a project. The focus is on web graphics because that is what p5.js is mainly used for. By dividing the question into four subquestions the comparison points become clear and the answers should cover many of the concerns a developer has when making decisions regarding web technologies. Readable code means that the code should be easy to understand for a developer reading it for the first time.

RQ2: How does p5.js compare to other creative coding frameworks on the web?

p5.js is not the only web-based framework for creative coding. Through investigating this question it will be made clear which its main competitors are and what the strong and weak sides of p5.js are. This helps answer the question of what choosing p5.js over an alternative framework will bring to a project. This question does not define which frameworks p5.js should be compared with and instead leaves it up to the research to identify which are the most important ones.

RQ3: How has p5.js been received and used by the creative coding community in web design, web graphics, and creative projects?

It is one thing to compare a tool with others and another to actually see what has been made with the tool. This gives further insight into the library and its use cases.
3  RESEARCH METHODS

3.1  Coding Experiments

To answer RQ1, four sketches were created in two versions, one that uses only plain JavaScript and one that makes use of the p5.js library. The reasoning was that greater variety could be achieved by creating several smaller sketches rather than one larger project. Sites like Codepen\(^2\) and OpenProcessing\(^3\) were browsed to find suitable code samples to start from. Finally, two sketches were selected that were written in plain JavaScript, one that was written using p5.js and one that was originally written in Processing and converted to p5.js. The goal was then for the researcher to rewrite each sketch in the other version, not necessarily in a completely identical way but at least in the way it looks and behaves.

A development diary with reflections on the process was kept and analyzed afterward, focusing on which approach was easier to develop with (RQ1a). The starting point was similar to that of the average web developer in that the researcher is proficient in JavaScript but had not worked with p5.js. The final code from both projects were compared on their readability (RQ1b) and amount of code (RQ1c). Readability means that the code should be easy to understand for a developer reading it for the first time. This includes meaningful naming of variables and functions and a clear code structure. The researcher that performed the code readability analysis is the same person that wrote the code, but they attempted to have a more detached, distanced perspective during the analysis. Finally, the resulting sketches were performance tested on file size, runtime performance, and memory usage (RQ1d).

By choosing two examples in plain JavaScript and two in p5.js and then rewriting them in the other version, a fair comparison could be made regarding the development process and performance assessment.

The selection of the four sketches was made based on a couple of criteria. As p5.js is primarily used for creative expression, the selection should emphasize interactive graphics and animations. Data visualization was equally thought of as a reasonable use case. The sketches should not be trivially simple but not too long and complex either as that would make comparison difficult. If possible the JavaScript part of the code should be contained in a single file. Finally, each selected sketch should have some unique aspect to it in comparison with the other selections, to enable exploration of different code concepts.

It turned out that it was difficult finding creative code ideas written in plain JavaScript, as most examples used an external library. Data visualization examples could not be found at all in either plain JavaScript or p5.js. In the end, a coding example from a book teaching data visualization in Processing [19] was selected as the data visualization part of the experiment and the rest of the sketches were more creative expressions.

\(^2\)https://codepen.io/
\(^3\)https://www.openprocessing.org/
3.1.1 Setup

The p5.js versions of all sketches use p5.js version 0.7.3\(^4\), which was released on January 20\(^{th}\), 2019. The downloaded package was “p5.js complete”, which includes the add-on libraries for DOM manipulation and sound; however, these were not used in any of the sketches and are not even loaded into the projects. This means that any features and performance characteristics specific to these add-ons are not included in the research.

All sketches have been tested in Firefox version 66.0.3 and Chrome version 73.0 using their respective developer tools\(^5\), on a 2017 iMac\(^6\) running macOS version 10.13.6. In order to see the size of the network requests in the browser, the sketches were put on a local Apache HTTP server. This is only needed for the analysis and has nothing to do with the nature of the projects. Sketches were run in incognito mode so that the browsers had no possible extensions affecting the results [20]. The developer tools section were in an undocked separate window, allowing the sketches to run in a full-sized browser window. Even though the developer tools allow for simulating constrained networks or different user agent CPUs, no restrictions were applied to the performance analysis. This is motivated by the fact that the purpose of the experiments is to investigate the difference in performance between two versions of the same sketch and not to evaluate their overall performance. For the same reason, tests have not been performed on mobile platforms or older machines.

3.2 Interviews

RQ2 and RQ3 were investigated through a survey and interviews with members of creative coding communities. Requests for interviews were posted on various online discussion forums and groups related to creative coding on Reddit\(^7\), Facebook\(^8\), and Slack\(^9\), as well as the official forum for the Processing Foundation\(^10\).

From this first call for interviews, three people agreed to participate. In order to increase the number of interviewees, a survey was created and a link to it posted on the same locations as before. As the purpose of the survey was primarily to recruit people to be interviewed, it was kept short and its questions were partially repeated in the interviews. Additionally, the survey included an invitation to a follow-up interview. The survey form can be found in Appendix B.

All of the survey respondents who indicated that they would be willing to participate further were contacted. Out of these, five people responded and agreed to an interview. In total, eight people were interviewed, seven of them through a recorded Skype conversation and one through a written email interview.

It should be noted that because most of the interviewees were recruited through the survey, their answers will be somewhat duplicated in their interview answers. However, they went into much more nuance and detail in the interviews.

\(\text{\textsuperscript{4}}\)https://github.com/processing/p5.js/releases/tag/0.7.3
\(\text{\textsuperscript{5}}\)For Chrome: https://developers.google.com/web/tools/chrome-devtools/
\(\text{\textsuperscript{6}}\)Exact specifications: https://support.apple.com/kb/SP759
\(\text{\textsuperscript{8}}\)https://www.facebook.com/groups/creativecodingp5/
\(\text{\textsuperscript{9}}\)https://genart.slack.com/ and https://creative-dev.slack.com
\(\text{\textsuperscript{10}}\)https://discourse.processing.org/
Interviews were primarily done over Skype and were planned for a duration of around 30 minutes. The audio was recorded and transcribed afterward. The interviews were set up in a semi-structured format. A pre-written question guide was used as the basis for the interviews, but follow-up questions could be added and questions could be altered depending on the subject and in what direction the interview went. Interview questions were split into two main themes. The first part (which relates to RQ3) focused on why and how the interviewees have used p5.js and how it has been to work with it. The second part (which relates to RQ2) focused on how the interviewees compare p5.js to other similar tools for working with web graphics and creative coding. The interview question guide is included in Appendix D and the exact questions asked in each interview can be found in Appendix E.

The only thing that was required of the survey respondents and the interviewees was that they had at least some experience with p5.js. Other than that, there were no limitations based on their backgrounds or interests. The interview requests were posted on creative coding community sites, meaning that only people with a self-identified interest in creative coding saw the requests.

3.3 Literature Review

A literature review was done in February 2019 on the topics of creative coding, Processing, and p5.js, to provide a foundation for the research in this thesis. The primary search strategy was to use the Scopus\(^\text{1}\) database of peer-reviewed articles, which ensures that the material holds an acceptable level of quality and accuracy. O’Reilly\(^\text{2}\) was used to search for textbooks on p5.js and Processing. The DiVA\(^\text{3}\) database was searched as well but did not return much of interest. A few articles in the search results could not be accessed through Scopus. Instead, these could be found on Google Scholar\(^\text{4}\).

The searches were done with the following parameters:

- search string: p5.js; search fields: All
- search string: “creative coding”; search fields: Article title, Abstract, Keywords
- search string: “creative coding” web; search fields: Article title, Abstract, Keywords
- search string: processing creative coding; search fields: Article title, Abstract, Keywords; Subject area: Computer Science, Engineering, Arts and Humanities

If there were many search results returned, results could be sorted by either “Relevance” or “Cited by (highest)”, to find the most significant articles. Abstracts were then analyzed to determine the relevance of each article to this thesis.

Some articles were found through the snowball sampling technique. When the articles were being read through, their reference lists were checked for possibly usable sources for this thesis.

\(^\text{1}\)https://www.scopus.com/
\(^\text{2}\)https://learning.oreilly.com/
\(^\text{3}\)http://bth.diva-portal.org/
\(^\text{4}\)https://scholar.google.com/
4 LITERATURE REVIEW

A lot of the material on creative coding and Processing focuses on their usage in educational contexts [6], [21], [22], [23], or as examples in more general discussions on creativity in programming [4], [5], [9], [24], [25], [26], [27], although there are some articles discussing Processing in a more general way [17], [28]. One of the seminal works on creative coding [7] unfortunately turned out not to be very useful in the context of a literature review, as it is mostly a visual catalog of art pieces in the creative coding discipline. There are some articles that focus on other tools in creative coding [14], [15], [29], [30] and some that include p5.js as a tool for various tasks [31], [32], but they do not really describe or discuss the library itself. In fact, there does not yet exist any academic studies on p5.js as the main focus (based on searching Scopus and Google Scholar). Another recent study on general web frameworks points out the general lack of research into more recent frameworks [18]. The literature that exists on p5.js is general guidebooks such as [33]. There exists similar books on Processing [8], [19], [34], [35]. These are mostly educational and instructive, aimed at beginner programmers, rather than being descriptive about the concept of creative coding, Processing, and p5.js.

This section will describe the context around p5.js by looking at the field of creative coding, go through some popular frameworks both on and off the web, and finally give a short introduction to p5.js. Some of the included references focus on subjects that are not relevant to this thesis, but they have still contained some information of interest.

4.1 Creative Coding

One of the first languages for creative coding was Logo. It was originally used for controlling a drawing robot but was later given capabilities to draw graphics on computer monitors, which increased its popularity during the 1980s and made it the subject for a lot of research at MIT. Later, during the 1990s, the Aesthetics and Computation Group at MIT developed the Design By Numbers environment. Students from this era later went on to create the Processing language a few years later. [15]

Creative coding projects (and the tools that are used in them) are usually focused on either visual expressions, audio, microcontrollers, or a combination of these; the common ingredients being computations of some kind and a sense of aesthetics. A related field is that of aesthetic computing [26], which is focused on applying elements of art and design to the field of computing, as the concept of aesthetics is much wider in art than it is in computing. Thinking about programming in the context of art brings about a new understanding of code as a material and a medium, just like paint to a painter or rock to a sculptor. This is something that has been discussed in numerous articles [6], [7], [21], [28], [36].

Another interesting, related aspect is that of authorship. Can computers create art and does that make them artists? The field of generative art is based on programming a set of tools and rules, and then letting the computer create something based on those premises. Joshua Davis considers this art as well, and its programmer an artist, in the same way that Jackson Pollock called himself a painter even though he mostly never touched the canvas with his brush [7].
Furthermore, code can be used to enhance traditional forms of art to make them interactive, which is investigated further in [28] and [37]. The latter describes how the visuals in a theatre performance were made interactive, reacting to the live audio, through using Processing. When all elements of the performance react to each other in real-time, it enhances the sense of the performance feeling alive and real, instead of a predetermined, well-rehearsed presentation. Additionally, it makes the performance a more playful experience for both performers and audience.

Communities whose interests to a large extent overlap with creative coding are those of the maker culture and the demo-scene. Originally mainly focused on tinkering with electronics platforms such as Arduino, the maker culture is now an inclusive DIY community for all kinds of crafting endeavors [4]. Its central principles are exploration and learning-by-doing, with a prototype often being the main goal of a project. Even closer to creative coding is the demo-scene subculture, whose members create audio-visual presentations, called demos, using code which is run in real-time [36]. These demos test their creators’ technical skills and aesthetic sense. The demo-scene can be seen as a form of creative craftsmanship, where code is the design material.

There exists a number of subtypes of creative coding, ways of practicing it, that have been identified in [24] and later expanded on in [9]. Bricolage programming, a term that has been used since the early 1990s, refers to a style of coding where the programmer constantly re-evaluates their work after every modification, an extreme form of iterative development. As it resembles the creative process of artists in that the creator is open to changes and never stops exploring new possibilities in their work, it is a style of programming that fits well for creative coding projects [5]. Sketching draws on an analogy with visual artistry, as the method consists of quickly throwing together prototypes for an idea, similar to how drawn sketches are created and used. This concept is one of the cornerstones for several popular tools in creative coding, such as Max/MSP and Processing. In live coding, the act of writing code is performed live and becomes part of the artwork as performance. This demands that the coding environment supports interpreting the code as it is written, without needing to restart or compile, and is supported in e.g. Max/MSP and SuperCollider. Code-bending, which takes its name from circuit-bending, means opening up and repurposing the internal APIs of open source software, which allows for new and unexpected uses of its source code. Finally, hacking is a practice where pre-existing software is modified or interfered with to change its intended behavior. It is similar to code-bending but hacking often requires deep knowledge of the computers and machine-level programming. All of these concepts can be contrasted with the standard notion of software engineering, a disciplined approach to software development where specifications are created, implemented and evaluated, following some kind of methodology, such as the waterfall model or an agile method. Although programming itself is inherently creative, which John Simon Jr. talks about in [7], there is a need for additional playfulness in programming, as shown by the research in [38].

Educators have taken an interest in creative coding as well, as a method for teaching people how to code. For young people, media is at the core of their technology interests and learning programming in a context that utilizes graphics and audio increases their motivation for learning [6], [23], [39]. At the same time as creative coding is a way of creating art through code, it is equally a way of understanding digital technology through creative projects [22]. Many of the tools used in creative coding, such as Processing, Arduino and Max/MSP, focus on a low barrier to entry and reduced complexity, as the people using these tools want to get started quickly and work by developing rapid prototypes. Simplicity is accomplished with the principle of convention over configuration, i.e. the platform makes assumptions about the setup of the tool and does not bother the user with all kinds of
options [4]. The aim is that with tools that are easier to use, artists will be more interested in learning them and starting to use them in their practice. Additionally, it makes these tools excellent for teaching anyone entry-level coding.

Although creative coding has been described here as a form of creative expression with the main purpose of creating art, a number of more practical applications has been considered, as in [28], which talks about its potential use in marketing and data visualization.

4.2 Frameworks for Creative Coding

What follows is a short, non-exhaustive list of some of the more popular tools and languages used for creative coding. The selection is based on which frameworks that have come up most often in research or there seems to be a consensus about regarding their noteworthiness [4], [5], [11], [14], [15], [25], [35], [40]. Table 1 summarizes these frameworks.

<table>
<thead>
<tr>
<th>Language based on / built in</th>
<th>Mainly used for</th>
<th>Strengths / weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing</td>
<td>Java</td>
<td>General purpose</td>
</tr>
<tr>
<td>OpenFrameworks</td>
<td>C++</td>
<td>General purpose</td>
</tr>
<tr>
<td>Cinder</td>
<td>C++</td>
<td>High-performance graphics</td>
</tr>
<tr>
<td>C4</td>
<td>Swift</td>
<td>Mobile-specific features</td>
</tr>
<tr>
<td>SuperCollider</td>
<td>Own programming language</td>
<td>Audio programming, live coding</td>
</tr>
<tr>
<td>ChucK</td>
<td>Own programming language</td>
<td>Audio programming, live coding</td>
</tr>
<tr>
<td>Max/MSP</td>
<td>Own VPL</td>
<td>Music and audio</td>
</tr>
<tr>
<td>vvvv</td>
<td>Own VPL, includes C# textual environment</td>
<td>Live coding</td>
</tr>
<tr>
<td>Arduino</td>
<td>Wiring (C/C++ dialect)</td>
<td>Electronics / physical computing</td>
</tr>
</tbody>
</table>

Table 1: Creative coding frameworks for the desktop environment

**Processing**

A coding environment built on top of Java that consists of the Processing language and a simple IDE. It is the framework that has perhaps the strongest focus on educational aspects, which, together with the fact that it is one of the earliest tools for creative coding, explains why it is one of the most well-known frameworks [40]. It has a minimal API and a simple development environment, which helps produce quick results and prototypes. Processing is the basis for the p5.js JavaScript library and their syntaxes are almost identical.
OpenFrameworks
This framework has the same origin as Processing, the MIT Media Lab, although OpenFrameworks is instead inspired by a C++ library called ACU. As it is written in C++, it allows for more low-level, powerful code, but it doesn’t have an IDE like Processing does, making it slightly more complicated to use.

Cinder
Built in C++, Cinder was created specifically to respond to needs for high-performance graphics by taking advantage of the features of OpenGL, a cross-platform library for rendering 2D and 3D graphics and interacting with the GPU. This makes Cinder a relatively hard framework to learn.

C4
A framework created specifically for iOS, built on top of the Swift language. It takes advantage of and provides easy access to mobile-specific features such as multi-touch, compass, motion and light sensors. It has a special focus on managing media objects easily. [14], [29]

SuperCollider
An audio programming environment that consists of a server to which commands are sent and processed in real-time, allowing for live coding. However, it does not include any graphical capabilities.

ChucK
Another language that allows for on-the-fly audio programming. It sticks out by using time as a fundamental part of the code, meaning that time has to be explicitly advanced in the instructions.

Max/MSP
A visual programming language, in which boxes, that each represents some kind of function, are placed on a canvas [24]. Wires are then drawn to interconnect these boxes and when data (some kind of input) enters the program, it is routed through the boxes, getting modified in each one, until it reaches the end and an output is produced. Max/MSP is one of the first tools that embraced the concept of sketches and is primarily used for music and audio purposes, although it can be extended to include video as well. As opposed to many other creative coding frameworks, Max is not open source.

vvvv
A visual programming language similar to Max/MSP, but it includes a C# textual development environment as well. It can be used for live coding a well but is exclusively available on Windows.

Arduino
A hardware platform for microcontrollers that includes its own programming language, Wiring, which is a dialect of C and C++, and an IDE that is based on the Processing IDE. It is meant as a beginner-friendly way of getting started with electronics and making physical objects interactive. The community around it invites people to come up with creative ideas for using the Arduino controllers.

4.2.1 Web-Based Frameworks
The frameworks that have been listed above are all desktop applications. In more recent years, creative coding tools that are built in JavaScript and run in the web browser have started to appear as well. As browsers have matured, their performance
has improved and this, in combination with the ubiquity and portability of web browsers, makes them an attractive platform for creative coding tools [30].

As mentioned at the beginning of this section, the research into web-based tools for creative coding tools is sparse. The following is a list of notable web-based frameworks. It has been distilled from a couple of sources [11], [41], [42], which although they might lack academic value should, as an aggregated selection, still provide some hint regarding popular frameworks. Table 2 summarizes these frameworks, which are all written in JavaScript.

<table>
<thead>
<tr>
<th></th>
<th>Mainly used for</th>
<th>Notable for</th>
</tr>
</thead>
<tbody>
<tr>
<td>p5.js</td>
<td>General purpose</td>
<td>JavaScript alternative to Processing</td>
</tr>
<tr>
<td>three.js</td>
<td>3D graphics</td>
<td>Simplifies working with WebGL</td>
</tr>
<tr>
<td>Paper.js</td>
<td>Vector graphics</td>
<td>Set of tools for scripting vector graphics</td>
</tr>
<tr>
<td>Sketch.js</td>
<td>General purpose</td>
<td>Minimal size</td>
</tr>
<tr>
<td>D3</td>
<td>Data visualization</td>
<td>Relatively complex</td>
</tr>
<tr>
<td>Gibber</td>
<td>Audiovisual live coding</td>
<td>Live coding environment</td>
</tr>
</tbody>
</table>

Table 2: Creative coding frameworks for web browsers

**p5.js**
This is Processing reimagined in JavaScript. It has been created by Lauren McCarthy and is developed by an active open source community, p5.js works as a separate web alternative to Processing, rather than being built on top of Processing or encapsulating Processing code.

**three.js**
The most well-known 3D graphics JavaScript library, using WebGL as the default renderer. Complex 3D graphics concepts, such as shaders, camera perspectives, and texture mapping, are made much simpler with this library.

**Paper.js**
A tool set for scripting vector graphics in the HTML5 canvas. Paper.js includes its own DOM called a Scene Graph for managing layers and other graphical elements. [43]

**Sketch.js**
A tiny creative coding framework, currently around 2 kB when gzipped. Still, it provides basic features for creating graphics and animations and managing input. [44]

**D3**
The number-one tool for data visualization on the web. D3 is popular both for creating traditional graphs and charts and more complex visualizations. It is primarily aimed at creating customized visualizations and is known for being both powerful and complex in comparison to other data visualization tools.

**Gibber**
An audiovisual live coding environment. Gibber is not mentioned in any of the sources that formed this list, but it is still worth mentioning, as what it brings is still quite unusual in the browser environment. Furthermore, its creator has submitted a number of research articles on its development, which gives interesting insights into the details of it. The main project is a standalone service, but it has been ported to work with p5.js as well. [15], [30]
4.3 Introduction to p5.js

As this thesis is mainly focused on p5.js, this section will give a short introduction to the library. This is for readers to get a grasp of what the library looks like and for those that are interested to learn the basics and start experimenting with it. For further reading, [33] or [45] is recommended, which are the sources for this section as well.

For drawing and all other basic functions, the base p5.js library is enough. However, there is a growing number of add-on libraries available, which add support for functionality such as DOM interaction and sound.

Using p5.js is as simple as including the p5.js file, that is available on their website [45], in the HTML of the project. There is an online editor available\(^\text{15}\), allowing for a quick start to the library.

There are two main functions that are part of any p5.js program. All code that sets up the project is put inside setup(). This code is the first to run (except for any code that is put in the global scope outside it) and it runs once. After that, the draw() function runs repeatedly, once for every animation frame. The standard screen refresh rate is 60 Hz, meaning that the code inside draw() runs around 60 times per second, although this can be easily adjusted in the code. It is with the code in draw() that objects are animated. Note that these functions should be declared in the code, but not called, as that is handled by the library.

Any code that includes a function call to the p5.js API or uses library-specific variables must be placed inside specific p5.js functions, such as setup() or draw(), or inside user-created functions that get called from a p5.js function, as the API is not exposed to the global scope.

Let’s look at an example from [33], shown in Figure 1, and go through it step-by-step:

```javascript
function setup() {
  createCanvas(480, 120);
}

function draw() {
  if (mouseIsPressed) {
    fill(0);
  } else {
    fill(255);
  }
  ellipse(mouseX, mouseY, 80, 80);
}
```

Figure 1: Simple p5.js code example. Source: [33]

As mentioned, the code inside setup() runs first. createCanvas() adds a new canvas element to the page, with the size in pixels sent in as the parameters.

Each time draw() is run, it starts by checking if the mouse button is currently being pressed by reading the p5.js variable mouseIsPressed. Depending on the outcome, the color that will be used to fill objects is sent in fill(). In the above case, only a single parameter is sent to fill(), which means that the color will be a grayscale value between 0 and 255.

\(^{15}\text{https://editor.p5js.org/}\)
This does not yet draw anything to the screen. That happens instead when `ellipse()` is called, which draws an ellipse with a width and height of 80 pixels, effectively creating a circle. The object is drawn at the current mouse position, whose coordinates are stored in the `mouseX` and `mouseY` variables.

When `draw()` runs the next time, a new circle will be drawn, leaving previously drawn objects underneath on the canvas.

### 4.4 Summary

The concept of creative coding brings interesting, new perspectives on the art of programming. Thinking about code in an aesthetic context reveals that code can be used as an artistic medium to explore and express ideas, just as a brush and paint for a painter. Creative coding can turn traditional art interactive and it can be used to create generative art, where the computer becomes the creator. Furthermore, it can be used in more practical applications, such as marketing and data visualization. Creative coding is becoming increasingly popular in beginner-level programming courses as well, as learning to code by using graphics and audio and being able to create prototypes quickly can be highly motivating. Creative coding tools often provide a low barrier to entry and offer simplicity through the convention over configuration principle.

Closely related communities are those of the maker culture and the demo-scene, where exploration and creative craftsmanship are important principles, with code as the design material and a prototype often the main goal. Some subtypes of programming, such as bricolage and sketching, are similar to the creative process of artists and therefore fit well into the spirit of creative coding. These programming practices can be contrasted with traditional software engineering where the focus is more on following specifications and solving tasks.

Frameworks and tools used by creative coders range from more traditional coding environments, such as Processing and OpenFrameworks, to visual programming languages such as Max/MSP and vvvv, and even hardware-focused platforms such as Arduino. In recent years creative coding tools that run in the web browser have become more popular because web browsers are ubiquitous today and the browser environment offers portability between platforms. Examples of web-based tools are p5.js, three.js, Sketch.js, and D3.
5 RESULTS

5.1 Coding Experiments

In this section, the results for the coding experiments are presented. First, the selected sketches will be shortly presented and motivated. Then in 5.1.2, thoughts and insights during development are presented by going through the main points from the development diary. In 5.1.3, the code is statically analyzed on readability and amount of code and 5.1.4 goes through different aspects of the performance measurements.

5.1.1 Selected Sketches

The source code for all the sketches can be found in Appendix A. The original sketches in the project repository have not been modified from the original sources in any way that affects performance. Only superficial details have been changed, such as file and directory names and comments to help navigate and understand the examples, as well as one case where deprecated syntax was updated.

Note that the different sketches have different licenses associated with them. Details can be found in Appendix A.

Sketch 1: “Attractor 0” by Masaki Yamabe\(^{16}\)

This sketch was originally created in p5.js and was rewritten in plain JavaScript. It uses 1000 particles that are attracted to the position of the mouse and creates an interesting visual and interactive sensation while relying on a rather small amount of code, which is why it was chosen. To try it out, simply move the mouse around the window.

![Figure 2: Static example of Sketch 1](https://www.openprocessing.org/sketch/394718)

Sketch 2: “circular trail” by Ana Tudor\(^ {17}\)

This sketch was originally created in plain JavaScript and was rewritten in p5.js. It creates a large number of particles (over 4000) and the animation is continuous, but there is no interaction. However, the animation responds nicely to screen resizing and

\(^{16}\)Source: https://www.openprocessing.org/sketch/394718

\(^{17}\)Source: https://codepen.io/thebabydino/pen/avwKGlw
if the screen is resized to a very small size, particles reaching the bottom of the screen start bouncing back up.

Figure 3: Static example of Sketch 2

Sketch 3: “never force” by Gerard Ferrandez

This sketch was originally created in plain JavaScript and was rewritten in p5.js. It creates simple visual shapes with a pseudo-3D effect and responds to the movement of the mouse. It seemed possible to move it over to p5.js and it even used similar function names. Thus, it would be interesting to see how well it would map to a new coding environment. To try it out, simply move the mouse around the window to see the visuals change. Clicking and holding the mouse button extends the shapes.

Figure 4: Static example of Sketch 3

Sketch 4: “US map example” by Ben Fry

This sketch is taken from chapter 3 in Visualizing Data by Ben Fry [19] and was originally created in Processing. The specific part used is found in the subdirectory “step15_framerate” in the source code for chapter 3. It was first converted to p5.js

---

18 Source: https://codepen.io/gel1doot/pen/jEgL.Ko
19 Source: https://bhfry.com/writing/archives/3/
before being rewritten in plain JavaScript. The first conversion was not too extensive as Processing and p5.js have very similar APIs [46].

It is a rather simple program, where values for all U.S. states are displayed as circles. Positive values are shown as blue circles while negative values are shown as red. The larger the circle, the larger the value is. By hovering the mouse over a circle the name of the state and its value is displayed. By pressing Space bar random new values are assigned to each state. This represents the possibility of the visualization getting live updates as data changes.

The reason for selecting this sketch is that no data visualization examples could be found in p5.js, but it was still thought to be of interest to include a more practical example.

![Figure 5: Static example of Sketch 4](image)

5.1.2 Development Diary

The first step with each sketch was to understand what the original code did, in great detail. Especially in Sketch 2 and Sketch 3, variables had not been named in an easily understandable way, meaning that during rewriting variables were renamed in order to understand what they contained. The next step was to first as directly as possible try and translate from the original version to the other and then see if the code could be refactored.

Sketch 1 got its first draft up and running in the plain JavaScript version without notable issues. The behavior of the particles seemed to be similar, but something in the visuals was not correct. Lines were too wide and too bright and seemed to move too quickly. After quite some troubleshooting it was discovered that alpha values were using different scales in p5.js than in plain JavaScript. The former uses a scale of 0-255, while the latter works with CSS colors which use an alpha value scale between 0 and 1. The former fits better with the RGB scale but the latter is what web developers are normally used to. Correcting the alpha value fixed the issue and work on Sketch 1 was complete.

For Sketch 2 the task was reversed; a plain JavaScript sketch now had to be moved over to p5.js. By this point, it started to become obvious that p5.js primarily revolves around the HTML5 canvas element. The original code demands many lines to

---

20Code excerpted from *Visualizing Data*. Published by O'Reilly Media, Inc. Copyright © 2006 Ben Fry. All rights reserved. Used with permission.
set up the canvas and draw elements to it, while these aspects are encapsulated in p5.js in function calls such as `createCanvas()` and `circle()`. The first draft of the rewritten version worked like the original except for the alpha values for each particle, which did not diminish over time as expected. It turned out that it was the alpha value scale that once again was causing the problem. Even though it had already been noted during the development of Sketch 1 the day before, the fact that p5.js uses another scale for this particular value continued to cause confusion. Another issue was that the p5.js version seemed to be performing considerably worse, especially when running in Firefox. Some time was spent on checking the code to see if part of the newly created p5.js code was poorly optimized, but nothing was discovered.

Sketch 3 in plain JavaScript works with the canvas and animation frames, and it turned out that p5.js could once again be used to good advantage in these aspects. Again there seemed to be a slight performance issue with the p5.js version, although only in Firefox.

Ending with Sketch 4, it was quite easy to understand what the original code was doing as the source contains a step-by-step explanation of it. The sketch first had to be converted from Processing to p5.js, which seemed to be an easy task at first [46]. However, there were a lot of surprising issues in getting the p5.js version to work, most of which seemed to be rooted in the researcher’s inexperience with Java and its differences to JavaScript. The details are outside the scope of this thesis, but one noteworthy aspect is that functions that work with loading file contents, such as `loadStrings()` and `loadImage()`, are asynchronous in JavaScript but not in Java. Therefore, they had to be put in a special `preload()` function in p5.js to ensure that the resources were loaded before the actual setup began. Once everything finally worked in p5.js, there were no notable issues in getting it rewritten in plain JavaScript, given all the experience from the previous sketches. For some reason that could not be discovered, the text and circle elements were a bit blurry both in the Processing and plain JavaScript version, but clear and sharp in the p5.js version.

5.1.3 Static Code Analysis

The focus for this section is which alternative that leads to a more readable code and a smaller amount of code. This does not imply better performing code, but less code might make it more approachable to a developer and easier to reason about, as long as it is not too short. Readable code means that the code should be easy to understand for a developer reading it for the first time. This includes meaningful naming of variables and functions and a clear code structure. The researcher that performed the analysis is the same person that wrote the code, but they attempted to have a more detached, distanced perspective during the static code analysis.

A general point to start with is that the differences between the code in p5.js and plain JavaScript are not that big. The core logic for the specific sketches does not differ much between versions. Differences can instead be found in how the sketches are framed and set up.

As p5.js primarily works with the canvas, there is no need for a variable holding the canvas rendering context object and using it to draw elements all the time. Instead of `ctx.fillRect()` (where `ctx` is the canvas context), in p5.js, `rect()` does the same thing. In that regard, p5.js provides for a cleaner looking code, although it can be confusing in the beginning for someone who is used to handling the canvas element. Additionally, p5.js creates the canvas element and adds it to the DOM, meaning there is no need to have the canvas tag in the HTML file.
One of the most notable differences between the p5.js and plain JavaScript versions is in how objects are drawn on the canvas. Consider the code excerpts in Figure 6, taken from Particle.draw() in Sketch 2, that both do the same thing, which is drawing a circle. Not only is the p5.js version much shorter here, but the code is more intuitive. Similar scenarios can be found in all four sketches.

```javascript
//Plain JavaScript version
ct.beginPath();
ct.arc(this.x, this.y, this.r, 0, 2*Math.PI);
ct.closePath();

//p5.js version
circle(this.x, this.y, this.r);
```

Figure 6: Code snippets from Sketch 2, illustrating the difference in how objects are drawn to the canvas

When using p5.js, function calls to the library can only be made inside specific p5.js functions, such as setup(), draw() or preload(), meaning that most or all of the setup part of the code is run inside setup(). In plain JavaScript, there is no such requirement and the easiest way is to run similar code in the global scope. Thus, p5.js forces a structure that makes the global namespace less polluted, which is good. However, if there are variables that need to be shared over different p5.js functions, they do need to be declared in the global namespace. This can be seen in all four sketches.

One of the most central functions in p5.js is draw(). The code inside draw() runs once every frame, which under normal circumstances means around 60 times per second (depending primarily on the refresh rate of the screen). It is through this function that objects are animated on the canvas. In plain JavaScript, a small setup is needed to get the same functionality. In the global scope of the sketch, a requestAnimationFrame() function call is needed, sending the function that should repeat every frame in as an argument. Inside this function that is called back later, the same requestAnimationFrame() call has to be made for the action to repeat continuously. An example of this difference can be seen in Figure 7.

```javascript
//Plain JavaScript version
function draw() {
    [...program-specific code...]
    window.requestAnimationFrame(draw);
}
window.requestAnimationFrame(draw);

//p5.js version
function draw() {
    [...program-specific code...]
}
```

Figure 7: Code snippets from Sketch 1, illustrating the difference in how the draw() functionality is implemented

Sketch 4 is shaped quite differently in the two versions. To a large extent, this has to do with the fact that local files are loaded into the program. p5.js has a function called preload(), which is guaranteed to run to completion before setup() is called. Therefore, slow file operations are done there and setup() can then safely act on the data. In the plain JavaScript version, this has to be done in a couple of steps. An asynchronous load() function contains await statements for each separate init() method. This method is itself asynchronous and contains an await statement to fetch
the data contents. As should be clear from this paragraph, the p5.js version is easier to reason about.

There are several helper functions and variables available in p5.js that make for shorter code. map() takes a number and remaps it from one range to another, which is cleverly used in Sketch 1 and Sketch 4. As both of them start as p5.js versions, additional code is needed when rewriting them to plain JavaScript. frameRate() is used in Sketch 4 to make the state transition animation more visible by setting the frame rate to 30 FPS, which constrains how often draw() is called. In the plain JavaScript version, frame durations have to be calculated and frame rate controlled manually. mouseX and mouseY hold the coordinate values for the mouse at the current frame. Three out of four sketches need to know where the mouse is located, and this is easily done on p5.js, while the plain JavaScript code needs additional setup for mouse movement, that can be seen in Figure 8. Sketch 3 uses a pointer object in the original plain JavaScript version, a structure that is not needed in p5.js. A small detail is that many methods and properties from the built-in Math object have been exported in p5.js in order for commonly used functions and constants, such as sin(), abs() and PI to be slightly easier available.

```javascript
//Additional setup for mouse events
var mouseX = 0;
var mouseY = 0;
c.onmousemove = (e) => {
  mouseX = e.pageX;
  mouseY = e.pageY;
}
```

Figure 8: Code snippet from Sketch 1, showing the additional setup needed in the plain JavaScript version to manage the mouse location location. Variable c holds the canvas element.

Sketch 2 and Sketch 3 are able to respond to screen resizing. In the original plain JavaScript versions, this is handled by adding an event listener to the resize event and handling the resized window in a callback function. In the p5.js version, the corresponding code has to be duplicated; first, it runs once in setup(), and then every time the window is resized in windowResized().

Finally, the differences between the versions can be quantified by counting the lines of code and the number of characters in the JavaScript files. The results are presented in Table 3. It should be noted that Sketch 4 only includes the main script files and that comments are ignored when counting the number of characters. As can be seen in the results, the p5.js versions consistently have smaller amounts of code.
<table>
<thead>
<tr>
<th></th>
<th>Sketch 1</th>
<th>Sketch 2</th>
<th>Sketch 3</th>
<th>Sketch 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code, p5.js version</td>
<td>61</td>
<td>111</td>
<td>119</td>
<td>105</td>
</tr>
<tr>
<td>Lines of code, plain JavaScript version</td>
<td>88</td>
<td>111</td>
<td>130</td>
<td>155</td>
</tr>
<tr>
<td>Difference in lines of code</td>
<td>-30%</td>
<td>0%</td>
<td>-8%</td>
<td>-32%</td>
</tr>
<tr>
<td>Number of characters, p5.js version</td>
<td>1201</td>
<td>2175</td>
<td>2934</td>
<td>2395</td>
</tr>
<tr>
<td>Number of characters, plain JavaScript version</td>
<td>1946</td>
<td>2223</td>
<td>3348</td>
<td>3269</td>
</tr>
<tr>
<td>Difference in number of characters</td>
<td>-38%</td>
<td>-2%</td>
<td>-12%</td>
<td>-27%</td>
</tr>
</tbody>
</table>

Table 3: Comparison of the amount of code in the p5.js and plain JavaScript versions of the coding experiments

5.1.4 Performance Analysis

Performance is measured and compared on three metrics: project file size, runtime performance, and memory usage. The project file size is measured by looking at the network monitor and comparing how big the resources being sent over the network are. This should be indicative of the load performance of each version, at least relative to each other. Runtime performance means how fast the code runs once it has been loaded into the browser and is up and running. The metrics used for evaluating this are the frame rate and looking at which parts of the code are most expensive in terms of time. Finally, memory usage will be measured by looking at how much memory is used by the sketches over time to see if there are memory leaks, i.e. if memory is continuously added without releasing unneeded memory. This could indicate improper memory garbage collection procedures. The test setup is described in section 3.1.1.

**File size**

As the project file size should not differ between different browsers, this test is only performed in the Firefox browser. Appendix A contains screenshots of the Network Monitor tool in Firefox being used for the different sketches. For the first three sketches, the results are similar; around 480 kB for the p5.js versions and around 34 kB for the plain JavaScript versions. As Sketch 4 loads additional resources, the total size lands on 605 kB for the p5.js version and 158 kB for the plain JavaScript version. The only real difference between the versions for all sketches is that the p5.js versions load the file p5.min.js, a minified version of the p5 library, with a file size of 450 kB.

**Runtime performance**

Runtime performance tests were done in both Chrome and Firefox in their performance tools. Measurement data was recorded for at least 10 seconds while letting the sketch run, interacting with the sketch if possible. The results were saved and can be found in Appendix A as JSON files. They can be imported in each browser’s Performance tool and viewed. For the runtime performance tests, the un-minified version of the p5.js library was used in order to get the proper variable names logged in the results [47].

As there is a lot of data and different ways of viewing the data available in the results, this section will only provide a summary and bring up notable details. Another point to be made is that the developers of p5.js recommend using Chrome for best performance [47]. Therefore, the results from Chrome will be the main point of focus,
with the results from Firefox serving as complementary whenever differences are noteworthy. A summary of the results is presented in Table 4.

<table>
<thead>
<tr>
<th></th>
<th>Sketch 1</th>
<th>Sketch 2</th>
<th>Sketch 3</th>
<th>Sketch 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average frame rate, p5.js version</td>
<td>60 FPS</td>
<td>20 FPS</td>
<td>60 FPS</td>
<td>Capped at 30 FPS</td>
</tr>
<tr>
<td>Average frame rate, plain JavaScript version</td>
<td>60 FPS</td>
<td>30 FPS</td>
<td>60 FPS</td>
<td>Capped at 30 FPS</td>
</tr>
<tr>
<td>Average frame time, p5.js version</td>
<td>7 ms</td>
<td>35 ms</td>
<td>Just above 1 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td>Average frame time, plain JavaScript version</td>
<td>4.5 ms</td>
<td>28 ms</td>
<td>Below 1 ms</td>
<td>Below 0.5 ms</td>
</tr>
<tr>
<td>Most time-expensive functions, p5.js version</td>
<td>fill()</td>
<td>ellipse()</td>
<td>color() fill() circle()</td>
<td>line() ellipse()</td>
</tr>
<tr>
<td>Most time-expensive functions, plain JavaScript version</td>
<td>fill()</td>
<td>ellipse()</td>
<td>arc() fill()</td>
<td>stroke() requestAnimationFrame()</td>
</tr>
</tbody>
</table>

Table 4: Runtime performance metrics from the coding experiments, measured in the Chrome web browser

To start off, it should be noted that the measuring itself can have a significant effect on performance. This can be verified by doing frame rate measurements directly in the code. The file “fpslogger.js” in Appendix A contains the code that was used for this. It simply looks at the time that has passed between each frame, converts it into a frame rate value and stores it in an array that can be viewed through the browser console. For example, running the p5.js version of Sketch 1 in Firefox gives an average frame rate of 28 FPS when running the performance tool, while using the code described above gives around 60 FPS. However, doing the same in Chrome does not yield any differences between frame rates, both tests result in around 60 FPS. For Sketch 2, running it in Chrome with the FPS logger code yields around 30 FPS for the p5.js version and around 40 FPS for the plain JavaScript version, which can be compared to the performance tool results below, which report lower FPS. All of this means that the measurement values should be looked at with some skepticism. However, the purpose of the tests is not to look at the absolute values but to compare the two versions against each other.

Sketch 1 has equal frame rates in both the p5.js and plain JavaScript versions on Chrome, averaging around 60 FPS, and the execution time for a single frame hovers around 7 ms for the p5.js version and around 4.5 ms for the plain JavaScript version, which is well under the recommended goal according to the RAIL performance model [48]. It should be noted that each frame goes through 1000 particles and draws them. There are no significant frame rate drops during the test. The most time-expensive functions in the sketch code are fill() and ellipse(), as well as rect() in the p5.js version, but as the results show, the code is performing well, considering that the refresh rate for most screens is 60 Hz. As noted above, in Firefox the p5.js version performs much worse with only 28 FPS, while the plain JavaScript version averages 50 FPS.

Sketch 2 shows the lowest frame rates of all sketches in the tests, with a pretty steady average of 20 FPS and a 35 ms frame execution time for the p5.js version and 30 FPS and just above a 28 ms frame execution time for the plain JavaScript version,
when recorded in Chrome. Looking at the code it is easy to see why; every frame 32 new particles are added to the animation until it reaches a maximum of 4032 particles, each of which is manipulated and drawn every frame. The most time-expensive functions in the p5.js version are color(), fill() and circle(), while in the plain JavaScript version they are arc() and fill(). It should be kept in mind however that these functions are all called 4000 times each frame. The recommendation to use Chrome over Firefox for p5.js becomes clear when testing the sketch in Firefox, where the p5.js version averages only 11 FPS, while the plain JavaScript version averages 43 FPS, which is actually a better result than in Chrome.

**Sketch 3** is much less demanding than the first two sketches. Both versions run at around 60 FPS in Chrome, with no frame rate drops. The p5.js version has a frame execution time of just above 1 ms, while the plain JavaScript version even goes below 1 ms on average. This means that there are no heavy functions in either version. The heaviest ones are line() in the p5.js version and stroke() in the plain JavaScript version. Running the tests in Firefox gives frame rates of 39 FPS for both versions, which is odd considering how light the sketch seems to be. It is likely that the reduced frame rate is mainly due to the heavy weight of the performance measurements.

Finishing off with **Sketch 4**, it should be noted that this sketch is capped at 30 FPS in the code. Both versions remain consistently at this frame rate in Chrome. In Firefox the results show that both versions run at 59 FPS, which indicates that the frame rate in the Firefox Performance tool is not calculated based on the actual number of frames drawn. Going back to the Chrome tests, the p5.js version has a frame execution time of 1 ms while the plain JavaScript version goes below 0.5 ms. The heaviest functions are ellipse() in the p5.js version and requestAnimationFrame() in the plain JavaScript version.

**Memory usage**

The Chrome Performance tool records memory usage as well, thus the same results store the measurements for how much memory is being used by the sketches at different times during execution. What is in focus here is the so-called JS Heap, i.e. the amount of memory used by JavaScript objects [49]. Once again, the results are available in Appendix A. The results are summarized in Table 5.

<table>
<thead>
<tr>
<th></th>
<th>Sketch 1</th>
<th>Sketch 2</th>
<th>Sketch 3</th>
<th>Sketch 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average memory added per frame, p5.js version</strong></td>
<td>8 MB</td>
<td>1.35 MB</td>
<td>100 KB</td>
<td>150 KB</td>
</tr>
<tr>
<td><strong>Average memory added per frame, plain JavaScript version</strong></td>
<td>640 KB</td>
<td>270 KB</td>
<td>20 KB</td>
<td>3 KB</td>
</tr>
<tr>
<td><strong>Maximum memory size, p5.js version</strong></td>
<td>8.3 MB</td>
<td>23.3 MB</td>
<td>17 MB</td>
<td>18.4 MB</td>
</tr>
<tr>
<td><strong>Maximum memory size, plain JavaScript version</strong></td>
<td>23.5 MB</td>
<td>23.4 MB</td>
<td>17 MB</td>
<td>17.3 MB</td>
</tr>
</tbody>
</table>

Table 5: Memory usage results from the coding experiments

**Sketch 1** shows no memory leaks in either version, although the patterns are a bit different. Each frame, the p5.js version produces 8 MB to the JS Heap and does garbage collection after each frame as well, while the plain JavaScript version produces only 640 KB each frame and does garbage collection more seldom. Because of the different rhythms, the p5.js version reaches a maximum of 8.3 MB in the JS Heap, while the plain JavaScript version goes up to 23.5 MB occasionally.
Sketch 2 does not show memory leaks either. The p5.js version adds 1.35 MB to the JS Heap every frame. Some of the memory is freed every frame as well, but the total memory usage increases over time and a bigger garbage collection is done every sixth frame. The JS Heap reaches a maximum of 23.3 MB during the test. On the other hand, the plain JavaScript version only adds 270 KB to the JS Heap every frame and garbage collection happens quite rarely as it takes a while for the used memory to gather up. It reaches a maximum of 23.4 MB.

Sketch 3 has similar memory patterns for both versions. Both have regular garbage collection at a rate of about once a second and both max out at 17 MB in the JS Heap. Each frame there is an average of around 100 KB added to memory in the p5.js version, while around 20 KB is added in the plain JavaScript version.

Sketch 4 shows completely different memory usage patterns for the two versions. The p5.js version looks similar to how it did in Sketch 3, with an average of 150 KB added to the JS Heap every frame and regular garbage collections about once a second and a maximum JS Heap size of 18.4 MB. The plain JavaScript version starts out right away with about 17.3 MB in the JS Heap and keeps it around that for a long time, before suddenly freeing up a lot of memory and then keeping the maximum to around 3.3 MB. Garbage collection happens but it is irregular and the memory addition every frame is minimal with around 3 KB.

5.2 Survey

The survey remained open for 20 days and gathered a total of 32 responses. The purpose of the survey was primarily to recruit people to be interviewed. For this reason, the survey was kept short and its questions were partially repeated in the interviews. Still, the responses were interesting in and of themselves and they are presented in this section. Each question will be presented along with a summary of the answers. The survey results in their entirety can be found in Appendix C.

5.2.1 “How much have you used p5.js?”

The answers were in a free text form due to the fact that it might be difficult to quantify the amount of time one has used a certain technology. The respondents were instead given the freedom to answer in any way they wanted.

The responses show a large variety in the amount of experience with p5.js and it seems quite evenly distributed. Answers range from having used it in “one finished demo-sized project and several small experiments” to “for the majority of my web projects for just over a year” to “sporadically for more than 5 years”. Seven answers point out in some way that all or most of their usage of p5.js has been of a hobby, experimental or side project kind.

5.2.2 “What have you used p5.js for?”

Respondents were given multiple choices for this question, as well as the ability to add their own answer through an “Other” option. The answers can be seen in Figure 9.

Art projects are clearly the most popular use case for p5.js, with around 80% of respondents having selected it, while more practical use cases such as website design and data visualizations have been selected by about a third of respondents. From the “Other” option it is interesting to note that four has added games and four has added education/teaching.
2. What have you used p5.js for?

Figure 9: Answers to survey question 2. Some answers are too long to fit in the graph, instead the full answers can be found in Appendix C.

5.2.3 “What other similar creative coding tools for the web have you used?”

This question had a free-text answer. From the responses, a total of 19 tools/frameworks/libraries can be identified, out of which 12 are JavaScript-based and 7 are based on other languages and platforms. The most popular one is three.js, which is used for creating 3D graphics, with 14 respondents having used it. 10 respondents answered D3, a framework for data visualization, while 5 noted paper.js which is used for vector graphics scripting. 7 answers noted having used Processing, although it is not primarily an application used on the web.

5.2.4 “How would you compare p5.js to other similar tools that you have used?”

The responses to this question were of varying length and focus. Still, three main points can be identified: good usability, performance issues and lack of advanced features.

The most common thread in the responses is that p5.js is easy to use and easy to learn. 12 respondents mention this in one form or another. The reasons for this are that the library is simple, well documented (at least in comparison with D3), accessible and has a good community around it. A couple of respondents that have used tools like Processing or openFrameworks point out that the similar API of p5.js made them choose it over alternative libraries.

Eight of the respondents note issues with performance and optimization in p5, in comparison to other tools. “Fairly slow”, “rather resource intensive” and “in its [sic] infancy in terms of […] it’s overall stability, it’s performance” are some of the comments on p5.js.

---

21https://threejs.org/
22https://d3js.org/
23http://paperjs.org/
The third major takeaway from the responses is that compared to other libraries, p5.js lacks advanced options and features. This was described in eight of the responses, where the comparison was mostly with D3 and three.js, which excel over p5.js when it comes to features in their own dedicated areas, which are data visualization and 3D graphics, respectively. Two responses mentioned that PixiJS is preferable when it comes to 2D sprite graphics. There was doubt regarding the use of p5.js in demanding or complex projects.

5.3 Interviews

In total, eight people were interviewed, seven of them through a recorded Skype conversation and one through a written email interview. The interviews have been transcribed and are available in Appendix E. The interviews have been anonymized in that the name of the interviewee or any specific detail such as current or past employers are not mentioned. Each interview is stored in its own document and is identified by the letters A to H. The presentation in this section will refer to specific interviews by these letters. The interview guide that served as the basis for the interviews can be found in Appendix D. This section is divided into a few parts that are broad categorizations of the interview questions.

5.3.1 Background

The only premise that was asked of the interviewees was that they had at least some experience with p5.js. Other than that, there were no limitations based on their background or interests. The interview requests were posted on creative coding community sites, meaning that only people with a self-identified interest in creative coding saw the requests.

The general backgrounds of the interviewees are quite varied. Four of them work or has worked in software/web development (A, B, D, F), one as a tech artist (H), one as a graphical designer (E), one studies and works as an interaction designer (C) and one is a student in computer science (G). Even the ones who are not in an explicitly visual field expressed having an interest in graphical design, either through previous work (D has a background in architecture while A and B works with frontend development) or as more of a hobby (F). Several (A and C) described being quite passionate about creative coding but not having the time to get into it more at the moment while (G) had to abandon a hobby project in p5.js due to a lack of time.

Most of the interviewees have used p5.js in one or a couple of major24 projects and besides that they have used p5.js for doodling, testing out ideas and creating small, personal experiments. (A) mentions having created a video tutorial series on p5.js, while (F) have been working on a personal research project regarding code submitted on OpenProcessing25. Only two (D and H) have used p5.js in commercial projects. None of the interviewees currently use p5.js in a major way in their work, (D) have used it a lot to create concepts or prototypes, but if the projects are developed to a production version they will go over to using another technical solution. (E) could consider using it in the future alongside their other design tools.

As for their general involvement in the creative coding community, most mention following online groups, forums, or chats focused on creative coding. Processing,

24The definition of a “major project” varies a lot between interviewees; for some it seems to mean just having gotten a project to completion and showed it somewhere, for some it means a professional-grade project.
25https://www.openprocessing.org/
p5.js, or generative design, occasionally commenting or reviewing other people’s submissions but no one has been posting a lot of their own work. There is no one that has been involved in the development of the p5.js library, although (A) mentions having helped with the p5.js website. Others keep up with trends by following creative coders on social media or participating in related meetups. Some specific mentions are Awwwards26, Purin Phanichphant27 and Dan Shiffman/The Coding Train28.

5.3.2 How p5.js Can Be Used

(A) used p5.js in a couple of bigger projects during their time in school. One was a large-scale music visualizer for the school radio station, where a large video wall was filled with a landscape and elements in the landscape responded to the music being played. Another was a live music tool, where audio samples were mapped to circles and these circles could be moved around on a 3D plane, which changed the volume and panning of the sample.

As a project for San Fransisco Design Week (B) built an installation where animated visuals were being projected onto canvases hanging on a wall. These canvases were rigged with Arduino Touch Boards and when a section of the canvas was touched, the animation and audio changed accordingly, making them interactive paintings. In this case, p5.js was at the center of the project, taking in the input from the sensors and creating and displaying the visuals in a browser window which was projected onto the canvases.

(C) had an idea for an art installation and created a proof of concept for it in p5.js. The idea was to ultimately have an interactive installation where people passing by could plant digital flowers, and then other people would have to tend to them to make sure they continue living, as a kind of way of building a community and making people feel more connected.

With their background in architecture, (D) developed an energy efficiency tool for an architecture research project, where p5.js was used to draw volumes that would represent buildings of different sizes, which could then be evaluated. Additionally, (D) used p5.js to create a genomics visualization where genome sequences were displayed. When hovering over specific genes more information would be shown. It was used for creating a tool for medical doctors in a project for helping them understand current genomic technologies.

In order to simplify their design workflow, (E) created a website using p5.js that could auto-generate a map visualization that their client used for social media posts. The map was used to visualize participants in a competition, data that changed frequently. Instead of having to open up Photoshop and use a template given by the designer, the client could simply input the new numbers and get a new graph easily.

(F) got interested in generative design a few years ago and has recently been exploring camera motion detection that drives the action happening on a screen, for use in bigger installation settings.

Curious about how people in their WhatsApp friend group interacted with each other, (G) wanted to visualize metadata from the chat. They created a website where group chat history could be sent. p5.js was then used to create a type of graph from this data, where people would be represented by circles sized according to their activity in

---

26https://www.awwwards.com/
27http://purin.co/Experiments-with-P5-js
28https://www.youtube.com/channel/UCyjgXvbBlbQiydfZU7m1_aw
the chat, and the circles would be joined together by links that were wider the more two people had chatted together.

An administrative region wanted a new year’s e-card, and (H) decided to create a website with p5.js. When visiting the website, with the name of the recipient as a parameter in the URL, the name would be displayed in the sketch that was the e-card.

5.3.3 Why Choose p5.js?

What are the primary use cases for p5.js? (C) finds p5.js best used as a learning tool, as its simplicity and documentation makes it approachable for someone who does not know how to program: “I think it’s pretty semantically logical from an English perspective, it doesn’t feel like such a stretch learning programming in p5”. (H) thinks it is a great tool for teaching JavaScript and creative coding and they plan to use it in their teaching. (D) sees p5.js as a good choice if a more original kind of data visualization is wanted than can be achieved with standard data visualization tools. (B) thinks it could be used for several purposes, like background animations on websites or for data visualization. (G and H) mention similar things although they stress that p5.js should only be used on a small scale. (G) describes a scenario of how something like an online newspaper that wanted a small animation alongside a news story might benefit from using p5.js, as it would be quick to prototype and create. (H) describes p5.js as a great tool for creating visual prototypes quickly. (C) mentions how sites like CodePen are a good fit for p5.js, as those sites showcase code snippets of cool experiments. As a designer, (E) sees as a great use case the possibility of having the client inputting data, which then generates an image based on the visual identity guidelines that the designer has created. This can be done easier in p5.js than in any other of their current tools, as the client can visit the site on any device and does not need any pre-installed software. (G) brought up how particle systems can be programmed in p5.js and how nature can be simulated through code. (F) talked about the interesting art work that is being done, and brought up that interesting things are being done with shaders in p5.js.

As a counterpoint, several interviewees point out that p5.js would not work very well for general use on standard websites. (A) said that they would only use p5.js in a very defined space, a specific web page just showing off a sketch that will only run under specific circumstances. As an example, they brought up what they did with the audio visualizer for their school radio station, that only ran on a video wall with specific dimensions. (C) finds it difficult to see how p5.js would be used in more practical ways, on a commercial-grade website. (G) was unsure but worried about the performance of p5.js, especially when viewed on mobile.

When asked why they chose to work with p5.js in the projects that they mentioned, frequently the main reason was that it is straightforward, simple to get started with and easy to learn. (D) mentioned the speed of development as an important factor, as with p5.js they could quickly get something to show. If the project was supposed to live in the browser, it was natural to go with p5.js rather than Processing, which was mentioned by several interviewees who had been using Processing previously. For (A) the project was to be displayed on a video wall and it was already running the Chrome browser in the background. Being able to host the visualizer on a site with the help of p5.js made it simple. (D) wanted to have their projects in the more flexible and maintainable browser environment rather than in a desktop environment, even though in one of the projects, “there was a strong inclination to use desktop software to represent the graphics and I proposed the whole thing on the web as a Software as a Service, to show that there was another way to do things and to make it massive and with better reach”.

32
In some cases, p5.js was the only platform interviewees knew about that could deal with the requirements for their project. (G) wanted their project to be available on the web and only knew of p5.js that could create the kind of visualization they wanted to create. (H) knew that p5.js could do vector graphics, which was what they were looking for. For (C), using p5.js was an obvious choice since it was part of a course they were taking and furthermore there seemed to be more resources for learning p5.js than other libraries. A big reason for why their teacher had chosen p5.js was that they really liked the teaching videos by Dan Shiffman, one of the people involved in the Processing Foundation, that were freely available online.

(E) started using programming in their design practice through learning the NodeBox visual programming language, then getting into Processing and Drawbot, both tools they currently use a lot. These tools allow them to create complex images that would be too laborious to do manually, especially if they want to explore and iterate on the image. Still, these tools are reserved for the designer. When they learned about p5.js they realized that they could use it to provide easily accessible design templates to their clients in the browser.

5.3.4 Working With p5.js

Interviewees were asked to elaborate on how their experience was working with p5.js. The opinion that was repeated by almost everyone was that p5.js is very easy to use and get started with. (B) likes the simplified drawing process compared to pure JavaScript, while (E) found it even easier to work with than Processing. When creating a live music tool, (A) found the setup simple for the audio part as well.

Despite p5.js being easy to use, some brought up difficulties they had during their projects, mainly with understanding bigger concepts that were needed in the project, such as animation, physics, and sprites. (B) had some issues following how elements got layered on top of each other. The physics behind animations was equally challenging, as they had not done any animation previously. For their project, (C) used the add-on library p5.play as well, because they needed to work with sprites. Understanding how to work with sprites and animations was confusing, as the add-on library does not have as many learning resources as p5.js. For (A), their biggest challenge was having to manually convert some of the drawings, that had been created in Adobe Illustrator, over to p5.js, in order for them to be animated. A library more focused on sprites might have made that part easier, but in general, they were happy with having chosen p5.js.

Performance issues in p5.js were brought up by several interviewees. (D) described the draw loop as a little expensive and that the performance issue has to be taken into account when making the choice of framework. (E) found the same thing in their design project, as the image that was being created had 15 000 individual points being calculated and drawn. (H) compared Processing and p5.js and thought that Processing was better performance-wise. (A) started running into memory issues in their audio project, which they thought might be because p5.js does not have a good garbage collection system. They added that the sound capabilities of p5.js probably are more aimed at game sounds, in which case audio clips only last a second. (G) put it this way: “Usually when you have programs that are simple to use then their efficiency isn’t as large as other, more complex programs”. Similar thoughts were voiced by (H): “Once you reach a certain level of visual and interactive complexity though, I don’t

29https://www.nodebox.net/
30https://www.drawbot.com/
31http://molleindustria.github.io/p5.play/
think it’s the best choice. But I don’t want p5 to become a monster framework anyway."

(F) found the lack and underdevelopment of related tools a big issue, especially when compared to other platforms with stable IDEs. They admit that the comparison might not be fair and that p5.js is a very recently created framework. The fact that p5.js is still in its infancy brings with it more errors than when using Processing, for example with the webcam. They add that this is due to the fact that Processing, which is based on Java, is closer to the native machine, which makes things like working with the webcam easier. They wished for a way that p5.js could integrate better with other JavaScript libraries, such as three.js.

5.3.5 Comparing p5.js With Other Tools for Creative Coding

Interviewees mentioned a number of other creative coding tools that they have used. D332, a data visualization library, is the most popular among the alternatives, having been used by all but one. three.js33, a 3D graphics library, has been used by four, and SVG.js34, which is used for manipulating SVG, by two. Other tools mentioned are create.js35, a library suite for rich interactive content, paper.js36, a vector graphics scripting engine, DrawBot37, a graphics scripting tool for macOS (it should be noted that it is not web-based), and PixiJS38, a 2D WebGL renderer engine.

(A) points out that it can be argued whether D3 is a creative coding tool or not. (B) adds that the intention with D3 is different from p5.js, the former being a data visualization tool while the latter is mainly for drawing objects. If the visualization is going to be more creative, (D) thinks p5.js is better. Many (C, D, F, G) agreed that D3, although cool and powerful, is very complicated and not for anyone impatient or new to coding. In these cases, (C) thinks p5.js is definitely to be preferred. However, on a general level, they feel that D3 is a more marketable skill, as they have seen it mentioned in job advertisements but never p5.js. For (F), p5.js would not be their first option for data visualization and would go with Python or R, or D3 if the project was for the web. (G) had a course in data visualization, using D3, but they quit it halfway as they didn’t like it. They found the documentation for D3 a bit sparse and for them, D3 would never be an option for a future project and they would prefer to use p5.js. However, they thought that there might be performance issues with p5.js if large amounts of data are used.

The 3D capabilities of p5.js are pretty minimal and several (A, F and H) suggest moving over to three.js for any serious 3D creation. On the other hand, three.js is probably worthless when it comes to 2D, according to (H), and (F) thinks they would go for p5.js if they wanted to create a 2D art image. (B) found that using three.js was a similar experience to using p5.js, but with the added 3D functionality, although they add that three.js is way more complicated precisely because of that. (H) would probably not use three.js for personal projects as they like the minimalism of p5.js. Furthermore, they might use p5.js for very basic 3D, if they are going for a flat or unrealistic visual look or complex shaders are not required. However, as soon as the project is commercial or needs better rendering or shadows, they would go with three.js.

32https://d3js.org/
33https://threejs.org/
34https://svgjs.com/docs/2.7/
35https://createjs.com/
36http://paperjs.org/
37https://www.drawbot.com/
38https://www.pixijs.com/
(A) found that SVG.js had an API similar to p5.js, although less expansive. (C) was unsure if and how motion could be used in SVG.js, at least in a way that is not very time-consuming. For them, SVG.js suits static graphics better while p5.js is better for animations and interaction. A benefit of SVG.js is that SVG files can be exported very easily, a capability (A) was pretty sure does not exist in p5.js.

(E) chose paper.js over p5.js for similar reasons for one project, as they needed an SVG file exported, and had heard that it was hard generating SVG or PDF files in p5.js. However, they found the documentation lacking and parts of the paper.js library unintuitive, and as the team around the library is very small, it might not be supported in the long run.

DrawBot is a tool that (E) has used a lot in recent years, and it is the one they would use if they needed a PDF, poster or vector image. It has interesting typographic technologies that are good for their design practice and something they did not think is available in Processing or p5.js. DrawBot only creates static images, meaning that for cases where the project is for the web or needs animation or interaction, they would probably use p5.js. A downside with DrawBot is that it only runs on macOS, while in comparison p5.js or Processing works on multiple operating systems.

Having used PixiJS a lot as a part of the Phaser framework, (H) would use that option whenever hardware accelerated 2D rendering was needed. They prefer PixiJS for its superior performance and interactivity capabilities, as it can track the local space of objects which is good for complex interaction and testing mouse hits on shapes, which is something that can’t be done in p5.js. Additionally, p5.js lacks some type of display list hierarchy, with parent/child shapes and objects, which would help with interactions and physics as well. Whenever they do use p5.js, (H) always manually codes in this capability.

Many expressed doubt using p5.js in projects other than personal experiments. (A) points out that p5.js has difficulties with scaling: “If you are making a website, something you have to think a lot about is, what does this look like on a desktop versus a phone, and making those kinds of calculations where you do things with pixels and stuff in p5 can be pretty difficult”. An additional reason that p5.js might not be used a lot on commercial projects is the extra file size that an additional library brings. (B), who has worked with user interface animations, said that the animations they created at work were simple and that there was no need to bring in an entire library. (D) explained that their development method often consists of starting out by prototyping projects in p5.js, but at some point during development, they convert their code to plain HTML and JavaScript, to make the code simpler and more performative.

5.3.6 Final Thoughts

(A) hopes that creative coding, and p5.js in particular because it is on the web, will become more popular, especially in teaching programming, because of the instant visual feedback that it provides to the code that is written. It makes learning to code more fun and accessible because it lives in the browser and it would be more welcoming for non-technical people. (D) learned to program in Processing because the graphical part appealed to them and recommends it as the best way for kids to get started programming, rather than something like Scratch, “because [p5.js] makes you think about the geometry and coordinate system right away”.

https://phaser.io/
(B) misses more work opportunities in the creative coding field, work that
demands both an understanding of code and art. Currently, creative coding is more of a
medium for creating art than a practical skill that is needed at a lot of jobs.

(G) finds the active community around p5.js a really positive aspect and something
that makes p5.js stand out: “For a good open source project you need a good
community engaged in trying to improve the product”. (F) points out that the big
challenge for p5.js is “how many people they can get working on the open-source side
to build up the libraries”. Furthermore, they feel that the p5.js developers are currently
focused more on the technology part of the library than the art part. This is visible in
its API, which (F) feels should be made easier. One example they give is the function
called noise(), which might not make a lot of sense to an artist when they are looking
to e.g. create a smoke effect on the screen. In that case, the natural way would be to
call a function called smoke() instead.
6 ANALYSIS

This section will start with summarizing the results in a number of main findings, comparing the results from different parts of the empirical research and the theory. The research questions are then revisited and answered. The section finishes with some thoughts regarding the validity threats to these findings.

6.1 Main Findings

6.1.1 Usability

Based on both the coding experiments and the interviews, there seems to be no doubt that p5.js is a tool that is easy to use. Its simplicity, approachability, documentation and learning resources are points that show strongly in the interview results in sections 5.3.3 to 5.3.5. The results presented in sections 5.1.2 and 5.1.3, detailing the development process and static code analysis from the coding experiments, show that p5.js is easy to understand, read and reason about. This is thanks to an intuitive API, a clear example of which is the `circle()` function in section 5.1.3, where a complex-looking piece of code turns into one elegant function call. Another takeaway from the coding experiments, specifically section 5.1.2, is that the canvas element is a very good way of working with graphics and interactivity in the browser, and p5.js shines especially when the canvas and animation frames are used. This is evidence to the claim put forth in the theory in section 4.1 that it is common that creative coding tools are built to be easy to use, to attract artists and others who might not have a technical background.

Despite it being a beginner-friendly tool in many ways, there are concepts related to p5.js that can still be hard for beginners to grasp, such as how to work with animations and physics in code, detailed in section 5.3.4. This is not a fault of p5.js, but perhaps its developers could work towards making these concepts easier to learn through the library? Working on the coding experiments, it took time getting used to working with the canvas, and it is not always easy to know what is happening under the hood of a running sketch. One interviewee mentioned the lack of a display list hierarchy in p5.js, and another one had difficulties understanding how elements were layered. This need for control and visibility over the program state is something that is discussed in [27] as well.

6.1.2 Performance

One finding that is present in all aspects of this study, from the coding experiments to the surveys and interviews, is the issue with performance in p5.js. The development diary in section 5.1.2 notes several times getting performance hits in the p5.js version. The main reason for this is the large number of particles that are drawn each frame in a couple of the sketches. The performance tests in section 5.1.4 confirm this and they pinpoint the most expensive functions, which are `fill()`, which sets the color to be used for filling shapes with, and the different functions for drawing shapes to the canvas, such as `circle()`, `rect()`, and `ellipse()`. In p5.js these functions seem to be doing more under the hood than the plain JavaScript counterparts, which is why, when they are run thousands of times each frame, p5.js versions have a noticeable drop in performance.
This is in line with what one of the interviewees said (section 5.3.4): “Usually when you have programs that are simple to use then their efficiency isn’t as large as other, more complex programs”. The theory in section 4.1 discusses the concept of convention over configuration as a way of achieving simplicity in creative coding tools, and p5.js seems to follow this principle, based on the code analysis in section 5.1.3. This principle makes p5.js quick to get started with, but with no ways of modifying parameters, it becomes difficult to improve performance. The developers of p5.js acknowledge that parts of the library are problematic performance-wise and they suggest using native JavaScript in these cases [47].

It was surprising to find that the performance differences were clearly visible between the two versions tested, as well as the differences between different browsers, as was noted in section 5.1.4.

6.1.3 Use Cases

Although a variety of use cases can be found for p5.js, it seems that the primary one is rapidly creating prototypes, experimenting and testing out ideas. It was brought up in many interviews (sections 5.3.2 and 5.3.3) and this finding fits well with the concept of bricolage programming from the theory in section 4.1, as the simplicity and usability of p5.js allow for getting quick visual feedback. Additionally, this finding fits in well with the sketching concept that is the basis for other popular frameworks such as Max/MSP and Processing. This way of using the library is similar to how the theory describes the maker culture, where the prototype is often the goal. One interviewee gave an interesting account in section 5.3.5 of how they usually created their prototypes in p5.js and then moved over the projects to plain HTML and JavaScript.

This brings us to the flip side of this finding: that p5.js is not used much in professional contexts, especially if the project is complex or large-scale. This is supported by the coding experiments, which show performance issues when a large number of elements are included, as described in sections 5.1.2 and 5.1.4. Furthermore, it is true that an extra library adds file size to a website. Still, it was surprising to find that none of the interviewees use p5.js directly in their current work, other than as a prototyping tool, as described in section 5.3.1. As a couple of interviewees mentioned in sections 5.3.5 and 5.3.6, neither p5.js nor creative coding in general are sought-after skills on the job market. There is hesitation regarding the usage of p5.js on standard websites because of scaling and portability issues. It works best in defined spaces under specific circumstances but should be avoided in contexts with a lot of diversity, as one interviewee put it. It might be that despite impressive features and advantages, the stability and popularity of the native language beats a library, as is the conclusion in [18].

At the same time, people have chosen to work with p5.js precisely because it lives in the browser, as detailed in section 5.3.3. This makes it easy to share over different platforms, even easier than programs written in Processing. Whether p5.js is portable or not might be a question of different definitions and exactly how different people use p5.js. The sketches in the coding experiments were not tested on mobile platforms, but quickly trying them out on a smartphone does not reveal big issues with most of the sketches. The theory in section 4.2.1 mentions how the web browser has become a more attractive platform for creative coding tools, partially thanks to its portability.

Another notable use case for p5.js is as a tool for learning how to code. This is reflected in both interviews, survey answers, and theory. The main arguments for this use case are the intuitive API, the quick visual feedback, and great documentation and support from the p5.js community.
According to the survey results presented in section 5.2.2, 80 % of respondents have used p5.js for art projects. What does this mean? The term art project can be hard to define and might have been interpreted differently by different people. The interviews provide many examples of art projects (section 5.3.2) and show an interesting variation in how p5.js has actually been used.

6.2 Answers to the Research Questions

RQ1: How does p5.js compare to plain JavaScript as a tool for web graphics?

This question was answered by doing coding experiments in both p5.js and plain JavaScript and comparing the process and results. The results are presented in section 5.1.

a) Which alternative is easier to develop with, for a proficient web developer with no previous experience of the p5.js library?

During the coding experiments, the researcher never had any problems getting started with p5.js. Working with the canvas and animation frames was easier and more intuitive in p5.js than in plain JavaScript. It is easier to work with file resources in p5.js, as file contents can be easily fetched before anything else happens. Still, the differences between working with the two alternatives are not that great in regards to the core logic of a specific program. In cases where performance is important, such as when a lot of elements are drawn to the canvas each frame, p5.js is more challenging to work with as it is hard to find and fix the resulting performance problems.

b) Which alternative leads to more readable code?

Readable code means that the code should be easy to understand for a developer reading it for the first time. The API of p5.js makes code that uses the canvas very easy to read and understand. The structure that p5.js enforces makes code that is more readable than is necessarily the case in plain JavaScript. Through a number of helper functions and variables, code written in p5.js is cleaner.

c) Which alternative leads to a smaller amount of code?

Programs written in p5.js are almost certain to contain fewer lines of code and fewer characters, up to 30 % less according to this study. This is thanks to its simple API and the approach of convention over configuration that p5.js seems to follow.

d) Which alternative loads faster in the browser?

This question can be answered in terms of the project file size, the runtime performance, and memory usage.

Measuring the project file size should indicate how fast resources can theoretically load into the browser. Programs that include the p5.js library are bigger in size, but depending on how many other resources that need to load, this might be slightly noticeable or not at all. Runtime performance, i.e. how fast the program runs once it is loaded, was mainly measured through the frame rate. Plain JavaScript had consistently better numbers, but it was most noticeable when the sketch contained a lot of elements being drawn. Furthermore, p5.js performs better in Chrome than in Firefox. Neither alternative shows any memory leaks or other notable details regarding memory usage.

In short, plain JavaScript loads faster, but it is only noticeable if the program is very demanding on computer resources.
RQ2: How does p5.js compare to other creative coding frameworks on the web?

This question was answered by doing a survey and interviews with members of creative coding communities and the results are presented in section 5.2 and 5.3. The most commonly recurring frameworks that were mentioned in the survey and the interviews are D3, three.js, paper.js and SVG.js. The research shows that in comparison with these similar tools, p5.js is easy to use but suffers from performance issues, lacks advanced features and is usually not recommended for professional and complex projects.

The minimalism of p5.js is a strong point in the comparisons and no alternatives beat p5.js in terms of simplicity. Additionally, it is well-liked because of its connection to Processing, which itself is a very popular tool for creative coding.

The most common comparisons were made with the three.js and D3 libraries. These are both more advanced in their respective areas of 3D graphics and data visualization, but more complex as well. If the demands on these specific areas are not high, p5.js might be preferable as it is easier to use, but its lack of advanced features makes it unfit for anything more complex. Still, p5.js can be a viable option for data visualization, as evidenced by one of the sketches in the coding experiments.

RQ3: How has p5.js been received and used by the creative coding community in web design, web graphics, and creative projects?

This question was answered by doing a survey and interviews with members of creative coding communities and the results are presented in section 5.2 and 5.3. The research shows that p5.js has been used in a large variety of circumstances, from creating tools for live music, design templates and architecture research to interactive visual installations and visualizations for data, music, and genomics. Its primary use cases are creating prototypes, art projects, and as a tool for learning how to code.

There is a general lack of confidence in using p5.js in professional contexts, partially due to its performance issues, and it is mostly recommended for personal projects and prototypes. At the same time, this can be seen as one of its strengths; p5.js is a simple library, simple in terms of features and capacity, but, therefore, simple to use as well.

6.3 Validity Threats

It turned out to be difficult to find creative coding projects that were written in plain JavaScript. This led to the final selection consisting of sketches that in some regards were pretty similar. There was a conscious choice to limit the scope of the experiments to focus on visuals only and not include any audio or DOM interaction, but it is possible that many interesting aspects regarding p5.js and its comparison with JavaScript were left undiscovered because of this.

Looking back at the development process of the sketches, it sometimes felt that what was being tested was not which alternative that was easier to develop with, but rather which alternative that it was easier to translate to. This is due to the choice to not create the sketches from scratch but use existing projects and convert them to the other version.

The biggest validity threat regarding the interviews is that the number of people interviewed, eight in total, is not very high. This puts into question if the represented
backgrounds, experiences and opinions are diverse enough. Most of the people interviewed have a computer science background, which leads to an underrepresentation of people from the art field, with little technical skills.

On the opposite side of the spectrum, none of the interviewees is involved in the development of p5.js. The original plan was to include at least one p5.js developer, to get their perspective on how p5.js has been received by the community and what impact it has had, but no one could be found that agreed to an interview.

Finally, because of the lack of research on p5.js and a general lack of more technically focused research on creative coding frameworks, the claims that are made in this thesis regarding p5.js are hard to verify against existing research.
7 Conclusion

Creative coding is about exploring code as a way of practicing art. This thesis has looked into the field of creative coding and different ways of doing it on the web, mainly through the Processing-inspired p5.js library and through plain JavaScript. The goal was to decide the feasibility of using p5.js for creating web graphics through code, compared to doing it in plain JavaScript or other web-based frameworks. Furthermore, this thesis has looked into how p5.js has been used by the creative coding community.

The thesis asked the question of how p5.js compares to plain JavaScript, in terms of how easy it is to develop in it as well as its code readability, amount of code and performance. By doing coding experiments, the conclusions are that p5.js is easier to get started with and the code is easier to read and reason about, thanks to the simple and intuitive API of the library. Using p5.js results in writing less code. However, using p5.js increases the total file size and if the project includes continuously drawing large amounts of elements, p5.js performs worse than plain JavaScript.

Similarly, the thesis asked how p5.js compares to other web-based creative coding frameworks. Similar conclusions are reached in these comparisons, which were made through interviews with members of creative coding communities. The strongest point in p5.js is its usability while its biggest drawback is its performance issues. It lacks advanced features for some purposes and is usually not considered for professional contexts. Instead, its primary use cases are creating quick prototypes, sketching out ideas, and using it as a tool for learning how to code.

Finally, the thesis asked how p5.js has been received and used by the creative coding community. Again, interviews were used to answer this question. The conclusion is that despite its flaws, p5.js has been used in a variety of contexts. It is most commonly used in different types of art projects, such as interactive visual installations and music visualizers, but it has been used for more practical purposes as well, such as design tools and data visualizations.

In summary, p5.js is a good library for getting started with coding creatively in the browser and it is an excellent choice for experimenting and creating visual prototypes quickly. Should project requirements be much more advanced than that, there might be other options that will work better.

There are many options for coding creatively both in a desktop and browser environment. The ubiquity and portability of web browsers make them a viable platform for creative coding, although web-based tools, such as p5.js, might suffer more from performance issues than desktop applications that are closer to the native machine, such as Processing.

The contribution that this thesis provides is an overview of the p5.js library. The focus on p5.js is a perspective that has been missing in academic articles in this field. The thesis evaluates the feasibility of using p5.js as a tool for creating interactive graphics and animations on the web. This thesis functions as a guide for developers looking into using p5.js or who are generally curious about creative coding. Additionally, it can serve as a starting point for further studies on p5.js.
8 Future Work

This thesis did not look under the hood of the p5.js library to try and understand exactly why it performed worse in some circumstances and if the issues could be mitigated in some way. A deeper study into this could be of value in the further development of the library, although it is most likely that the developers are aware of the issues.

There are a number of add-on libraries available for p5.js. There are official ones for adding sound capabilities, interacting with the DOM and making p5.js more accessible, as well as many other contributed libraries that adds support for such things as creating games or serial communication. These could all be explored and might even modify one of the conclusions in this thesis that p5.js is lacking in advanced features.

One of the main findings in this thesis was that p5.js is very easy to use. However, visual programming languages, such as Max/MSP, might be even easier for someone who does not yet know how to code. A comparative study of how easy it is to work with a creative project in a coding environment and a more visual one could provide interesting results.

Finally, looking more at the art side of the topic, it would be interesting to investigate in more detail the technical requirements and considerations of art projects, such as a large-scale interactive installation, that are heavily dependent on technology and code.
9 REFERENCES


10 ANNEXES

Appendix A—Coding experiments: source code and performance analysis


This appendix is available online, as it is more convenient when it comes to source code. The online repository includes files containing the results for the network and performance analysis as well.

For quick access, the file index.html in the base directory contains links to all sketches in both versions.

The general license for this repository is MIT. However, as the repository consists of code sketches from different authors, the license for each sketch is included in their respective directories.

Appendix B—Survey form

Survey on p5.js and creative coding

This survey consists of a few questions about the p5.js creative coding library. In order to be able to participate in it, you should have at least some experience with p5.js.

This study is part of a bachelor’s thesis about p5.js and creative coding. Answers will be treated confidentially and the results will be used to present the views of the creative coding community on the p5.js library.

Thank you very much for your participation and perspective.

Emil Sandberg

Contact me for any questions at emilalsandberg@gmail.com

* Required

1. How much have you used p5.js? *
   (Long answer text)

2. What have you used p5.js for? *
   Please select all that apply.
   (Checkboxes)
   • Data visualizations
   • Website design
   • Other kinds of web graphics
   • Art projects
   • Other:
3. What other similar creative coding tools for the web have you used?
E.g. Sketch.js, Three.js, D3.
(Long answer text)

4. How would you compare p5.js to other similar tools (question 3) that you have used?
Leave empty if you only have experience with p5.js.
(Long answer text)

5. Would you be willing to do a follow-up interview where you would get a chance to elaborate on these topics? If Yes, please leave your email address and you will be contacted shortly.
(Short answer text)

Appendix C—Survey results
Available online at https://docs.google.com/spreadsheets/d/19F_9nPDDiqsJU-ZwbI1mS_ue9m-tg2eJCFhZ_35yA7Y/edit?usp=sharing

The results have been anonymized.

Appendix D—Interview question guide

Introduction / Warm-up
1. What is your background / area of work/study?

Part 1: p5.js
2. How much have you used p5.js?
   How many projects?
   How many years?
3. How involved are you in the creative coding community?
4. Have you ever been involved in the development of p5.js?
   How much have you worked with developing it?
   What has been your main contributions?
5. What types of projects have you used p5.js for?
   data visualizations
   website design
   other web graphics
   art projects
   other, e.g. games, teaching, AI
6. Can you give some examples of projects where you have used p5.js and how p5.js was used?
   What was the project about?
   In what way was p5.js used?
   Why was p5.js chosen?
   How was it to work with p5.js?
   For art projects: Why did you use a web platform, and not e.g. Processing?
7. What do you see as the primary/best use case for p5.js?
8. What are your coolest/most impressive examples for how p5.js has been used?
Part 2: Other tools

9. What other similar creative coding tools for the web have you used?
10. Could you shortly describe these other tools?
   What are their primary use cases?
11. How would you compare p5.js to other similar tools that you have used?
   What are the pros and cons?
   In your p5 projects, why did you choose to use it?
   In your non-p5 projects, why did you choose not to use it?
   In which cases might you use vs. not use p5.js?

Conclusion

12. Do you have anything you want to add / final thoughts / opinions on p5.js?

   + Specific questions for each interview based on what I know beforehand about them

Appendix E—Transcriptions of interviews

Available online at https://github.com/ems16/thesis-interviews

The text from the transcribed interviews would measure 27 pages in a single document and instead of including them in this thesis document, they are available as an online appendix. The interviews have been anonymized and their file names correspond to the references in the thesis.