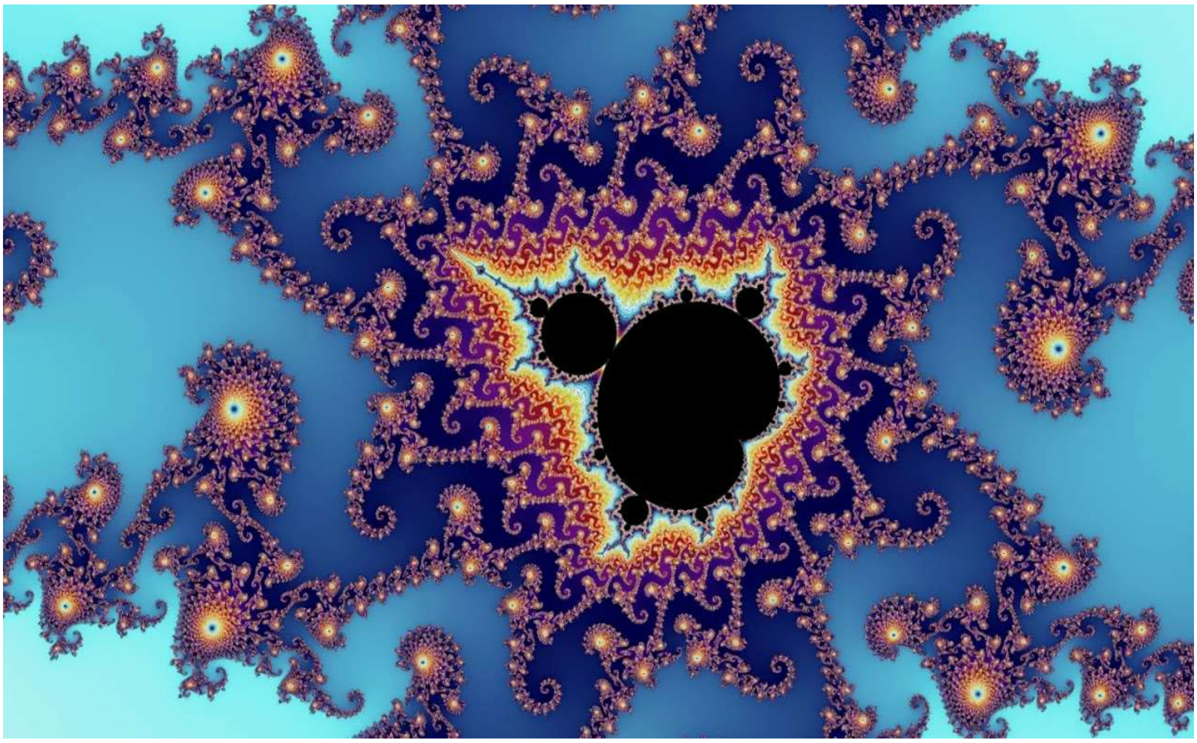
MANDELBROT SET VISUALIZATION



DONE BY

(1) DAVE PAUL JOSEPH

TVE21EC023

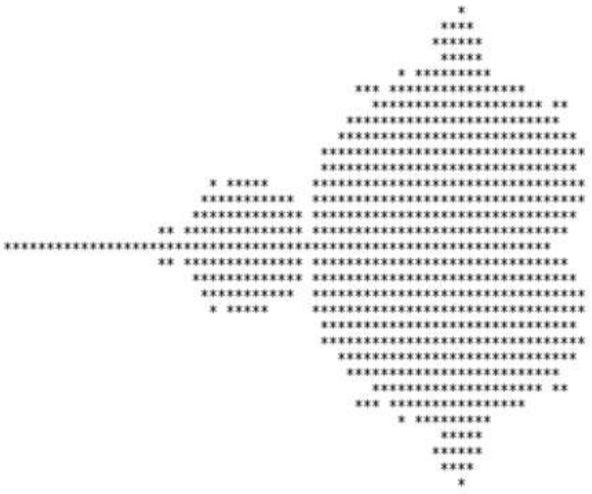
Roll no: 19

(2) DEENA MARIYA DAVID

TVE21EC024

Roll no:20

INTRODUCTION



The term **Mandelbrot** set is used to refer both to a general class of fractal sets and to a particular instance of such a set. In general, a Mandelbrot set marks the set of points in the complex plane such that the corresponding Julia set is connected and not computable.

The Mandelbrot set is the set obtained from the quadratic recurrence equation,

$$Z_{n+1}=Z_n^{2} +C$$

with Z_0=C, where points in the complex plane for which the orbit of Z_n does not tend to infinity are in the set. Setting Z_0 equal to any point *in the set* that is not a periodic point gives the same result. The Mandelbrot set was originally called a molecule mu by Mandelbrot. J. Hubbard and A. Douady proved that the Mandelbrot set is connected.

This set was first defined and drawn by Robert W. Brooks and Peter Matelski in 1978, as part of a study of Kleinian groups. Afterwards, in 1980, Benoit Mandelbrot obtained high quality visualizations of the set while working at IBM's Thomas J. Watson Research Center in Yorktown Heights, New York.The Mandelbrot set is a compact set, since it is closed and contained in the closed disk of radius 2 around the origin.

The Mandelbrot set is generated by iteration, which means to repeat a process over and over again. In mathematics this process is most often the application of a mathematical function. For the Mandelbrot set, the functions involved are some of the simplest imaginable: they all are what is called quadratic polynomials and have the form f(x) = x2 + c, where *c* is a constant number. As we go along, we will specify exactly what value *c* takes.

To iterate *x2 + c*, we begin with a *seed* for the iteration. This is a number which we write as *x0*. Applying the function x2 + c to *x0* yields the new number

$x1 = x0^2 + c.$

Now, we iterate using the result of the previous computation as the input for the next. That is

$x2 = x1^2 + c$

$x3 = x2^2 + c$

$x4 = x3^2 + c$

$x5 = x4^2 + c$

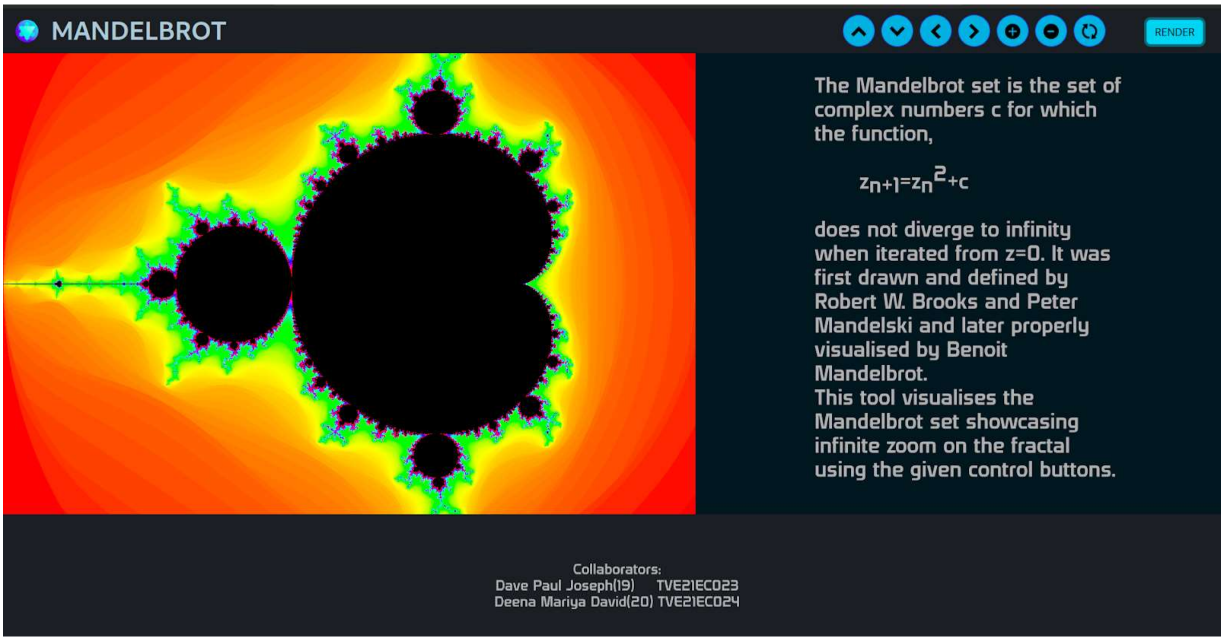
and so forth. The list of numbers x0, x1, x2,... generated by this iteration has a name: it is called the *orbit* of x0 under iteration of x2 + c.

The Mandelbrot set is known for its intricate, self-similar structure and has been studied extensively in the field of mathematics. It has also been used in a variety of applications, including computer graphics and data visualization.

METHODOLOGY

The Mandelbrot function is a mathematical function that is used to generate fractal images.. The set is visualized using a color-coded plot, with the points inside the set appearing black and the points outside the set appearing colored. The implementation of this visualizer is divided in two major part:

The frontend



The Mandelbrot set is the set of complex numbers c for which the function,

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

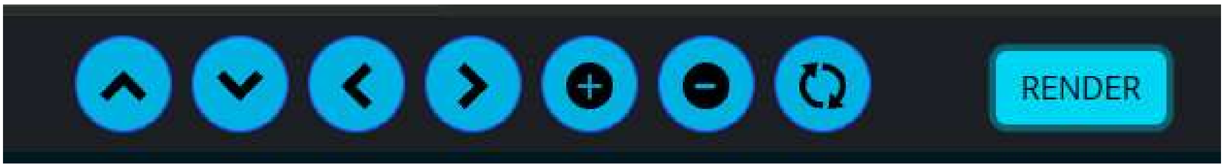
does not diverge to infinity when iterated from z=0. It was first drawn and defined by Robert W. Brooks and Peter Mandelski and later properly visualised by Benoit Mandelbrot.
This tool visualises the Mandelbrot set showcasing infinite zoom on the fractal using the given control buttons.

Collaborators:
Dave Paul Joseph(19) TVE21EC023
Deena Mariya David(20) TVE21EC024

The project has a user interface in the form of a web-app. The web-app has been coded using ReactJs and styled using CSS.

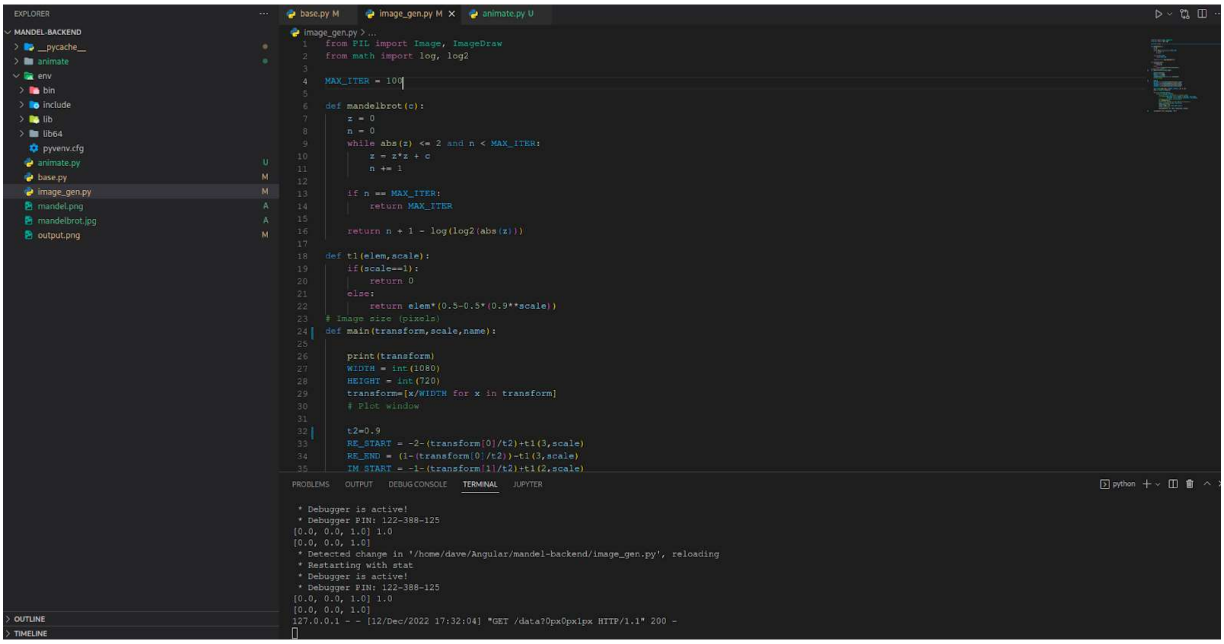
ReactJS is a powerful JavaScript library for building user interfaces. It is designed to be used on the frontend, which is the part of an application that the user interacts with. ReactJS is known for its efficient rendering and its ability to handle complex and dynamic user interfaces.

The UI is provided with buttons for vertical and horizontal transforms, zoom-in and zoom-out and to finally render the output to the screen. These buttons update local variables that are parsed into a suitable form and act as arguments in the API call. There is a reset button to set the local variables to their initial state.



GET and POST are used for communication with the API. The API returns an image which is then displayed on screen. The UI also has a brief introduction to the Mandelbrot set and links to the appropriate data pages for relevant information.

The Backend



```python
from PIL import Image, ImageDraw
from math import log, log2

MAX_ITER = 100

def mandelbrot(c):
    z = 0
    n = 0
    while abs(z) <= 2 and n < MAX_ITER:
        z = z*z + c
        n += 1

    if n == MAX_ITER:
        return MAX_ITER

    return n + 1 - log(log2(abs(z)))

def t1(elem,scale):
    if(scale==1):
        return 0
    else:
        return elem*(0.5-0.5*(0.9**scale))
# Image size (pixels)
def main(transform,scale,name):

    print(transform)
    WIDTH = int(1080)
    HEIGHT = int(720)
    transform=[x/WIDTH for x in transform]
    # Plot window

    t2=0.9
    RE_START = -2-(transform[0]/t2)+t1(3,scale)
    RE_END = (1-(transform[0]/t2))-t1(3,scale)
    IM_START = -1-(transform[1]/t2)+t1(2,scale)
```

```
 * Debugger is active!
 * Debugger PIN: 122-388-125
[0.0, 0.0, 1.0] 1.0
[0.0, 0.0, 1.0]
 * Detected change in '/home/dave/Angular/mandel-backend/image_gen.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 122-388-125
[0.0, 0.0, 1.0] 1.0
[0.0, 0.0, 1.0]
127.0.0.1 - - [12/Dec/2022 17:32:04] "GET /data?0px0px1px HTTP/1.1" 200 -
```

A Flask backend is a type of web application framework that is built using the Flask microframework for Python. Flask is a lightweight framework that is designed to be simple and flexible, making it a popular choice for creating web applications. A Flask backend is typically used to handle the server-side logic of a web application, such as managing requests and responses, accessing a database, or rendering templates. Flask makes it easy to create a web application by providing a web server and tools for routing, handling requests and responses, and working with templates. Flask is known for its simplicity and ease of use, making it a popular choice for building web applications.Flask API acts as a backend in this project.

The code is mainly divided into three files.

- base.py

  This file contains the code for receiving and dealing with API calls. The GET request is parsed into the appropriate parameters. It then calls the main function in image_gen.py to render the image providing it with the required values. The image is then stored in a local temporary file and sent as a POST request to the front end.

- image_gen.py

  This file contains the code that deals with the generation of the image. Its main function has the input of the transform, scaling and the name of the temporary file. It also has variables to control the steps up to which the convergence is checked.

- animate.py

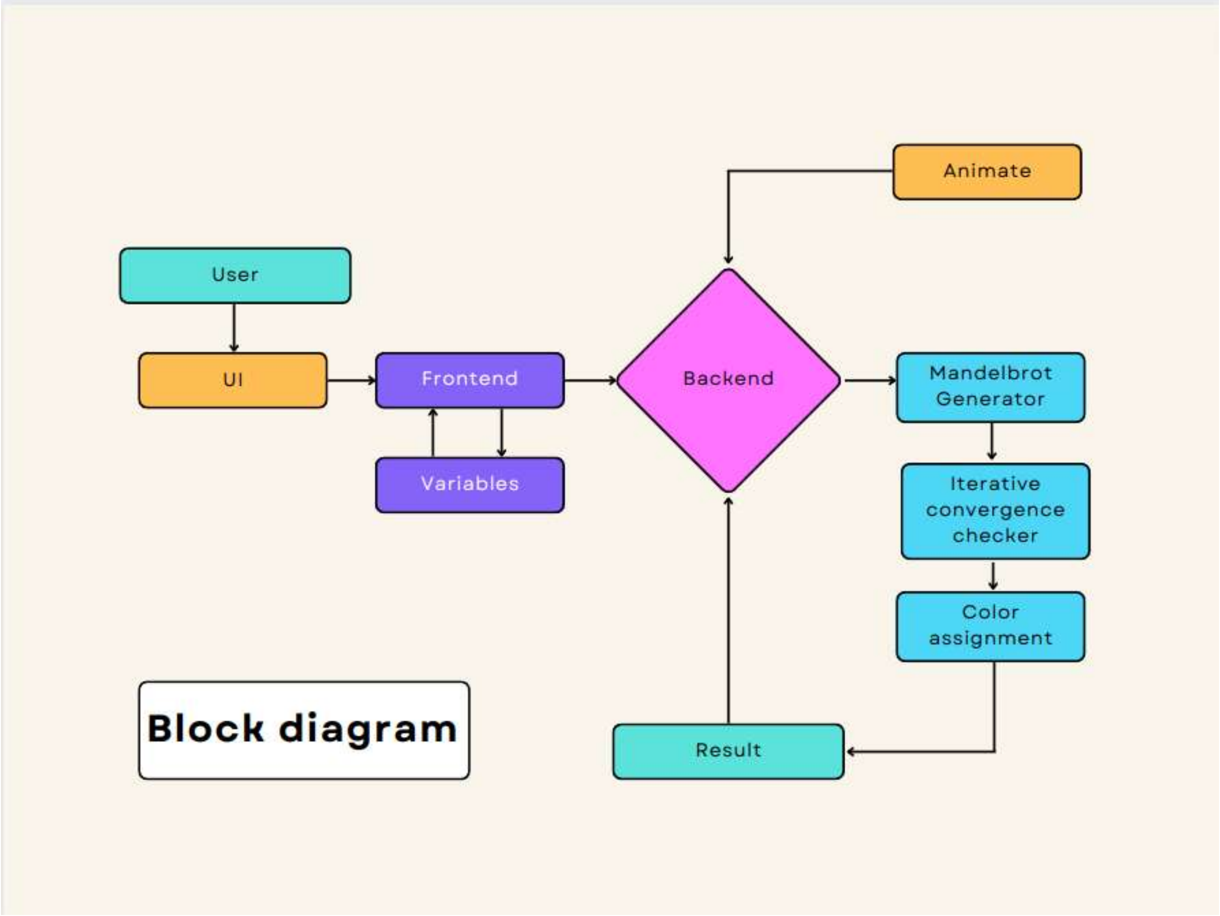
  This file contains the code to generate an image sequence. It calls the main function in image_gen with the required set of varying parameters. The generated images are stored in a folder named animate for later stitching into a video by a video editing program.

To visualize the Mandelbrot set in the first step is to define a function that calculates the Mandelbrot set for a given complex number. This function uses a loop to iterate over a range of complex numbers, and for each number it applies the Mandelbrot set formula to determine whether the number is part of the set.

The Mandelbrot set formula involves iteratively squaring and adding a complex number. For each complex number, the function will iterate over a fixed number of times, and for each iteration it will square the number and add the original number. If the magnitude of the resulting number is greater than 2, it is assumed to be outside of the Mandelbrot set and the iteration is stopped. If the magnitude is less than or equal to 2 after the maximum number of iterations, the number is considered to be part of the Mandelbrot set.

Once the function has been defined, it can be used to iterate over a range of complex numbers and calculate the Mandelbrot set for each number. The resulting data can then be plotted using a visualization library to create a graphical representation of the Mandelbrot set. This visualization is further customized by changing the color scheme, zoom level, and other parameters.

In real time view the steps used are 100 and transform is set to 0px horizontal and 0px vertical with a scaling of 1.



WHY MANDELBROT SET VISUALIZATION?

The Mandelbrot set is a mathematical object that is known for its intricate, detailed structure and its visual appeal. Visualizing the Mandelbrot set can be an interesting and challenging project for anyone interested in computer graphics, mathematics, or fractal geometry. It can also be a good way to learn about complex numbers, the concept of infinity, and the Mandelbrot set itself. In addition, a Mandelbrot visualizer can be a fun and engaging tool for exploring and navigating the set, and it can be a great way to generate interesting and beautiful images.

This project has covered the course outcome of scientific computing lab which includes data visualization using python.Mandelbrot being a mathematical object is created using mathematical data provided.The backend is mainly based on the math behind mandelbrot.

Even though mandelbrot math is simple ,there are certain challenges to make the final result so soothing to vision.The fact which makes this project bit tough is the property of infiniteness of the image.As we zoom through the image we can see the most beautiful part of mandelbrot which is   actually the repeated part of the image itself.Also ,mandelbrot set is the perfect example how wonderful math and data can create wonders in the field of visualization.Hence ,we find this project interesting and informative.

CODE

BACKEND

BASE

```
# Import flask and datetime module for showing date and time
from flask import Flask
import datetime
import cv2
from flask import send_file
from flask import request
import image_gen
x = datetime.datetime.now()

img = cv2.imread('mandelbrot.jpg')

# Initializing flask app
```

```
app = Flask(__name__)

@app.route('/data', methods=['GET'])

def home():
    data=(request.url.split("?")[1].split('px'))
    transform=[float(data[0]),float(data[1]),float(data[2])]
    scale=float(data[2])
    print(transform,scale)
    image_gen.main(transform,scale,'output.png')
    return send_file('output.png', mimetype='image/gif')


# Running app
if __name__ == '__main__':
    app.run(debug=True)
```

IMAGE GENERATION

```
from PIL import Image, ImageDraw
from math import log, log2

MAX_ITER = 1500

def mandelbrot(c):
    z = 0
    n = 0
    while abs(z) <= 2 and n < MAX_ITER:
        z = z*z + c
        n += 1

    if n == MAX_ITER:
        return MAX_ITER

    return n + 1 - log(log2(abs(z)))

def t1(elem,scale):
    if(scale==1):
        return 0
    else:
        return elem*(0.5-0.5*(0.99**scale))
# Image size (pixels)
```

```
def main(transform,scale,name):

    print(transform)
    WIDTH = int(1080)
    HEIGHT = int(720)
    transform=[x/WIDTH for x in transform]
    # Plot window

    t2=0.9
    RE_START = -2-(transform[0]/t2)+t1(3,scale)
    RE_END = (1-(transform[0]/t2))-t1(3,scale)
    IM_START = -1-(transform[1]/t2)+t1(2,scale)
    IM_END = (1-(transform[1]/t2))-t1(2,scale)

    im = Image.new('HSV', (WIDTH, HEIGHT), (0, 0, 0))
    draw = ImageDraw.Draw(im)

    for x in range(0, WIDTH):
        for y in range(0, HEIGHT):
            # Convert pixel coordinate to complex number
            c = complex(RE_START + (x / WIDTH) * (RE_END - RE_START),
                        IM_START + (y / HEIGHT) * (IM_END - IM_START))
            # Compute the number of iterations
            m = mandelbrot(c)
            # The color depends on the number of iterations
            hue = int(255 *( m *2)/ (MAX_ITER))
            saturation = 255
            value = 400 if m < MAX_ITER else 0
            # Plot the point
            draw.point([x, y], (hue, saturation, value))

    im.convert('RGB').save(name, 'PNG')
```

ANIMATION

```
import image_gen
import shutil
scale=1
frame=0
for i in range(0,300,1):
    name="temp"+str(frame)+".png"
    image_gen.main([300,-200],scale,name)
    source="./"+name
    destination="./animate/"+name
    shutil.move(source, destination)
    scale+=1
    frame+=1
```

FRONTEND(PART1)

```
import logo from './fractal.png';
import load from './spinner.gif';
import './Mandel2.css';
import 'bootstrap/dist/css/bootstrap.css';
import { Navbar } from 'react-bootstrap';
import { useState } from 'react';
import Button from "react-bootstrap/Button";
import up from './up.svg';
import down from './down.svg';
import zoomOut from './zoom-out.svg';
import zoomIn from './zoom-in.svg';
import refresh from './refresh.svg';
import text from "./mandel_text.png"
import text2 from "./mandel_text2.png"
var vertical=0;
var horizontal=0;
var zoom=1
const Mandel = () => {

      const [Render, setRender] = useState();

      function sayMandel() {
        const url="https://en.wikipedia.org/wiki/Mandelbrot_set";
        window.open(url);
      }
      function sayBenoit() {
        const url="https://en.wikipedia.org/wiki/Benoit_Mandelbrot";
        window.open(url);
      }
```

```
    function getRender(){
      setRender(load);

      return render()
    }

    const render = async () => {
      try{
        var
Transform=horizontal.toString()+"px"+vertical.toString()+"px"+zoom.toString
()+"px";
      }
      catch{}
      console.log(Transform)
      const res = await fetch("/data?"+Transform)

      const imageBlob = await res.blob();
      const imageObjectURL = URL.createObjectURL(imageBlob);
      setRender(imageObjectURL);
    };
    function updateVert1(){
      vertical-=100;
    }
    function updateVert2(){
      vertical+=100;
    }
    function updateHorz1(){
      horizontal-=100;
    }
    function updateHorz2(){
      horizontal+=100;
    }
    function updateZoom1(){
      zoom+=1;
    }
    function updateZoom2(){
      zoom-=1;
    }
    function reset(){
      vertical=0;
      horizontal=0;
      zoom=1;
    }

    return (

        <div className='App1'>

        <header className="App-header2">

            <Navbar>
            <img onClick={sayBenoit}
            src={logo}
            alt="Sample Brand Logo"
            width="35"
            className="align-top d-inline-block"
            height="35"
          />
            <p onClick={sayMandel}>MANDELBROT</p>
            <div className='panel'>
              <Button variant='primary' style={{ backgroundColor: "#00B1E1"
}} className='rounded-circle' onClick={updateVert1}>
                <img src={up} className='btn-img' alt="btn"></img>
              </Button>
              <Button variant='primary' style={{ backgroundColor: "#00B1E1"
}} className='rounded-circle' onClick={updateVert2}>
                <img src={down} className='btn-img' alt="btn"></img>
              </Button>
              <Button variant='primary' style={{ backgroundColor: "#00B1E1"
}} className='rounded-circle' onClick={updateHorz1}>
                <img src={down} className='btn-img' alt="btn" style={{
rotate:"90deg" }}></img>
              </Button>
              <Button variant='primary' style={{ backgroundColor: "#00B1E1"
}} className='rounded-circle' onClick={updateHorz2}>
                <img src={down} className='btn-img' alt="btn" style={{
rotate:"-90deg" }}></img>
              </Button>
              <Button variant='primary' style={{ backgroundColor: "#00B1E1"
}} className='rounded-circle' onClick={updateZoom1}>
                <img src={zoomIn} className='btn-img' alt="btn"></img>
              </Button>
              <Button variant='primary' style={{ backgroundColor: "#00B1E1"
}} className='rounded-circle' onClick={updateZoom2}>
```

```
                <img src={zoomOut} className='btn-img' alt="btn"></img>
              </Button>
              <Button variant='primary' style={{ backgroundColor: "#00B1E1"
}} className='rounded-circle' onClick={reset}>
                <img src={refresh} className='btn-img' alt="btn"></img>
              </Button>
            </div>
          <Button variant="info" onClick={getRender}>
            RENDER
          </Button>

            </Navbar>

        </header>
        <div className='container2'>
          <div className='canvas'>
            <img src={Render} className='render' alt=""></img>
          </div>
          <div className='controls'>
            <p id="text">
              <img src={text} alt="" className='text'></img>
            </p>
          </div>
        </div>
        <div className='footer'>
        <img src={text2} alt="" className='text2'></img>
        </div>

        </div>
    );
}

export default Mandel;
```

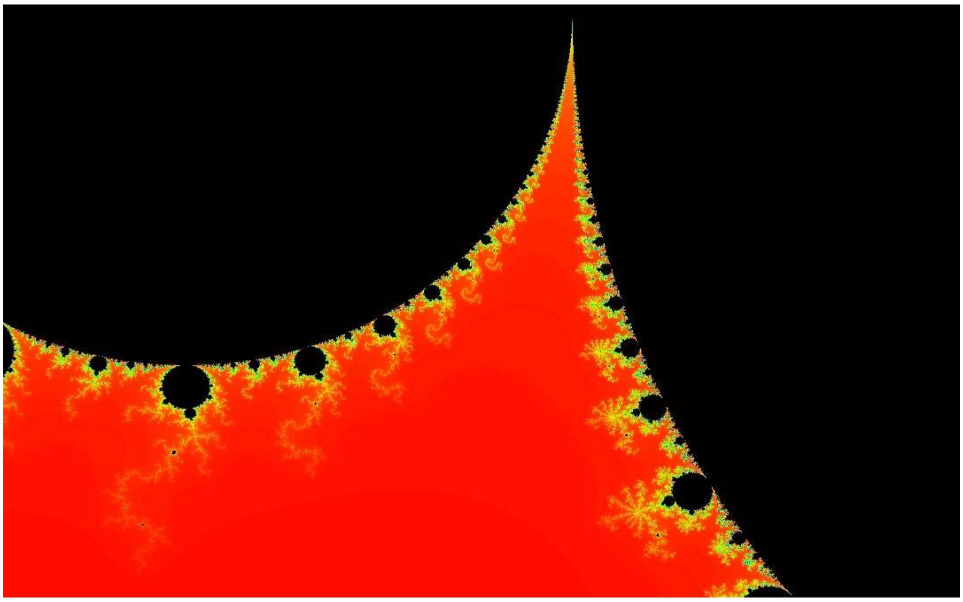
FRONTEND(PART2)

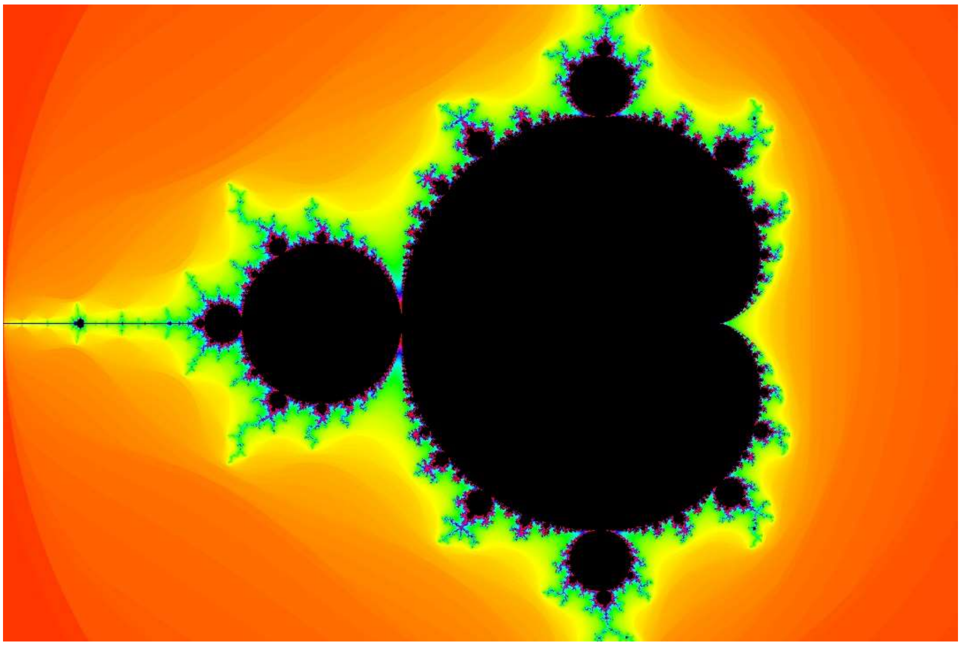
```
.App-header2{
  height: 70px;
  width: auto;
  position: relative;
  display: grid ;
  grid-template-rows:max-content;
  background-color: #282c34;
  overflow: hidden;
}
.bg-App-col {
  background-color: #1c1f24;
}
.navbar{
  display: flex;
  justify-items: center;
  align-items: center;
  height: 70px;
  width:100%;
}
img{
  position: inherit;
  left: 20px;
}
p{
  position: inherit;
  left:40px;
  top:7px;
  width: 80%;
  color:rgb(180, 204, 219);
  font-weight: 700;
  font-size: 40px;
  -webkit-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
  user-select: none;
}
.p2{
  position: inherit;
  left:0%;
  top:10px;
  color:rgb(255, 255, 255);
  font-weight: 700;
  font-size: 23px;
}
.cont{
  width: 5%;
  max-width: 300px;
  align-items: center;
```

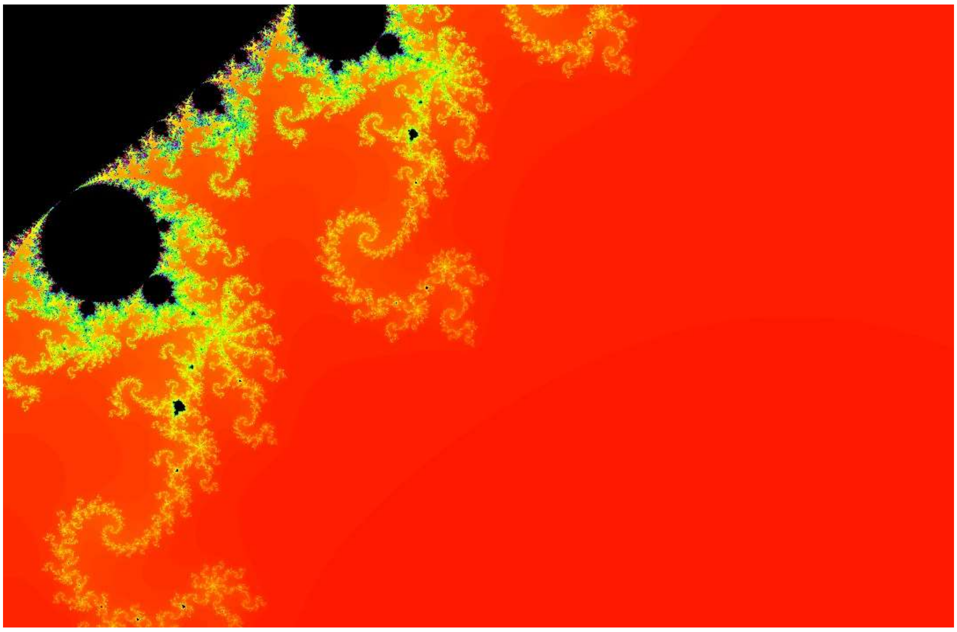
```
}
.btn-info{
  position: absolute;
  right: 2.5%;
  top: 25%;
  align-self:center;
  box-ordinal-group: aquamarine;

}

.canvas{
  background-image: rgb(177, 31, 31);
  position:relative;
  display: grid;
  grid-template-columns: 50% 10px;
  width: fit-content;
  margin: 0;
  overflow: hidden;
}
.render{
  position: relative;
  left: 0px;
}

.controls{
  background-color: antiquewhite;
}
.footer{
  background-color: #1c1f24;
  width: 100%;
  bottom:0;
  height:192px;
  overflow-y:-moz-hidden-unscrollable;;
  position: relative;
}


.disable-select {
  -webkit-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
  user-select: none;
```

```
}

.container2{
  height: auto;
  width: auto;
  position: relative;
  background-color: antiquewhite;}
```

RESULT







REFERENCES

- https://en.wikipedia.org
- https://mathworld.wolfram.com
- https://math.hws.edu