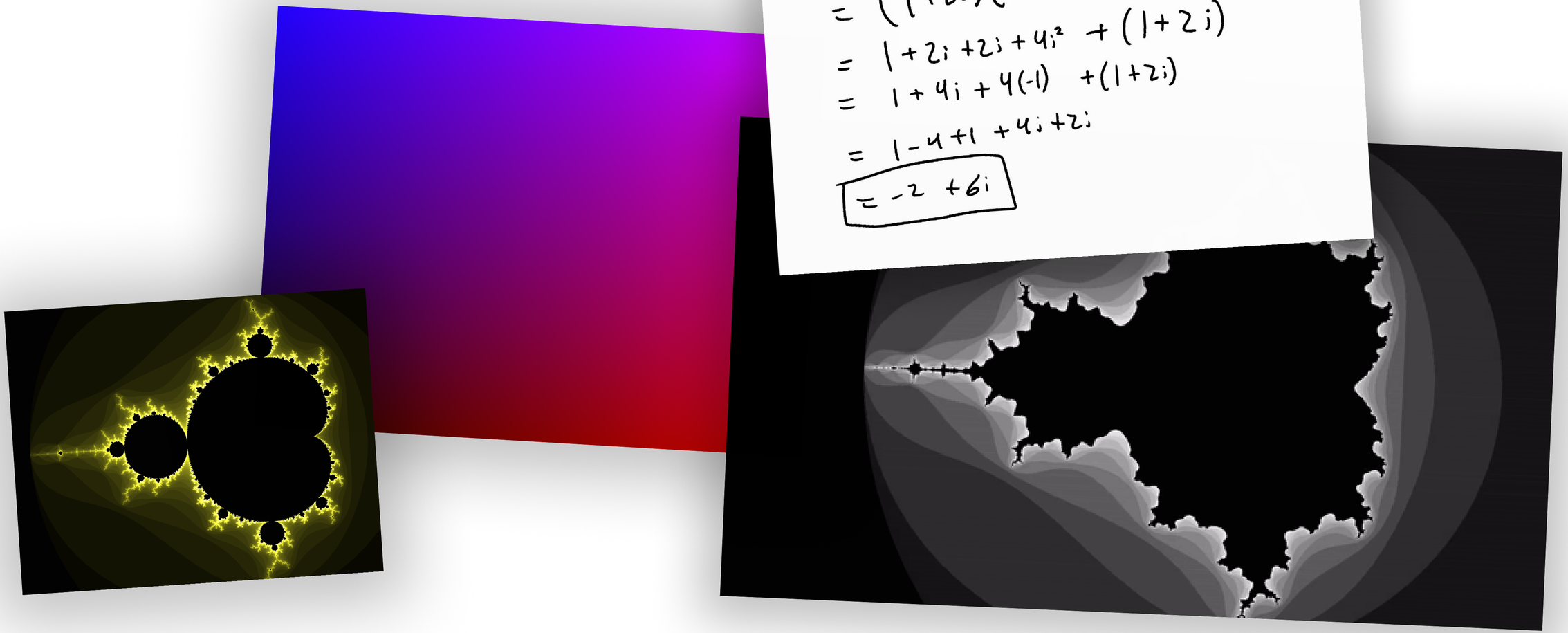
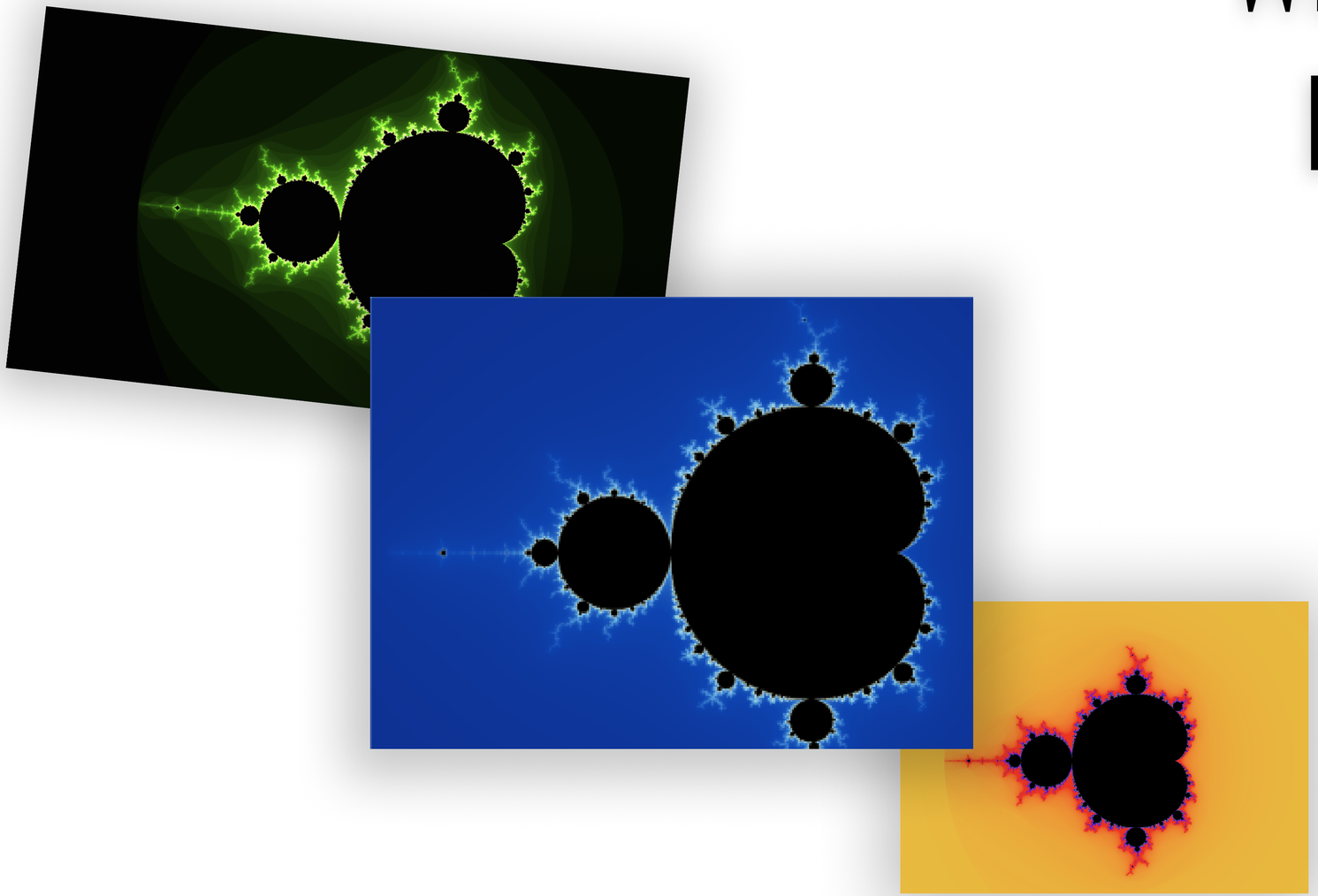
Create **INFINITE ART** with **MATH** (and code)





What is the **Mandelbrot** set?

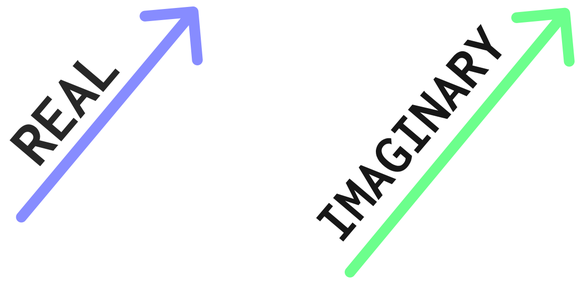
Complex Numbers

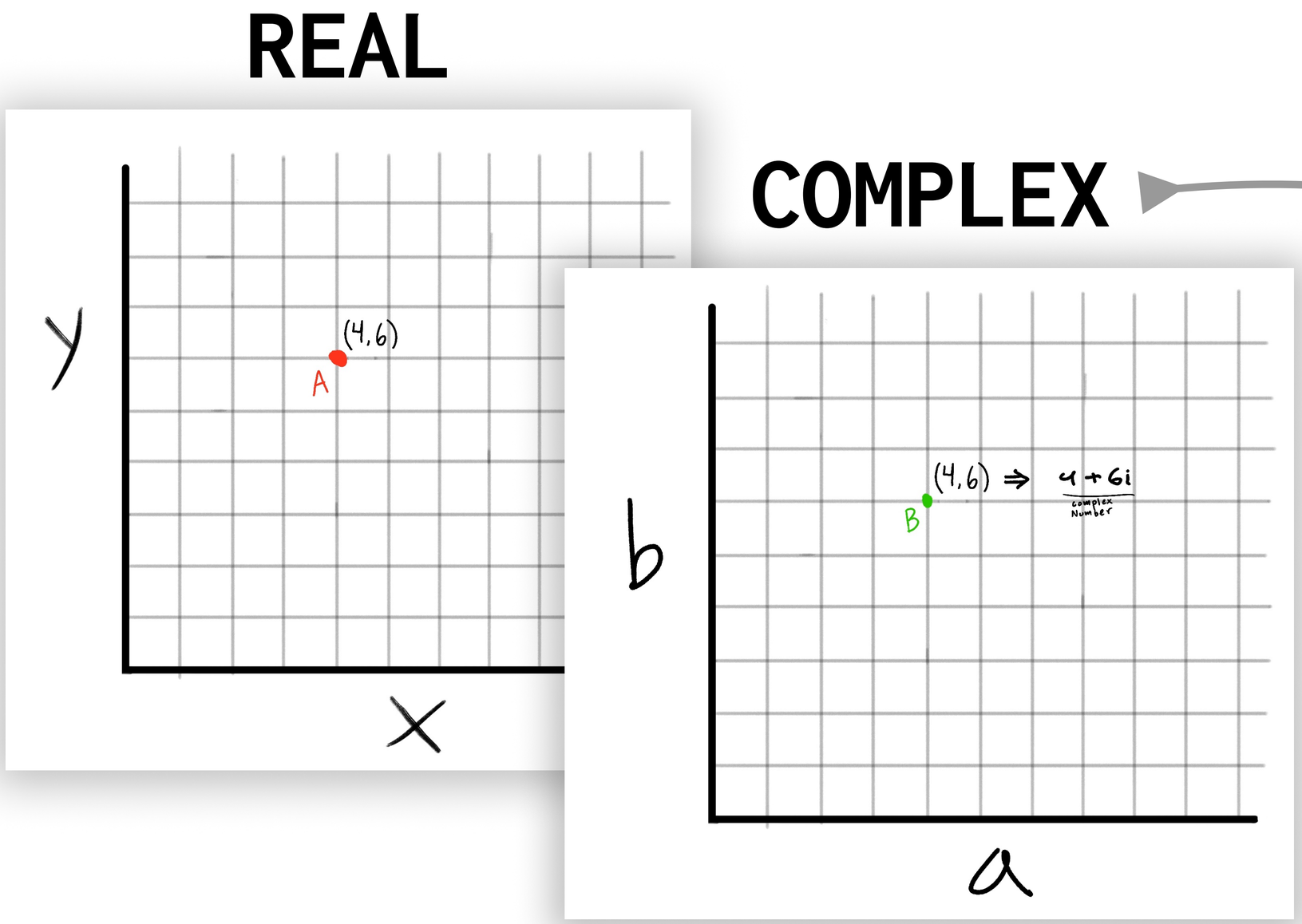
The imaginary number i is equal to the square root of -1

$$i = \sqrt{-1}$$

A complex number is just a number with a real and imaginary part

$$a + bi$$




REAL
y
(4,6)
A
x
COMPLEX
b
(4,6) ⇒ 4 + 6i
complex Number
B
a
a + bi

The **MANDELBROT EQUATION**

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

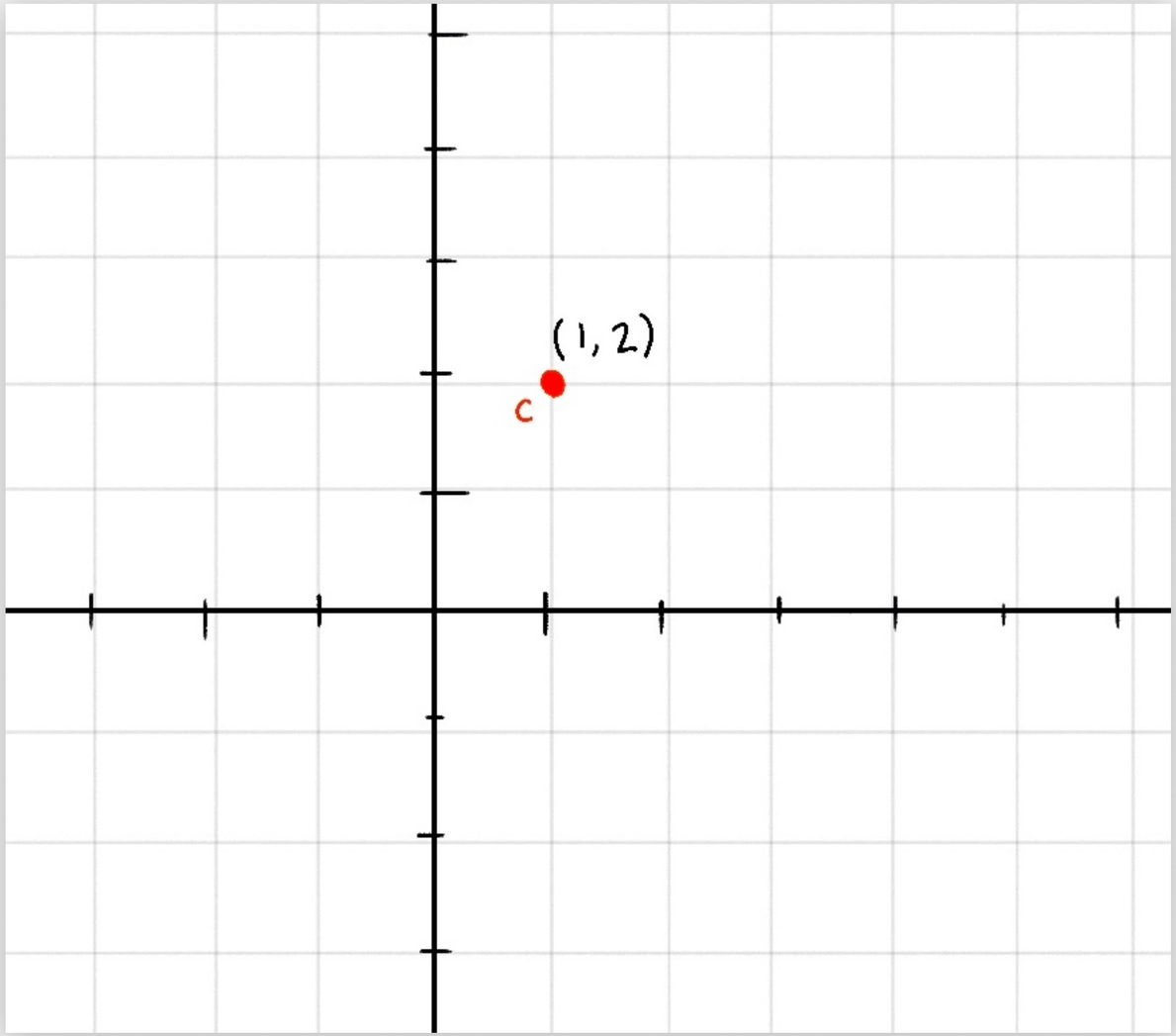
Squared

Complex point

Next Iteration

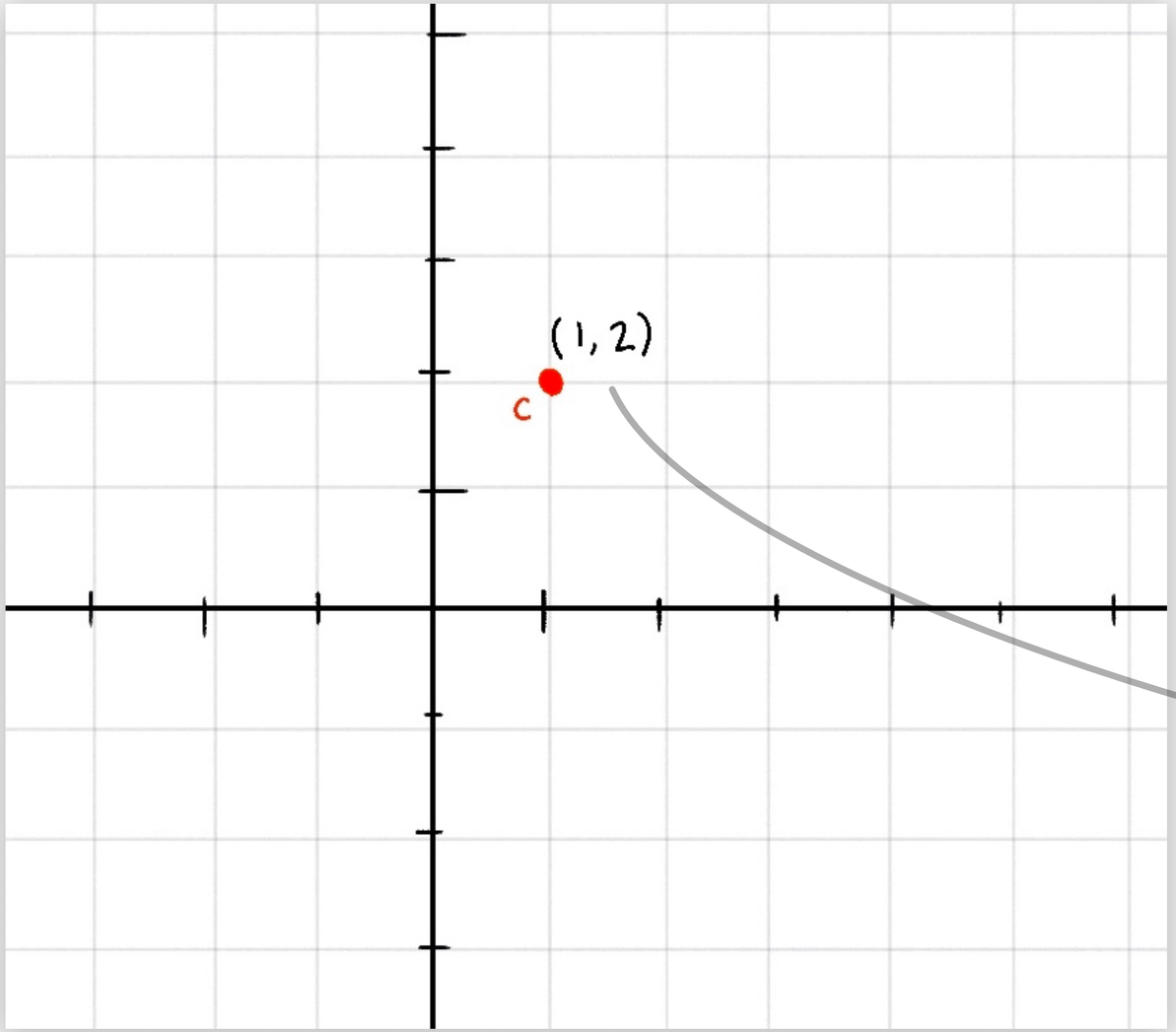
Current Iteration

ITERATION **0**



sequence
[0]

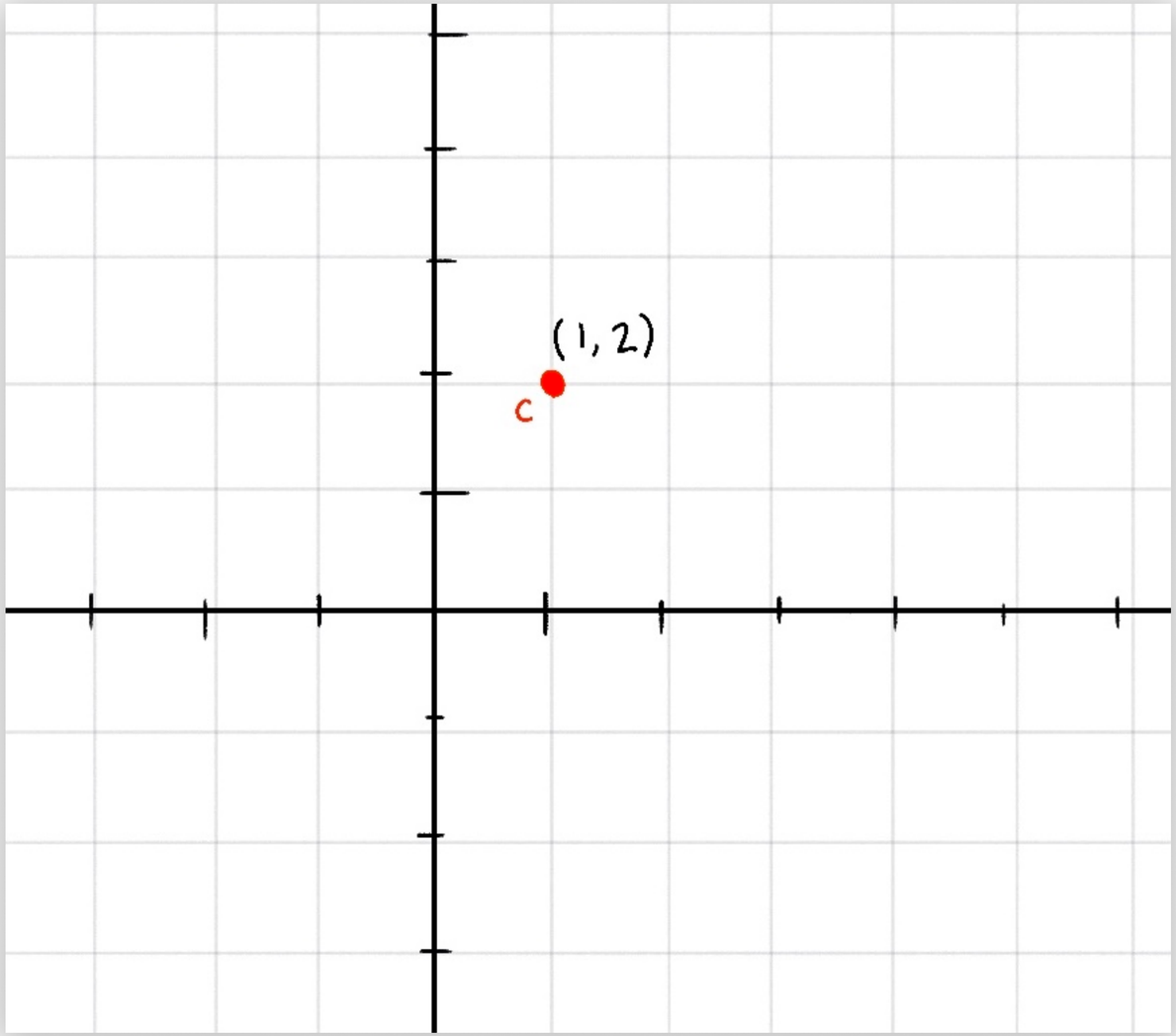
ITERATION 1



sequence
[0]

$$0^2 + (1 + 2i)$$

ITERATION 1



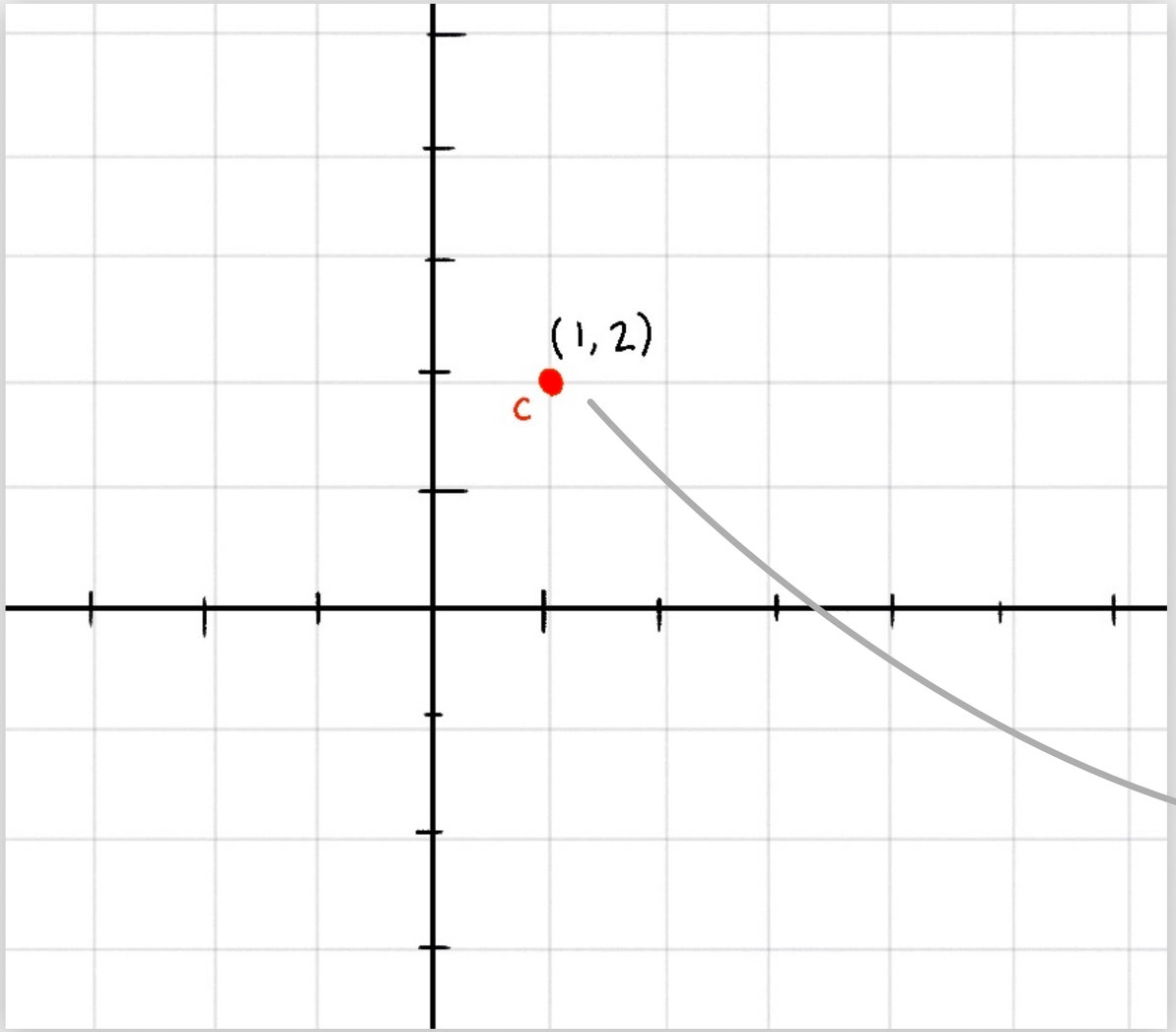
sequence

```
[0, 1 + 2i]
```

$$0^2 + (1 + 2i)$$

$$= 1 + 2i$$

ITERATION **2**

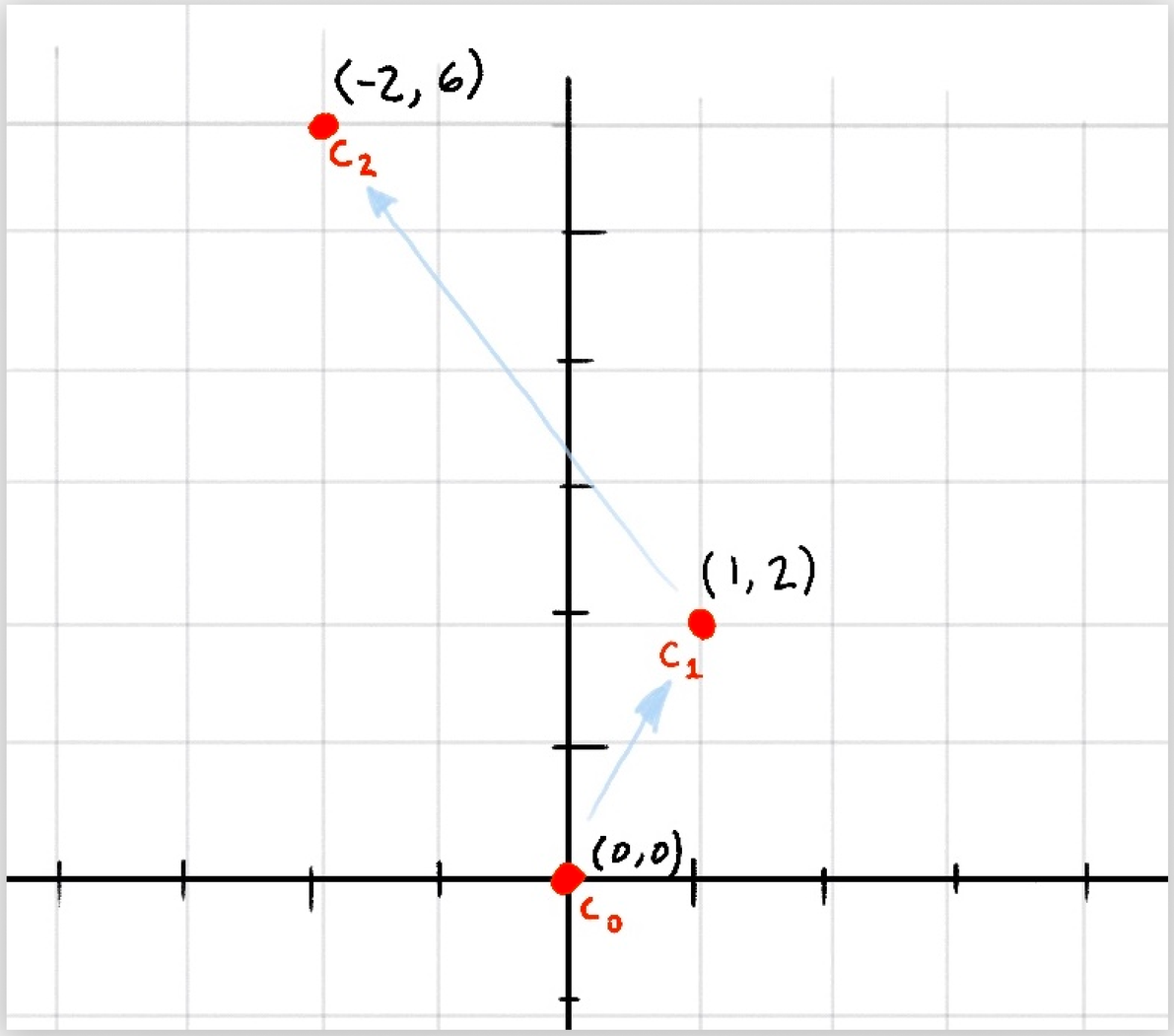


sequence

[0, 1 + 2i]

$$(1+2i)^2 + (1+2i)$$

ITERATION 2

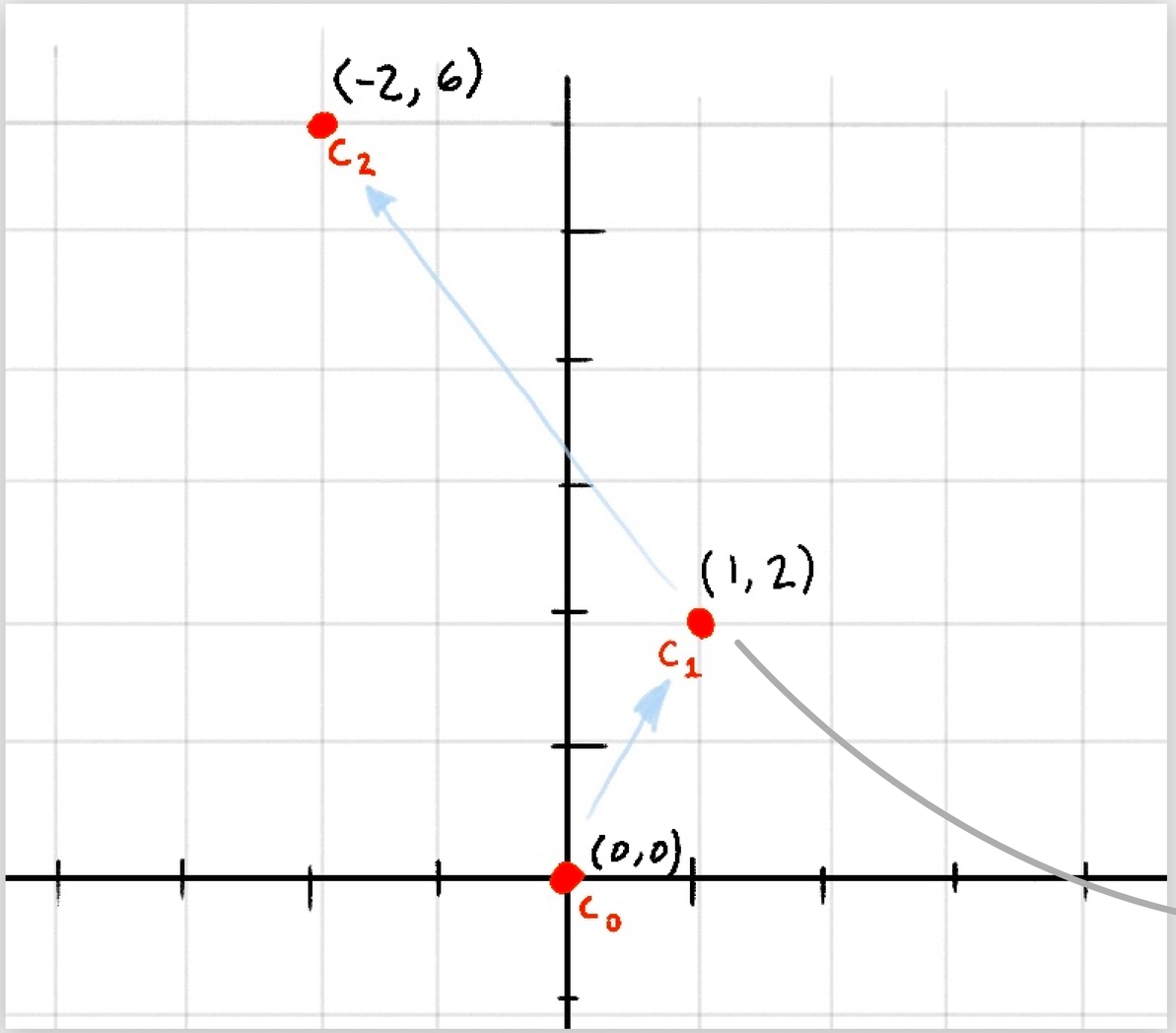


sequence

[0, 1 + 2i, -2 + 6i]

$$(1+2i)^2 + (1+2i)$$
$$= (1+2i)(1+2i) + (1+2i)$$
$$= 1 + 2i + 2i + 4i^2 + (1+2i)$$
$$= 1 + 4i + 4(-1) + (1+2i)$$
$$= 1 - 4 + 1 + 4i + 2i$$
$$= -2 + 6i$$

ITERATION 3

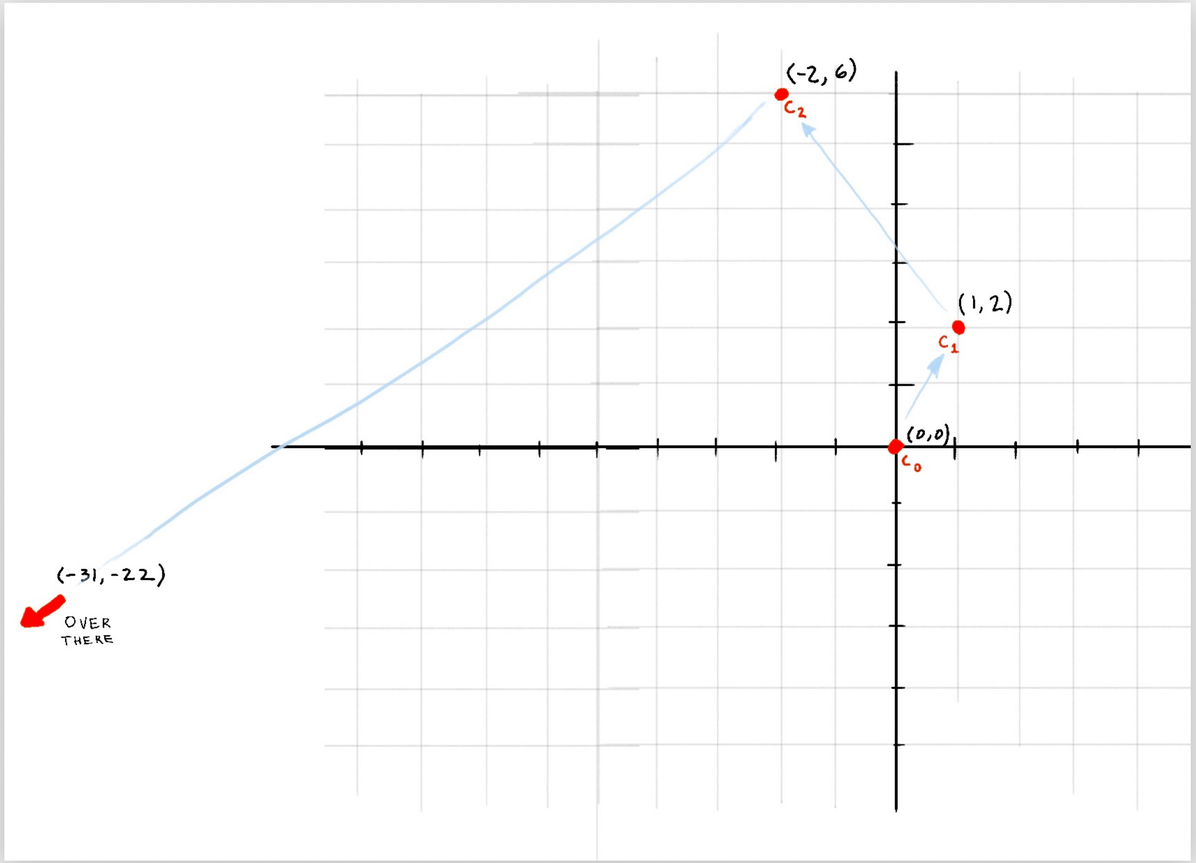


sequence

[0, 1 + 2i, -2 + 6i]

$$(-2+6i)^2 + (1+2i)$$

ITERATION 3



sequence

[0, 1 + 2i, -2 + 6i, -31 - 22i]

$$(-2+6i)^2 + (1+2i)$$

$$= (-2+6i)(-2+6i) + (1+2i)$$

$$= 4 - 12i - 12i + 36i^2 + (1+2i)$$

$$= 4 - 24i + 36(-1) + (1+2i)$$

$$= 4 - 36 + 1 - 24i + 2i$$

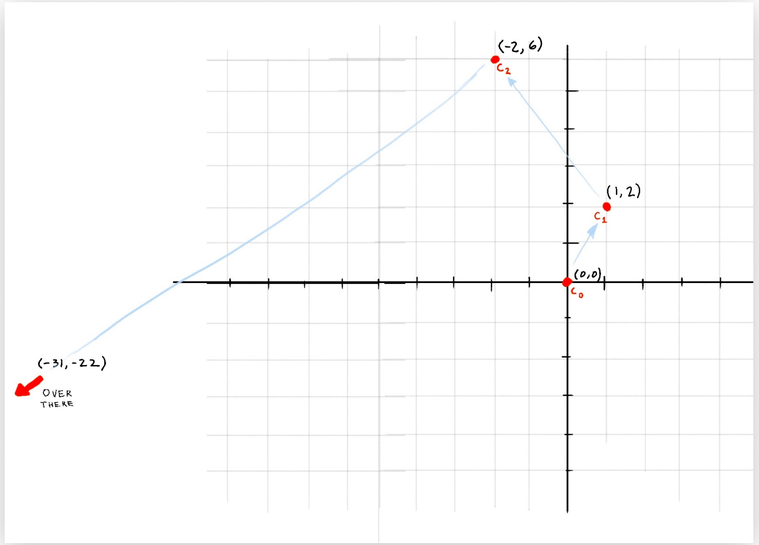
$$= -31 - 22i$$

What did we **just do?**

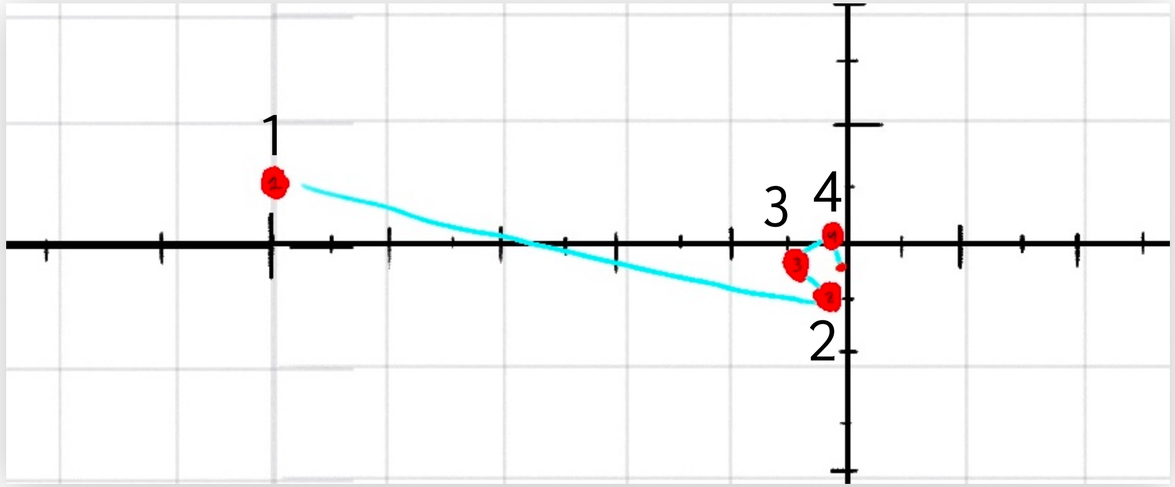
1. Choose a point on the complex plane

(1, 2) = 2 + 1i

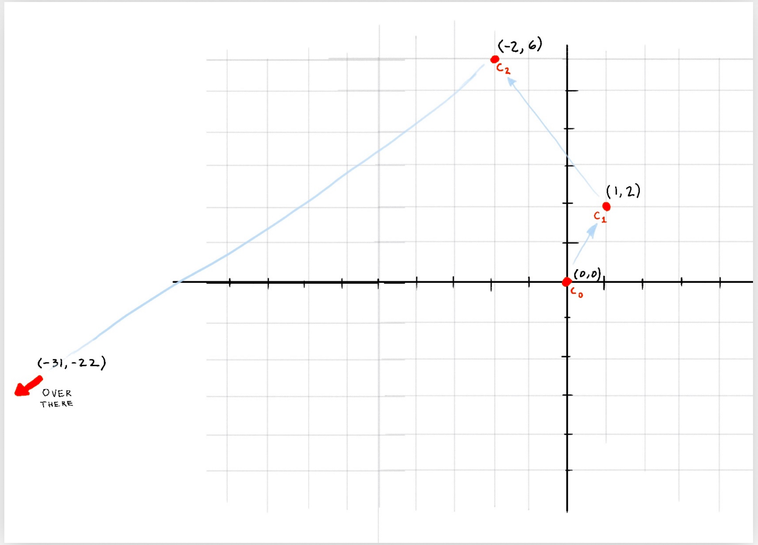
2. Recursively call the Mandelbrot Formula on the point and watch where it ends up



3. Classify that point



YES!



NO!

Shadertoy BETA

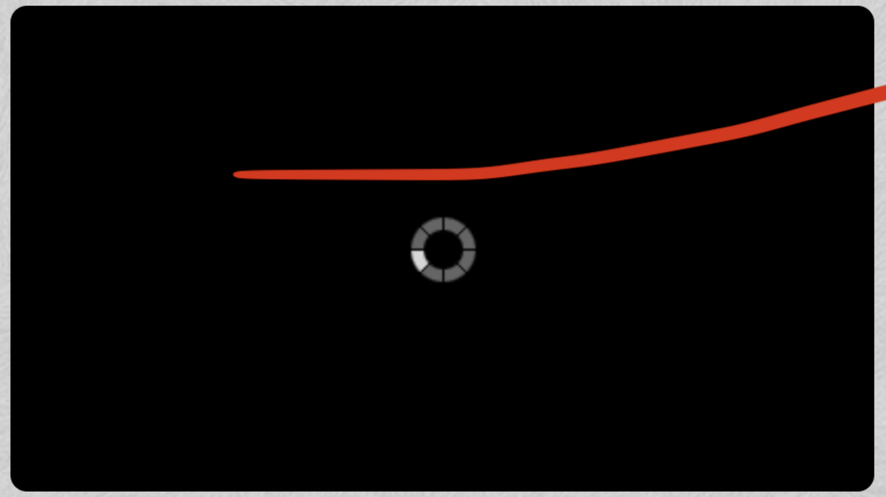
shadertoy.com

Shadertoy

Search...

Welc notmysql | Browse New Logout

Shader of the Week



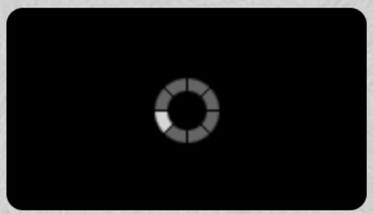
Build and Share your best shaders with the world and get Inspired

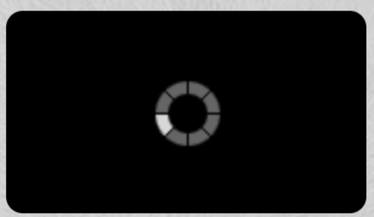
PayPal Donate

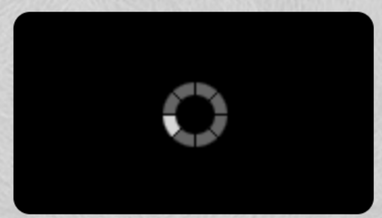
Become a patreon

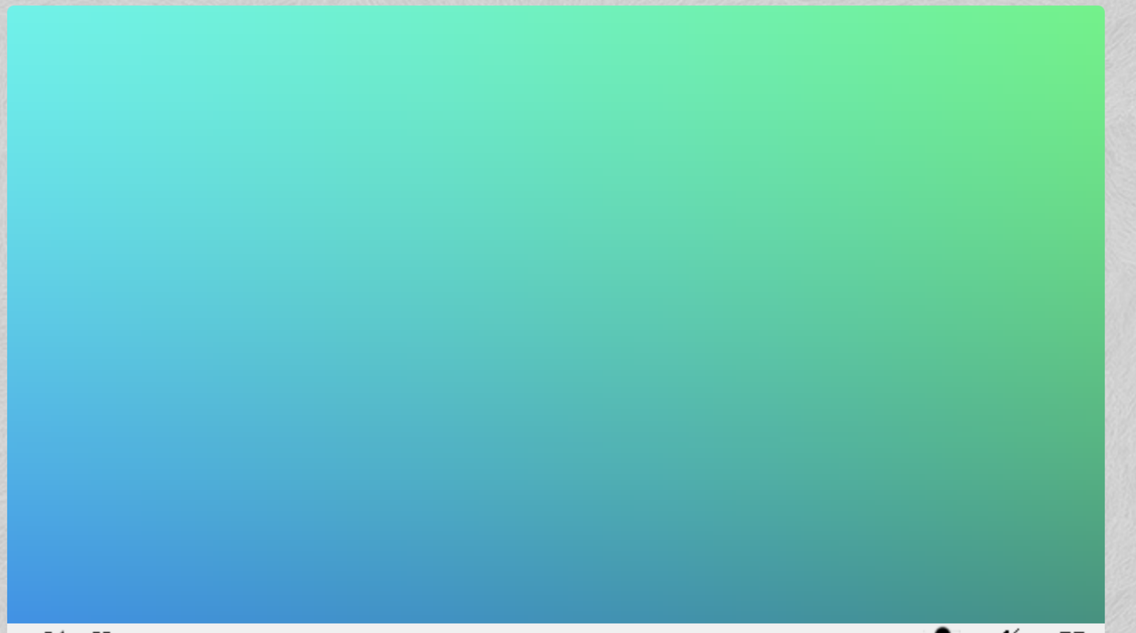
Latest contributions: **"polar doodle"** by **eenzil** 48 minutes ago, **"Linear/XYZ by PuzzleAndy"** by **PuzzleAndy** 54 minutes ago, **"Polar stuff"** by **GBA** 2 hours ago, **"Linear/sRGB by PuzzleAndy"** by **PuzzleAndy** 2 hours ago, **"Vibrance by PuzzleAndy"** by **PuzzleAndy** 3 hours ago

Featured Shaders









78.26 119.8 fps 1280 x 720 REC

Mandelbrot Lesson

fractal

A fractal shader

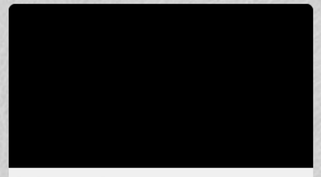
unlisted Submit

Image

Shader Inputs

```
1  void mainImage( out vec4 fragColor, in vec2 fragCoord )
2  {
3      // Normalized pixel coordinates (from 0 to 1)
4      vec2 uv = fragCoord/iResolution.xy;
5
6      // Time varying pixel color
7      vec3 col = 0.5 + 0.5*cos(iTime+uv.xyx+vec3(0,2,4));
8
9      // Output to screen
10     fragColor = vec4(col,1.0);
11 }
```

Compiled in 0.0 secs 158 chars



iChannel0

iChannel1

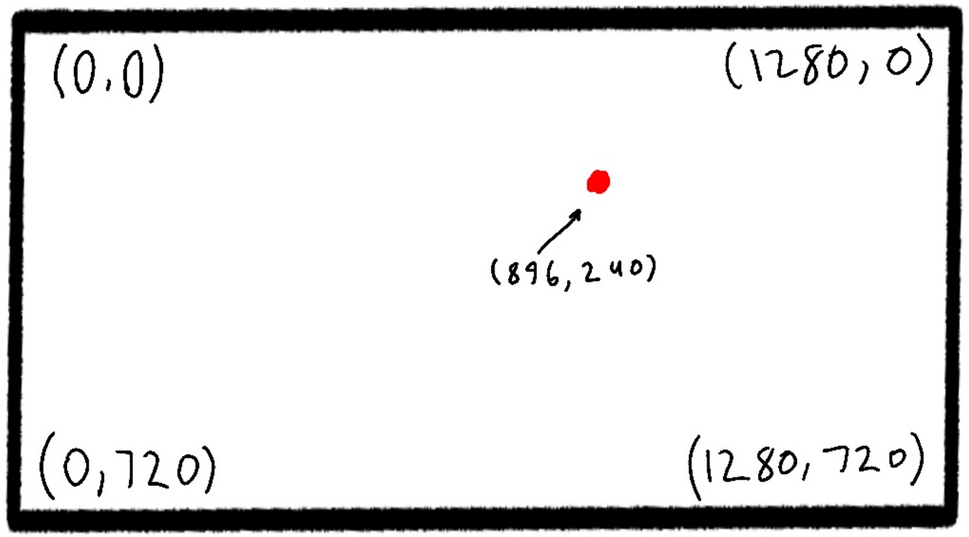
iChannel2

iChannel3

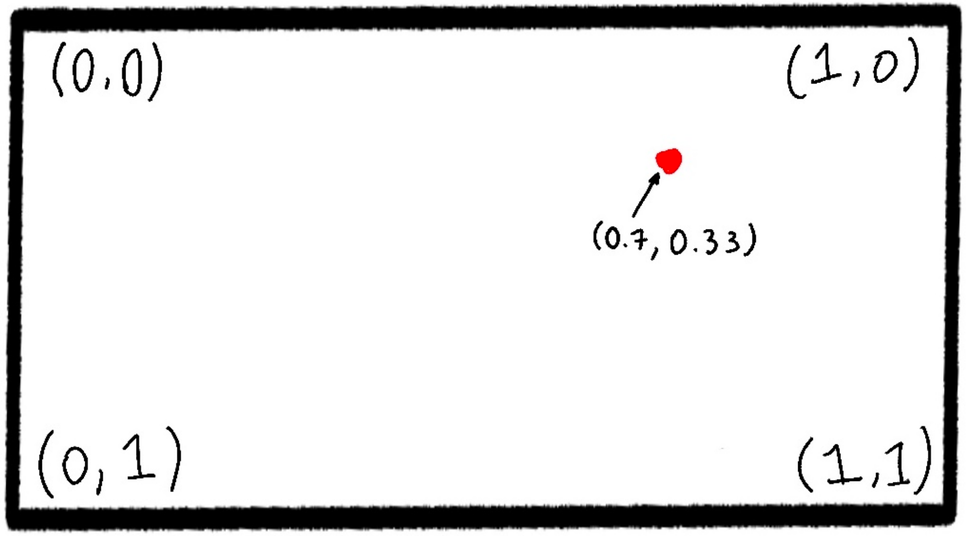
GLSLFractals

```glsl
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.xy;

    // Output to screen
    fragColor = vec4(0.0, 0.0, 0.0, 1.0);
}
```

Texture Coordinates

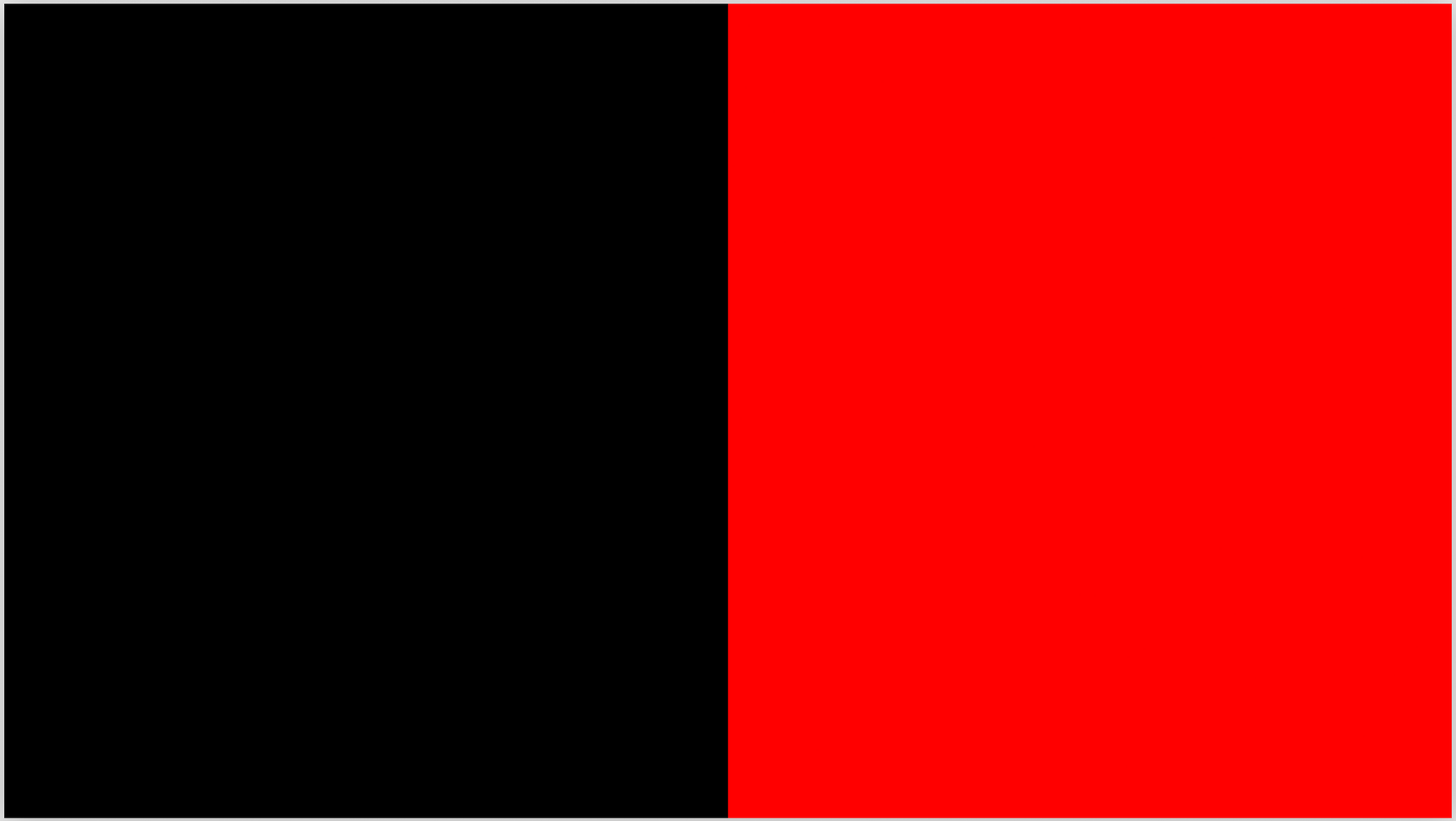


UV Coordinates



GLSLFractals

```
// Vectors example

// 2D Vector
vec2 position = vec2(2.0, 3.0)

position.x // => 2.0
position.y // => 3.0

// OR

position[0] // => 2.0
position[1] // => 3.0


// 3D Vector
vec3 position_3D = vec3(2.0, 5.3, 1.0)

position_3D.x // => 2.0
position_3D.y // => 5.3
position_3D.z // => 1.0
```



GLSLFractals

```glsl
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.xy;

    // Output to screen
    fragColor = vec4(uv.x, 0.0, 0.0, 1.0);
}
```

GLSLFractals

```
fragColor = vec4(uv.x, 0.0, 0.0, 1.0);
```

GLSLFractals

```glsl
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 c = fragCoord/iResolution.xy;

    // Output to screen
    fragColor = vec4(0.0, 0.0, 0.0, 1.0);
}
```

GLSLFractals

```
// Squares a given complex number
vec2 cSquare( in vec2 z )
{
    return vec2(
            z.x * z.x - z.y * z.y,
            2.0 * z.x * z.y
        );
}
```

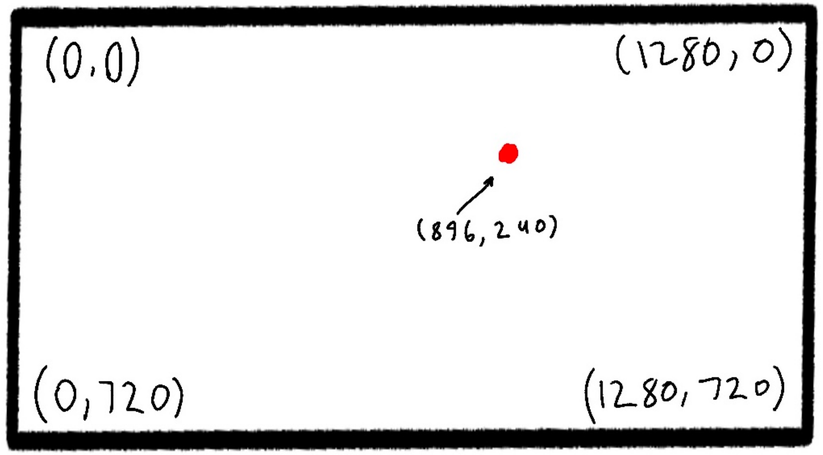
GLSLFractals

```
for(int i = 0; i < maxIterations; i = i + 1)
{
    // Z(n+1) = Z(n)^2 + C
}
```

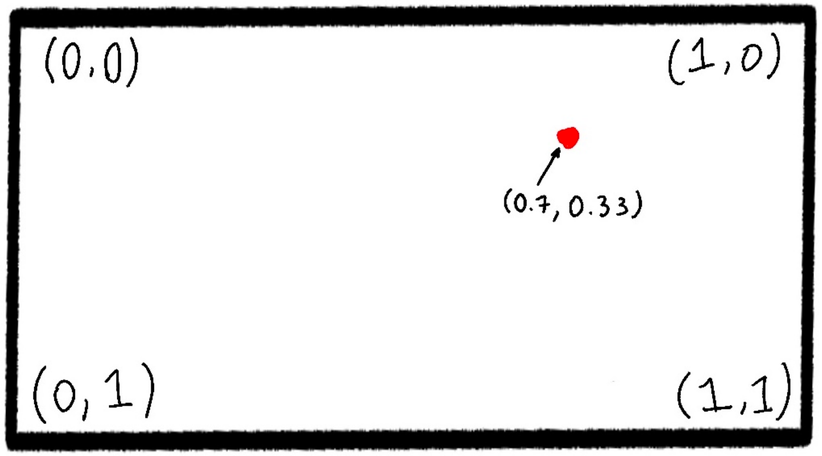
Render Issues:

1: **Framing**
2: **Stretching**
3: **Colors**

Texture Coordinates



UV Coordinates

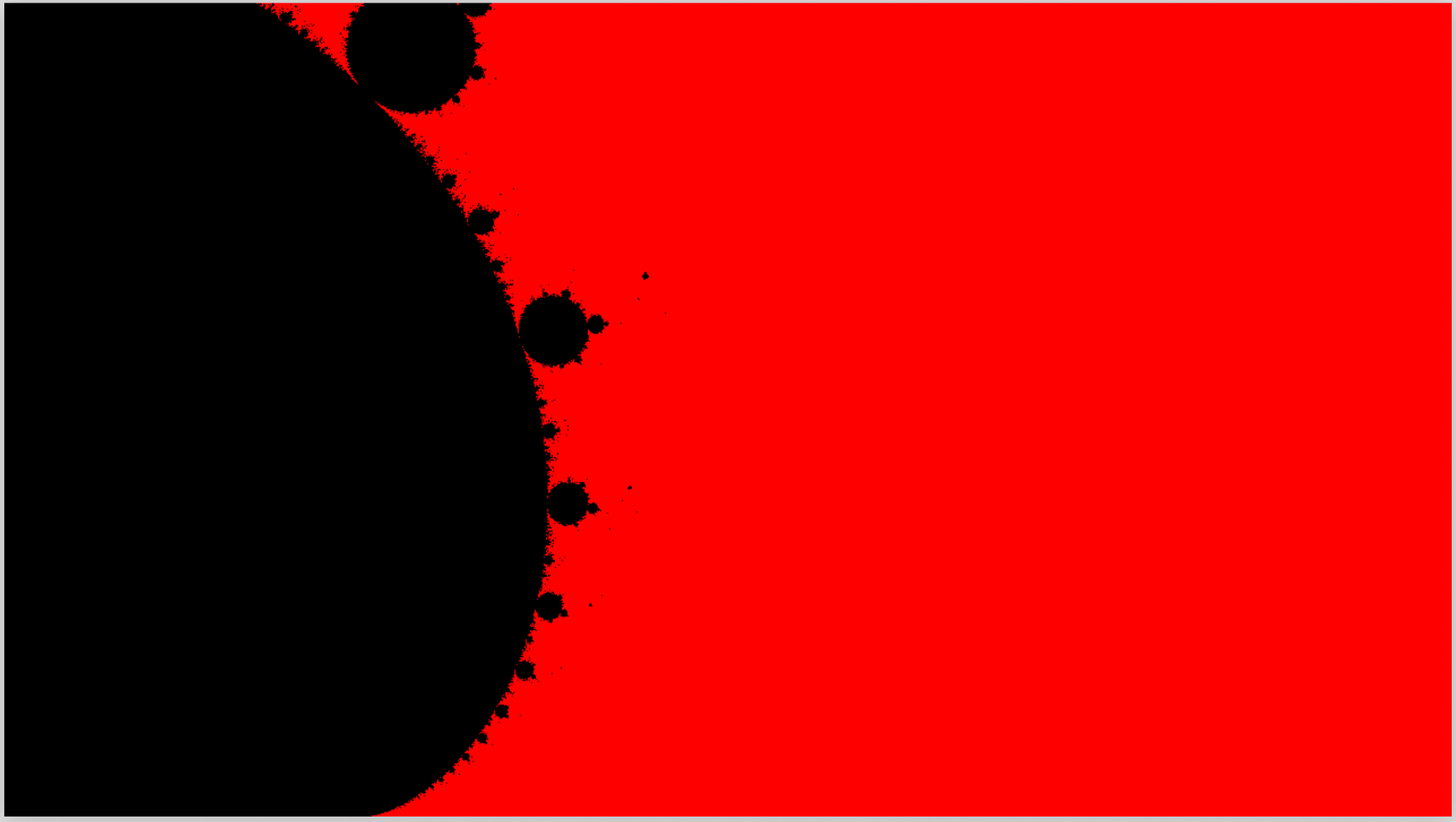


GLSLFractals

```
// Normalized pixel coordinates (from 0 to 1)
vec2 c = fragCoord/iResolution.xy;
```

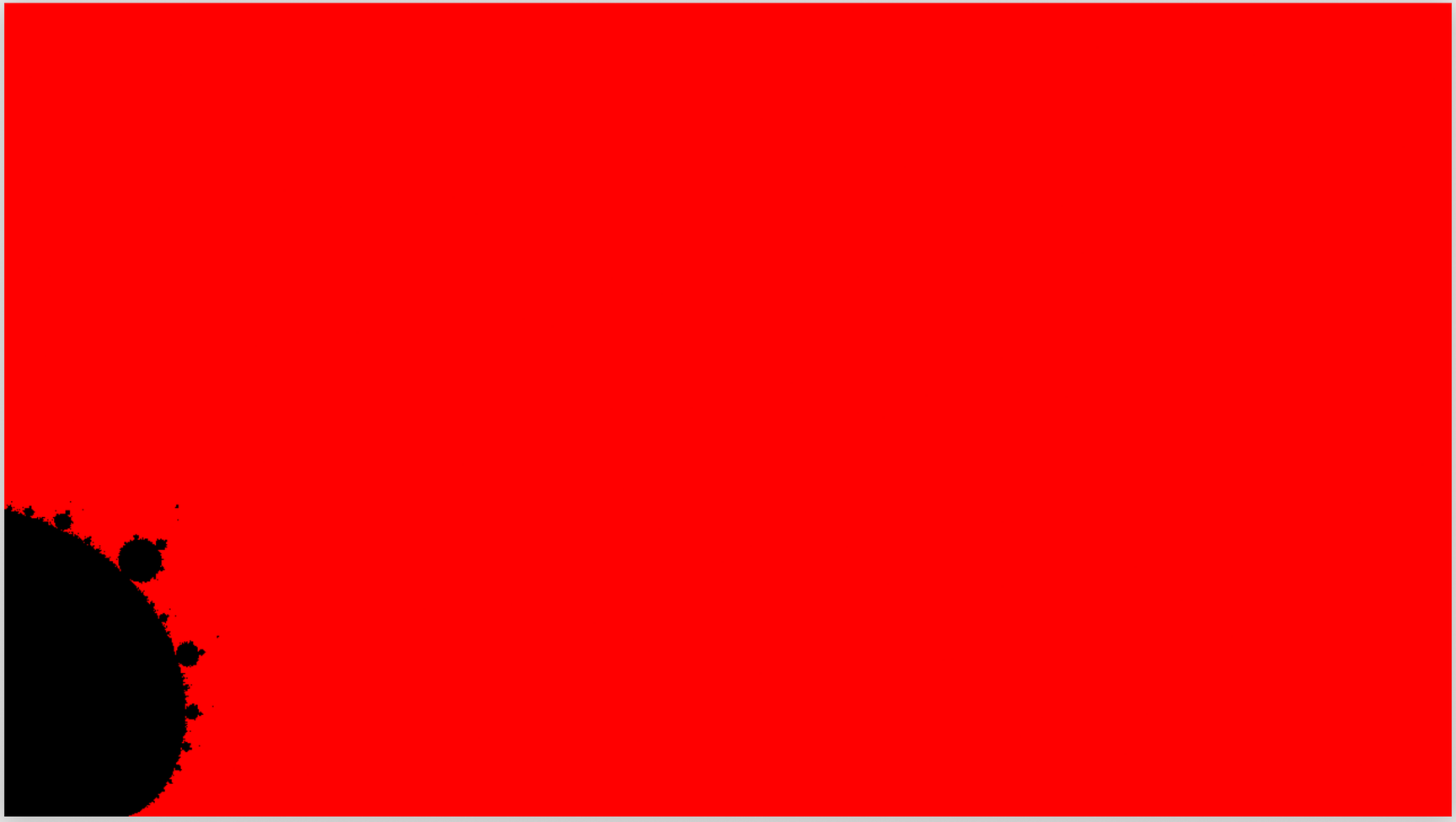
GLSLFractals

```
// Normalized pixel coordinates (from 0 to 1)
vec2 c = fragCoord/iResolution.x;
```



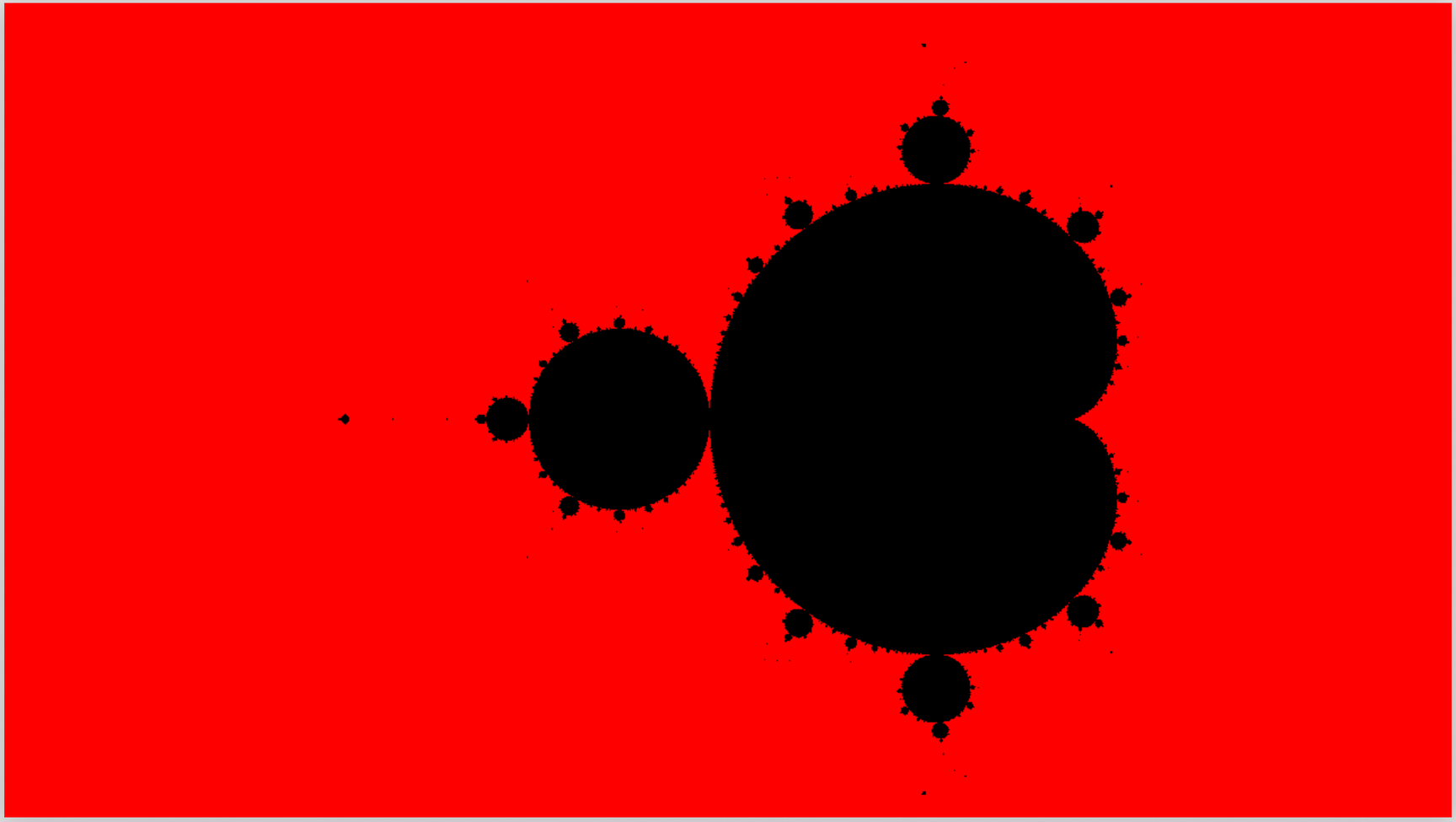
GLSLFractals

```
// Normalized pixel coordinates (from 0 to 1)
vec2 c = fragCoord/iResolution.x;

// Scale [lower is more zoomed in]
float scale = 4.0;

c *= scale;
```



GLSLFractals

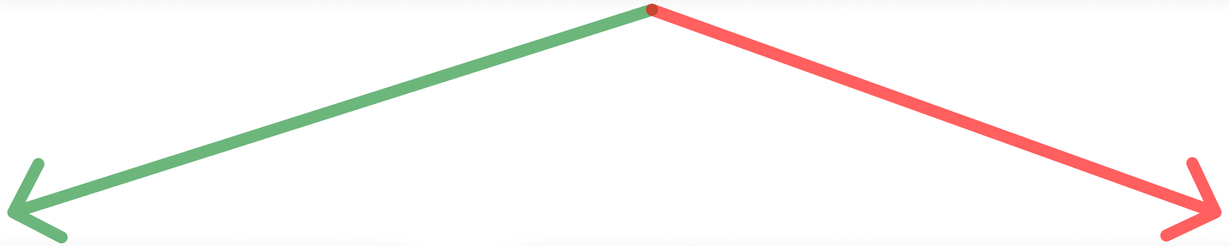
```
// Normalized pixel coordinates (from 0 to 1)
vec2 c = fragCoord/iResolution.x;

// Scale [lower is more zoomed in]
float scale = 4.0;

c *= scale;

// Move set into view
c += vec2(-2.7,-1.1);
```



COLORS!!!

after maxIterations

Does this point escape to infinity ?



NO: Color it **black**

part of Mandelbrot set

NO: Color it based on **how many iterations is took to escape**

not part of Mandelbrot set

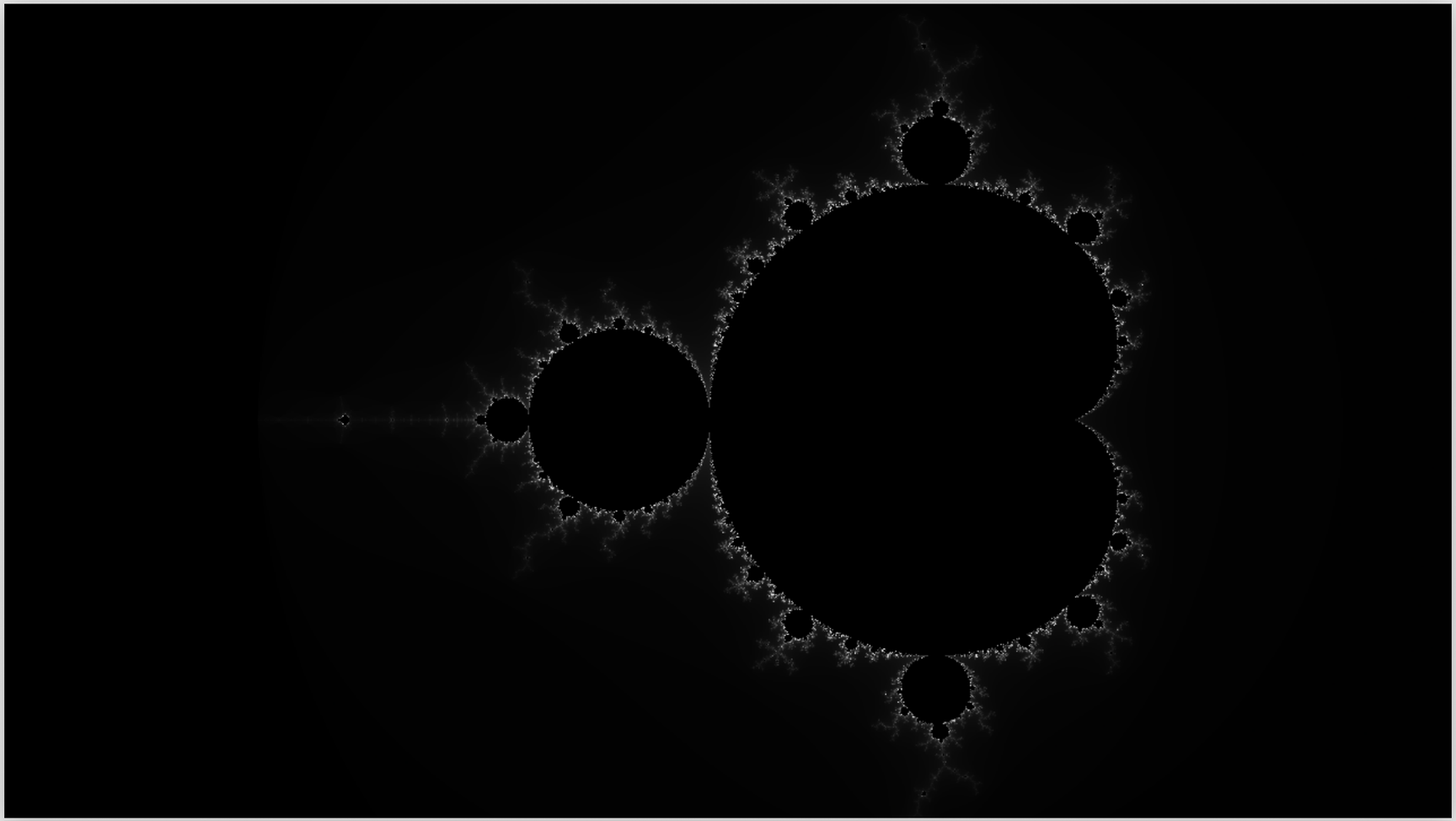
Vector constructors 🤩

GLSLFractals

```
a = vec3(1.0) // => vec3(1.0, 1.0, 1.0)
b = vec4(2.4) // => vec4(2.4, 2.4, 2.4, 2.4)

vec4(a, 5.0) // => vec4(1.0, 1.0, 1.0, 5.0)
```

GLSLFractals

```
if(dot(Z, Z) > 4.0) {
    // Normalize i for brightness
    float brightness = float(i) / float(maxIterations);

    fragColor = vec4(vec3(brightness), 1.0);
    break;
```



Current method:

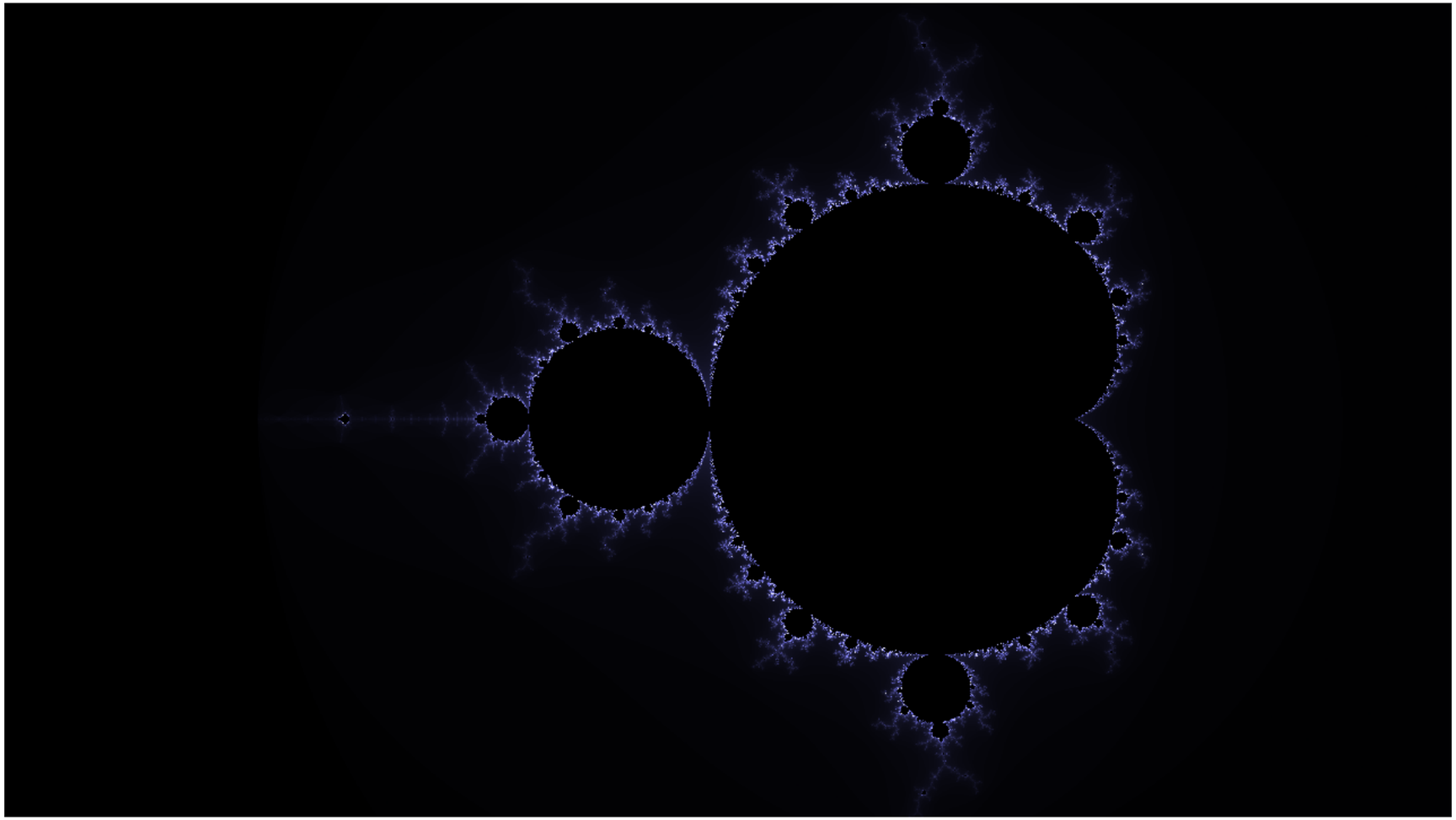
New weighted method:

GLSLFractals

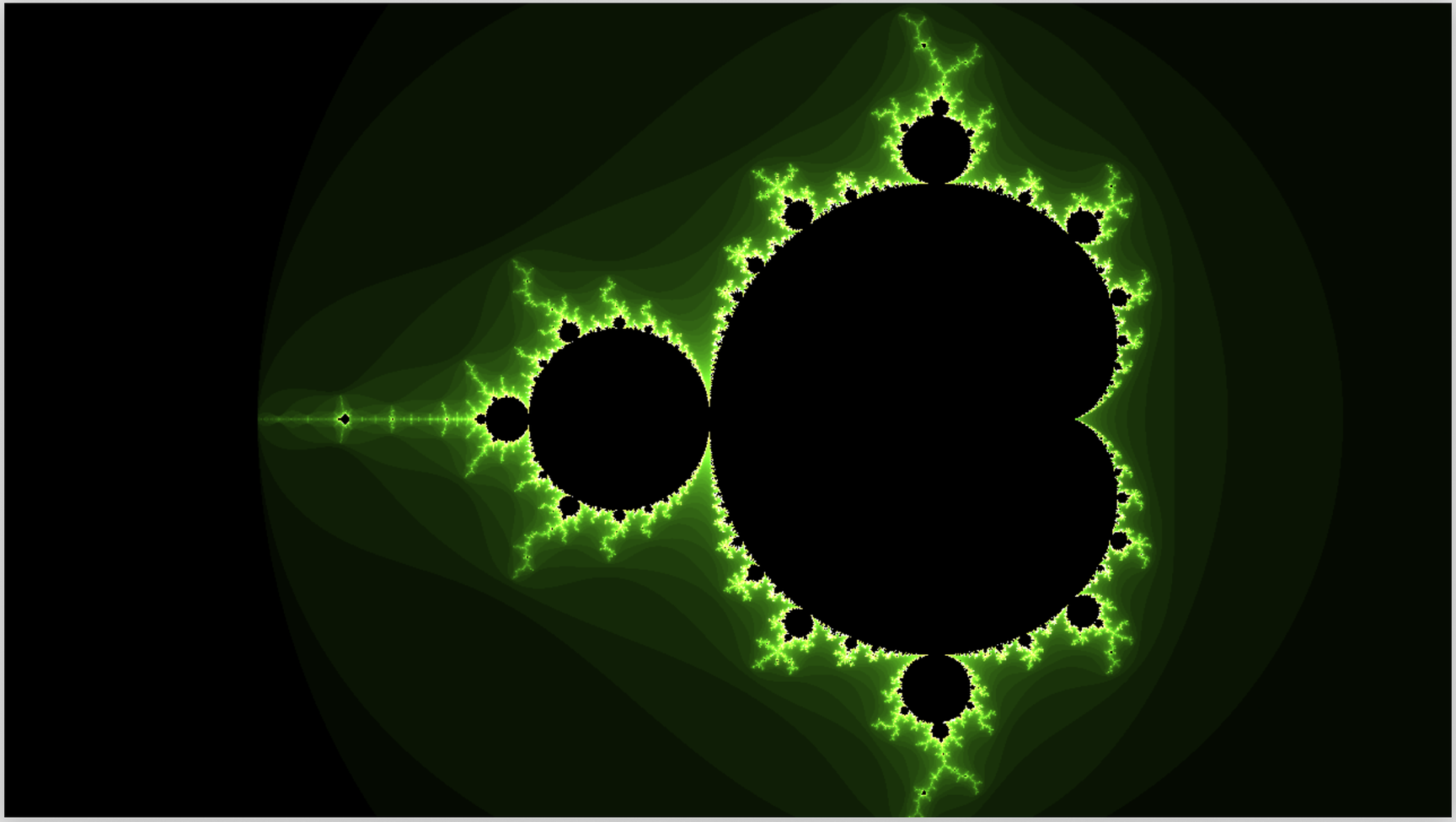
```
// multipliers for ( R    G    B ) values
vec3 weights = vec3(1.0,1.0,1.0);

fragColor = vec4(vec3(brightness) * weights, 1.0);
break;
```

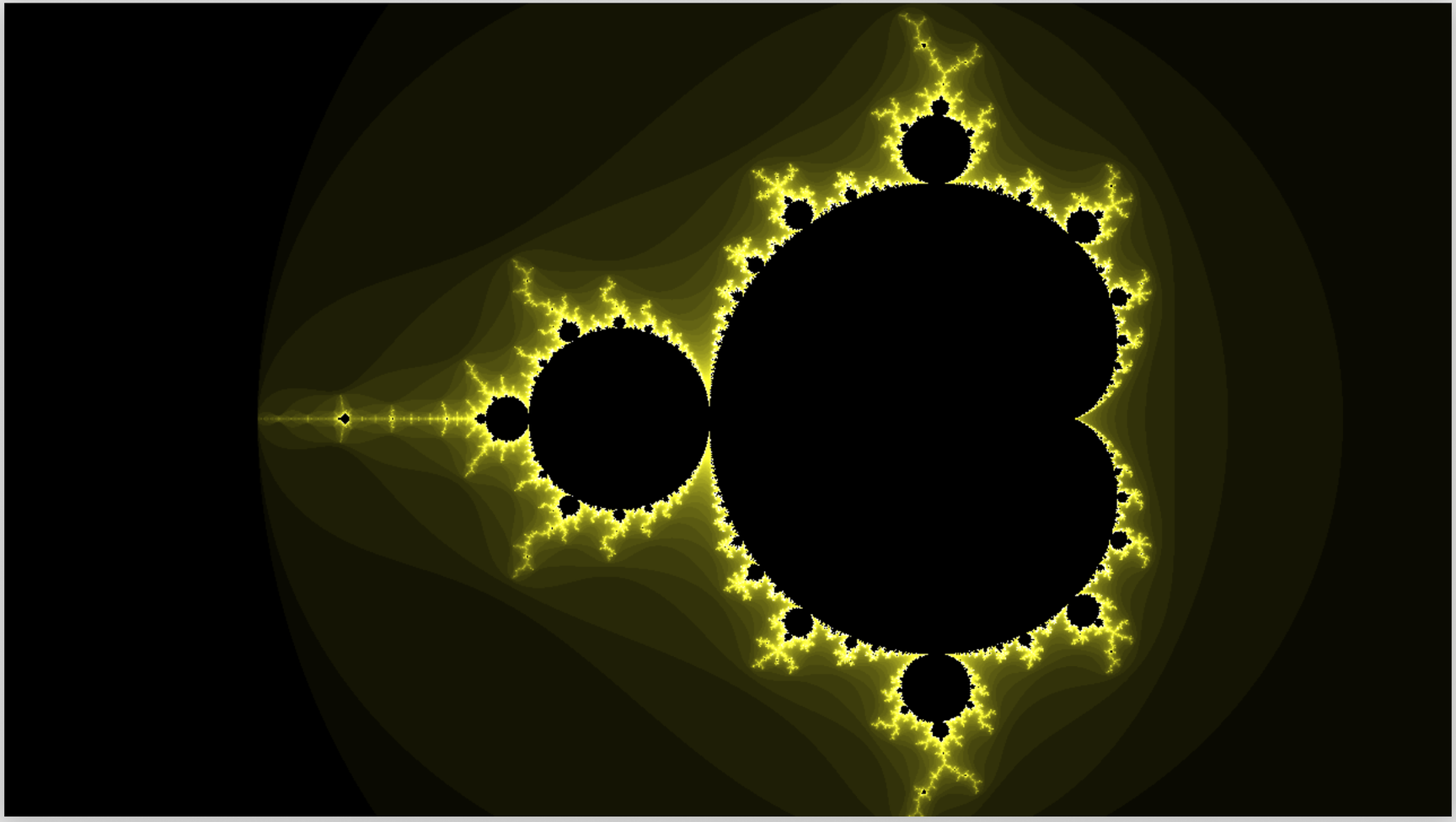
GLSLFractals

```
vec3 weights = vec3(1.0,1.0,2.0);
                    R   G   B
```



COOL! Now try experimenting with bigger / different **weight values**





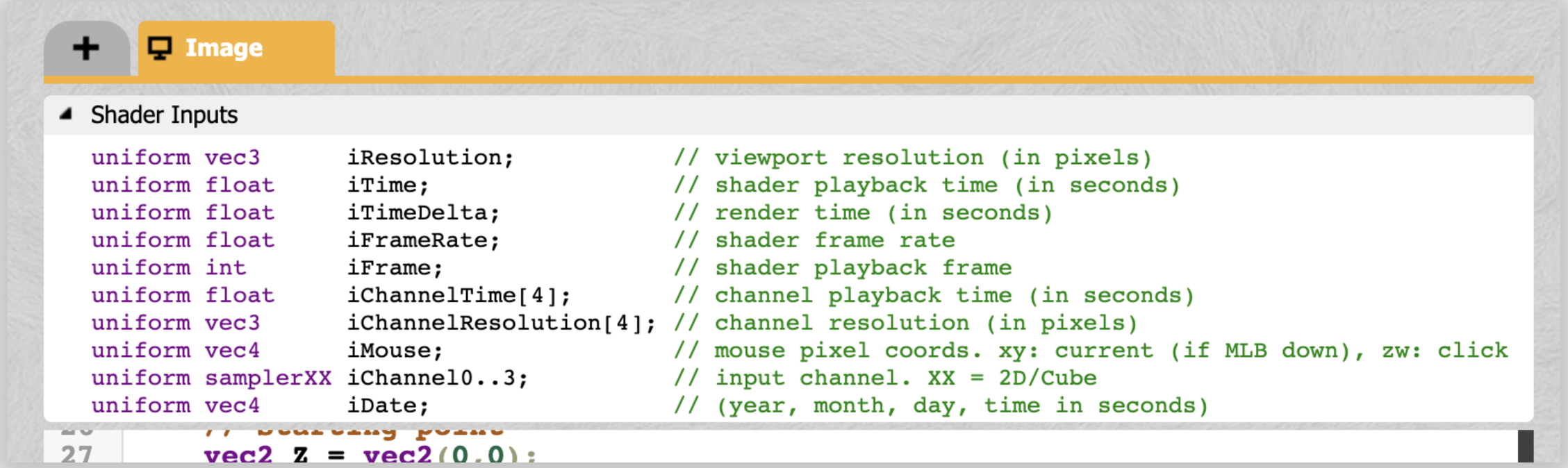


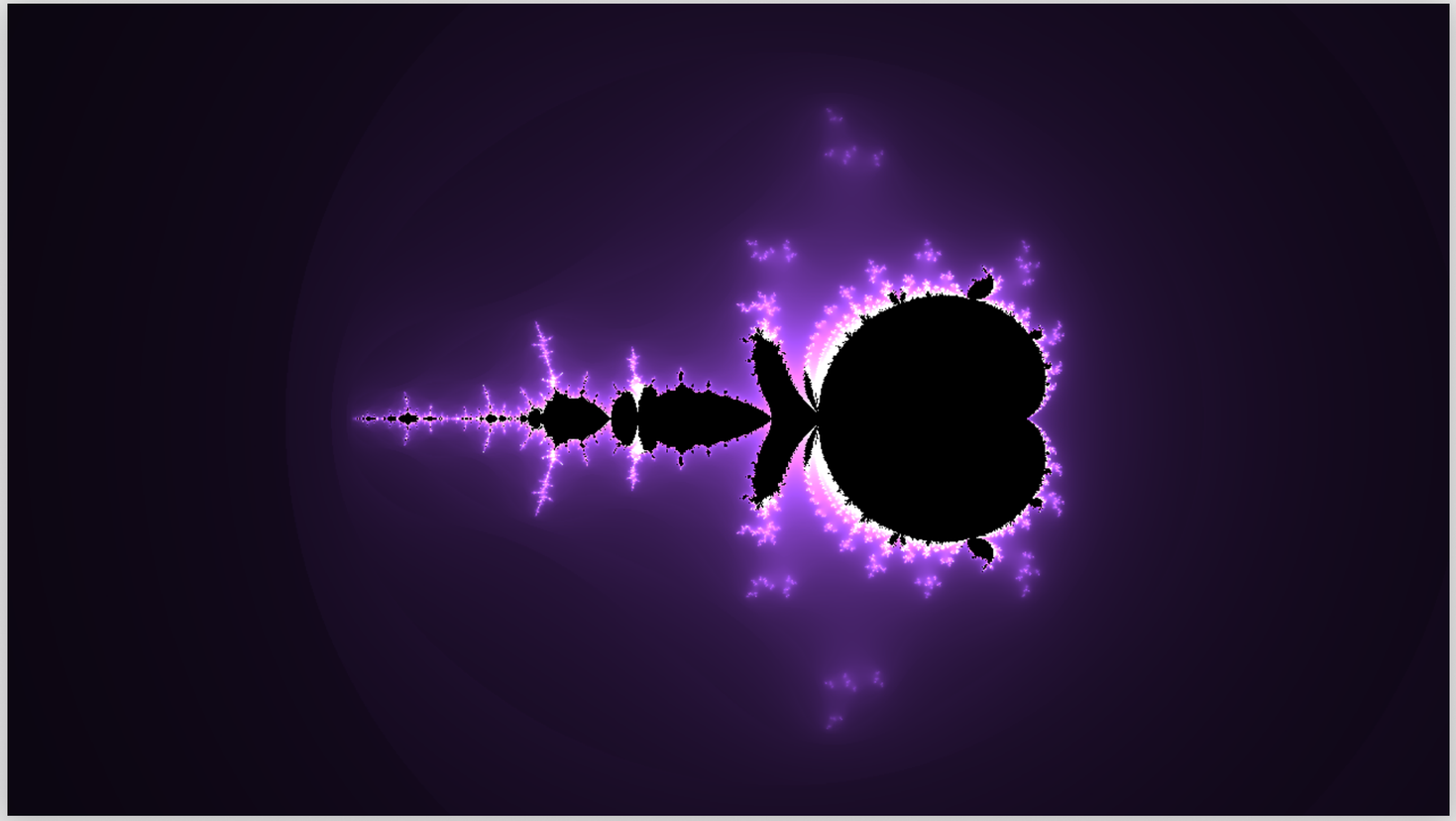
Image
Shader Inputs
uniform vec3 iResolution; // viewport resolution (in pixels)
uniform float iTime; // shader playback time (in seconds)
uniform float iTimeDelta; // render time (in seconds)
uniform float iFrameRate; // shader frame rate
uniform int iFrame; // shader playback frame
uniform float iChannelTime[4]; // channel playback time (in seconds)
uniform vec3 iChannelResolution[4]; // channel resolution (in pixels)
uniform vec4 iMouse; // mouse pixel coords. xy: current (if MLB down), zw: click
uniform samplerXX iChannel0..3; // input channel. XX = 2D/Cube
uniform vec4 iDate; // (year, month, day, time in seconds)
27
vec2 z = vec2(0.0);

GLSLFractals

```
// multipliers for ( R    G    B ) values
vec3 weights = vec3(sin(iTime) * 2.0 + 5.0, cos(iTime) * 3.0 + 5.0,
sin(iTime) * 10.0);

fragColor = vec4(vec3(brightness) * weights, 1.0);
```

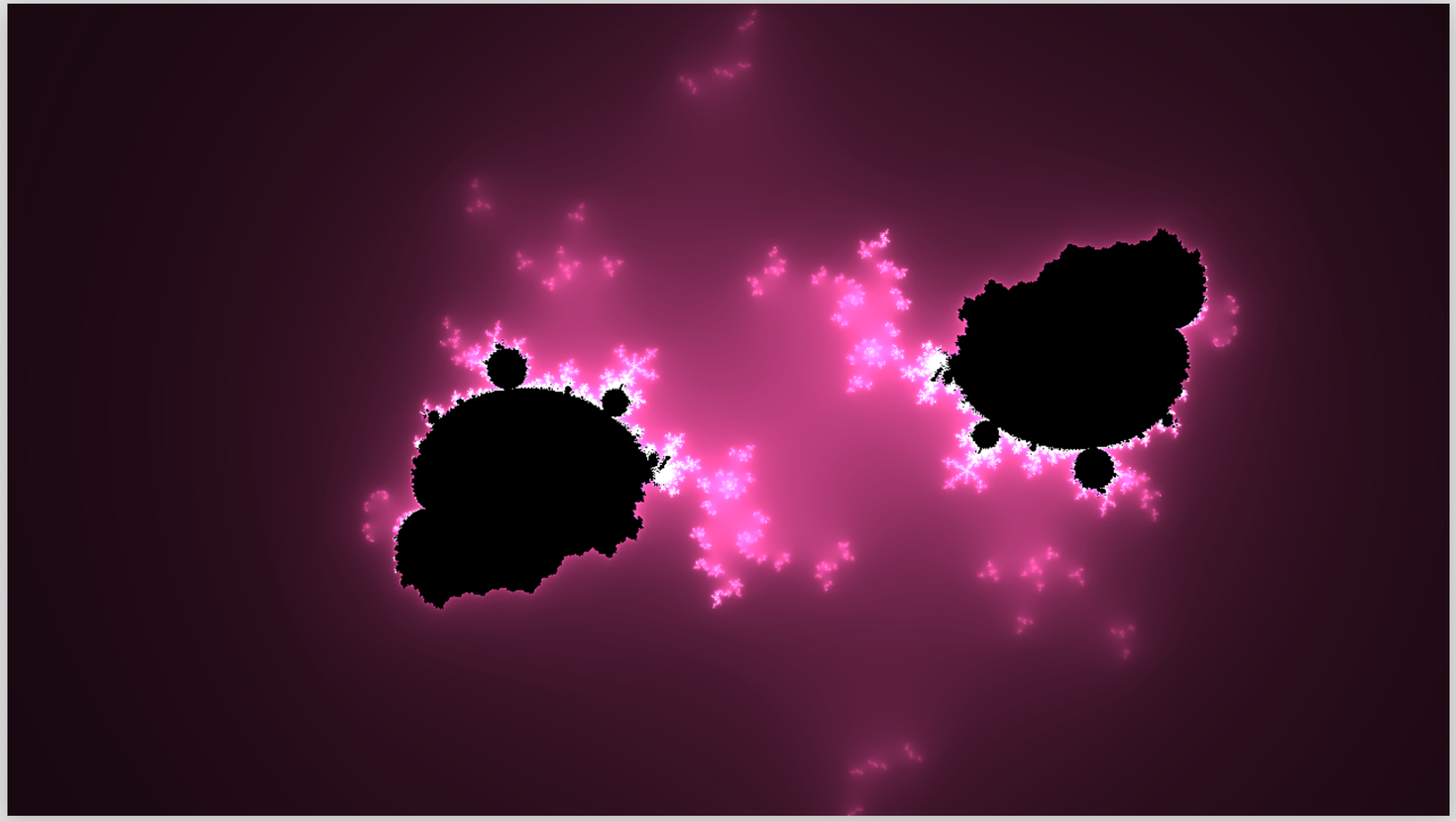
GLSLFractals

```
// Z(n+1) = Z(n)^2 + C
vec2 oldZ = Z;
Z = cSquare(oldZ) + c;
```



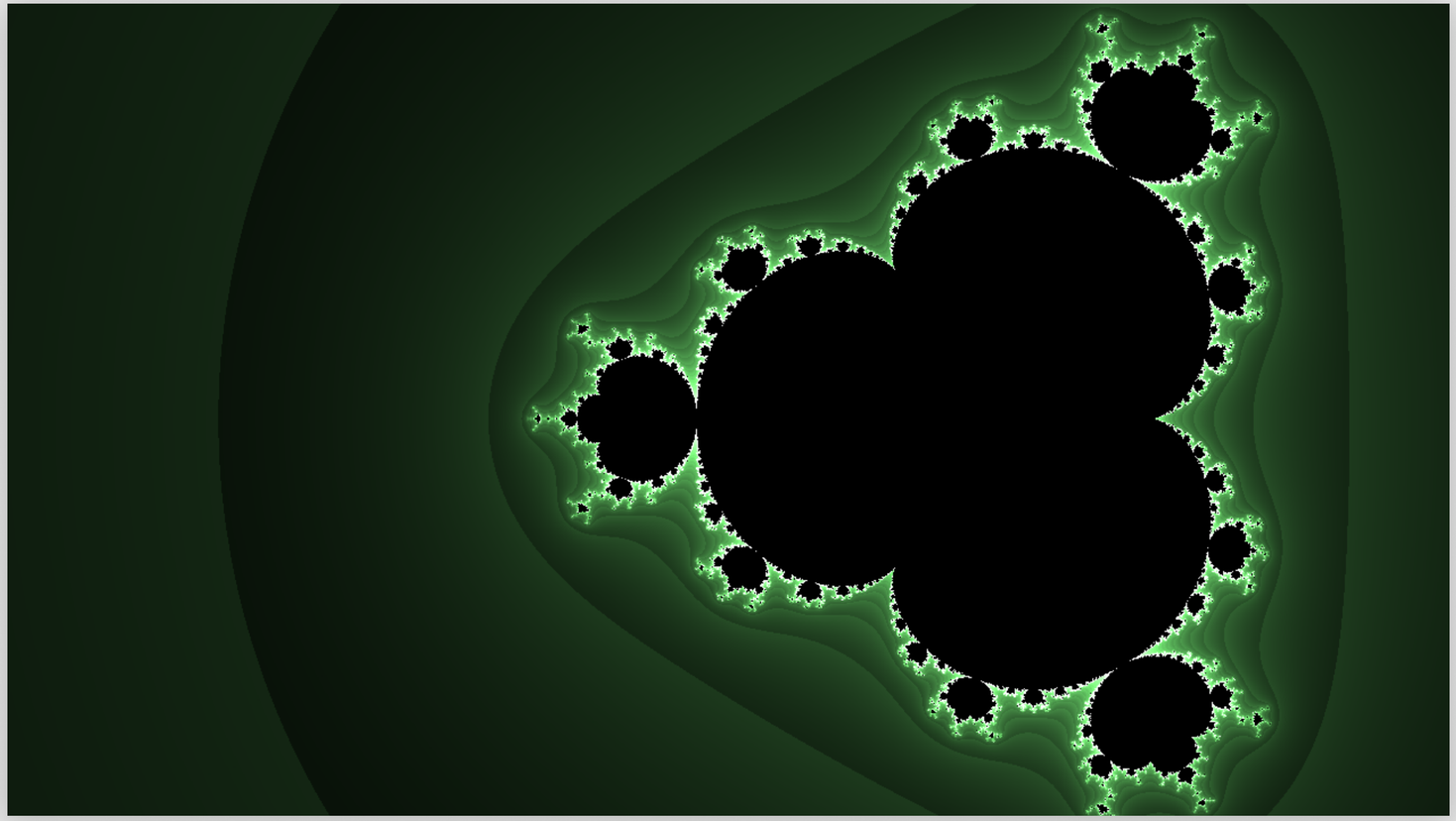
GLSLFractals

```
vec2 oldZ = Z;
Z = cSquare(oldZ + c) + c + cSquare(c * sin(iTime));
```



GLSLFractals

```
vec2 oldZ = Z;
Z = cSquare(c) + c + cSquare(oldZ - sin(iTime));
```



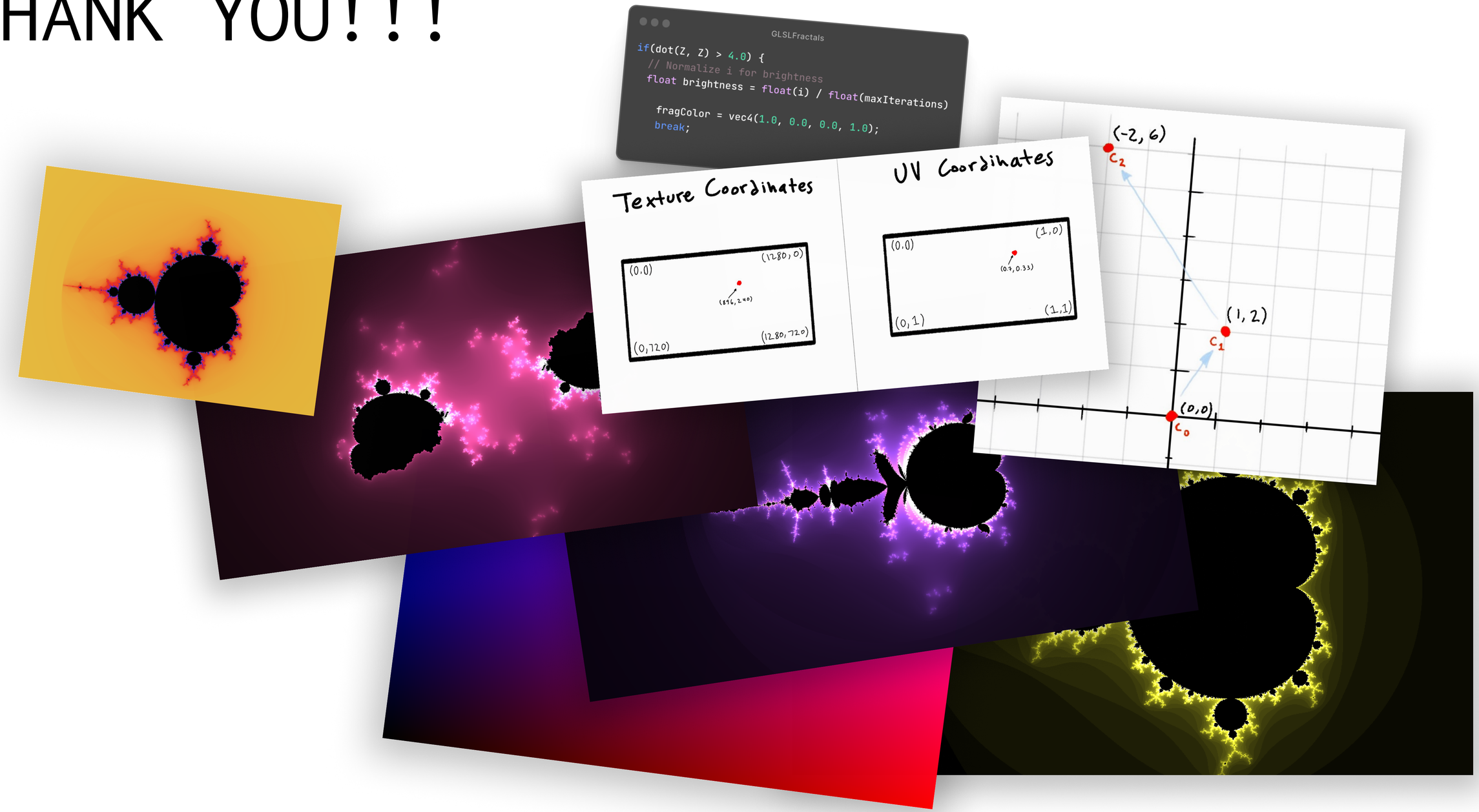
GLSLFractals

```
vec2 oldZ = Z;
Z = cSquare(cSquare(oldZ) + c);
```

Where to go from here

1. Mess with cSquare()
2. Increase / Decrease max iterations and escape radius
3. Think of other color formulas
4. Use other builtin variables(iMouse)
5. Other colors for points inside Mandelbrot set

THANK YOU!!!


GLSLFractals
if(dot(Z, Z) > 4.0) {
// Normalize i for brightness
float brightness = float(i) / float(maxIterations)
fragColor = vec4(1.0, 0.0, 0.0, 1.0);
break;
Texture Coordinates
(0,0)
(1280,0)
(896,240)
(0,720)
(1280,720)
UV Coordinates
(0,0)
(1,0)
(0.7,0.33)
(0,1)
(1,1)
(-2,6)
C2
(1,2)
C1
(0,0)
C0