

UNIVERSITÄT DUISBURG-ESSEN

BACHELOR THESIS

---

# Development and Comparison of Overview Techniques for Extreme Resolution Datasets

---

*Author:*

Danyun LEI

*Supervisor:*

Prof. Dr. Jens KRÜGER

*Examiners:*

Prof. Dr. Jens KRÜGER

Prof. Dr. Josef PAULI

*A thesis submitted in fulfillment of the requirements  
for the degree of Bachelor of Science*

*in*

Computer Engineering  
International Studies in Engineering (ISE) PO08  
Fakultät für Ingenieurwissenschaften

*for*

The High Performance Computing Group  
Department Engineering

September 18, 2019

## Versicherung an Eides Statt

Ich, Danyun LEI, versichere an Eides statt durch meine untenstehende Unterschrift,

- dass ich die vorliegende Arbeit - mit Ausnahme der Anleitung durch die Betreuer - selbstständig ohne fremde Hilfe angefertigt habe und
- dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus fremden Quellen entnommen sind, entsprechend als Zitate gekennzeichnet habe und
- dass ich ausschließlich die angegebenen Quellen (Literatur, Internetseiten, sonstige Hilfsmittel) verwendet habe und
- dass ich alle entsprechenden Angaben nach bestem Wissen und Gewissen vorgenommen habe, dass sie der Wahrheit entsprechen und dass ich nichts verschwiegen habe.

Mir ist bekannt, dass eine falsche Versicherung an Eides Statt nach §156 und nach §163 Abs. 1 des Strafgesetzbuches mit Freiheitsstrafe oder Geldstrafe bestraft wird.

---

Ort, Datum

---

Unterschrift

*“To see a world in a grain of sand, hold infinity in the palm of your hand. ”*

William Blake

UNIVERSITÄT DUISBURG-ESSEN

## *Abstract*

Fakultät für Ingenieurwissenschaften

International Studies in Engineering (ISE) PO08

Computer Engineering

Bachelor of Science

### **Development and Comparison of Overview Techniques for Extreme Resolution Datasets**

by Danyun LEI

Matr. No. 2265625

[danyun.lei@stud.uni-due.de](mailto:danyun.lei@stud.uni-due.de)

In this thesis, three main overview techniques and two secondary preview techniques of the overviews are developed and implemented using the Mandelbrot set as the source of extreme resolution datasets. For these datasets, a hierarchical structure is used to present the index information of the current region of interest. Using the developed software in this thesis, it is possible to intuitively understand the hierarchical state of the current observing area with the whole. This thesis also compares the different overview techniques which in combination consists of six different ways.

It is worth mentioning that the Mandelbrot set that are implemented in this thesis is a very good example for extreme high resolution datasets because it is a dataset that can theoretically provide infinitely high resolution.

The technology stack used in this thesis is pure web technology, a classic combination of HTML / JavaScript / CSS, and the program offers the possibility to adapt and replace the pure front-end technology with front and back end separation solution easily.

## *Acknowledgements*

Foremost, I hereby express my deep sense of gratitude and indebtedness to Prof. Dr. Jens Krüger, for your valuable guidance, encouragement and support. Your patience and faith in me was a key reason that I could finish the work.

I would like to also express my thanks to Mr. Andrey Krekhov, Mr. Michael Michalski, Mr. Sebastian Cmentowski, and all colleagues in the group who have willingly helped me and offered valuable advices with their excelled abilities.

Also I express my thanks to Prof. Dr. Yunqi Lei, Dr. Bixia Wu and Dr. Franz-Josef Schmitz, in no particular order. Without your support, I could not have finished the task successfully.

# Contents

<b>Versicherung an Eides Statt</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>1 Introduction and Motivation</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Motivation For More Complicated Situations . . . . .	2
1.3 Inspirations for the Arrangements of Context Views . . . . .	5
1.3.1 Dock And Scrollbar . . . . .	6
1.3.2 Stacked Cards . . . . .	7
1.3.3 Tabs . . . . .	7
1.3.4 Previews Of Contexts . . . . .	8
1.4 Related Work . . . . .	11
<b>2 Background</b>	<b>13</b>
2.1 Web Technology Stack . . . . .	13
2.2 Web, HTML5 And Canvas API . . . . .	14
2.2.1 Browsers . . . . .	14
2.2.2 Markup Language . . . . .	15
2.2.3 HTML5 . . . . .	15
2.2.4 Canvas API . . . . .	17
2.3 Mandelbrot Set . . . . .	17
<b>3 Architectural Designs</b>	<b>21</b>
3.1 Basic Structure . . . . .	21
3.2 Front End UI . . . . .	22
3.3 Back End Resolver . . . . .	25
<b>4 Implementation</b>	<b>26</b>
4.1 Files And Folders . . . . .	26
4.1.1 Folders . . . . .	26
4.1.2 Top Level Files . . . . .	28
4.2 Start the Project . . . . .	29
4.3 Front End . . . . .	31
4.3.1 HTML Entry index.html . . . . .	31
4.3.2 Main JavaScript index.js . . . . .	31
4.3.2.1 Class MandelWorker . . . . .	32
4.3.2.2 Class MapVisualPair . . . . .	34
4.3.2.3 Class Minimanager . . . . .	37
4.3.2.4 Class EffectManager . . . . .	38
4.3.2.5 Instantiation, Variables and the Rest . . . . .	38

4.3.3	CSSs for Overview Effects . . . . .	38
4.3.4	Scrollbar + Dock Effect . . . . .	39
4.3.5	Stacked Cards Effect . . . . .	39
4.3.6	Tabs Effect . . . . .	39
4.4	Back End Calculation . . . . .	39
4.4.1	Global Scope . . . . .	39
4.4.2	Message Reception . . . . .	40
4.4.3	Iteration Limit . . . . .	40
4.4.4	Iteration Count for One Point . . . . .	40
4.4.5	Image Generation . . . . .	40
4.4.6	High Precision Version . . . . .	40
4.5	Utility Assets . . . . .	40
4.5.1	Folder ./js . . . . .	40
4.5.2	Folder ./fa . . . . .	40
4.5.3	Folder ./bs . . . . .	40
4.5.4	Folder ./css . . . . .	40
<b>5</b>	<b>Results and Comparison</b>	<b>41</b>
5.1	General Results . . . . .	41
5.2	Comparison Between Different Arrangement Methods . . . . .	41
5.3	Future Work . . . . .	41
<b>A</b>	<b>Frequently Asked Questions</b>	<b>42</b>
A.1	Where can I find the source code of this project? . . . . .	42
A.2	Is there L <sup>A</sup> T <sub>E</sub> X source code for this thesis? . . . . .	42
	<b>Bibliography</b>	<b>43</b>

# List of Figures

1.1	Overview Plus Details On Map . . . . .	1
1.2	Overview Plus Details In Photoshop . . . . .	2
1.3	Overview Plus Details In Computer Games . . . . .	3
1.4	Focus Region Becomes A Dot On Context Region . . . . .	3
1.5	Multiple Levels of Overview Plus Details . . . . .	4
1.6	Zoomed-in Fractal Image . . . . .	4
1.7	Occupied Screen . . . . .	5
1.8	MacOS Dock . . . . .	6
1.9	MacOS Dock Hover . . . . .	6
1.10	MacOS Dock With More Apps . . . . .	6
1.11	Modern-looking Scrollbar . . . . .	7
1.12	Means of Arrangements On iOS Safari . . . . .	8
1.13	How Google Chrome Arranged Objects of Interest . . . . .	8
1.14	“Preview” Mechanism on Windows 10 . . . . .	9
1.15	Fade-in Effect in jQuery . . . . .	10
1.16	Moving Preview Plane On Context Views . . . . .	10
2.1	Mandelbrot Set Graphical Presentation . . . . .	19
3.1	Basic Structure . . . . .	21
3.2	Different Reponse Types . . . . .	22
3.3	A Pair of Focus + Context . . . . .	23
3.4	Manager of All The F+Cs . . . . .	24
3.5	Manager of Effects . . . . .	25
4.1	File Structure . . . . .	26
4.2	DOM Body Structure . . . . .	32
4.3	Magnification Level . . . . .	33
4.4	Map Visual Pair . . . . .	34
4.5	Pairing Multiple Levels of Maps . . . . .	35
4.6	Message Exchange . . . . .	39



# List of Tables

# List of Abbreviations

## Acronyms

**API** Application Programming Interface.

**CPU** Central Processing Unit.

**CSS** Cascading Style Sheets.

**DOM** Document Object Model.

**GIS** Geographic Information System.

**GPU** Graphics Processing Unit.

**HTML** Hypertext Markup Language.

**HTTP** Hypertext Transfer Protocol.

**IT** Information Technology.

**JS** JavaScript.

**JSON** Java Script Object Notation.

**LOD** Level of Detail.

**PC** Personal Computer.

**UI** User Interface.

**URL** Uniform Resource Locator.

## Glossary

**CSS3** In latest evolution of the standard that defines **HTML**, comes together with the extended **CSS**, which is often referred as CSS3..

**F + C** Focus + context, overview + details, mini-map or **minimap**, same concept as the entry **Map**.

**Full HD** 1920 x 1080 px; also known as **Full HD** or **FHD** and **BT.709**.

**HTML5** Latest evolution of the standard that defines **HTML**.

**Map** Overview + details, focus + context, mini-map or **minimap**.

**O + D** Overview + details, focus + context, mini-map or **minimap**, same concept as the entry **Map**.

*Special thanks to Dr. Zhonghua Xu, Ms. Meng Wang and Mrs. Vivian E. Rice, I wish I could share the joy of this achievements with all of you - here or in Azeroth.*

## Chapter 1

# Introduction and Motivation

### 1.1 Introduction

In many modern-day applications and computer programs, we may encounter frequently a situation where users have to browse or interact with a large set of data or information, however, only focusing only on a certain part of them. In many situations, this part is usually a lot smaller than the original dataset relatively. The most commonly seen scenario would be a **Geographic Information System (GIS)** programs on computers, for example an **O + D** display shown in **Figure 1.1 Overview Plus Details On Map**. These type of applications usually have a zoomable interface to allow users to zoom into a specific region of the original dataset. Also they usually allow users to browse around the dataset from the current focus region and navigate and interact with the whole large dataset with the sense of the “large picture” by the mechanism of **O + D** techniques.

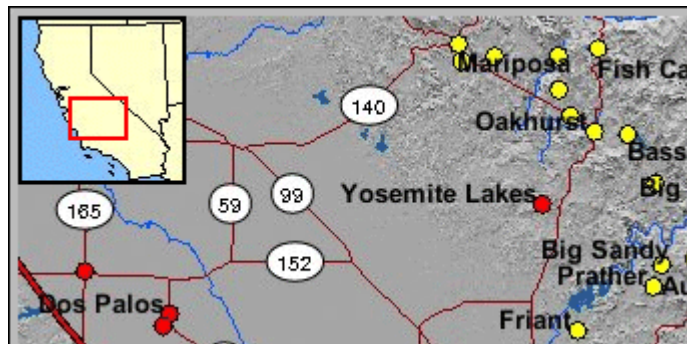


FIGURE 1.1: From <http://wildfire.usgs.gov>, an overview of the graphics next to a zoomed “detail view”.

Other examples besides **GIS** include also some image processing, image generation applications such as Photoshop shown in **Figure 1.2 Overview Plus Details In Photoshop**, because usually the resolution of the image being processed or generated are larger than the resolution of one single screen monitor. Another interesting example would be in the modern computer gaming industry, that the concept of *Mini-maps* were invented, illustrated in **Figure 1.3 Overview Plus Details In Computer Games**, with the same concepts due to the fact that the virtual world of digital games are mostly a lot larger than how much one single screen can contain.

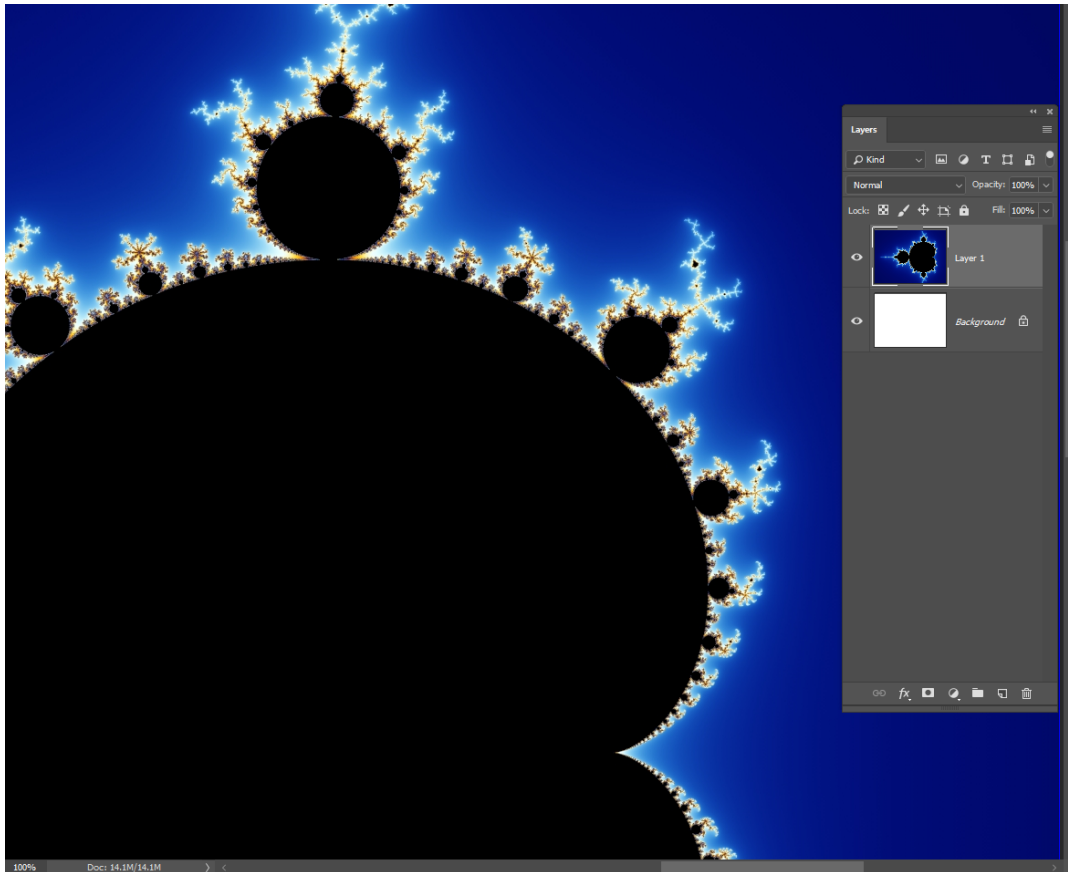


FIGURE 1.2: The basic mechanism in the application Photoshop, layers, offering an overview functionality, with detailed focus area on the main visual area.

Note that the term of *Overview + Details* sometimes are also referred to as *Focus + Context*, *Mini-map* or some other terms, however, they are referring to a similar techniques or concepts.

## 1.2 Motivation For More Complicated Situations

Most of these techniques and examples above are feasible and can improve how human comprehend the information or datasets is because the whole context area, although larger than how much one screen can hold, comparing to the focus area, is still not so much bigger. If you're looking at a street plan of a city, the focus area can still be visibly represented as a rectangle if the context area is only as large as a city. If you're looking at an image with the resolution of twice as wide and tall as your screen monitor, the focus area still has a quarter of the size of the context area.

However, there are situations when these techniques will not be able to improve our comprehension of the whole dataset in a very good way, that is when the resolution of the original dataset is getting too high and the focus area is zoomed in into an extremely detailed state. That way, the focus region becomes a "dot" instead of a region to be represented on the context view, losing its width and height properties



FIGURE 1.3: The computer game *Freeciv* has a mini-map in the bottom left corner. This is a similar concept as  $O + D$ . On this mini-map the white rectangle represents the area of the map currently visible on the main screen. The different colors represent land and ocean and the territories of the different players. The white dots are the position of cities and the blackness are the unexplored areas (the “fog of war”)[25].

and stops giving intuitive information. A simple example is shown in **Figure 1.4 Focus Region Becomes A Dot On Context Region** indicating that when the resolution of the dataset is extreme, new approaches need to be taken in order to let the users able to “grab the whole picture”.

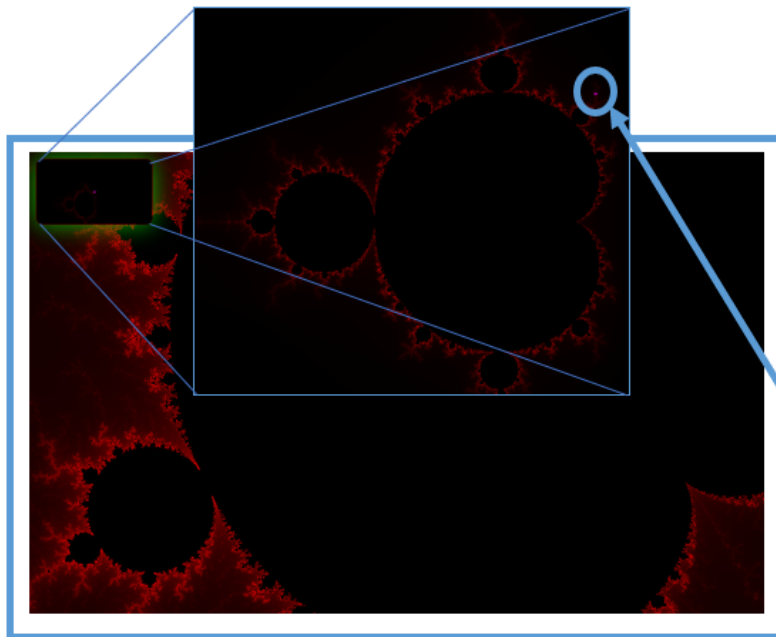


FIGURE 1.4: Traditional  $O + D$  techniques stop to provide intuitive information on a highly zoomed-in fractal image.

In these situations, a new level of the **O + D** techniques is introduced, which is to provide more levels of overviews to the user. Take what's shown in **Figure 1.5 Multiple Levels of Overview Plus Details** as an example, multiple levels of information of objects are presented as overviews for the users to browse and search. For a similar solution as shown previously in **Figure 1.4 Focus Region Becomes A Dot On Context Region**, a solution shown in **Figure 1.6 Zoomed-in Fractal Image** is to give multiple hierarchical overviews to the user to preserve the intuitiveness of the techniques.

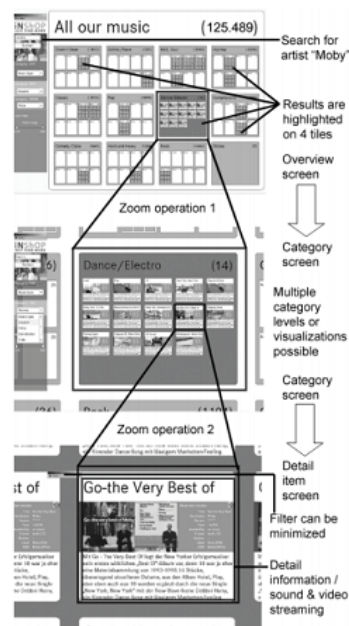


FIGURE 1.5: ZEUS from overview to detail view by zoom in[8].

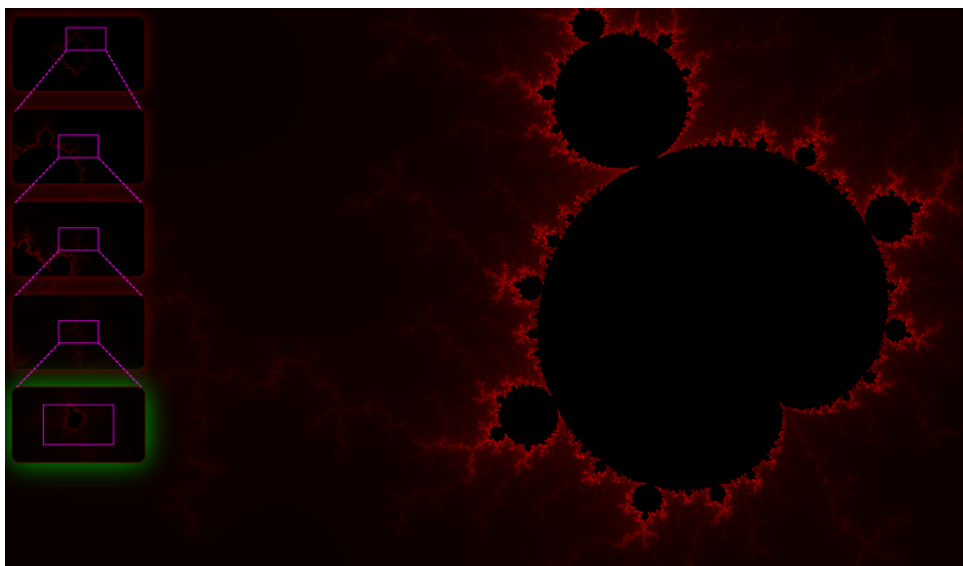


FIGURE 1.6: A graphical representation of Mandelbrot set zoomed in for around 35 thousand times compared to the original.



This solution is straightforward, however, we can still raise questions to push the topic into new levels — what happens when the resolution of the dataset increases even more?

In this case, more overviews need to be added and presented to the users. When more overviews are added providing more **Levels of Detail (LODs)** and contexts and put to the screen, the original problem emerges again — these overviews are going to occupy a large portion even the entire screen monitor so the most zoomed-in detail area, the most important area of interest, is not going to be easily comprehended or even not able to be seen, as shown in **Figure 1.7 Occupied Screen**.

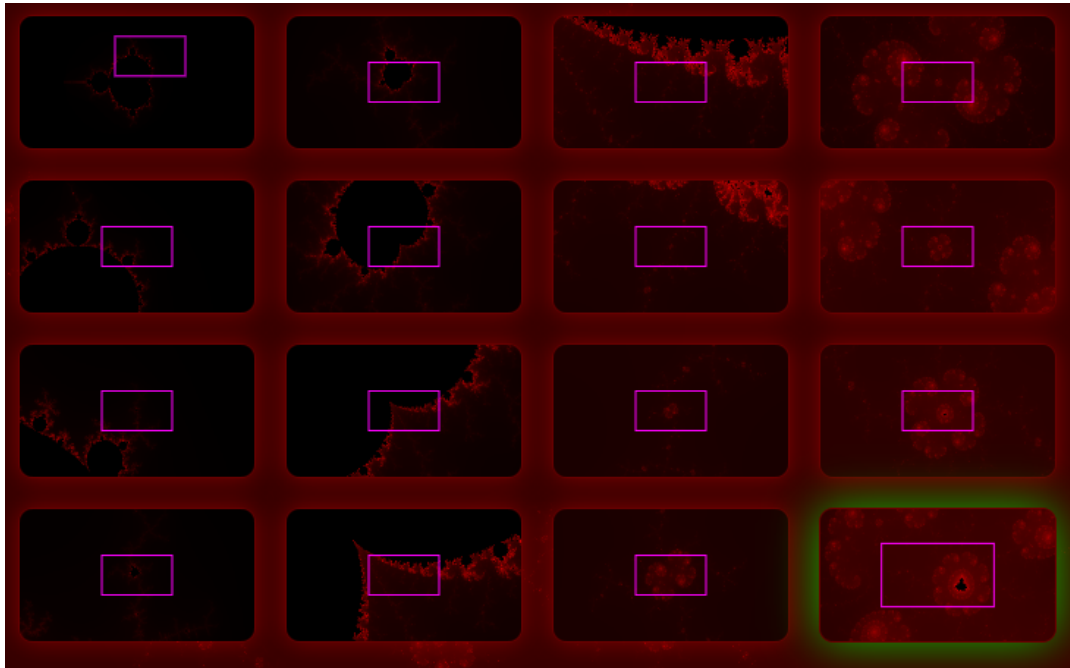


FIGURE 1.7: Too many overviews occupying entire screen monitor.

Therefore, the topic of this thesis is focused on solving this problem, to arrange these overviews in certain ways that they can all provide hierarchies of overviews of information with respect to the original extreme resolution dataset, at the same time, being intuitive enough for human users to understand the **LODs** as well as the most detailed region with the help of all these arrangements.

### 1.3 Inspirations for the Arrangements of Context Views

As we mentioned above, in order to prevent situations like shown in **Figure 1.7 Occupied Screen** that all tiled context views occupying the entire focus view from happening, we figured out several ways of arranging these context views, and the inspirations of which came from various ways of how we interact with **IT** related objects.

### 1.3.1 Dock And Scrollbar

First thing that came into my mind was the state-of-the-art designs from *Apple*. They invented the concept of macOS Dock or *Dock* as shown in [Figure 1.8 macOS Dock](#), which can be used as an idea of arranging multiple objects of interest on one screen. Not that as shown in [Figure 1.9 macOS Dock Hover](#), that it adds an eye-catching visual effect to the current focused object and some other coherent objects.

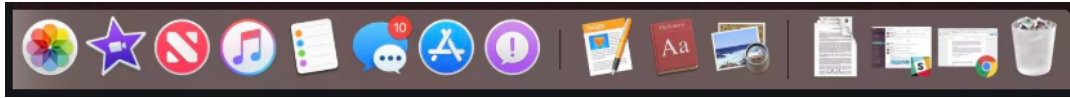


FIGURE 1.8: Apple gives users the ability to put interested Apps on one side of the screen, which can be the interested contexts in our project.



FIGURE 1.9: When user hovers over an object of interest on the Dock, the Dock can show a visually distinguishable effect.

However, in situations that when user tries to add more interested objects to the Dock, this design only shrinks the sizes of the objects and cannot provide capabilities to embody more objects and advance further, as shown in [Figure 1.10 macOS Dock With More Apps](#).



FIGURE 1.10: Apple Dock has certain limitations of holding more objects inside.

At the same time, to display more object in a smaller container, there is an obvious way of putting a scrollbar beside it, as shown in [Figure 1.11 Modern-looking Scrollbar](#). Therefore, the first idea of combining a scrollbar together with the visual effects of the macOS Dock were formed.

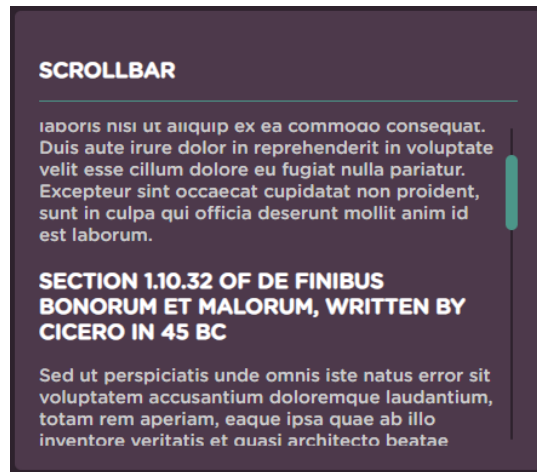


FIGURE 1.11: An example of using a scrollbar on the side of a container to allow users to browse through more information inside this container.

### 1.3.2 Stacked Cards

We all browse through different web pages for some information nowadays, and we all had the necessity to keep several web pages open at the same time at some point. The amount of opened websites can easily grow out of hand and how the modern web browsers are handling these web pages of interest can easily catch our attention. The next thing that caught my attention is the *iOS Safari* browser. It is a famous browser installed by default on a popular mobile phone which is a device that has limited amount of space for visual area. How it's displaying the opened web pages are shown in [Figure 1.12 Means of Arrangements On iOS Safari](#).

Visually, they look like cards stacked upon each other. What if we arrange it in a way that this stack of cards can dynamically adjust their positions on screen and the angles they pile up with as their numbers grow? This can be the next good way of arranging these context views of our project.

### 1.3.3 Tabs

As mentioned before, how browsers arrange the open web pages can be really good examples for our project, therefore, we can't neglect the conventional way of arranging the open web pages on the top of the visual area as "opened tabs", like shown in [Figure 1.13 How Google Chrome Arranged Objects of Interest](#). This is the third idea of cases we make in this project.

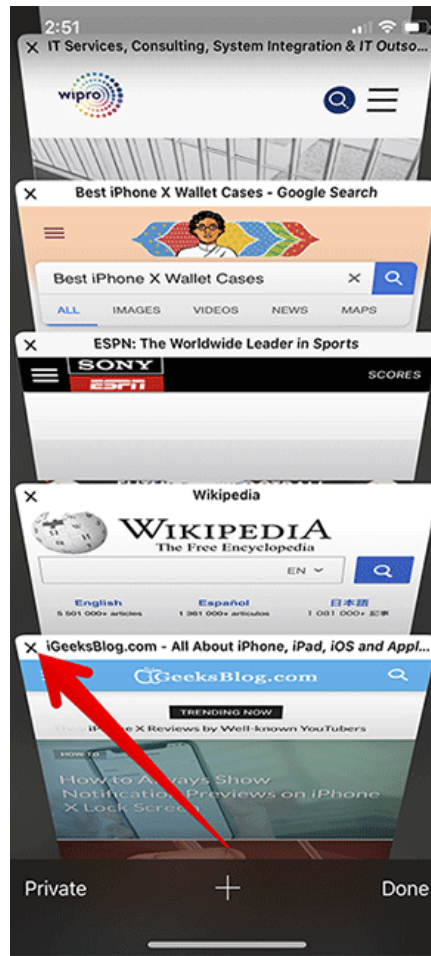


FIGURE 1.12: How iOS Safari arranges multiple open web pages on an iPhone.

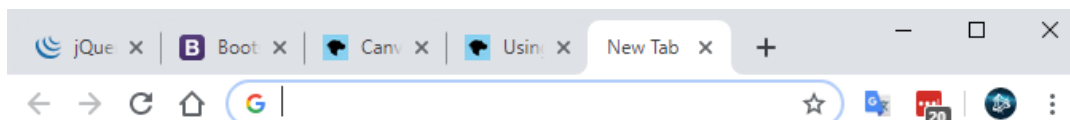


FIGURE 1.13: The way Google Chrome handles the open web pages are to put a so-called “tab” on top of visual area shaped like a tag, with informative titles which sometimes are shortened with ellipsis at the end.

### 1.3.4 Previews Of Contexts

As shown in [Figure 1.6 Zoomed-in Fractal Image](#), the context views can be relatively small when put to action. Therefore, we need some sort of mechanism to let the user “preview” the shrunk context views, in a similar sense where *Windows 10* users can hover their cursor over the bottom right corner of their screen monitor to have a “glance” of how their desktop looks like shown in [Figure 1.14 “Preview” Mechanism on Windows 10](#), hiding the **User Interface (UI)** of the current open applications.

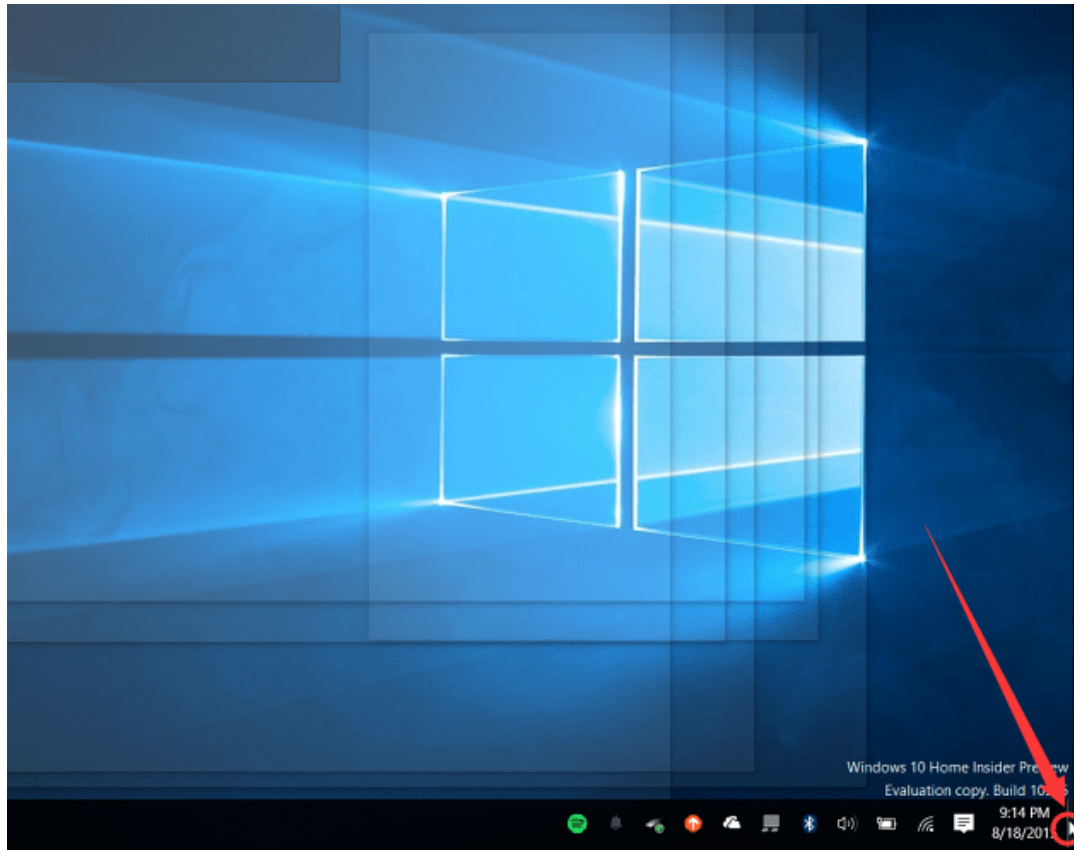


FIGURE 1.14: When users hover their mouse cursor on bottom right corner of a Windows PC and have a glance of what is on their Desktop.

A most natural preview setup would be like this, when the user hovers over one of the context view, display an enlarged image in **Full HD** that shows the context. That is in fact the default setting of this project.

As inspired by the same *Windows* effect, the second visual effects type becomes a fade-in and fade-out effect. This effect, as shown in **Figure 1.15 Fade-in Effect in jQuery**, is adding a small transition in the transparencies of the context to be displayed on screen — smoother and can give positive user experience in the comprehension of the **F + C** problem.

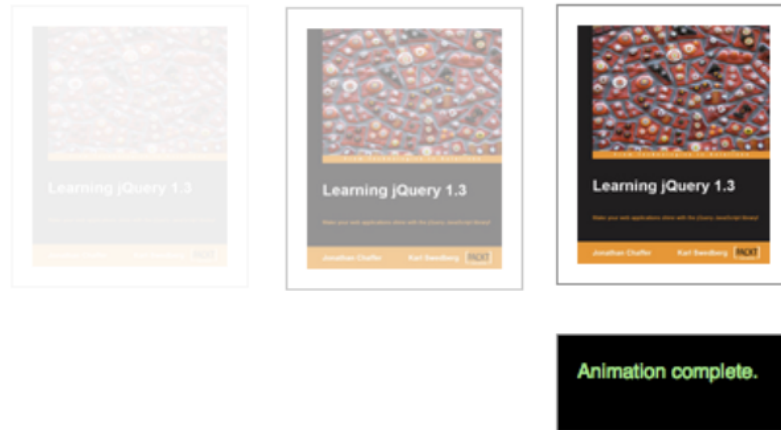


FIGURE 1.15: Illustration of the `fadeIn()` effect in *jQuery*, a transparency change of objects to be displayed.

Another way of giving previews of the contexts is inspired by some modern graphics and video work, such as a deep zooming video of Mandelbrot set<sup>1</sup> or some special scenes in a movie such as *Limitless*<sup>2</sup> or *Man In Black*<sup>3</sup>.

As shown in **Figure 1.16 Moving Preview Plane On Context Views**, it would be really nice if we implement this visual feature that when user wants to preview one of the contexts, instead of magically showing that context, we show this process of letting the preview image raise or sink from the “current” depths to the “intended” depths of the context. Therefore, this second ways of previewing the contexts got to be implemented into this project.

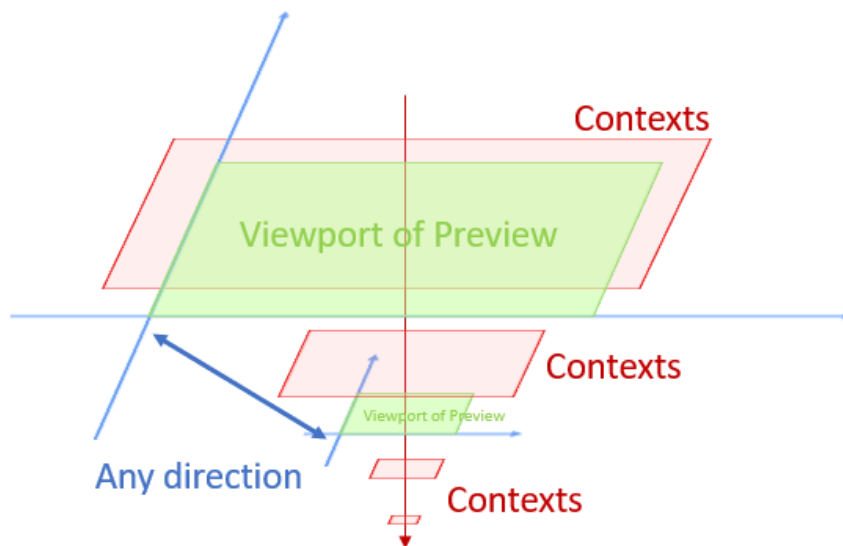


FIGURE 1.16: The viewport of preview is like a plane raising or sinking to the desired depths on the z axis.

<sup>1</sup> See <https://www.youtube.com/watch?v=pCpLWbHVNhk> for more information.

<sup>2</sup> See <https://www.youtube.com/watch?v=xqv1maJaDtQ&t=37> for more information

<sup>3</sup> See <https://www.youtube.com/watch?v=0KnpPCQyUec> for more information.



## 1.4 Related Work

There are many researchers who have already done plenty of work related to the topic of **O + D**, **F + C** or **LOD**.

Some researchers did work on a single level of **F + C** techniques:

Cockburn et al. [4][5] reviewed and compared **O + D**, zooming, and **F + C** interfaces. They described that **O + D** uses a spatial separation between focused and contextual views, zooming is temporal separation of views and **F + C** displays the focus within the context.

Hornbæk et al. [11] compared zoomable user interfaces with and without an overview by experiments to understand the navigation patterns and usability of these interfaces. In a later work, Hornbæk et al. [12] extensively reviewed papers that used the notion of overview and developed a model which highlights the awareness that makes up an overview, the process, by which users acquire it, the usefulness of overviews, and the role of user-interface components in developing an overview.

In [2] considered Baudisch et al. mainly the technique focus plus context screens. They experimented and compared this technique with the other two techniques overview plus detail and zooming/panning. They noted, for interaction with dynamic views, the technique focus plus context screens alone seems not to be enough and needs an additional monitor. In a related work, Baudisch et al. [1] present the technique focus plus context screens and implemented a system with this technique by combining multiple display units of different resolution. Their means is particularly suitable for situations where all information is shown in only one view and switching between multiple views should be avoided.

Roto et al. [18] developed a Web page visualization method minimap for mobile phones that shows pages in a modified original layout and navigates a Web page with a mini map view.

Gutwin et al. [9] compared three techniques: fisheye, zoom and panning, to find out what is the best ways to redesign a large UI to fit a smaller screen of mobile devices.

Holmquist [10] developed a flip zooming as a new focus+context technique: Mini Pages are placed in a simple left-to-right, top-to-bottom ordering on the display. When a page is brought into focus, it is enlarged and placed approximately in the middle of the display, with the other Mini Pages arranged around it.

There are also some researchers did work related with multiple **LODs**:

Pan et al. [26] present a summary on the mesh simplification techniques which they used for creating models at multiple **LOD**. In this paper they also compared typical methods.

Clark [3] proposed hierarchical geometric models for visible surface algorithms for producing computer pictures. He described the benefits of using more than one representation of a model for image rendering and pointed out that objects that cover a small area of the screen can be rendered from a simplified version of the object and that this allows more efficient rendering of a scene.

Crow described the benefits of having both simple and complex representations of an object in his paper on an image generation environment.[6] He gave an example of a chair that is represented in high detail, medium detail and very low detail. These

models with three level of detail were created by hand. He suggested that creating the lower levels of detail is a process that should be automated.

Gundelsweiler et al. [8] developed a system of hierarchical information structure. Objects and categories are organized in groups on different hierarchy level visualized as tiles on the screen by the user. The search is supported by zooming/panning navigation.



## Chapter 2

# Background

In this chapter, we'll introduce some necessary background knowledge in order to let the user understand better how this project came into its form.

Not only the technology stack that this project is using is important to further understanding of this project, but also some theoretical part that's behind them. We'll be introducing them gradually.

## 2.1 Web Technology Stack

Before starting to program on the software that shows the different ways to show an interface providing **F + C** with many **LOD**, we must first choose the nature and language of the program. A web technology stack program stands out easily and as a matter of fact, was also the first choice for the development of the immature prototype in the first place. There are several reasons behind it. For web technology stack developed programs:

- **Compatibility** Code once, it works everywhere. It saves lots of time for users and testers to read a long manual. All we need is a working browser with loose requirements to it.
- **Simple Installation** Almost nothing to be installed in order to see the project up and running. Besides a browser of some version specifications, only a web server of *any kind* is required, as described in [section 4.2 Start the Project](#).
- **Appearance** Can look modern with almost zero code. Can have cool visual effects a lot easier than desktop development.
- **Successful Case** If it works for Google Maps, it should work for our project as well.
- **Performance** Slower in a sense because hardware resources are not easily fetched for browser-based **UI** heavy applications.

On the other hand for desktop development:

- **Compatibility** Really common to be “working on my computer” but not on users’.
- **Simple Installation** Lots of preconditions and require lots of planning before the delivery of a standalone package. If not planned and all aspects chosen

carefully, applications might not work on Macs and gives pop-ups to require “vcredist” dependencies on other operating systems.

- **Appearance** Requires lots of coding to have the most basic appearance. Visual effects need to be programmed manually one by one, not to mention the planning of the software structure.
- **Successful Case** Works also fine with lots of deep zooming applications such as *Ultra Fractal* and *Kalles Fraktaler*.
- **Performance** Can be a lot faster than web applications if choosing C / C++ or similar languages since they can access directly your hardware resources.

Since this project is focused a lot on the front end **UI**, web technologies is chosen even when performance can sometimes hinder the advancement of some applications. If planned well, performance on the deep zooming part of the project can be improved by coding techniques and algorithms, however, on the other hand, programming front end **UI** effects together with planning a good structure to use the advantages of calculation speed of desktop applications would not be productivity efficient.

Some other approaches were also considered, for example to use an intermediate solution like Python because it can be easily structured and coded, however, most of them were not as good choices as web technologies. Taking Python as an example, it cannot do conventional iterations over pixels fast enough unless complicated techniques are used, which actually brings it back on the same level with desktop programming, not to mention the invocation of hardware **GPU** / **CPU** resources and fast image manipulations — they all bring lots of complications to the implementation of the project.

## 2.2 Web, HTML5 And Canvas API

After the choice of web technology stack, we’ll be talking about the equipments that are used in the project.

Technically speaking, web wechnologies are focused on **UI** and communications between servers and computers since the computers and servers on the internet can’t communicate with each other the way people do, however, the communication part is not hugely focused in this project. First thing to know is that web technology stack includes the markup languages and multimedia packages servers and computers use to communicate, however, it’s the markup languages and multimedia packages that we focus on, not the communication part, since the communication involved currently is still on local machines.

### 2.2.1 Browsers

Web technology runs on browsers. Browsers are the tools that request information and then they show us users in the way we can understand, as the interpreters of the web programs. There are many famous web browsers around:

- **Google Chrome** Currently the most popular browser by Google
- **Safari** Apple’s web browser

- **Firefox** Open-source browser supported by the Mozilla Foundation
- **Internet Explorer** Microsoft's browser

And since Google Chrome has the most popularity and supports all of our requirements for the project, we chose Google Chrome as our browser for the project.

### 2.2.2 Markup Language

There are some fundamental things for web programming. The first things we need to know about are **Hypertext Markup Language (HTML)**, **Cascading Style Sheets (CSS)** and **JavaScript (JS)**.

**HTML** **HTML** is one of the first thing to be introduced in this section. Thanks to **HTML**, the browsers can know what to present when we hit them we a request. It is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as **CSS** and scripting languages such as **JS**.<sup>[22]</sup>

**CSS** The technology assisting **HTML** describes how **HTML** elements are to be displayed on the screen, if to be put straightforward. It is a style sheet language used for describing the presentation of a document written in a markup language like **HTML**.<sup>[20]</sup> **CSS** is a cornerstone technology of the web technologies, alongside **HTML** and **JavaScript (JS)**.<sup>[7]</sup>

**JavaScript** Alongside **HTML** and **CSS**, **JavaScript (JS)** is one of the core technologies of the World Wide Web.<sup>[7]</sup> **JavaScript** enables interactive web pages and is an essential part of web applications. The vast majority of websites use it,<sup>[19]</sup> and major web browsers have a dedicated **JavaScript** engine to execute it.<sup>[23]</sup>

### 2.2.3 HTML5

It is worth mentioning that there is more to the technologies this project is using than the traditional **HTML** / **CSS** / **JS**, which is the larger set — **HTML5**. **HTML5** is the latest evolution of the standard that defines the original. The term **HTML5** represents two different concepts:

- It is a new version of the markup language **HTML**, with new elements, attributes, and behaviors
- A larger set of new web technologies that allows the building of more diverse and powerful applications.

Therefore, this set is sometimes called **HTML5 and friends** and often shortened to just **HTML5**.

There are lots of new features that **HTML5** brings to the original **HTML**, but we're not describing all them there in full<sup>1</sup>. The core features we care about and using in this project are the following two:

---

<sup>1</sup> See <https://en.wikipedia.org/wiki/HTML5> to know more details

**Graphics and Effects** Allowing a much more diverse range of presentation options, with respect to 2D / 3D graphics and effects.

**Performance and Integration** Providing greater speed optimization and better usage of computer hardware.

**Styling** Allowing more sophisticated themes and **UI** experience.

Solving exactly the problems of our focus points in this thesis. Some more specific details to each of the above points and their applications in this project will be shortly described.

In the graphics part, the new element `<canvas>` element that allows delicate drawing on web pages are heavily used.

In the performance and integration part, *Web Workers* are used allowing delegation of **JS** evaluation to background threads, and allowing these activities to prevent slowing down interactive events. The separation used in the project can also be easily advanced by replacing this multithreading calculation part with the new *XMLHttpRequest* allowing fetching information asynchronously, also allowing it to display dynamic content, varying according to the time and user actions<sup>2</sup>.

In the styling part, since the new **CSS** in **HTML5**, **CSS3**, has been extended to be able to style elements in a much more complex way, many new features of **CSS3** are used, such as:

**Background Styling** The possibility to put shadows on elements using `box-shadow` and **CSS** filters. These advanced box effects are used in this project for the display of halos of the context views and also internally in the dependencies of this project, *jQuery*<sup>3</sup> and *Bootstrap*<sup>4</sup>.

**Animating of Styles** Using **CSS** Transitions to allow the animation between different states or using **CSS** Animations to animate parts of the page without a triggering event, for example some movement animations of the context views in this project.

**New Presentational Layouts** Two new layouts have been added to **CSS3**: the **CSS** multi-column layouts and **CSS** flexible box layout, and the latter is used in multiple places in this project, for example, the default layouts for the context views and the tabs layouts for the context views. The dependency *Bootstrap* also relies largely on this new feature.

---

<sup>2</sup> This is the technology behind *Ajax*. To know more information about *Ajax*, see [https://en.wikipedia.org/wiki/Ajax\\_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming)).

<sup>3</sup> *jQuery* is an open-source, fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and *Ajax* much simpler with an easy-to-use API that works across a multitude of browsers. See <https://jquery.com/> for more information.

<sup>4</sup> *Bootstrap* is an open source toolkit for developing with **HTML**, **CSS**, and **JS**. See <https://getbootstrap.com/> for more information.

### 2.2.4 Canvas API

**HTML5** brought exciting new advantages to the **HTML** coding world as mentioned above. Amongst them all, one of the most thrilling one, the new `<canvas>`, allows us to render delicate graphics powered by **JS**. We'll be talking about what it is and then the **Application Programming Interface (API)** that supports its usages.

`<canvas>` is an **HTML** element which can be used to render graphics via scripts<sup>5</sup>, here in our project **JS**. It can, for instance, be used to draw graphs or create simple (and not so simple) animations. In our project, `<canvas>` is used for the graphical representation of the extreme resolution datasets.

It's also worth mentioning that after first introduced in *WebKit* by *Apple* for the *OS X Dashboard*, `<canvas>` has since been implemented and supported in all major browsers and today.

The Canvas **API** itself is a sense of how to use scripts to draw on this newly introduced **HTML** element. It is largely focuses on 2D graphics. The reason to bring it up here together with the `<canvas>` element is that there are many other **APIs** that come together with it, for example the **WebGL API**, which also uses the `<canvas>` element, draws hardware-accelerated 2D and 3D graphics. In another word, the context of the `<canvas>` element<sup>6</sup> is "2d" through the Canvas **API**, not "webgl" context or any other.

## 2.3 Mandelbrot Set

As for the datasets themselves, we use a kind of computable dataset instead of a static and deep zoomable image information dataset. The advantage is obvious that by using a dataset that can be computed for its information of any specific point in the set, we don't have to store the information anywhere anymore, as if we did, it would require a lot of investment and investigation for the storage or fetching of the dataset for this project. The downside of it is also clear that we'll have to trade time for space, since the calculation is usually going to take more time than only querying from a static set of data. It is proved to be solid to do this trade, since in this project, the visually inspected results shows that the waiting time is still inside human tolerable margins.

The dataset we chose is the Mandelbrot set. The Mandelbrot set is a famous example of a fractal in mathematics. It is named after Benoît Mandelbrot, a Polish-French-American mathematician<sup>7</sup>. The reason this thesis is using Mandelbrot set as the source of extreme resolution dataset is because Mandelbrot set can provide theoretically infinitely high resolution datasets. In this section, we'll introduce briefly Mandelbrot set, its algorithms ideas for visualization.

<sup>5</sup> To know about the `<canvas>` element, see [https://en.wikipedia.org/wiki/Canvas\\_element](https://en.wikipedia.org/wiki/Canvas_element).

<sup>6</sup> Here not the same concept as the "context" in the previously mentioned term **F + C**.

<sup>7</sup> Its definition and name are due to Adrien Douady, in tribute to the mathematician Benoît Mandelbrot[24].

## What Is Mandelbrot Set

Firstly, we recursively define a sequence  $(z_n)_{n \in \mathbb{N}}$  as follows:

$$z_0 = 0 \quad (2.1)$$

$$z_n = z_{n-1}^2 + c \quad (2.2)$$

For different constant  $c$ , the absolute value of  $z_n$  could remain bounded, could also be divergent, if  $n$  is increased.

The Mandelbrot set  $M$  is the set of  $c$  in the complex plane for which the sequence  $(z_n)_{n \in \mathbb{N}}$  remains bounded.

## Important Properties of the Mandelbrot Set

A property of Mandelbrot set is as follows:

A complex number  $c$  belongs to the Mandelbrot set  $M$ , if and only if the absolute value of  $z_n$  is not larger than 2, for all  $n = 0, 1, 2, \dots$

## Simple Graphical Presentation

The following figure [Figure 2.1 Mandelbrot Set Graphical Presentation](#) is a simple graphical representation of Mandelbrot set. A point  $c$  in the complex plane is conventionally colored *black* if it belongs to the Mandelbrot set  $M$ , and *white* if not.

## Algorithms for Visualization

For each  $c$  in complex plane, the following is done in an outer loop:

```
{
  For current  $c$ , we calculate in an internal loop sequence  $z_n, n = 0, 1, \dots, max$ :
  {
     $z_0 = 0$ , (initial value),
     $z_{n+1} = z_n^2 + c$ 
    If the value of  $z_{n+1}$  is larger than 2, then:
       $c$  does not belong to  $M$ . In this case, we set the color of  $c$  to white and the
      calculation of sequence is stopped.
    If  $n = max$ , then:
       $c$  belongs to  $M$  and we set the color of  $c$  to black.
  }
}
```

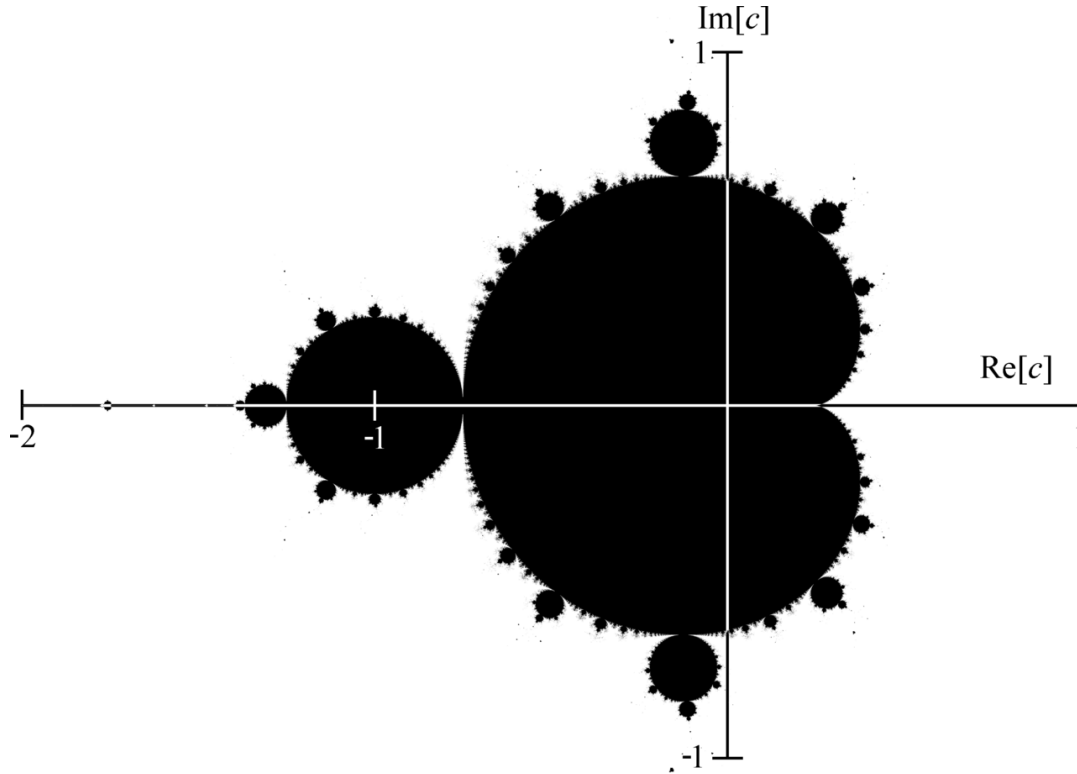


FIGURE 2.1: A simple graphical representation of Mandelbrot set.

### Algorithms Idea for Graphical Representation with Grayscale

In the above algorithm in [2.3 Algorithms for Visualization](#),  $c$  is set to either *black* or *white*. If the iteration number  $n$  is equal to the maximum iteration number  $max$  and the value of  $z_n$  still less than 2, then  $c$  has the color *black*. If the iteration number  $n$  is smaller than  $max$  then at this moment the absolute value of  $z_n$  becomes larger than 2. In this case, we cannot set the color of  $c$  to *black*, because  $c$  does not belong to Mandelbrot set. However, we set a color with a portion of *black*, to indicate how close  $c$  is to be in *black* area.

For each  $c$  in complex plane, the following is done in an outer loop:

{

For current  $c$ , we calculate in an internal loop sequence  $z_n, n = 0, 1, \dots, max$ :

{

$z_0 = 0$ , (initial value),

$$z_{n+1} = z_n^2 + c$$

If the value of  $z_{n+1}$  is larger than 2, then:

$c$  does not belong to  $M$ . In this case, we set the color of  $c$  to *white grayscale*<sup>8</sup> depending on the number of iteration  $n$  and the calculation of sequence is stopped.

If  $n = max$ , then:

<sup>8</sup> In the current implementation, this color is set to be grayscaled red,  $\text{rgb}(\text{grayscale} \% \times 255, 0, 0)$ .

$c$  belongs to  $M$  and we set the color of  $c$  to *black*.

}

}



## Chapter 3

# Architectural Designs

Before starting to talk about the implementations of the project, we'll first describe the architecture of the project on a design level, without describing the details on the tech ends.

### 3.1 Basic Structure

The basic structure of this project is fairly simple, as shown in **Figure 3.1 Basic Structure**.

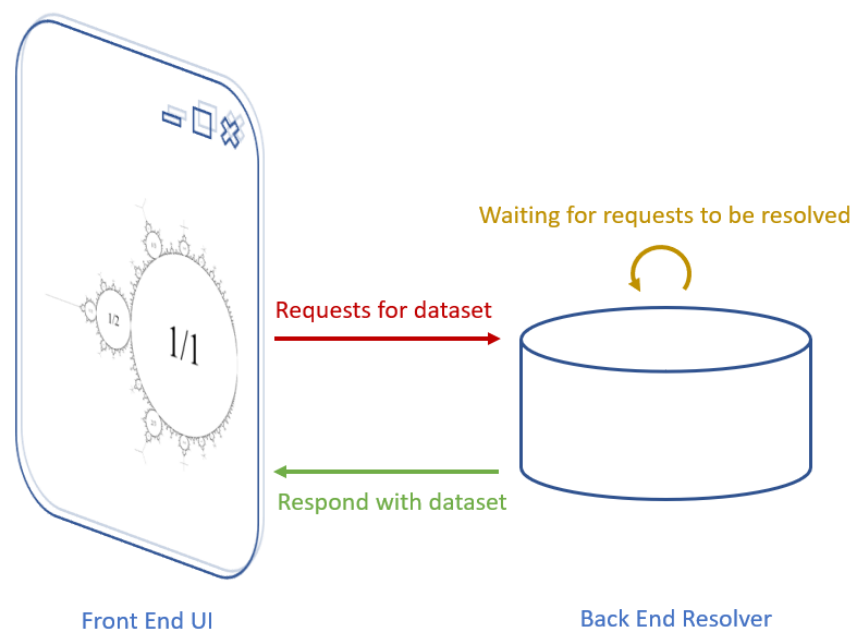


FIGURE 3.1: An illustration of the basic structure of the project, a front end that makes requests and a resolver that responds to the connected front end.

First of all, a front end **UI** that organizes the content of the project, handling user interactions and so on, and making requests when some actual data of the extreme resolution dataset is needed.

Secondly, a back end resolver that resolves the requests coming from the connected front end, through necessary methods such as fetching an image from the database

or calculate some values out of an equation. In the current state, however, this back end resolver should “resolve” the “problem” by calculation and not data fetching, calculating a portion of the dataset Mandelbrot set. It should return an image as an answer together with some necessary parameters so the front end can verify and use them as either a complete result image or as a intermediate buffer image, as shown in [Figure 3.2 Different Reponse Types](#).

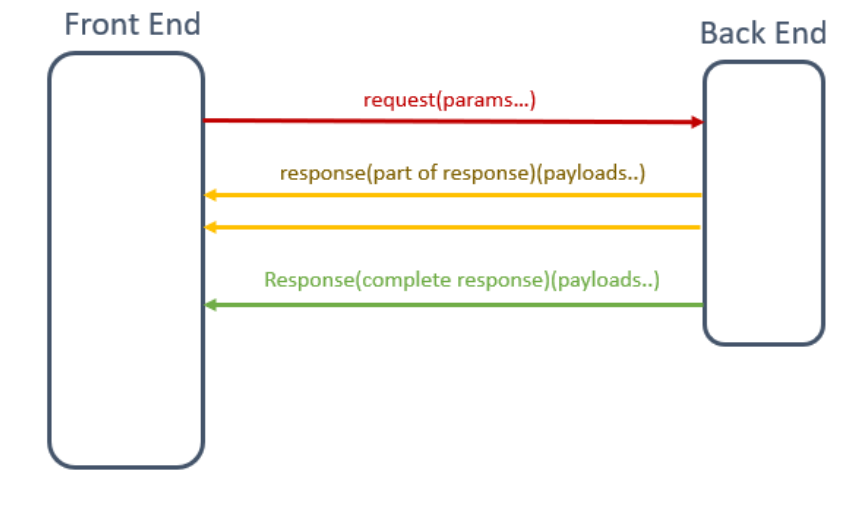


FIGURE 3.2: Responses of different types that the resolver can answer to the front end.

## 3.2 Front End UI

For starters, the front end should consist of three major parts: models of the **F + C**, a manager to manage all these **F + Cs** and a effect manager to manage the effects to be displayed and some other **UI** interactionss and controls from the user which in turn results to more effects.

First of all, let's look at the model of a pair of **F + C**. This model should represent a tiny manager that manages one focus view together with a context view, as shown in [Figure 3.3 A Pair of Focus + Context](#). It's connected with these two targets and when it's being instructed to send messages to the outside world(not inside front end criteria), requests and receives responses and handle the render event, putting them on corresponding canvases.

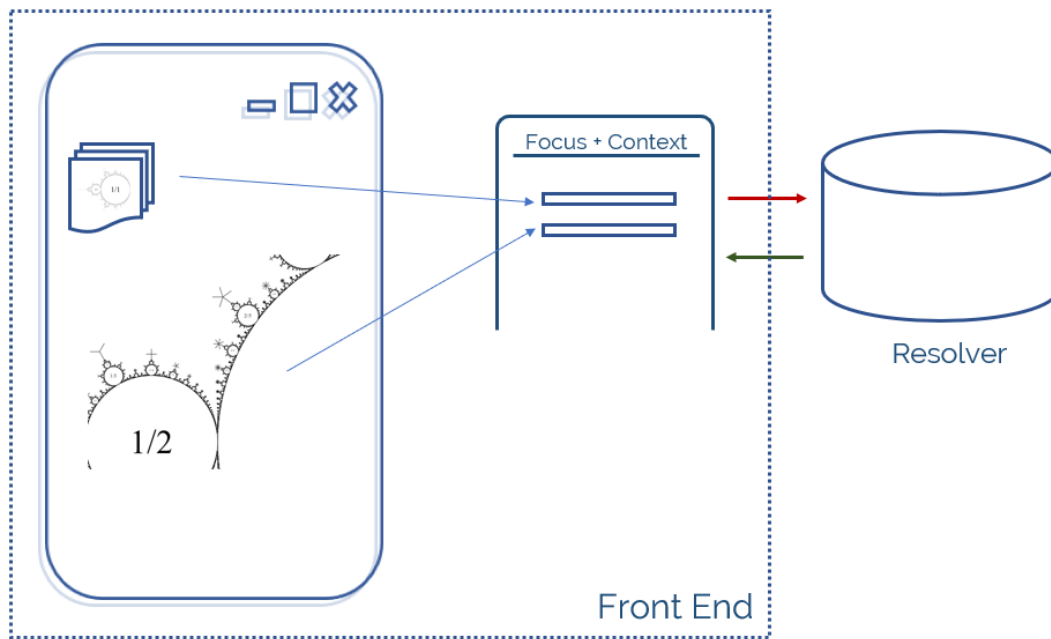


FIGURE 3.3: Model of a pair of **F + C**, a tiny manager that connects a focus view (the one in the center of the browser in this illustration) with a context view (the smaller view displaying dataset information), as well as sending pings to the outside.

And then a single instance of that model is not enough, let's look at the manager that manages all generated **F + C** models shown in **Figure 3.4 Manager of All The F+Cs**. We can see that this manager should manage the connections between each **F + C** model with their corresponding canvases, as well as controlling them when to send or receive requests to the outside world, in turn controls the message exchange between the entire front end with the outside world resolver.

Note that from a insider point of view, this manager controls all the message exchange for all the **F + C** models, however, to the outside world, this resolver is still handling all the different connections like treating multiple clients and don't have knowledge of the presence of this manager. Another thing that worth mentioning is that this manager should also handle the initializations of all the **F + C** models. Non-effect based **UI** events should also be handled here as some user interactions like zooming and panning require data exchange between the front end side and the outside world.

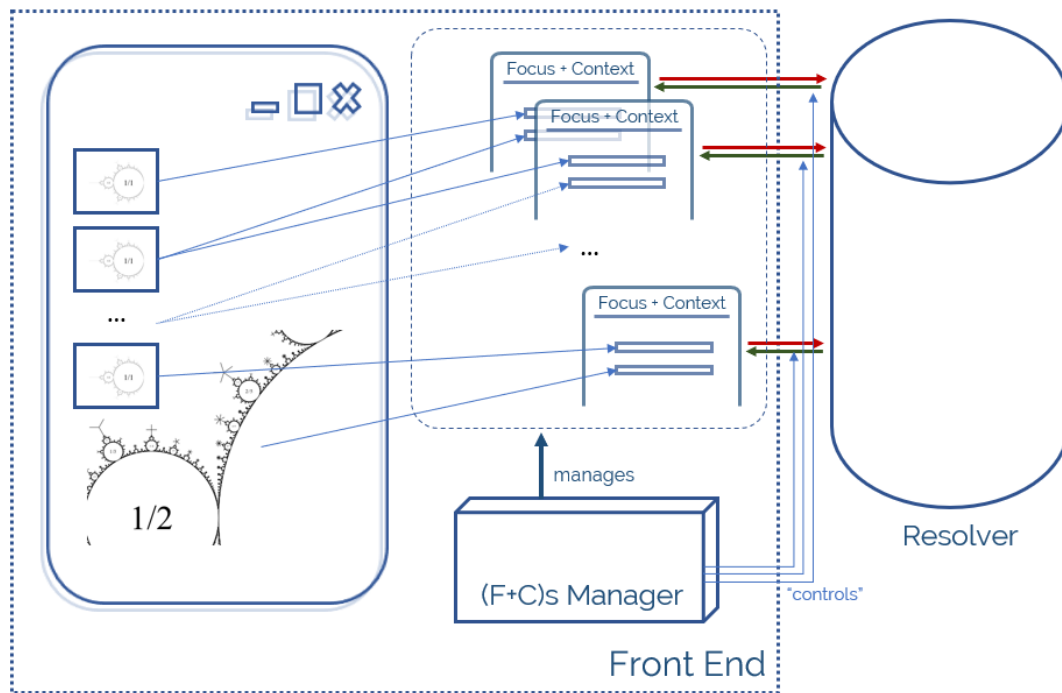


FIGURE 3.4: A manager that manages all the **F + C** models that are connected properly with corresponding views and outside world, in turn controls the data exchange between the front end **UI** with outside world.

Finally we need a manager that manages only the visual effects. All related affairs should be within the criteria of not requiring data exchange with the outside world, as shown in **Figure 3.5 Manager of Effects**. This manager will manage the arrangements of the **F + C** models but only on the front end, for example the activation / deactivation of effects, the positions and visual effects occurring during these processes, and user interactions with the control panel that triggers rearrangements of the on-screen canvases, and the mechanics of previews.

It is worth mentioning that the preview mechanics, since it being purely on front end, is managed by this effect manager. This manager detects this user interaction when trying to activate one of the previews, and then goes into its storage and find corresponding recorded dataset data, finally instruct corresponding **F + C** model to render them on the canvas that's for previewing purpose.

To summarize:

- Manages **F + C** models with respect to effects.
- Manages user interactions on **F + C** models with respect to their context canvases and preview mechanics.
- Manages user interactions on control panels.
- Manages everything else that stays in front end criteria.

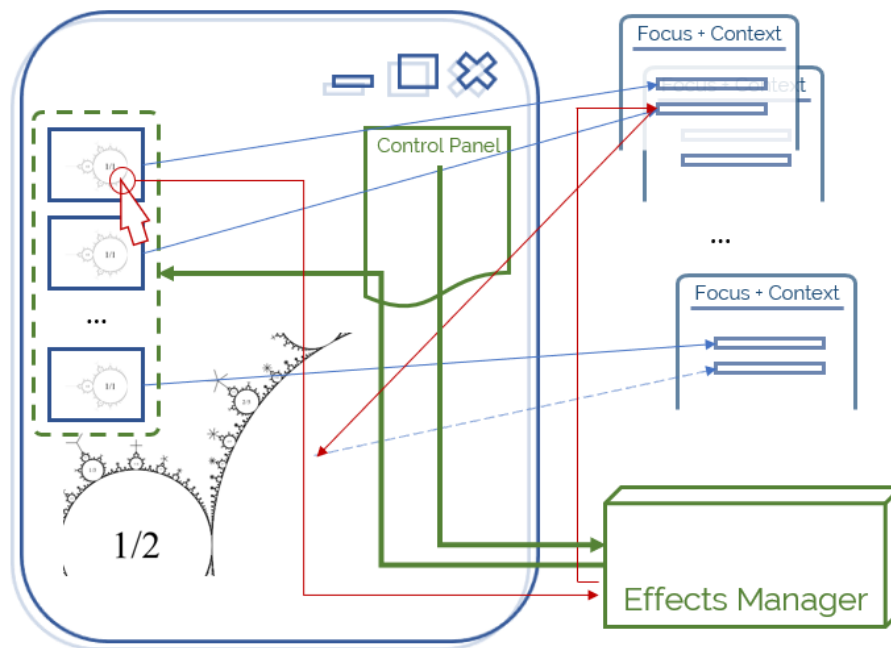


FIGURE 3.5: A manager that manages all effects related routines, including: (green) getting feedback of first hand user interactions from control panel and reflect back to the arrangements of the pile of **F + C** models; (red) some other user interactions such as mouse hovering event triggering the preview mechanics of putting preview image data onto the browser.

### 3.3 Back End Resolver

As for the back end resolver, it can be fairly simple or can be a complete cluster of servers or supercomputers to solve the response.

In the current state of the project, however, is a relatively simple script that compute for the required dataset with a given solution and location information, and send the results back to the front end, gracefully with the help of modern technology **HTML5** introduced in **2.2.3 HTML5**.

The structure of it can be narrowed down to three essential part:

- Kickstarting of the idle state.
- Reception of requests, necessary parsing of the requests and sending them to calculation instances.
- Responses during the calculation / fetching process of the extreme resolution dataset, and / or at the end of the calculation / fetching as final results.

## Chapter 4

# Implementation

In this chapter, the overall structure of the project, how the files are arranged and the functionalities of each components, will be described in details.

### 4.1 Files And Folders

The folder names and file names are mostly self-explanatory or conventional in this project. They'll be described briefly in this section.

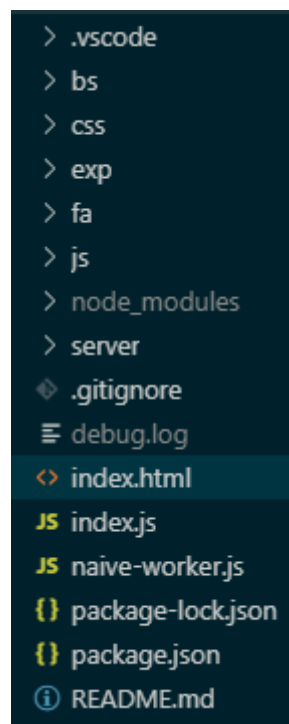


FIGURE 4.1: A glimpse of files and folders.

#### 4.1.1 Folders

**Folder** `./vscode` The configured Visual Studio Code workspace settings file. This file is included and stored inside the workspace and only apply when the workspace

is opened which overrides Visual Studio Code's default user settings. The author tweaked this file to make some parts of VS Code's editor, user interface, and functional behavior more fitting to review or to base future work upon this project<sup>1</sup>.

VS Code provides two different scopes for settings:

- User Settings - Settings that apply globally to any instance of VS Code you open.
- Workspace Settings - Settings stored inside your workspace and only apply when the workspace is opened.

Workspace settings override user settings[13].

**Folder ./js** All the third-party open source JS dependencies are stored in this folder. Sometimes third-party open source projects include a bundle of JavaScript (JS) and CSS files, here only the pure JS projects' files are included.

**Folder ./css** The CSS files of the projects are included. Firstly there is a ./css/common.css file, which sets the overall styles of the project, basically whatever the users can see at the very first glance when they open this project. Then there are several other CSS files, each sets a specific portion of the styles in this project. These files include:

- CSS File ./css/dock.css sets the iOS-Dock look-like styles, making the focused item larger with larger margins and adjacent items smaller and smaller margins with their corresponding nearby items.
- CSS File ./css/minibar.css sets the customized scrollbar styles that's being added upon the default styles of the dependency *MiniBar* which is used to create custom scrollbars.
- CSS File ./css/stacked.css sets the styles of the stacked cards effect.
- CSS File ./css/tabs.css sets the related styles of the tabs effect.

Note that most of the effects require not only the CSS stylings but also JS actions in order to work.

**Folder ./fa** Assets of the dependency *Font Awesome*, including all resources of the open source part. This dependency is used for the fonts of the icons in this project.

**Folder ./bs** Assets of the open source project *Bootstrap* by *Twitter*. This dependency is used for the stylings of the web elements inside the control panel, such as input boxes, dropdown menus and font styles in control panel. It also comes with some nice utilities for general web elements style setting.

---

<sup>1</sup> To learn more about this file, see <https://code.visualstudio.com/docs/getstarted/settings>.

**Folder** `./node_modules` Packages pulled from the **JS** dependency management tool *npm*<sup>2</sup> are stored in this folder. The required dependency here is the package `minibarjs` under this folder – in folder `./node_modules/minibarjs`. Conventionally this folder shouldn't be included or committed to the version control system<sup>3</sup>, because all the packages info are recorded in the file `package.json` and `package-lock.json` and if any dependencies are missing, running the *npm* command `npm install` should be able to pull all necessary dependencies into this folder, however, considering this project sometimes can be run in an environment without internet connection, this folder is included in the final static zipped package.

**Folder** `./exp` Some trivial *Python*, **JS** and **HTML** codes left from the prototypes of implementation at the beginning of this project. Some of them are using different algorithms and different scripts trying to achieve similar results to this project. They are not in use anymore and only kept for future references.

### 4.1.2 Top Level Files

**File** `index.html` This entry **HTML** file of this project. When a server is being run on the local machine, this is the first file getting executed. When a different implementation of the back end using techniques other than a web worker, for example a `WebSocket`, is developed and being adapted to this project, double-clicking on this file should also start this project.

**File** `index.js` The main **JS** script file of the project. This file gets included at the very end of the **HTML** file `index.html`.

**File** `naive-worker.js` The back end calculation **JS** script. The only job of this script is to receive information of the image the front end is asking for, and post the result message back to the front end. This piece of scripts not only post the complete results back, but also slices of results when the calculation takes longer than a certain amount of time and let the front end decide what to do with the partial results<sup>4</sup>.

**File** `package.json` A description file of the **JS** package management tool *npm*. This file can have many descriptions about what *npm* should do for this workspace<sup>5</sup> but here it most importantly specifies which packages to pull from the global repository, in the **JavaScript Object Notation (JSON)** field `'dependencies'`. Dependencies are specified in a simple object that maps a package name to a version range. The version range is a string which has one or more space-separated descriptors. Dependencies can also be identified with a tarball or git URL[17].

<sup>2</sup> Build amazing things — Essential JavaScript development tools that help you go to market faster and build powerful applications using modern open source code[16]. To know more about *npm*, see <https://www.npmjs.com/>.

<sup>3</sup> This is actually also what this project is following.

<sup>4</sup> In this project, what the front end will do after receiving partial results is that it will still render the slices of images onto the canvas and high light the painted partial image with green borders.

<sup>5</sup> For detailed information, see <https://docs.npmjs.com/files/package.json>.



**File** `package-lock.json` A generated file from *npm* package manager which locks the version of the dependencies of this specific workspace. Take the current project as an example, in file `package.json` there is this part in the **JSON** body:

```
{
  ..
  "dependencies": {
    ..
    "minibarjs": "^0.4.0",
    ..
  },
  ..
}
```

This piece of code only specified that the version of the package `minibarjs` that we require will match all `0.x.x` releases including `0.5.x`, but will hold off on `1.x.x`. This file `package-lock.json` will “lock” the version inside current workspace to a specific version with a hashed fingerprint of the files, in the current project with a version number of `0.4.0` and a hash fingerprint `sha512-iCUE/YVWn+0ht+NV2fLBS8bAVxED/916A5i1qJ20csCrc0tXHamgpWCo7uL+23HQ0UyFPvpw1izw2l3vzVKkXg==`.

**File** `README.md` A brief introduction file for the global version control system *GitHub*. Trivial.

**File** `.gitignore` Version control settings file, telling which files should not be committed to *Git* system. Not relevant to the project but the version control during the development phase of this project. Trivial.

## 4.2 Start the Project

Although this project is a pure web project, it cannot be started by simply double-clicking on the entry file `index.html`, because modern browsers usually don’t allow local scripts to directly start *Web Workers*<sup>6</sup> for security concerns. However, *Web Worker* is being used in this project as simple means for doing heavy calculations in background threads without interfering with the **UI**, therefore, in order to start the project, a simple **Hypertext Transfer Protocol (HTTP)** server must be up and running on the local machine.

It is also worth mentioning that this project should be running with Google Chrome browser as it supports most of the advanced visual effects and modern web technology syntax, known as *HTML5*<sup>7</sup>. The recommended version of Google Chrome is `76.0.x`.

To start and keep a **HTTP** server running, the simplest and recommended way would be to use *Python*’s `http.server`. To do that[14]:

<sup>6</sup> Web Workers are a simple means for web content to run scripts in background threads.[15]

<sup>7</sup> See <https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5> for more detailed information.

**Install Python** If you are using *Linux* or *macOS*, it should be available on the system already. If you are a *Windows* user, *Python* installer can be downloaded from the *Python* homepage and the instructions can be followed to install it:

- Go to [python.org](https://python.org)
- Under the *Download* section, click the link for Python 3.xxx.
- At the bottom of the page, choose the *Windows x86 executable installer* and download it.
- When it has downloaded, run it.
- On the first installer page, make sure you check the “Add Python 3.xxx to PATH” checkbox.
- Click *Install*, then click *Close* when the installation has finished.

**Verification** Open a *Command Prompt* (Windows) / *Terminal* (macOS / Linux). To check *Python* is installed, enter the following command:

```
python -V
```

**Navigation** The above command should return a version number. If this is OK, navigate to the directory that the files of this project is inside, using the *cd* command.

```
# include the directory name to enter it, for example  
cd Desktop/fractals
```

**Start the Server** Enter the command to start up the server in that directory:

```
# If Python version returned above is 3.X:  
python -m http.server
```

```
# Or simply:  
py -m http.server
```

```
# If Python version returned above is 3.X  
# and on non-Windows machines:  
python3 -m http.server
```

```
# If Python version returned above is 2.X,  
# or if on macOS using the default Python  
# installed:  
python -m SimpleHTTPServer
```

By default, the above actions will run the contents of the directory where the files of this project are located on a local web server, on port 8000. In order to view this project now, simply go to this server by going to the **Uniform Resource Locator (URL)** `localhost:8000` in your web browser, to be specific and recommended in Google Chrome. Here the project entry `index.html` will be run by default and users can see directly the result.

## 4.3 Front End

Since this project is a pure web project, the front end occupies a large portion of the codes.

### 4.3.1 HTML Entry `index.html`

The entry of the project is where this program gets started, in similar concept of the `main()` function in C or the public static void `main(String[] args)` function in Java. The entry point is a **HTML** file and as expected named `index.html`. It introduces the front end structure of the project in raw.

First part of the **HTML** file is the `<head>` part. In this part, the character set of this web page is defined as *UTF-8*, the size of the entire **HTML** document as fullscreen size, scaling not allowed and not shrinking to display its content.

```
<meta charset="utf-8">
<meta name="viewport" content="width=device-width,
    initial-scale=1, shrink-to-fit=no">
```

And then all the needed **CSS** files are included to end the `<head>` part. Besides the **CSS** files which will be described in **subsection 4.3.3 CSSs for Overview Effects**, the necessary **CSS** files from third-party open source vendors are also included, including *Bootstrap's CSS* part, *FontAwesome* and *MiniBar CSS* assets.

The `<body>` part is the essential part of the **HTML** entry, which describes the structure of what users can “actually see”. It begins first with three `<div>` tags for the most important three parts of this project, the container for main background canvases, the container for mini-maps, and the container for the control panel floating on the top right corner of the **UI** screen. The positioning, sizes and container behaviours of these `<div>`s are defined in the **CSS** files which are already included. Before users set any effects up, these properties mostly come from the file `./css/common.css`.

After the visual `<div>` part, several `<script>` tags come after it to include what's necessary for the essential coding part. Here firstly are the dependencies of the project, including *jQuery*, *Bootstrap's JS* part, and *MiniBar's JS* part. And then at the very end the main **JS** file `index.js` is included and all the core programs of this project goes in there.

Worth noting that conventionally all **JS** files should be included at the very end of the page as what we are doing now, unless the **JS** file is needed before the render phase of the web page. This way if the **JS** file is a little bit bigger than usual, the loading of the **JS** files won't affect the rendering process of the **DOM** documents.

### 4.3.2 Main JavaScript `index.js`

The main **JS** file `index.js` is where the core codes are. In this file there is firstly the definition of required classes from bottom level to the top, then the instantiation of them and putting the front end **HTML** elements into action to display the overall results.

There in total four classes defined.

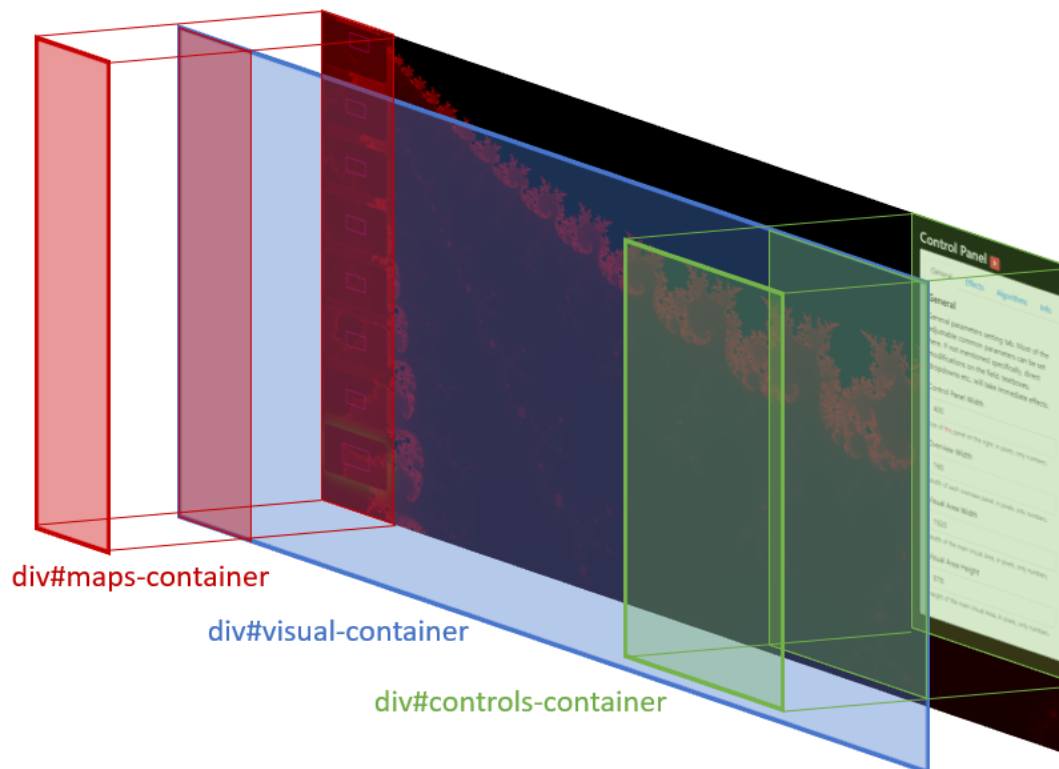


FIGURE 4.2: Document Object Model (DOM) structure in `<body>` tag.

#### 4.3.2.1 Class MandelWorker

The class `MandelWorker` is in charge of sending a message to the back end and when a result is sent back, doing some other actions.

When instantiated, an instance of a native *Web Worker* will also be created as a private property of this class. `MandelWorker` instantiates the *Web Worker* by the script `naiveworker.js`, which means that the script `naiveworker.js` will be the core of the worker and this worker will be doing whatever in that script when it is asked to<sup>8</sup>.

**Function** `work(params...)`

The function `work(params...)` is the interface between `MandelWorker` and the outside invoker. To get an image from the source, one must invoke this function with the needed parameters as follows:

- `magnif` The magnification level of the result image to be expected from the *Web Worker*.
- `centerX` The *x* component of the center coordinates on the mathematical plane of the result image to be expected from the worker.
- `centerY` The *y* component of this coordinates.
- `width` The width in pixels of the result image.

<sup>8</sup> See [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Workers\\_API/Using\\_web\\_workers#Spawning\\_a\\_dedicated\\_worker](https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers#Spawning_a_dedicated_worker) for more detailed information of the process of instantiating a *Web Worker*.

- `height` The height in pixels of the result image.
- `callback` The function to execute when a result message is received.
- `callbackThis` The “this” context where the `callback` function should be executed under.

Here what’s worth mentioning is the parameter `magnif`. The magnification level is a number representing the number of pixels that together has a length of 1 on the mathematical axis. As shown in **Figure 4.3 Magnification Level** is an image with the magnification level of 2, since 2 pixels have the length of 1 on the mathematical axis.

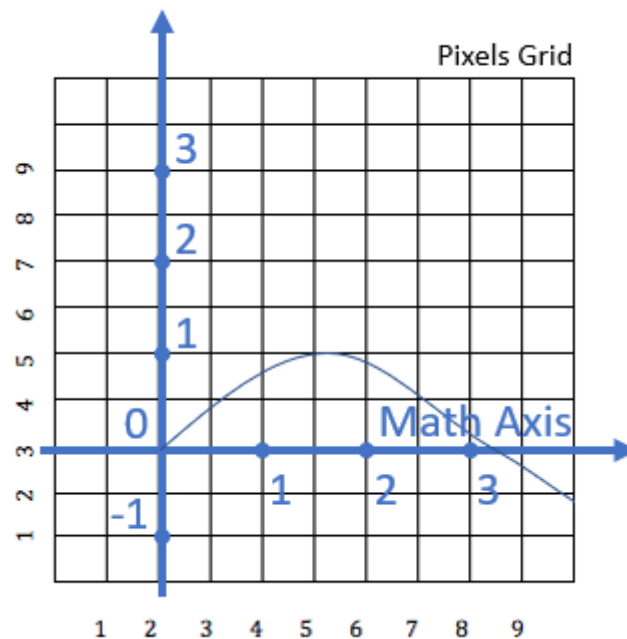


FIGURE 4.3: Magnification level in aspect of mathematical axis.

Once this function is invoked, `MandelWorker` will tell its own *Web Worker* to start working on datasets fetching<sup>9</sup>, and if any sorts of results come through that worker, hit the `workerResponse(e)` function of the current `MandelWorker`.

#### Function `workerResponse(e)`

The function `workerResponse(e)` will be called when a response from the *Web Worker* is sent back. It basically does one thing: checking if the parameters of `callback` and `callbackThis` were set when function `work(params...)` got invoked in the first place. If they were set to any function, call it under the context of the parameter `callbackThis`.

#### Function `destroy()`

The function `destroy()` as the name implies is the method to destroy and release the resources for current `MandelWorker`. It terminates the *Web Worker*, sets the response

<sup>9</sup> In the context of the current project, is actually image generation.

method to null so no responses will be dealt furthermore and sets all other references to null as well so the internal JS engine can garbage collect<sup>10</sup> all these instance to avoid memory leakage when the calculation gets heavy.

#### 4.3.2.2 Class MapVisualPair

An abstract concept of pairing a minimap<sup>11</sup> with an active focus region with higher resolution, as Figure 4.4 Map Visual Pair is an example of what they actually are respectively.

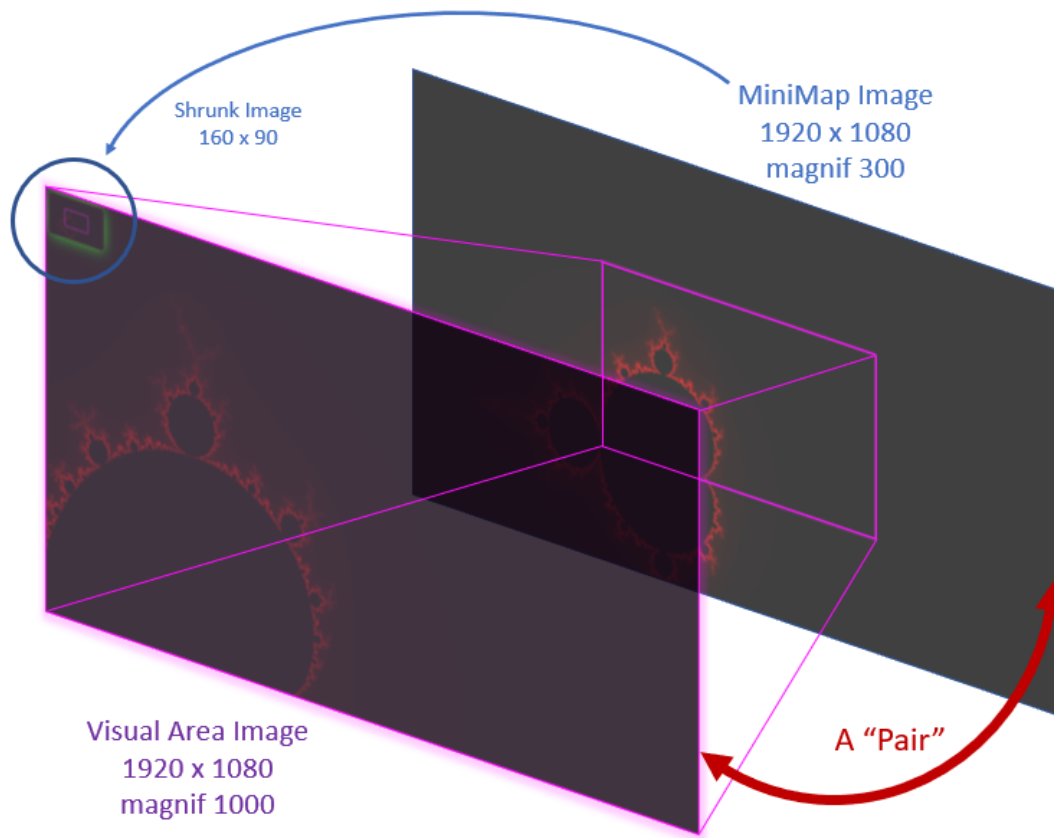


FIGURE 4.4: Map area image and visual area(current focus region) image.

The minimap representing a relatively “larger” area in the dataset and the visual area, which represents the active observing region, actually have both same resolution so they can both be displayed on a Full HD screen. Although the visual area seems “smaller” in comparison with the Map area, since it’s an active region that is being observed by the user, the size of this visual area only represents its dimensions on the mathematical plane and not smaller pixels-wise.

<sup>10</sup> In computer science, garbage collection (GC) is a form of automatic memory management. The garbage collector, or just collector, attempts to reclaim garbage, or memory occupied by objects that are no longer in use by the program[21].

<sup>11</sup> Same concept in current project as an *overview*.

The **Map** image, however, will normally be shrunk and put on the left side of the screen, till the user hovers over a specific **Map** that will trigger the preview process of this program so he can have a glance of this image in a **Full HD** way.

The reason to pair up a visual area with a **Map** area is that whenever the user wants to browse around the dataset, the focusing coordinates on the visual area should also reflect back to the **Map** area, and the rectangle representing the current observing area should also get updated automatically. When there are more hierarchies of **Maps** present, this pairing mechanism becomes extremely helpful because the changes on the active region will flow all the way up to all levels of **Maps** necessary. Here **Figure 4.5 Pairing Multiple Levels of Maps** is a simple illustration of it.

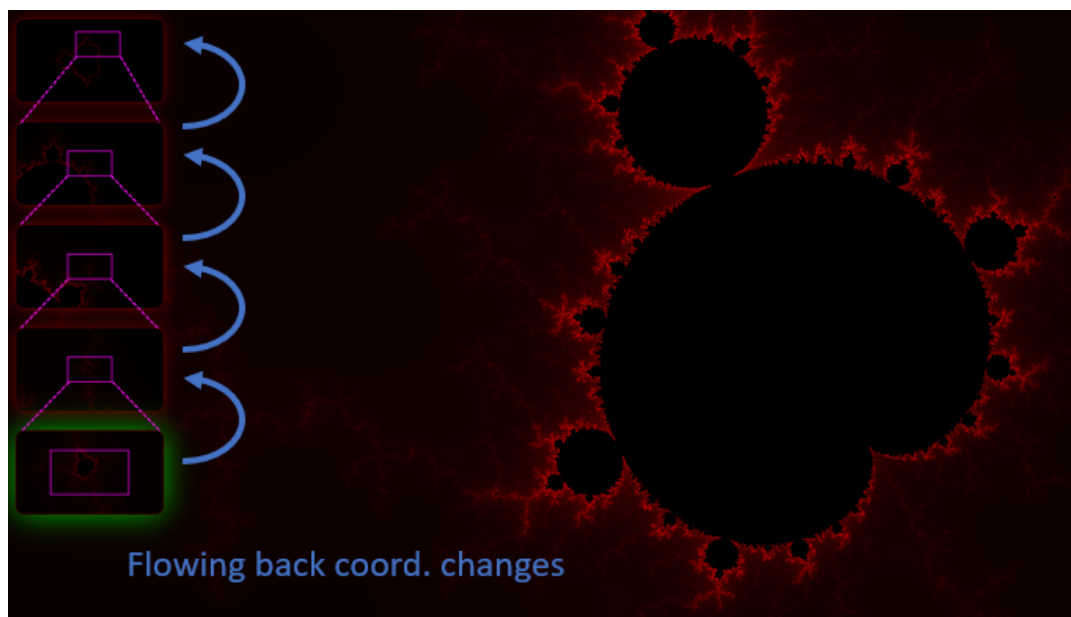


FIGURE 4.5: When multiple hierarchies of **Maps** present, visual area changes, all **Maps** changes.

**Function** `init(mapCanvas, previewCanvas, visCanvas = null)`

This function will initialize the current `MapVisualPair`, bounding a `mapCanvas`, a `previewCanvas` and a `visCanvas` all of type `<canvas>` element to this class. When internally a drawing action should be performed, related images will be drawn on corresponding canvas.

The canvas `mapCanvas` will be used to draw the shrunk version of the **Map** image. `previewCanvas` will be used to draw the normal(**Full HD**) version of the **Map** image. And `visCanvas` will be used to draw the visual area image in **Full HD** if set.

After bounding the canvases, this function will also check if any event handlers are bound to the events `mouseover` and `mouseout` to the `mapCanvas` element, and record corresponding info to the element to avoid attempts to rebound event handlers to the same `<canvas>` element. In this way, only one event handler for `mouseover` and `mouseout` will be triggered when the user hovers their mouse on the `<canvas>` and when the user put their cursor out of the element.

The handlers this function is going to bound to the `mapCanvas` element are going to first add `CSS` class `.nearby` to adjacent siblings of this canvas, and check if any additional callbacks this class should call. The callbacks, if any, which are set in the private properties `mouseoverCallback` and `mouseoutCallback`, will be called under set context `mouseoverCallbackThis` and `mouseoutCallbackThis` when user performs corresponding actions. As mentioned before, they will also be called only once because the bounding information is recorded. In the project, these callbacks and the contexts of them are set by a manager class [4.3.2.4 Class EffectManager](#).

**Function** `destroy()`

This function as the name implies releases all in-use resources, including `Workers`, unbinding bound event handlers, clearing references to the canvases and their 2d contexts.

**Function** `drawMapHoverArea(offsetRealX = 0, offsetRealY = 0)`

When the class is told to draw images on corresponding canvas elements, it will not automatically draw the purple rectangle indicating the current observing visual area, since the cached image data does not include this rectangle — it doesn't belong to the extreme resolution dataset itself, therefore, this function exists to draw this current focus area using a purple rectangle to indicate it, drawing this rectangle on the `mapCanvas` whenever invoked. Note that whenever the image from the calculation side is fetched, without invoking this function, no current observing area rectangle will be shown, since the newly fetched image data will cover the old one also the old drawn rectangle.

The parameters `offsetRealX` and `offsetRealY` can also be set, as the purple rectangle to be drawn will then have an offset of `(offsetRealX, offsetRealY)` with respect to the center of the current observing area on the mathematical complex plane.

**Function** `drawPreviewHoverArea(offsetRealX = 0, offsetRealY = 0)`

Like the function `drawMapHoverArea`, this function will also draw a rectangle representing the current observing visual area, but on another canvas `previewCanvas`. The reason to separate these two functions is that the canvas `previewCanvas` isn't always visible and if these two functions are combined, it'll draw unintended purple rectangles on wrong canvases.

**Function** `moveTo(x = null, y = null)`

This function is used to move the current `MapVisualPair` around. The parameters of coordinates are on the mathematical complex plane, i.e. with respect to the entire datasets.

Note that these two parameters can be omitted and when omitted, the current `MapVisualPair` will simply send a ping to the calculation side and grab new image data for current observing coordinates, like a "refresh" action.

**Properties** `visMagnif` and `mapMagnif`

The magnification for the pair of these two canvases. See [Figure 4.3 Magnification Level](#) for the explanation of magnification level.

**Properties** `visCenterX` and `visCenterY`

The coordinates of the center point of the current observing visual area, with respect to the mathematical complex plane.



**Properties** visCanvasWidth and visCanvasHeight

The width and height in **pixels** of the image data of the whole visual area. In **Figure 4.4 Map Visual Pair**, they're the dimensions of the whole visual area image. These properties have to exist because the dimensions of this area cannot always be fetched from the visCanvas property, as it is an optional parameter during the init(params...) phase and sometimes is absent.

**Properties** visImgOffsetX and visImgOffsetY

These two properties are set for the dragging actions that are realized in **4.3.2.3 Class MinimapManager**. They represents the current dragging offset with respect to the top left corner of the visCanvas. The reference point being top left corner not the center point is because in web canvas drawing system, the (0, 0) point is the top left corner, unlike in mathematical complex plane it being the center.

**Properties** mapCenterX and mapCenterY

Like the properties visCenterX and visCenterY, these two properties represents the coordinates of the center point of the current **Map** area, with respect to the mathematical complex plane. In **Figure 4.4 Map Visual Pair**, they're the center coordinates of the **Map** image.

#### 4.3.2.3 Class MinimapManager

The manager class of all the minimaps, controlling all their behaviours on the top level.

##### Inits

```
initMaps(visCanvas, previewCanvas, mapsContainer, visualContainer, hoverX=0,
hoverY=0)
```

```
initPairMainDrag()
```

##### Mouse Down(drag to browse around)

```
pairMainMouseDown(e)
```

```
pairMainMouseMove(e)
```

```
pairMainMouseUp(e)
```

```
pairMainStepDrag(timestamp)
```

##### Mouse Wheel(zoom in and out)

```
initPairMainWheel()
```

```
pairMainWheel(e)
```

```
pairMainTimeoutWheel()
```

```
pairMainStepWheel()
```

#### 4.3.2.4 Class EffectManager

This class manages the behaviours, activation and deactivation of the overview effects and the preview effects of the overviews.

General functions: `init()`, `getInfo(el)`

Fade in / out related functions: `fadeMouseOver(e, currentPair)`, `fadeMouseOut(e, currentPair)`

Zoom in / out through related functions: `zoomMouseOver(e, currentPair)`, `zoomStep(timestamp)`, `zoomMouseOut(e, currentPair)`

General preview effects functions: `updatePreview()`, `updateFadePreview()`, `updateZoomPreview()`, `destroyPreview()`

General overview effects functions: `destroy()`, `update()`

Scrollbar + Dock effects specific functions: `initScrollbar()`, `destroyScrollbar()`, `updateScrollbar()`

Stacked Cards effects specific functions: `initStacked()`, `destroyStacked()`, `updateStacked()`

Tabs effects specific functions: `initTabs()`, `destroyTabs()`, `updateTabs()`

#### 4.3.2.5 Instantiation, Variables and the Rest

- `screenWidth`
- `screenHeight`
- `mapWidth`
- `mapHeight`
- `controlPanelWidth`
- `visMagnif`
- `mapMagnif`
- `hoverX`
- `hoverY`
- `mainCanvas`
- `previewCanvas`
- `$('#visual-container')`
- `$('#maps-container')`

### 4.3.3 CSSs for Overview Effects

Folder `./css` includes five **CSS** files, each setting up some visual effects of the project.

File `./css/common.css` first sets up all general appearance of the elements on the web page when no parameters or effects are set. File `./css/dock.css` sets up the appearance when *Scrollbar + Dock* is activated, only the iOS Dock part and file

`./css/minibar.css` sets up the scroll bar part. File `./css/stacked.css` sets up the effects of stacked cards. File `./css/tabs.css` sets up the effects of the tab selection on the top.

#### 4.3.4 Scrollbar + Dock Effect

#### 4.3.5 Stacked Cards Effect

#### 4.3.6 Tabs Effect

### 4.4 Back End Calculation

The back end calculation is done in the JS file `naive-worker.js`. This file is being used for initializing the *WebWorkers* inside `index.js` dynamically. Whenever a calculation or extraction for a specific region of a dataset is needed, the main JS file `index.js` is going to send a message to `naive-worker.js` with desired parameters and this back end will respond with corresponding image data.

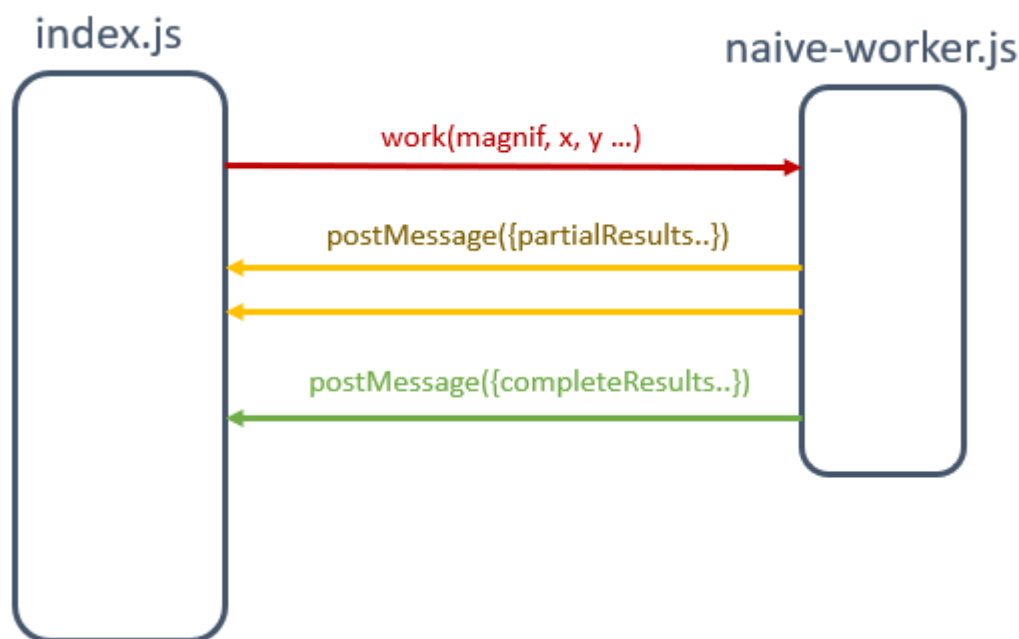


FIGURE 4.6: Message exchange between `index.js` and `naive-worker.js`.

#### 4.4.1 Global Scope

In the global scope of this file, the following things were done.

**Includes** The `decimal.js` dependency is included for high-precision floating points calculation. Default parameters for the dependency is set.

**Constants** Constants of default screen width and default screen height are defined in case the front end doesn't give these parameters.

**Canvas** An `OffscreenCanvas` instance is created and instantiated with the dimensions of by default the values of the defined constants. The `OffscreenCanvas` will be used as the canvas to generate the desired image on, and since it's not being shown on the screen, will occupy less system resources and boost the calculation speed. Corresponding variables is declared after the instantiation, respectively `canvas` for the `OffscreenCanvas` itself and `ctx` as the 2d context of the canvas.

#### 4.4.2 Message Reception

See [Figure 4.6 Message Exchange](#).

#### 4.4.3 Iteration Limit

#### 4.4.4 Iteration Count for One Point

#### 4.4.5 Image Generation

#### 4.4.6 High Precision Version

### 4.5 Utility Assets

Other open source third-party utilities lie in different folders with corresponding names.

#### 4.5.1 Folder `./js`

In `./js` folder, all **JS** third-party files are here, including:

- File `decimal.min.js` is for high-precision floating points calculation for **JavaScript (JS)**.
- File `jquery-3.4.1.min.js` is for **DOM** traversal and manipulation, event handling and animation.
- File `bootstrap.bundle.min.js` is for some basic styling of the control panel sitting on top right corner of the screen.

#### 4.5.2 Folder `./fa`

#### 4.5.3 Folder `./bs`

#### 4.5.4 Folder `./css`

## **Chapter 5**

# **Results and Comparison**

### **5.1 General Results**

### **5.2 Comparison Between Different Arrangement Methods**

### **5.3 Future Work**

## Appendix A

# Frequently Asked Questions

### A.1 Where can I find the source code of this project?

This source code of the project is hosted on *GitHub* as a private repository on <https://github.com/divyinfo/fractals.git>. You could ask the author at [danyun.lei@stud.uni-due.de](mailto:danyun.lei@stud.uni-due.de) for further assistance such as asking for a view or collaboration access.

### A.2 Is there $\text{\LaTeX}$ source code for this thesis?

Yes, it is hosted on *GitHub* as a private repository on <https://github.com/divyinfo/fractals-report.git>. You could contact the author at [danyun.lei@stud.uni-due.de](mailto:danyun.lei@stud.uni-due.de) if further assistance is needed.

# Bibliography

- [1] Patrick Baudisch, Nathaniel Good, and Paul Stewart. "Focus plus context screens: combining display technology with visualization techniques". In: *Proceedings of the 14th annual ACM symposium on User interface software and technology*. ACM. 2001, pp. 31–40.
- [2] Patrick Baudisch et al. "Keeping things in context: a comparative evaluation of focus plus context screens, overviews, and zooming". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2002, pp. 259–266.
- [3] James H Clark. "Hierarchical geometric models for visible surface algorithms". In: *Communications of the ACM* 19.10 (1976), pp. 547–554.
- [4] Andy Cockburn, AMY Karlson, and Benjamin B Bederson. "A review of focus and context interfaces". In: *HCIL Tech Report 2006-09* (2006).
- [5] Andy Cockburn, Amy Karlson, and Benjamin B Bederson. "A review of overview+ detail, zooming, and focus+ context interfaces". In: *ACM Computing Surveys (CSUR)* 41.1 (2009), p. 2.
- [6] Franklin C Crow. "A more flexible image generation environment". In: *ACM SIGGRAPH Computer Graphics* 16.3 (1982), pp. 9–18.
- [7] David Flanagan. *JavaScript: the definitive guide*. 6 ed. "JavaScript is part of the triad of technologies that all Web developers must learn: HTML to specify the content of web pages, CSS to specify the presentation of web pages, and JavaScript to specify the behaviour of web pages." O'Reilly Media, Inc., 2006, p. 1.
- [8] Fredrik Gundelsweiler, Thomas Memmel, and Harald Reiterer. "ZEUS—zoomable explorative user interface for searching and object presentation". In: *Symposium on Human Interface and the Management of Information*. Springer. 2007, pp. 288–297.
- [9] Carl Gutwin and Chris Fedak. "Interacting with big interfaces on small screens: a comparison of fisheye, zoom, and panning techniques". In: *Proceedings of Graphics Interface 2004*. Canadian Human-Computer Communications Society. 2004, pp. 145–152.
- [10] Lars Erik Holmquist. "The Zoom Browser: Showing Simultaneous Detail and Overview in Large Documents". In: *Human IT: Journal for Information Technology Studies as a Human Science* 2.3 (1998).
- [11] Kasper Hornbæk, Benjamin B Bederson, and Catherine Plaisant. "Navigation patterns and usability of zoomable user interfaces with and without an overview". In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 9.4 (2002), pp. 362–389.
- [12] Kasper Hornbæk and Morten Hertzum. "The notion of overview in information visualization". In: *International Journal of Human-Computer Studies* 69.7-8 (2011), pp. 509–525.
- [13] Microsoft. *User and Workspace Settings*. [Online; accessed 14-September-2019]. 2018. URL: <https://code.visualstudio.com/docs/getstarted/settings>.

- [14] Mozilla and individual contributors. *How do you set up a local testing server? - Learn web development* | MDN. [Online; accessed 14-September-2019]. 2019. URL: [https://developer.mozilla.org/en-US/docs/Learn/Common\\_questions/set\\_up\\_a\\_local\\_testing\\_server](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server).
- [15] Mozilla and individual contributors. *Using Web Workers - Web APIs* | MDN. [Online; accessed 14-September-2019]. 2019. URL: [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Workers\\_API/Using\\_web\\_workers](https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers).
- [16] npm, Inc. *npm* | *build amazing things*. [Online; accessed 14-September-2019]. 2019. URL: <https://www.npmjs.com/>.
- [17] npm, Inc. *npm-package.json* | *npm Documentation*. [Online; accessed 14-September-2019]. 2019. URL: <https://docs.npmjs.com/files/package.json#dependencies>.
- [18] Virpi Roto et al. "Minimap: a web page visualization method for mobile phones". In: *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM. 2006, pp. 35–44.
- [19] w3techs.com. *Usage Statistics of JavaScript for Websites, March 2018*. [Online; accessed 15-September-2019]. 2018. URL: <https://w3techs.com/technologies/details/cp-javascript/all/all>.
- [20] Wikipedia. *Cascading Style Sheets - Wikipedia*. [Online; accessed 15-September-2019]. 2019. URL: [https://en.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](https://en.wikipedia.org/wiki/Cascading_Style_Sheets).
- [21] Wikipedia. *Garbage collection (computer science) - Wikipedia*. [Online; accessed 14-September-2019]. 2019. URL: [https://en.wikipedia.org/wiki/Garbage\\_collection\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)).
- [22] Wikipedia. *HTML - Wikipedia*. [Online; accessed 15-September-2019]. 2019. URL: <https://en.wikipedia.org/wiki/HTML>.
- [23] Wikipedia. *JavaScript - Wikipedia*. [Online; accessed 15-September-2019]. 2019. URL: <https://en.wikipedia.org/wiki/JavaScript>.
- [24] Wikipedia. *Mandelbrot set - Wikipedia*. [Online; accessed 14-September-2019]. 2019. URL: [https://en.wikipedia.org/wiki/Mandelbrot\\_set](https://en.wikipedia.org/wiki/Mandelbrot_set).
- [25] Wikipedia. *Mini-map - Wikipedia*. [Online; accessed 14-September-2019]. 2019. URL: <https://en.wikipedia.org/wiki/Mini-map>.
- [26] Pan Zhigeng, Ma Xiaohu, and Shi Jiaoying. "Overview of Multiple Level of Detail Creation [J]". In: *Journal of Image and Graphics* 9 (1998).