# Realtime And Editable Mandelbrot Set on GPU

Qihe Chen
chenqh@shanghaitech.edu.cn

Haiyu Song
songhy@shanghaitech.edu.cn

January 2, 2023

**Abstract**

In this final programming project, we implement Mandelbrot Set on GPU with editable features and realtime rendering high performance. Since the Mandelbrot set is generated by iteration, it is almost impossible to make the parallelization for single pixel color calculation. Therefore, we focus on the parallelization of warps of pixels instead of single pixel. To the rendering part, we implement two techniques on GPU, i.e. phong lighting and stripe average coloring. To the experimental part, we implement two kinds of Mandelbrot Set, i.e. basic algorithm and escaped time based algorithm. To the evaluation part, we compare our method with serial method on basic algorithm.

## 1 Introduction

We implement the Mandelbrot Set on GPU and build a rendering framework by OpenGL and Dear Imgui.

### 1.1 Compile & Build

We recommend to compile the project on Windows 10 operation system. For compiling successfully, g++ should support at least C++ 17 standard and nvcc should support at least C++ 17 also.

We recommend we to build the project by CMake with version at least 3.21.

```
mkdir build
cd ./build
cmake ..
cmake --build . --config Release
./Release/main.exe
```

It should compile successfully and automatically run the executable file.

### 1.2 Benchmark Configures

| Config | Value |
|---|---|
| CPU | Intel(R) Core(TM) i9-10900x CPU @ 3.70GHz |
| GPU | NVIDIA GeForce RTX 3090 |
| Memory | 64 GB DDR4 3200 MHz 2 of 8 slots |
| GPU Memory | 24576 MB GDDR6X (Micron) |
| OS | Windows 10 21H2 19044.2251 |
| Compiler | MSVC 19.33.31630.0, nvcc V11.8.89 |
| Optimize | MSVC: /O2, nvcc: -O3 |
| CUDA Cores | 10496 |
| Bandwidth | 936.2GB/s |
| Driver Version | 31.0.15.2206 (NVIDIA 522.06) DCH / Win 10 64 |

## 1.3 Mandelbrot Set

The Mandelbrot set is the set of complex numbers $c$ for which the function $f_c(z) = z^2 + c$ does not diverge to infinity when iterated from $z = 0$, i.e., for which the sequence $f_c(0)$, $f_c(f_c(0))$, etc., remains bounded in absolute value.[wik]

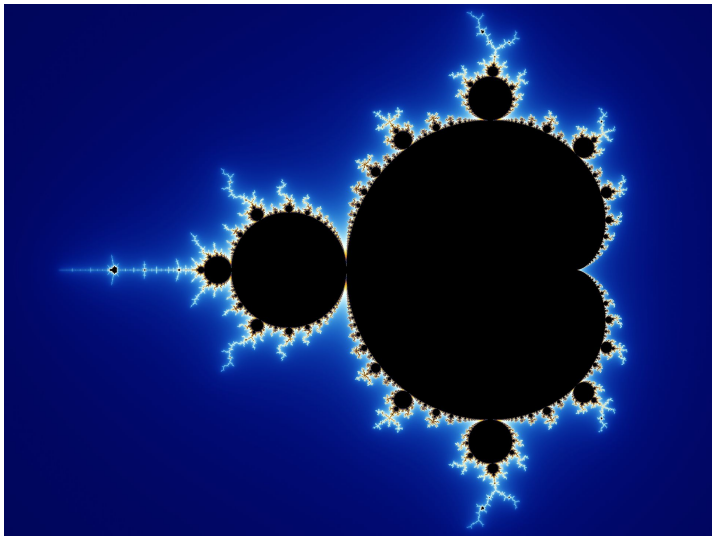


Figure 1: MandelbrotSet

For a complex $z$, given parameter complex $c$, we define $f$ function below:

$$f_c(z) = z^2 + c \tag{1}$$

If the orbit of the critical point $z = 0$ under iteration of quadratic marginparwidth

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

remains bounded. We say that the complex number $c$ is a member of the Mandelbrot Set, when starting with $z_0 = 0$ and applying the iteration repeatedly, the absolute value of $z_n$ remains bounded for all $n > 0$.

## 1.4 Speed up

We evaluate the performance of our GPU-version Mandelbrot Set by comparing the running time to render a $800 \times 600$ frame with CPU-version under different max iterations. For stability, we continue to render frames until the average fps converging. And the CPU-version is accelerated by OpenMP.

| maxiter | CPU ms/frame | GPU ms/frame | CPU fps | GPU fps | speed up |
|---|---|---|---|---|---|
| 100 | 113.081 | 7.225 | 8.8 | 138.4 | 15.651 |
| 200 | 217.070 | 10.148 | 4.6 | 98.5 | 21.390 |
| 300 | 307.077 | 14.200 | 3.3 | 70.5 | 21.625 |
| 500 | 495.950 | 22.303 | 2.0 | 44.9 | 22.237 |
| 1000 | 942.430 | 42.444 | 1.1 | 23.6 | 22.204 |
| 2000 | 1876.767 | 83.336 | 0.5 | 12.0 | 22.520 |
| average | | | | | 20.92 |

# 2 Preliminary

## 2.1 Mandelbrot Set

### 2.1.1 Definition.

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 $z_{t+1} = z_t^2 + c$, where $c$ is a constant number.

### 2.1.2 Mathematics.

From the mathematical view, the Mandelbrot set is a mathematical set of points that is defined in the complex plane. The set is define using an iterative process that involves performing a simple mathematical operation on a complex number repeatedly. This operation is defined by the equation:

$$z = z^2 + c \tag{3}$$

where $z$ and $c$ are complex numbers. The mandelbrot set is the set of all complex numbers $c$ for which the above equation does not diverag, that is , the absolute value of $z$ remains less than 2 for all iterations. The equation is repeated a large number of times to determine whether a given number is in the set or not.

**Lemma 2.1.** *A point c belongs to the Mandelbrotset if and only if* $|z_n| \leq 2$ *for all* $n \geq 0$.

*Proof.* if $|z_j| > 2$, we can write $|z_j|$ as $|z_j| = 2 + \varepsilon$ for $\varepsilon \in \mathcal{R}^+$

$$\begin{aligned} |z_j^2| &= |z_j^2 + c - c| \\ &\geq |z_j^2 + c| + |c| \\ |z_{j+1}| &= |z_j^2 + c| \\ &\geq |z_j|^2 - |c| \\ &\geq |z_j|(|z_j| - 1) \\ &\geq |z_j|(1 + \varepsilon) \end{aligned}$$

So, after k iterations, we have

$$|z_{j+k}^2 + c| \geq |z_j|(1 + \varepsilon)^k$$

which escapes to infinity.
Since $c = z_1$, it follows that $|c| \leq 2$, establishing that $c$ will always be in the closed disk of radius 2 around the origin. □

### 2.1.3 Background

The Mandelbrot set is interesting because it exhibits a complex and intricate structure when plotted, with a seemingly infinite level of detail. It has become famous for its aesthetic appeal and has been featured in many works of art, music, and literature. It is also studied in mathematics for its connection to the field of fractals, which are geometric shapes that are self-similar and exhibit a repeating pattern at different scales.

## 2.2 Phong Lighting

Phong lighting is a type of 3D computer graphics shading techique, which is extended to 2D Mandelbrot Set shading in this project. In the Phong lighting model, the intensity of the reflected light at any point on a surface is a combination of three components: ambient light, diffuse relfection, and specular reflection.

- Ambient light is the base level of light that is present in a scene, and it is evenly distributed over all surfaces.

- Diffuse reflection is the light that is scattered evenly in all directions after it his a rough surface.
- Specular reflectuion is the light that is reflected in a specific direction, producing a highlight or a shiny spot on a surface.

The Phong lighting model is typically implemented in computer graphcis using the following equation:

$$L = I_a K_a + I_d K_d \max(0, n \cdot I) + I_s K_s \max(0, n \cdot h)^p \tag{4}$$

### 2.3 Stripe Average Coloring

Stripe average coloring is a post-processing technique for creating color schemes for fractals. The technique involves creating color bands or "stripes" of uniform color and then blending or averaging the colors of adjacent stripes to create a smooth gradient effect. In this project, it can be used to create colorful and visually appealing images of fractals.

## 3 Method

### 3.1 Basic Algorithm

**Algorithm 1:** Escape Algorithm

**Input:** complex c, maxiteration
**Output:** iteration count

```
z ← 0
count ← 0
for i=1 to maxiteration and abs(z) < 2 do
    z ← z^2+c
    count ← count+1
end
return count
```

### 3.2 Visualization

In this project, we use OpenGL to help us rendering the image realtime and use third library "imgui" to better GUI to control the input parameter and selcet the drawing mode.

#### 3.2.1 Coloring

We use colortable to color the pixel according to their iteration count.
We produce the colortable using sin function and convert from [0,1] to [0,255] later.

**Algorithm 2:** ColorTable Algorithm

**Input:** theta, colorsize
**Output:** colortable

```
dx ← 1 / colorsize
colors ← empty vector
for i=0 to colorsize do
    color ← (vec3(dx * i) + theta) * 2 * PI
    color ← 0.5+0.5*sin(y)
    colors[i] ← color
end
return colors
```

### 3.2.2 Color smooth

We use smoother function to smooth number of iteration in the Mandelbrot set for given c.
In this funciton, we combine the frequency parameter and memory parameter to control the stripe average Coloring.
In particular, we use "Jussi Harkonen On Smooth Fractal Coloring Techniques" to stripe average coloring and use "Smoothing + linear" to interpolate.

**Algorithm 3:** Smoother Algorithm

```
Input: complex c, stripe_s,stripe_sig
Output: niter, stripe_a, dem,normal
z ← 0
dz ← 1
esc_radius ← 1e5
stripe_a ← 0
for n=0 to maxiter do
    dz ← (2*z*dz)+1
    z ← z*z+c
    stripe_tt ← (sin(stripe_s*atan2(z.imag, z.real)) + 1) / 2
    if abs(z) > esc_radius then
        modz ← abs(z)
        log_ratio ← log(modz) / log(esc_radius)
        smooth_i ← 1 - log(log_ratio) / log(2)
        stripe_a ← (stripe_a * (1 + smooth_i * (stripe_sig-1)) + stripe_t * smooth_i * (1 -
          stripe_sig))
        stripe_a ← stripe_a / (1 - pow(stripe_sig, n) * (1 + smooth_i * (stripe_sig-1)))
        normal ← z/dz
        dem ← modz * log(modz) / abs(dz) / 2
        return (n+smooth_i, stripe_a, dem, normal)
    end
    stripe_a ← stripe_a * stripe_sig + stripe_tt * (1 - stripe_sig)
end
return (0,0,0,{0, 0})
```

### 3.2.3 Phong lighting

We use Lambert normal shading for better visualization.
The color of a pixel is

$$\text{result} = \text{ambient} + \text{diffuse} + \text{specular}$$

**Algorithm 4:** Phong lighting Algorithm

```
Input: complex normal, light
Output: brightness
normal ← normalize
diff ← normal.real*cos(φ)*cos(θ) + normal.imag*sin(φ)*cos(θ) +sin(θ)
diff ← diff/(1+sin(θ))
thetahalf ← (pi/2 + θ)/2
spec ← normal.real*cos(φ)*sin(thetahalf) + normal.imag*sin(φ)*sin(thetahalf)
  +cos(thetahalf)
spec ← spec/(1+cos(thetahalf))
spec ← spec ** shininess
bright ← k_ambient + k_diffuse*diff + k_specular*spec
bright ← bright * opacity + (1-opacity)/2
return bright
```

### 3.3 Optimization

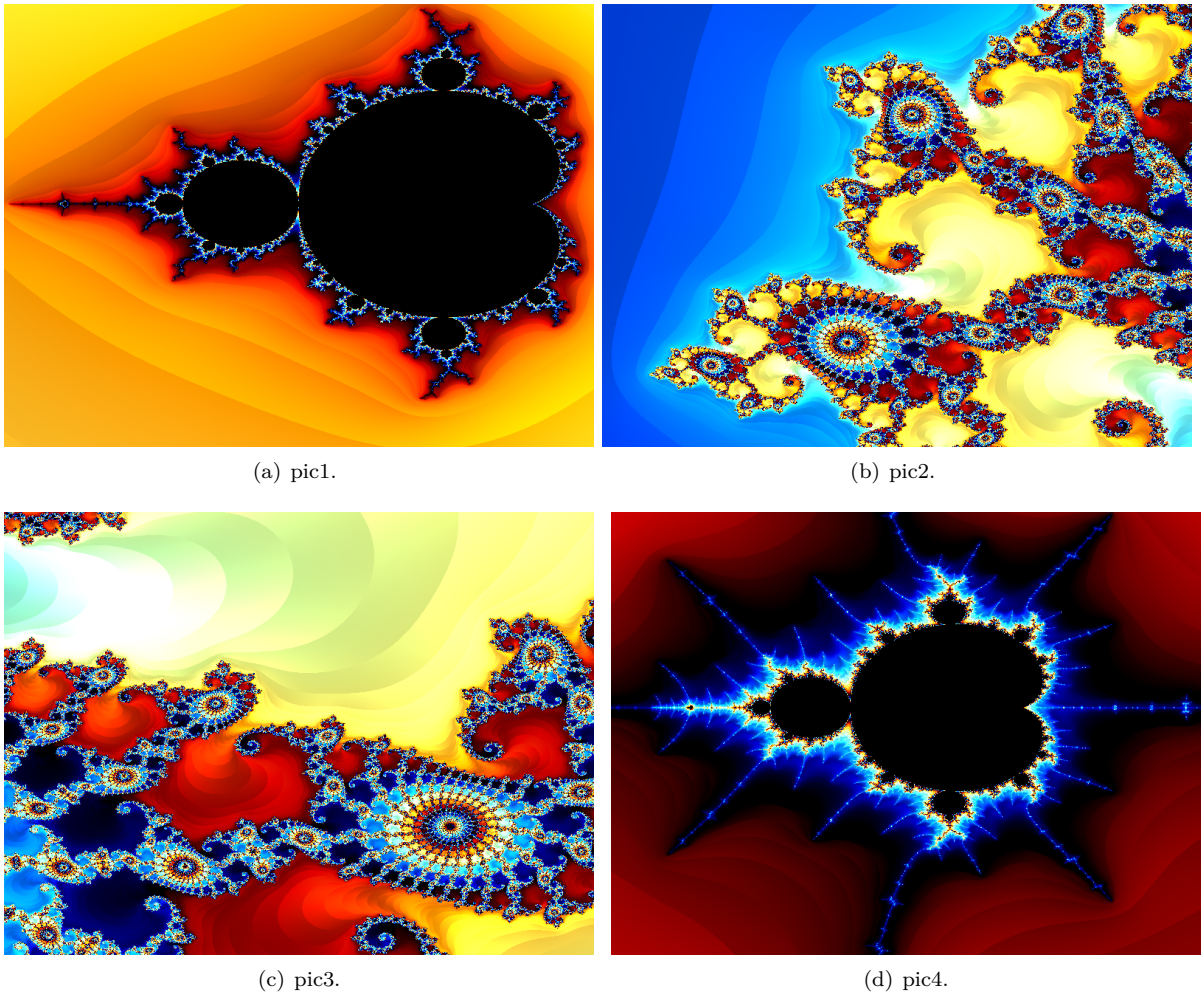
#### 3.3.1 Cuda parallel

Since calculating for each pixel are independent and the workload are nearly balanced. Thus, we can use Cuda to help us generate the image quickly.
Naturally, we use $*data$ to store the whole pixel with size $width * height$. And we equally divide the data calculation work for each block.

#### 3.3.2 Memory access and allocation

Since there are many data pass from device to host when we draw pictures, we don't need to malloc cudamemory each time when we call Cuda kernel.
Instead, we just assign memory for host and device respectively once, and each time we call cuda kernel, we just pass the pointer to cuda. When we need to feed back the data from device, we simply transfer the the data from device to host with the help of <thrust/host_vector.h>and <thrust/device_vector.h>.

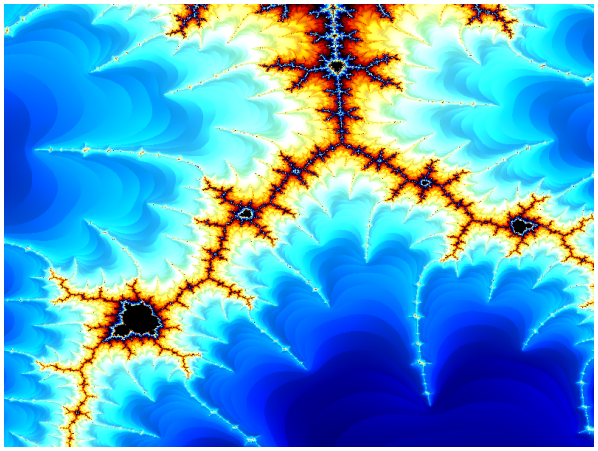
## 4 Experiment



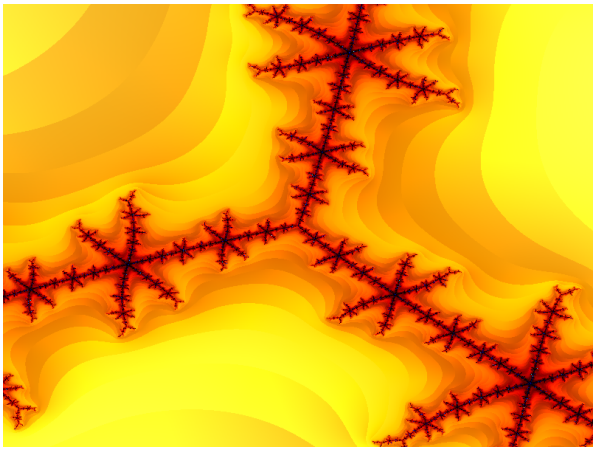
(a) pic1. (b) pic2.

(c) pic3. (d) pic4.

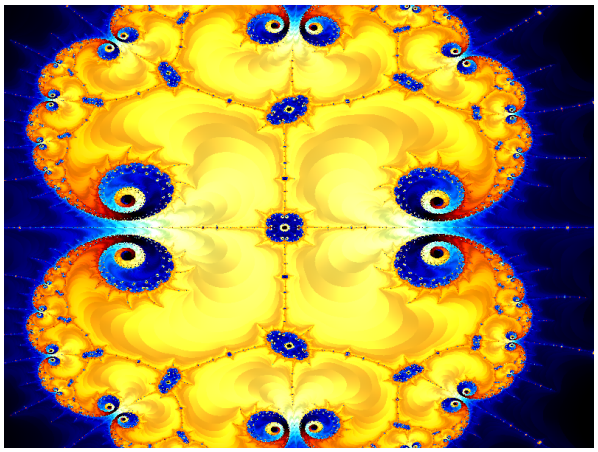
Figure 2: pics



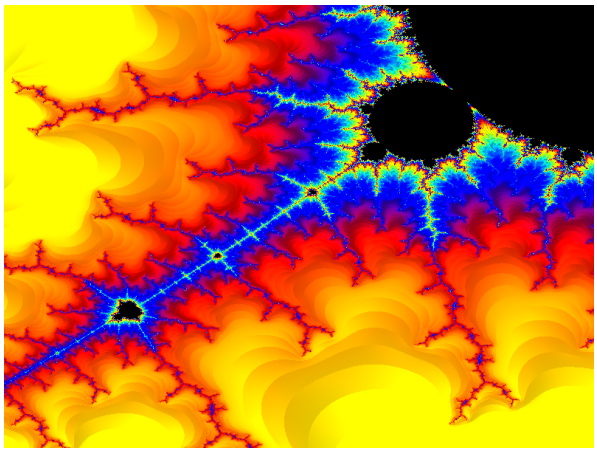
(a) pic1.



(b) pic2.

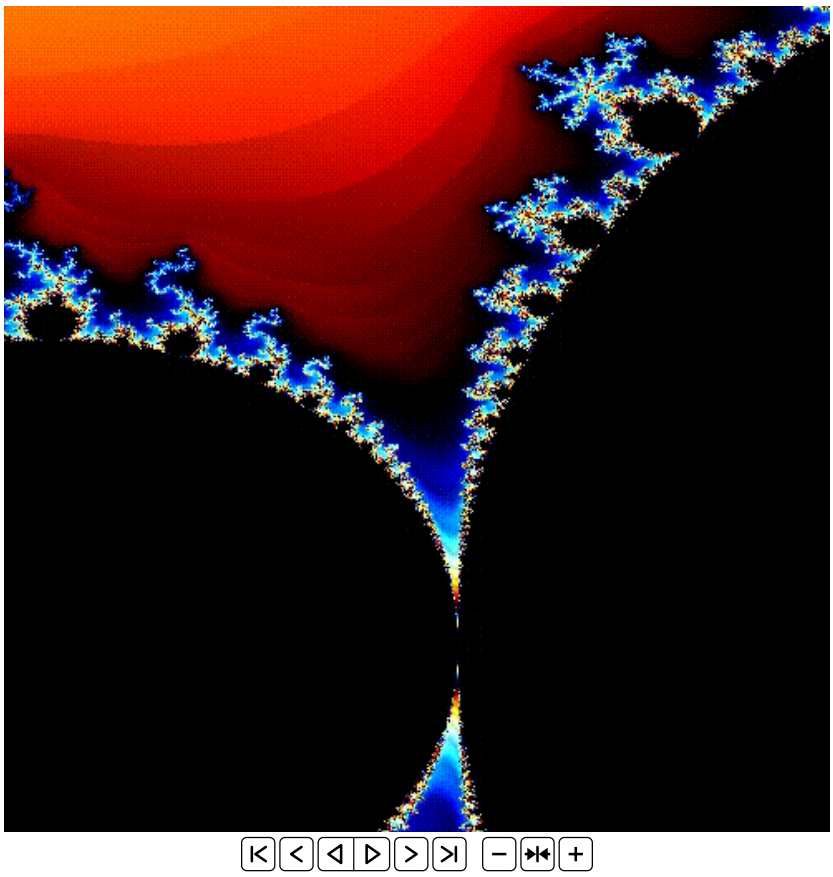


(c) pic3.



(d) pic4.

Figure 3: pics



## 5 Conclusion

In this final programming project, we implement Mandelbrot Set on GPU with editable features and realtime rendering high performance. Since the Mandelbrot set is generated by iteration, it is almost impossible to make the parallelization for single pixel color calculation. Therefore, we focus on the parallelization of warps of pixels instead of single pixel. To the rendering part, we implement two techniques on GPU, i.e. phong lighting and stripe average coloring. To the experimental part, we implement two kinds of Mandelbrot Set, i.e. basic algorithm and escaped time based algorithm. To the evaluation part, we compare our method with serial method on basic algorithm. Finally, we achieve 20x times speed up than traditional openmp serial algorithm.

## References

[wik] wikipedia. Mandelbrot set. "https://en.wikipedia.org/wiki/Mandelbrot_set". 2022.