

# Hands On: Mandelbrot

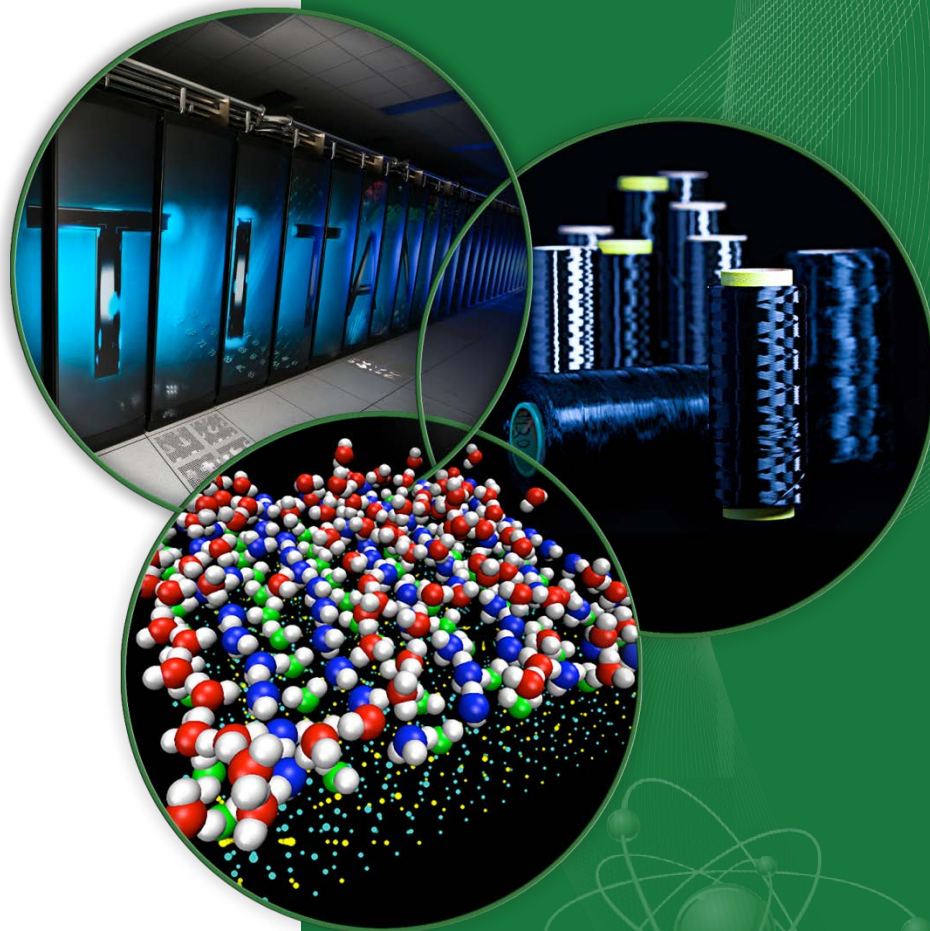
Verónica G. Vergara Larrea

Adam B. Simpson

Tom Papatheodore

**CSGF 2017**

**July 26, 2017**



# Definition

The set of values of  $c$  in the complex plane for which  $f_n$  does not diverge for arbitrarily large values of  $n$

$$f_0 = c$$

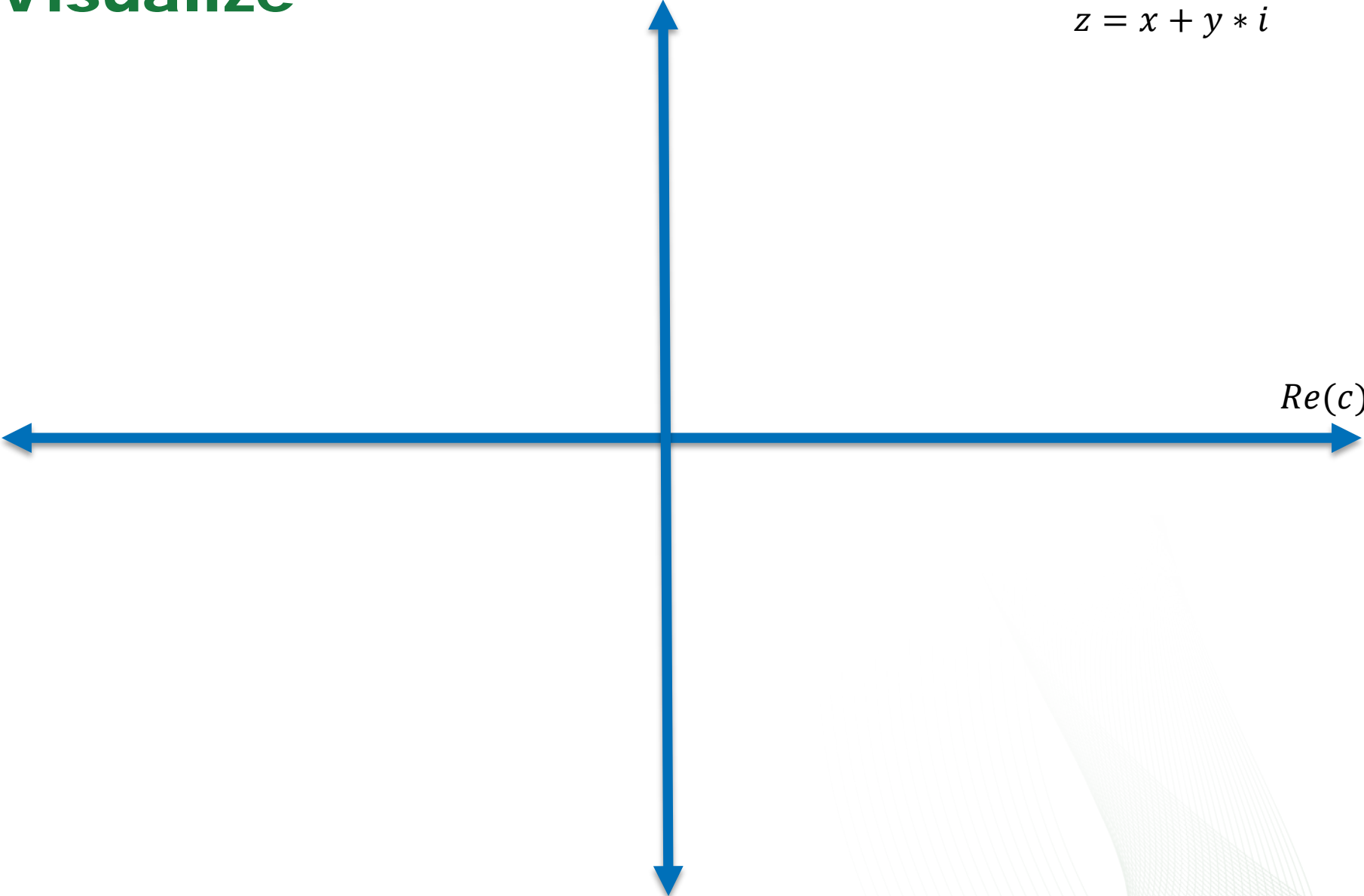
$$f_{n+1} = f_n^2 + c$$

# Visualize

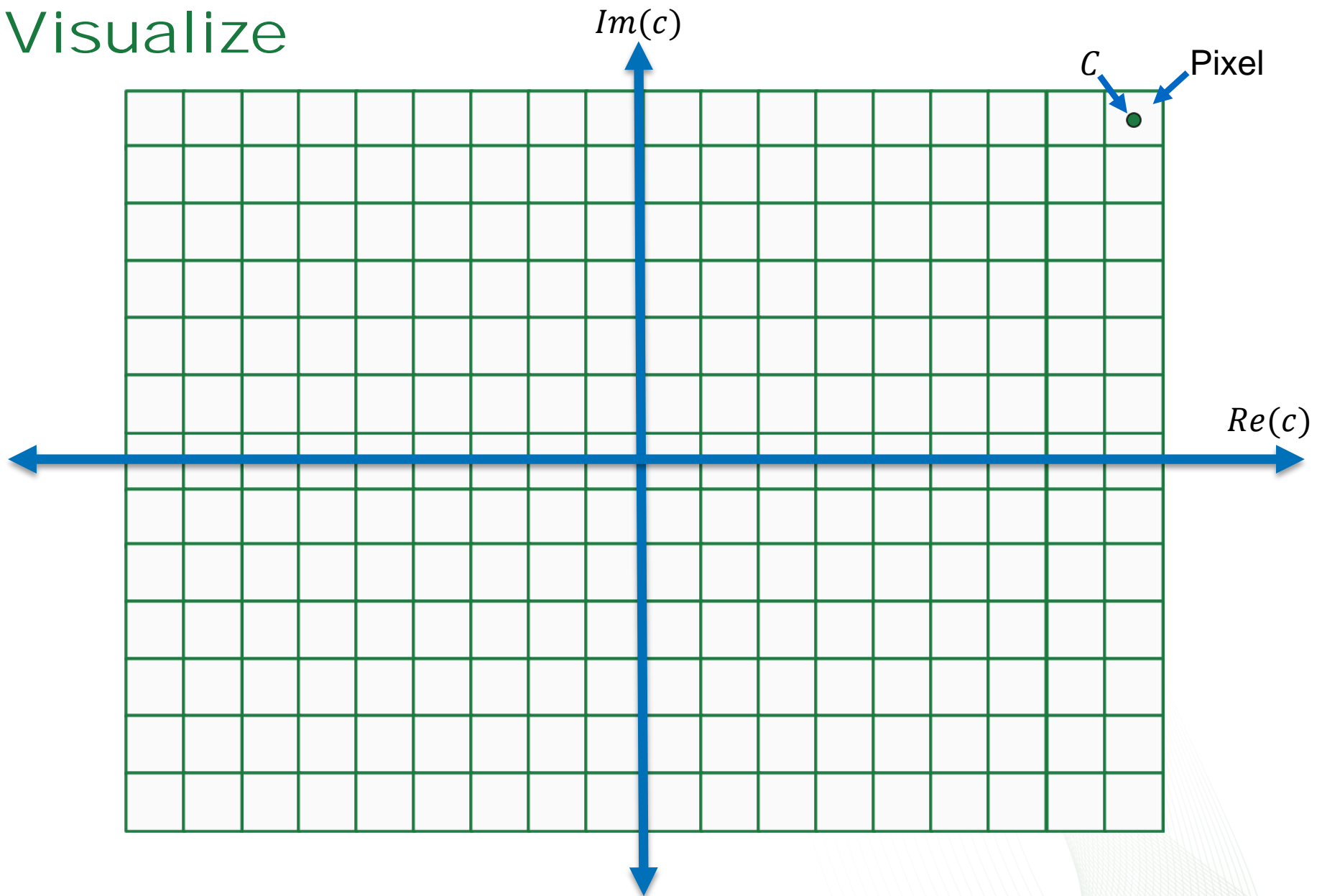
$Im(c)$

$$z = x + y * i$$

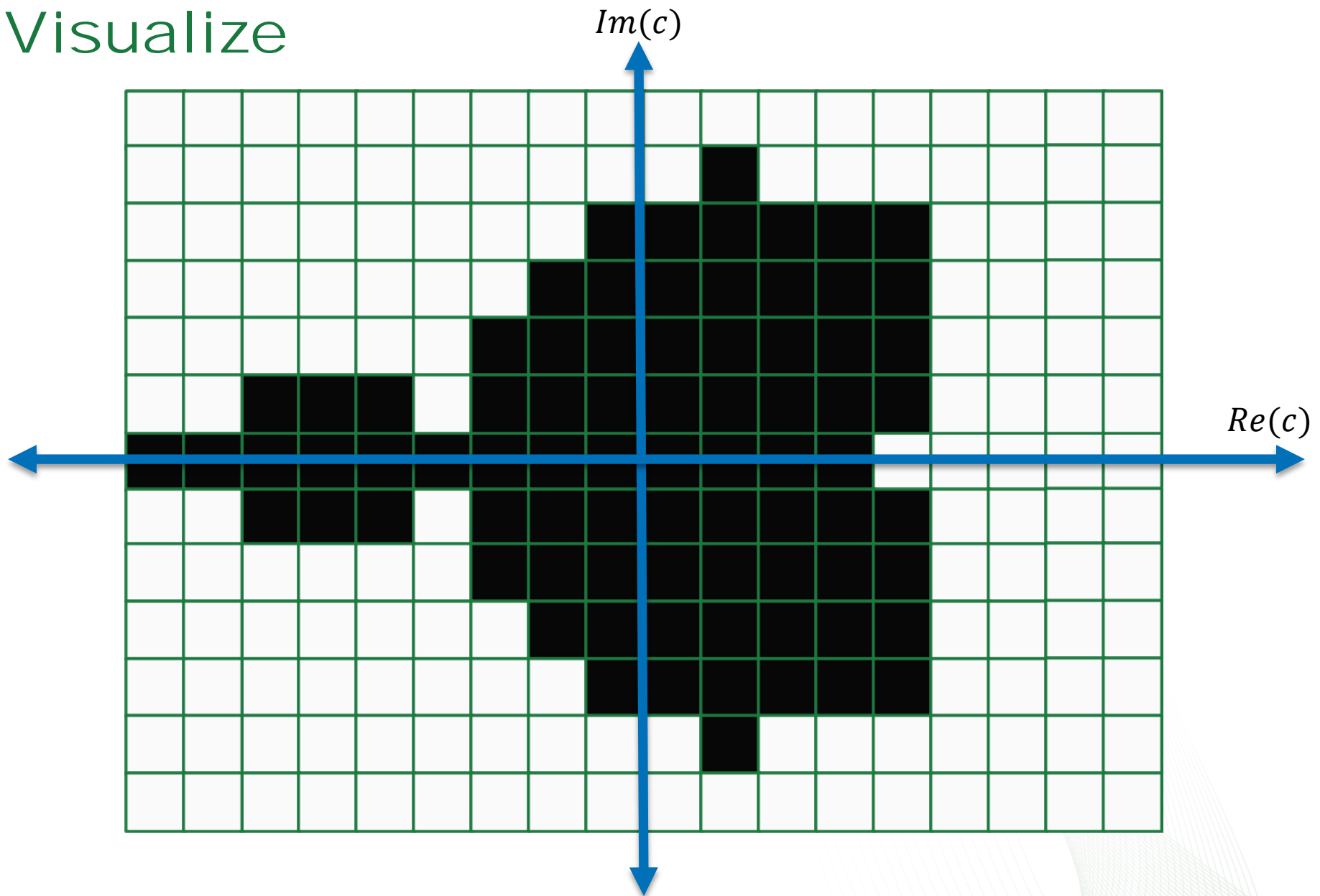
$Re(c)$



# Visualize



# Visualize



# App: Overview

- Define image and grid dimensions
- Allocate memory
- Iterate over all pixels and determine color
- Write pixels to image file

# App: Image and grid

## Recommended image/grid dimensions

$$\begin{aligned}center_x &= -0.75 \\center_y &= 0.00\end{aligned}$$

$$\begin{aligned}length_x &= 2.75 \\length_y &= 2.0\end{aligned}$$

$$min_x = center_x - \frac{length_x}{2.0}$$

$$max_y = center_y + \frac{length_y}{2.0}$$

$$pixel\_count_x = 8192$$

$$pixel\_size = length_x / pixel\_count_x$$

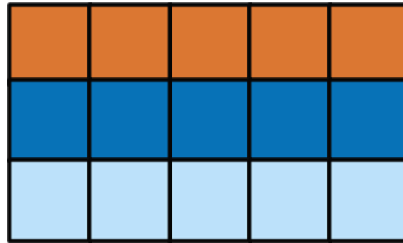
$$pixel\_count_y = \frac{length_y}{pixel\_size}$$

# App: Image and grid

## Allocating pixels

Recommended: **one** contiguous chunk of memory to store pixels

Image:



$$\begin{aligned} pixel\_count_x &= 5 \\ pixel\_count_y &= 3 \end{aligned}$$

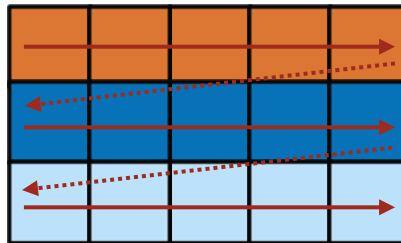


# App: Image and grid

## Allocating pixels

Recommended: **one** contiguous chunk of memory to store pixels

Image:



$$\begin{aligned} \text{pixel\_count}_x &= 5 \\ \text{pixel\_count}_y &= 3 \end{aligned}$$

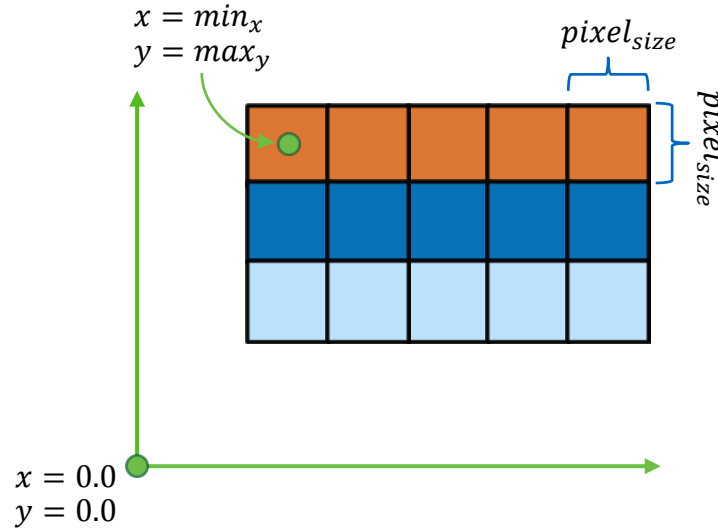
Memory:



## Row Major format

# App: Image and grid

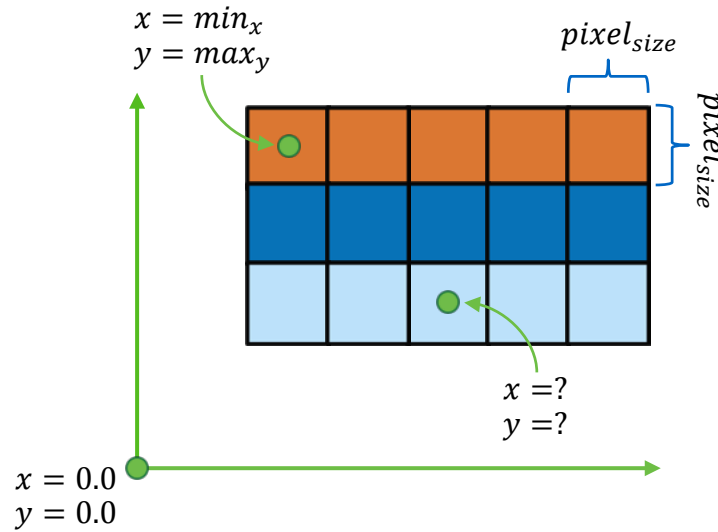
## Iterating pixels



$$pixel\_count_x = 5$$
$$pixel\_count_y = 3$$

# App: Image and grid

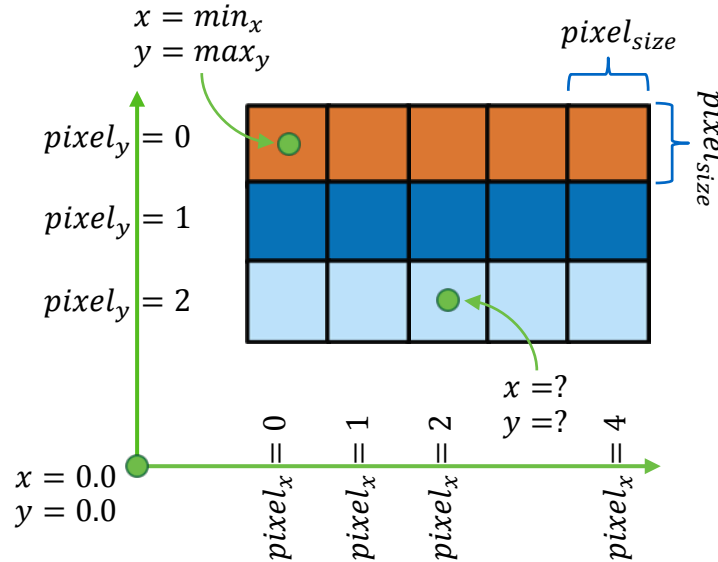
## Iterating pixels



$$pixel\_count_x = 5$$
$$pixel\_count_y = 3$$

# App: Image and grid

## Iterating pixels

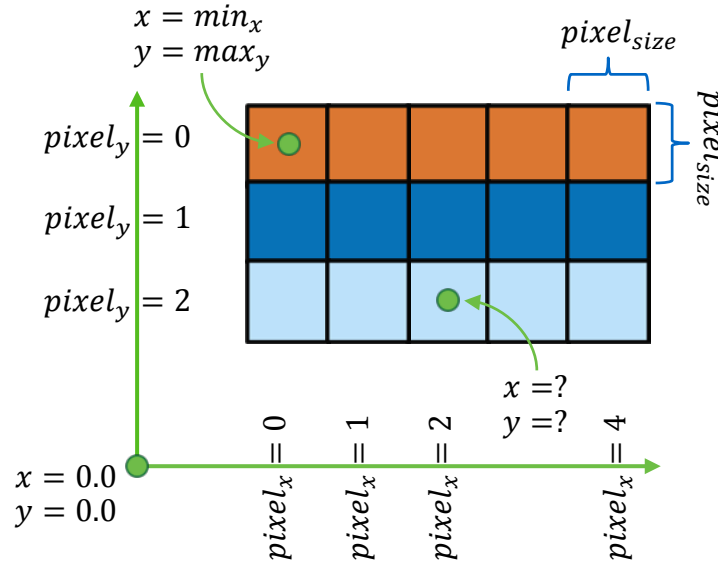


$pixel\_count_x = 5$   
 $pixel\_count_y = 3$

*for  $pixel_y$  in  $pixel\_count_y$*   
*for  $pixel_x$  in  $pixel\_count_x$*

# App: Image and grid

## Iterating pixels

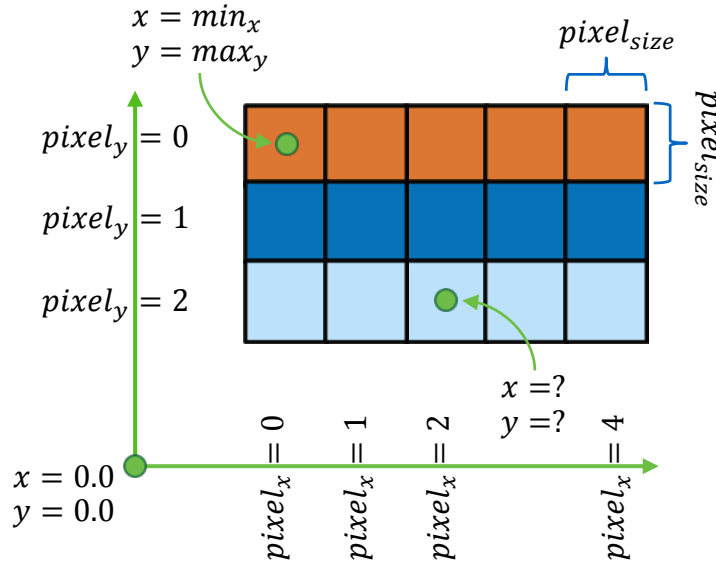


$pixel\_count_x = 5$   
 $pixel\_count_y = 3$

```
for  $pixel_y$  in  $pixel\_count_y$ 
  for  $pixel_x$  in  $pixel\_count_x$ 
     $x = \dots$ 
     $y = \dots$ 
     $i = \dots$ 
     $pixels[i] = Mandelbrot(x, y)$ 
```

# App: Image and grid

## Iterating pixels

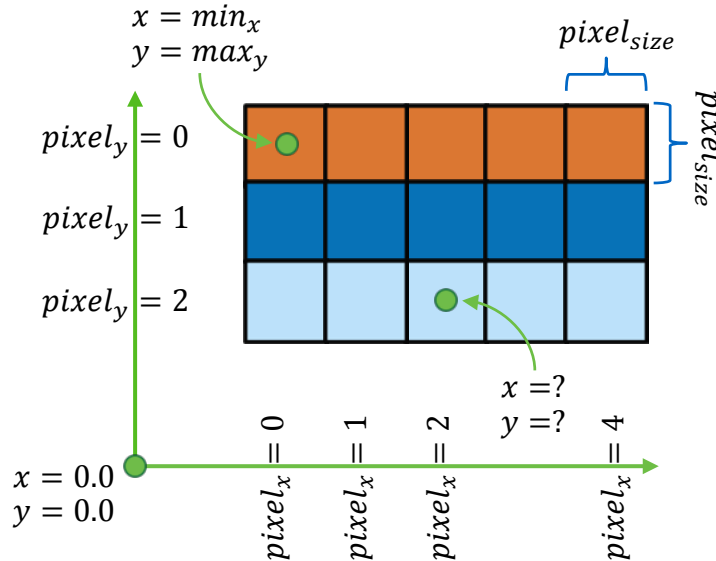


$$pixel\_count_x = 5$$
$$pixel\_count_y = 3$$

```
for  $pixel_y$  in  $pixel\_count_y$ 
  for  $pixel_x$  in  $pixel\_count_x$ 
     $x = \min_x + pixel_x * pixel\_size$ 
     $y = \dots$ 
     $i = \dots$ 
     $pixels[i] = Mandelbrot(x, y)$ 
```

# App: Image and grid

## Iterating pixels

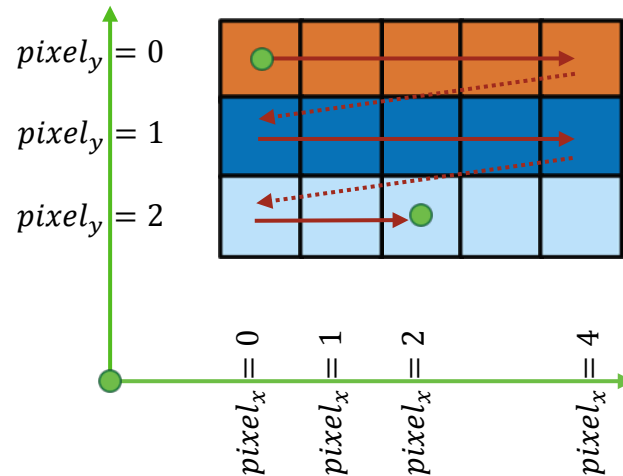


$$pixel\_count_x = 5$$
$$pixel\_count_y = 3$$

```
for  $pixel_y$  in  $pixel\_count_y$ 
  for  $pixel_x$  in  $pixel\_count_x$ 
     $x = min_x + pixel_x * pixel\_size$ 
     $y = max_y - pixel_y * pixel\_size$ 
     $i = \dots$ 
     $pixels[i] = Mandelbrot(x, y)$ 
```

# App: Image and grid

## Iterating pixels



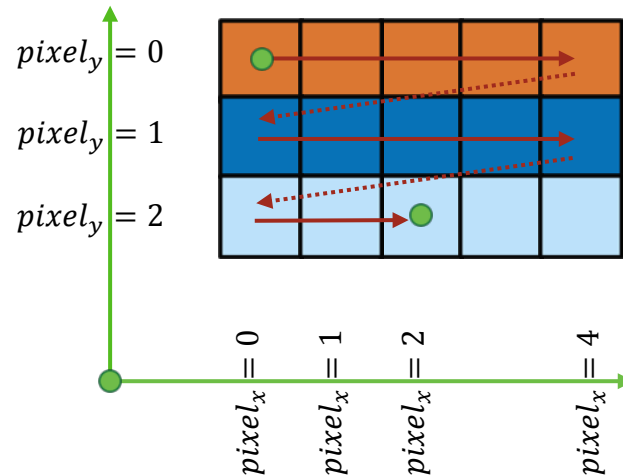
$pixel\_count_x = 5$   
 $pixel\_count_y = 3$

```
for  $pixel_y$  in  $pixel\_count_y$ 
  for  $pixel_x$  in  $pixel\_count_x$ 
     $x = min_x + pixel_x * pixel\_size$ 
     $y = max_y - pixel_y * pixel\_size$ 
     $i = \dots$ 
     $pixels[i] = Mandelbrot(x, y)$ 
```



# App: Image and grid

## Iterating pixels



$pixel\_count_x = 5$   
 $pixel\_count_y = 3$

```
for  $pixel_y$  in  $pixel\_count_y$ 
  for  $pixel_x$  in  $pixel\_count_x$ 
     $x = min_x + pixel_x * pixel\_size$ 
     $y = max_y - pixel_y * pixel\_size$ 
     $i = pixel_y * pixel\_count_x + pixel_x$ 
     $pixels[i] = Mandelbrot(x, y)$ 
```

# App: Escape Time(dwel) Black & White

## Coloring pixels

Iterate  $f_n$  many times and see if the value escapes. It can be shown if  $f_n$  is greater than 2.0 the point will eventually escape to infinity and is not part of the set.

*Mandelbrot(c):*

*iter* = 0

*iter*<sub>max</sub> = 1000

*radius, z* = 0.0

*radius*<sub>max</sub> = 2.0

*while radius* < *radius*<sub>max</sub> && *iter* < *iter*<sub>max</sub>

*z* = *z*<sup>2</sup> + *c*

*radius* = |*z*|

*iter* = *iter* + 1

*if iter* < *iter*<sub>max</sub>

*return white*

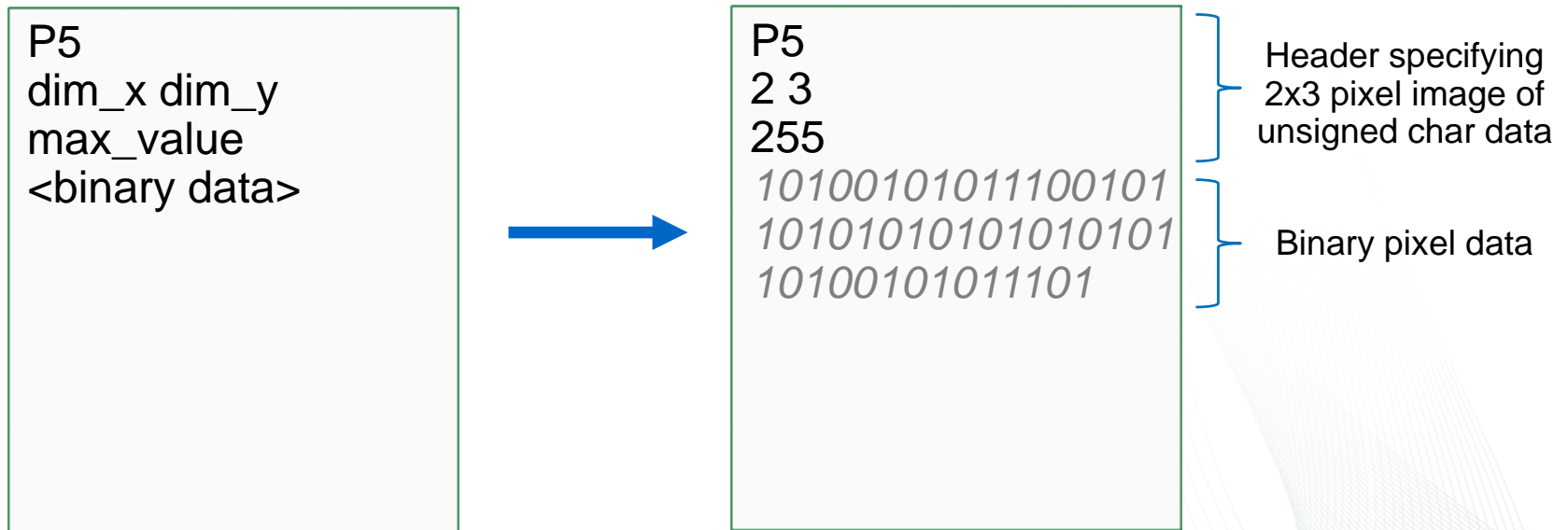
*else*

*return black*

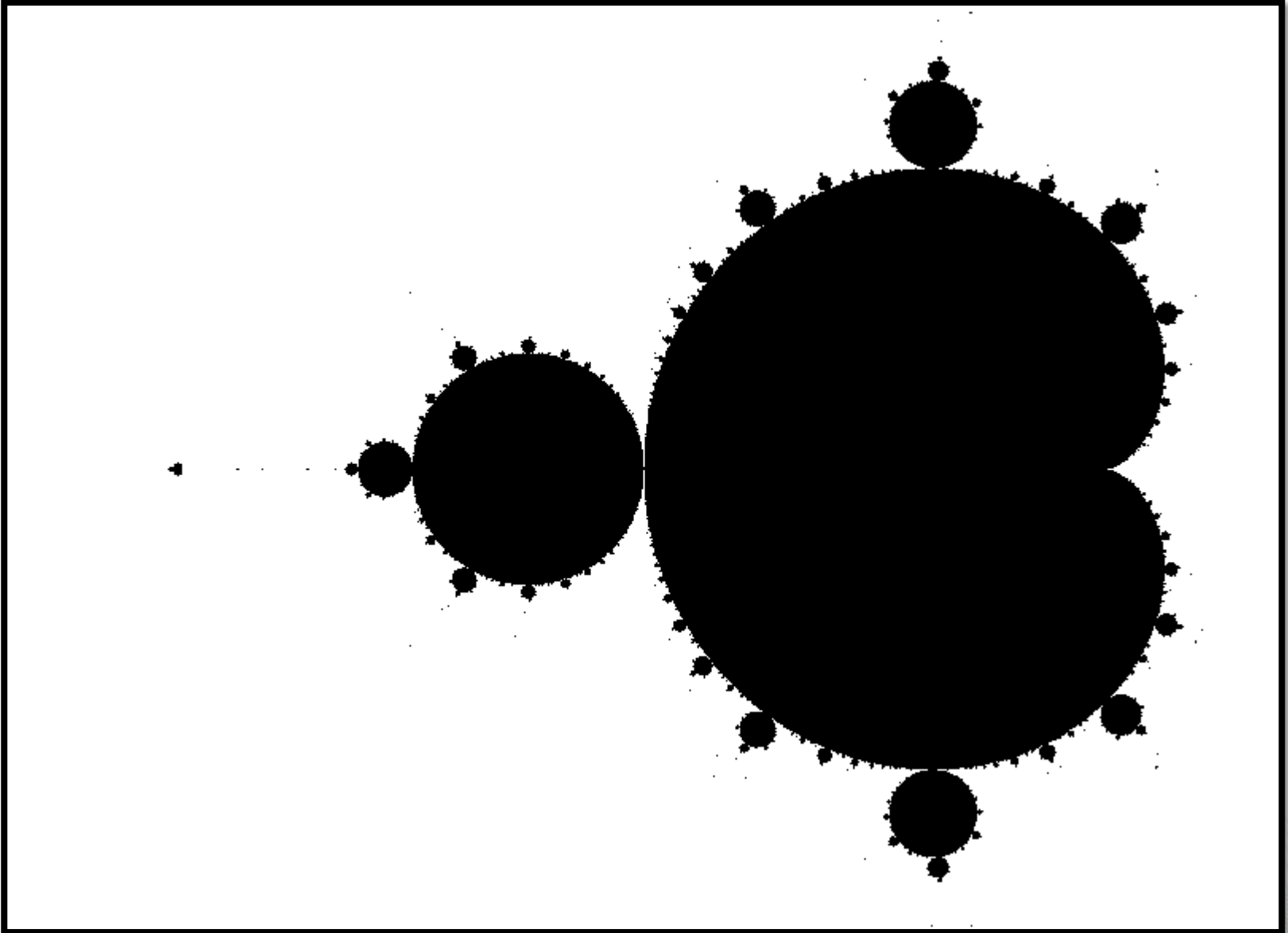
# App: PGM Image file

## Portable GrayMap (PGM,P5)

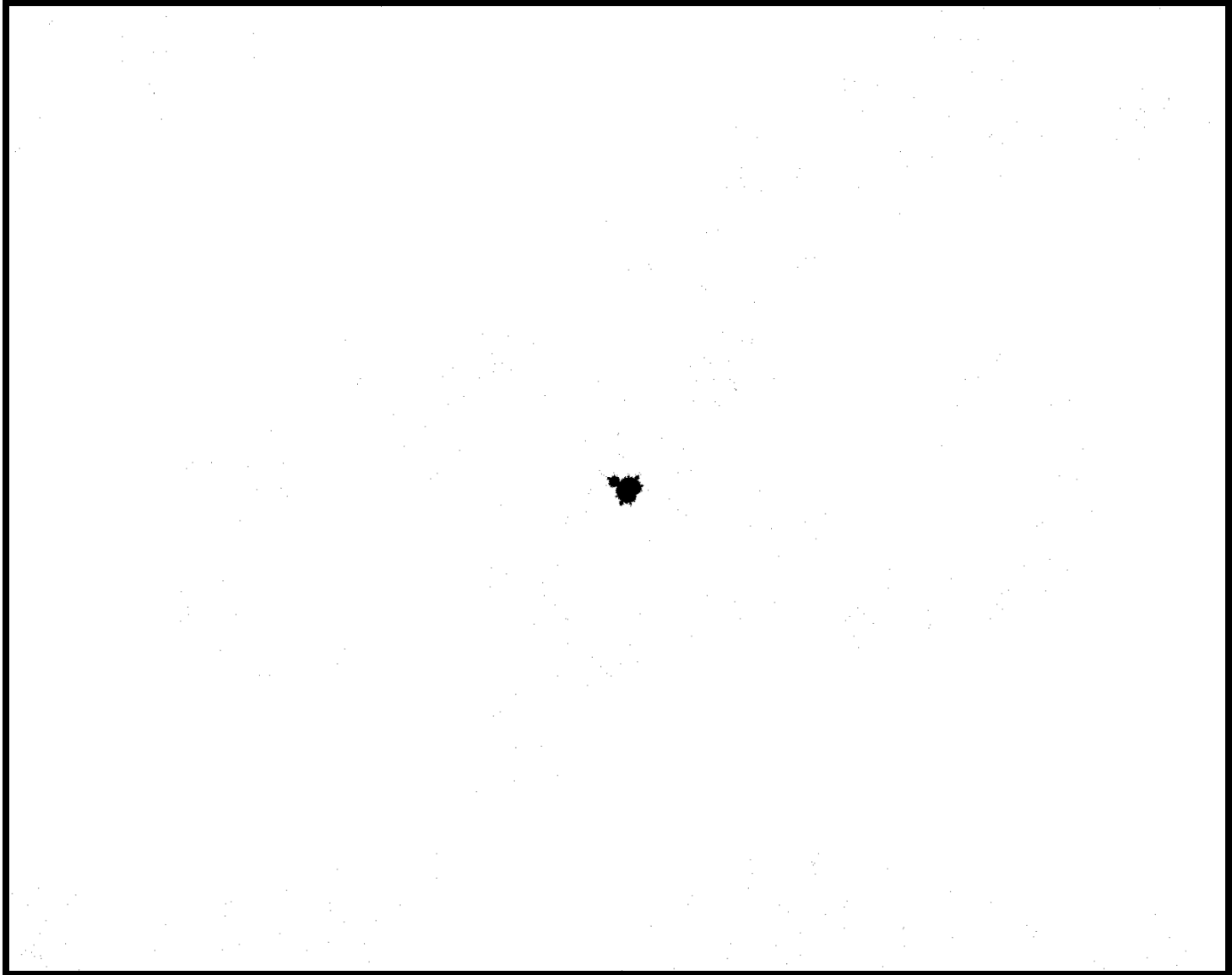
- Simple image format for Grayscale images
- File consists of ASCII header followed by binary 8 bit pixel data in range 0-255 (generally expects pixels written top left first row-major order)



# App: Escape Time(dwel) Black & White



# App: Escape Time(dwel) Black & White



# App: Distance Estimator Grayscale

## Coloring pixels

- While the exact center of a pixel may not be in the Mandelbrot set some point within the pixel may be
  - The escape time algorithm doesn't account for this

# App: Distance Estimator Grayscale

## Coloring pixels

- While the exact center of a pixel may not be in the Mandelbrot set some point within the pixel may be
  - The escape time algorithm doesn't account for this
- It's possible to estimate the distance from a given point to the nearest point on the Mandelbrot set

# App: Distance Estimator Grayscale

## Coloring pixels

- While the exact center of a pixel may not be in the Mandelbrot set some point within the pixel may be
  - The escape time algorithm doesn't account for this
- It's possible to estimate the distance from a given point to the nearest point on the Mandelbrot set
  - Color pixel based upon how far nearest point is to center of the pixel



# App: Distance Estimator Grayscale

## Coloring pixels

- While the exact center of a pixel may not be in the Mandelbrot set some point within the pixel may be
  - The escape time algorithm doesn't account for this
- It's possible to estimate the distance from a given point to the nearest point on the Mandelbrot set
  - Color pixel based upon how far nearest point is to center of the pixel

$$d = 2 * \log(|z|) * \frac{|z|}{\left| \frac{dz}{dc} \right|}$$

# App: Distance Estimator Grayscale

## Coloring pixels

*Mandelbrot(c):*

*iter* = 0

*iter*<sub>max</sub> = 10000

*radius, z, dz* = 0.0

*radius*<sub>max</sub> = 1 << 18

*distance* = 0.0

*while radius* < *radius*<sub>max</sub> && *iter* < *iter*<sub>max</sub>

*dz* = 2 \* *z* \* *dz* + 1

*z* = *z*<sup>2</sup> + *c*

*radius* = |*z*|

*iter* = *iter* + 1

*distance* = 2 \* ln(|*z*|) \* |*z*|/|*dz*|

return *Color(distance)*

# App: Distance Estimator Grayscale

## Calculating distance

*Mandelbrot(c):*

*iter* = 0

*iter*<sub>max</sub> = 10000

*radius, z, dz* = 0.0

*radius*<sub>max</sub> = 1 << 18

*distance* = 0.0

*while radius* < *radius*<sub>max</sub> && *iter* < *iter*<sub>max</sub>

*dz* = 2 \* *z* \* *dz* + 1

*z* = *z*<sup>2</sup> + *c*

*radius* = |*z*|

*iter* = *iter* + 1

*distance* = 2 \* ln(|*z*|) \* |*z*|/|*dz*|

return *Color(distance)*

Max iterations and escape radius should be increased to get accurate distance estimate

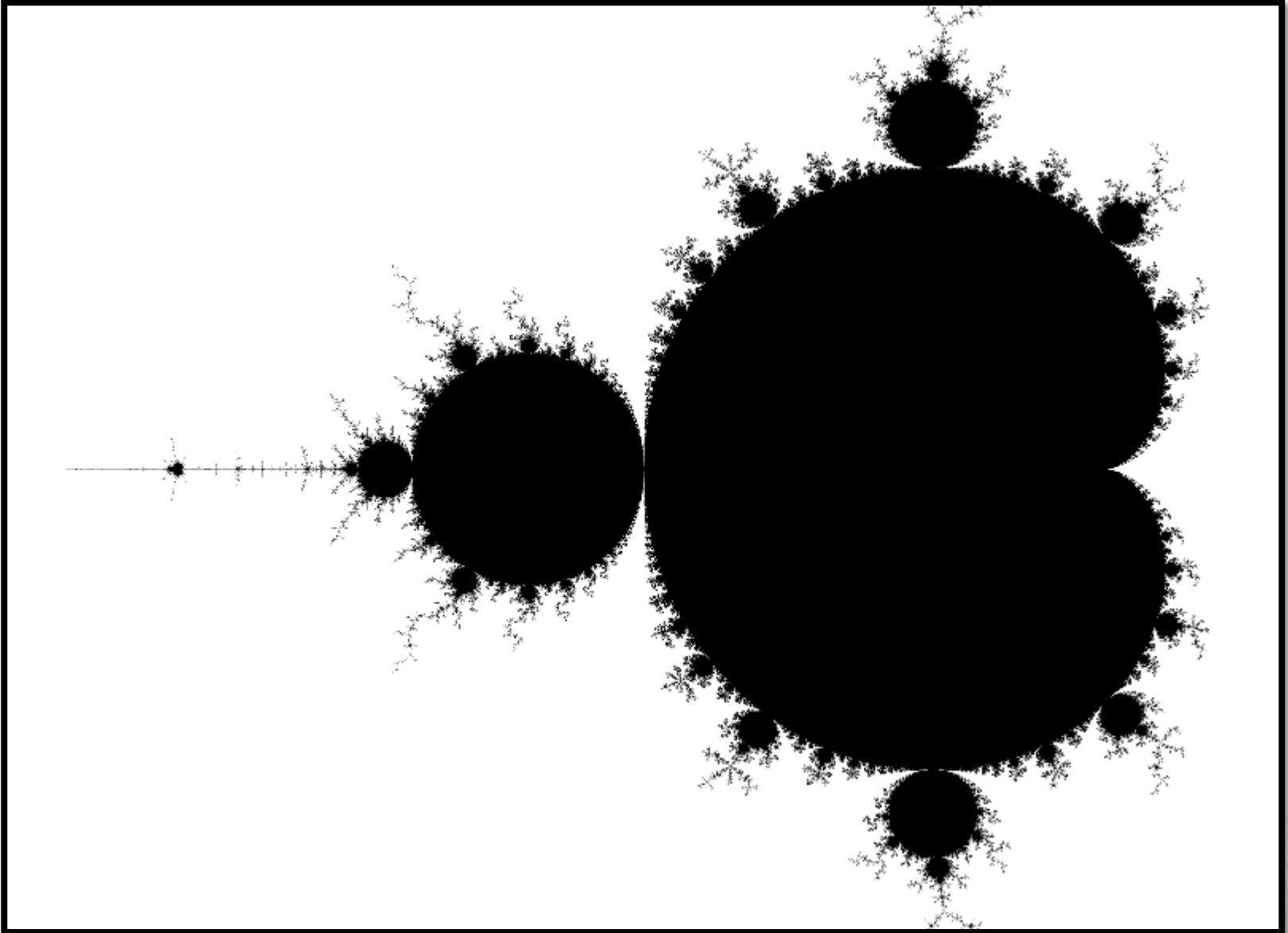
# App: Distance Estimator Grayscale

## Coloring pixels

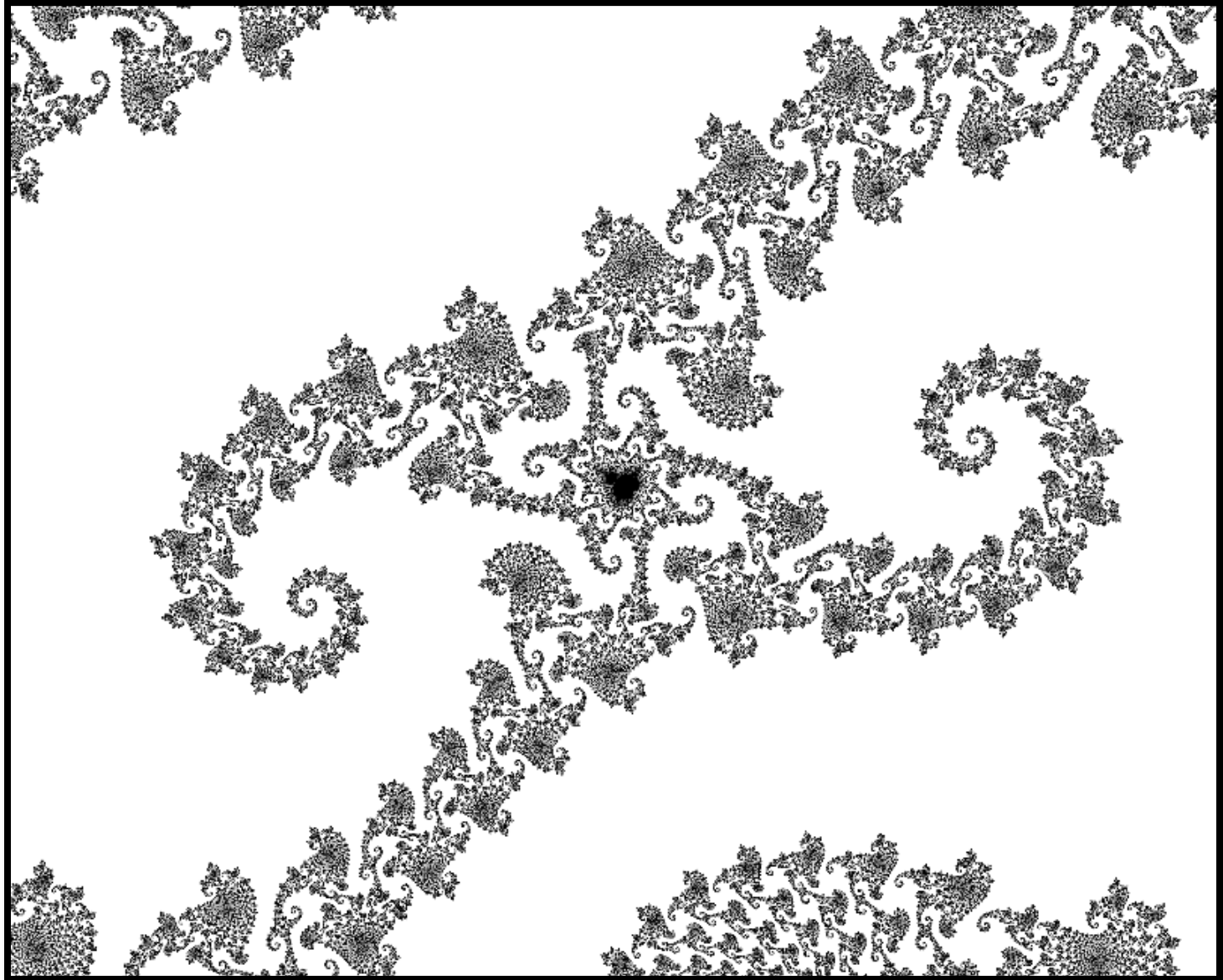
If the distance from the center of our pixel to the closest point within the set is within the pixel we scale the color between pure black, at zero distance from the set, to pure white at the pixel edge. If the distance is greater than that of the pixel we set it to white and return.

```
Color(distance):  
  if distance <  $\frac{1}{2} * pixel_{size}$   
    return pow $\left(\frac{distance}{\frac{1}{2} * pixel_{size}}, \frac{1}{3}\right) * 255$   
  else  
    return 255
```

# App: Distance Estimator Grayscale



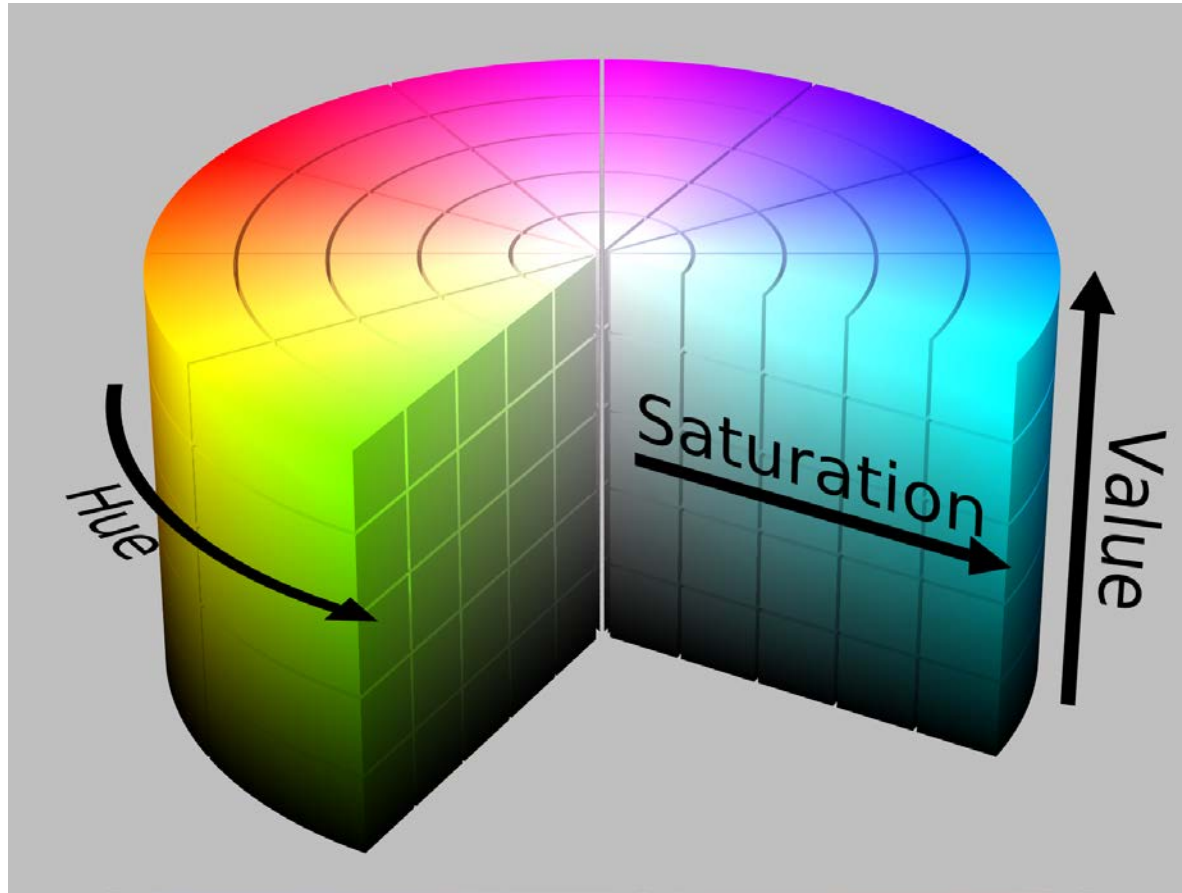
# App: Distance Estimator Grayscale





# App: Adding color

## HSV Color Space



derivative work: [SharkD](#) licensed under [CC BY-SA 3.0](#)

# App: Distance Estimator Color

## Coloring pixels

The previous grayscale method corresponds to setting the Value component. Saturation can be set to a constant that's aesthetically pleasing. The Hue can be set to a log scaled value within the range [0.0,1.0], the factor of 10 controls how many times the hue range is circled.

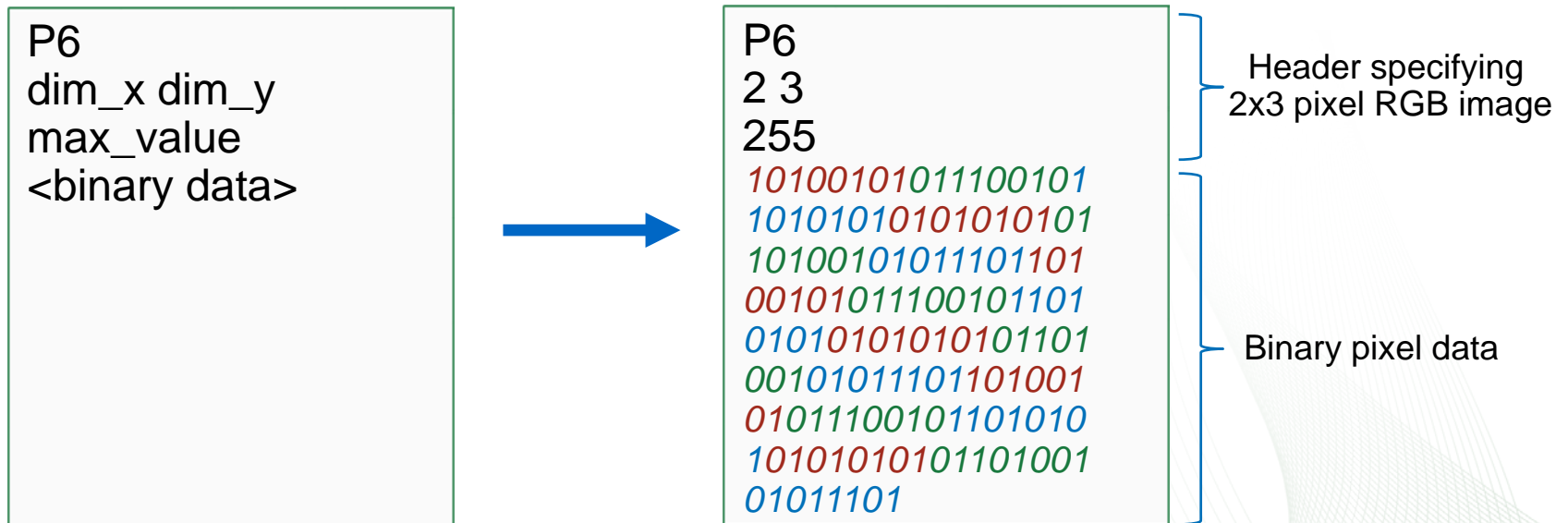
```
Color(distance, iterations):  
    if iterations >= itermax  
        return white  
    if distance <  $\frac{1}{2} * \text{pixel}_{size}$   
        Value = pow $\left(\frac{\text{distance}}{\frac{1}{2} * \text{pixel}_{size}}, \frac{1}{3}\right)$   
    else  
        Value = 1.0  
  
    Saturation = 0.7  
  
    Hue =  $\frac{\ln(\text{iter})}{\ln(\text{iter}_{max})}$   
    Hue = Hue * 10  
    Hue = Hue - floor(Hue)  
    return Hue, Saturation, Value
```



# App: PPM Image file

## Portable PixMap (PPM,P6)

- Simple image format for RGB color images
- File consists of ASCII header followed by binary 24 bit pixel data(8 bits per RGB channel) in range 0-255 (generally expects pixels written top left first row-major order)



# App: Distance Estimator Color

## Converting HSV to RGB

Input:  $H \in [0, 360]$   $S \in [0, 1]$   $V \in [0, 1]$

Output:  $R \in [0, 1]$   $G \in [0, 1]$   $B \in [0, 1]$

$HSVtoRGB(H, S, V)$ :

$$C = V * S$$

$$H' = \frac{H}{60}$$

$$X = C * (1 - |\text{fmod}(H', 2.0) - 1|)$$

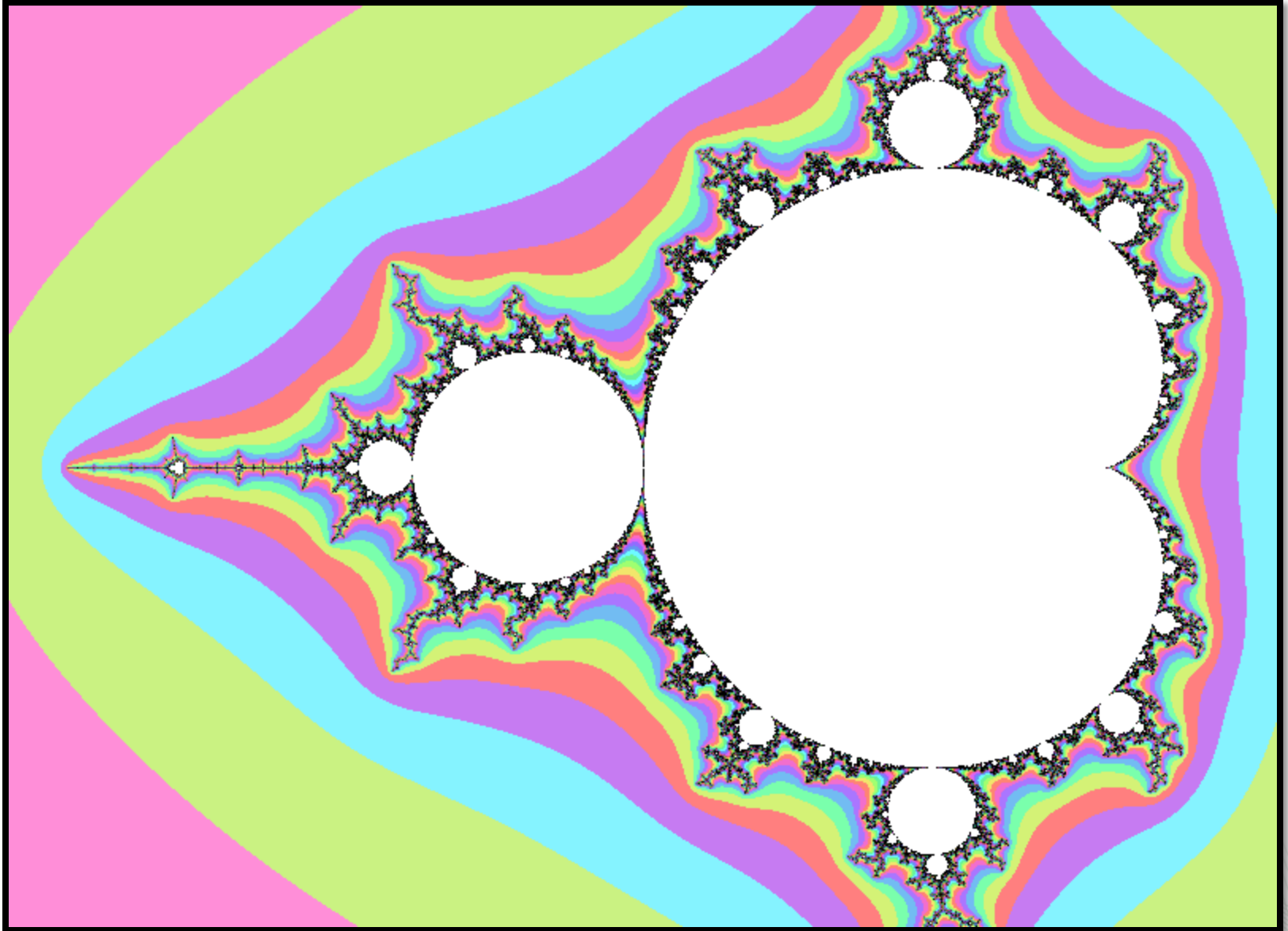
$$(R_1, G_1, B_1) = \begin{cases} (C, X, 0) & \text{if } 0 \leq H' \leq 1 \\ (X, C, 0) & \text{if } 1 \leq H' \leq 2 \\ (0, C, X) & \text{if } 2 \leq H' \leq 3 \\ (0, X, C) & \text{if } 3 \leq H' \leq 4 \\ (X, 0, C) & \text{if } 4 \leq H' \leq 5 \\ (C, 0, X) & \text{if } 5 \leq H' \leq 6 \end{cases}$$

$$m = V - C$$

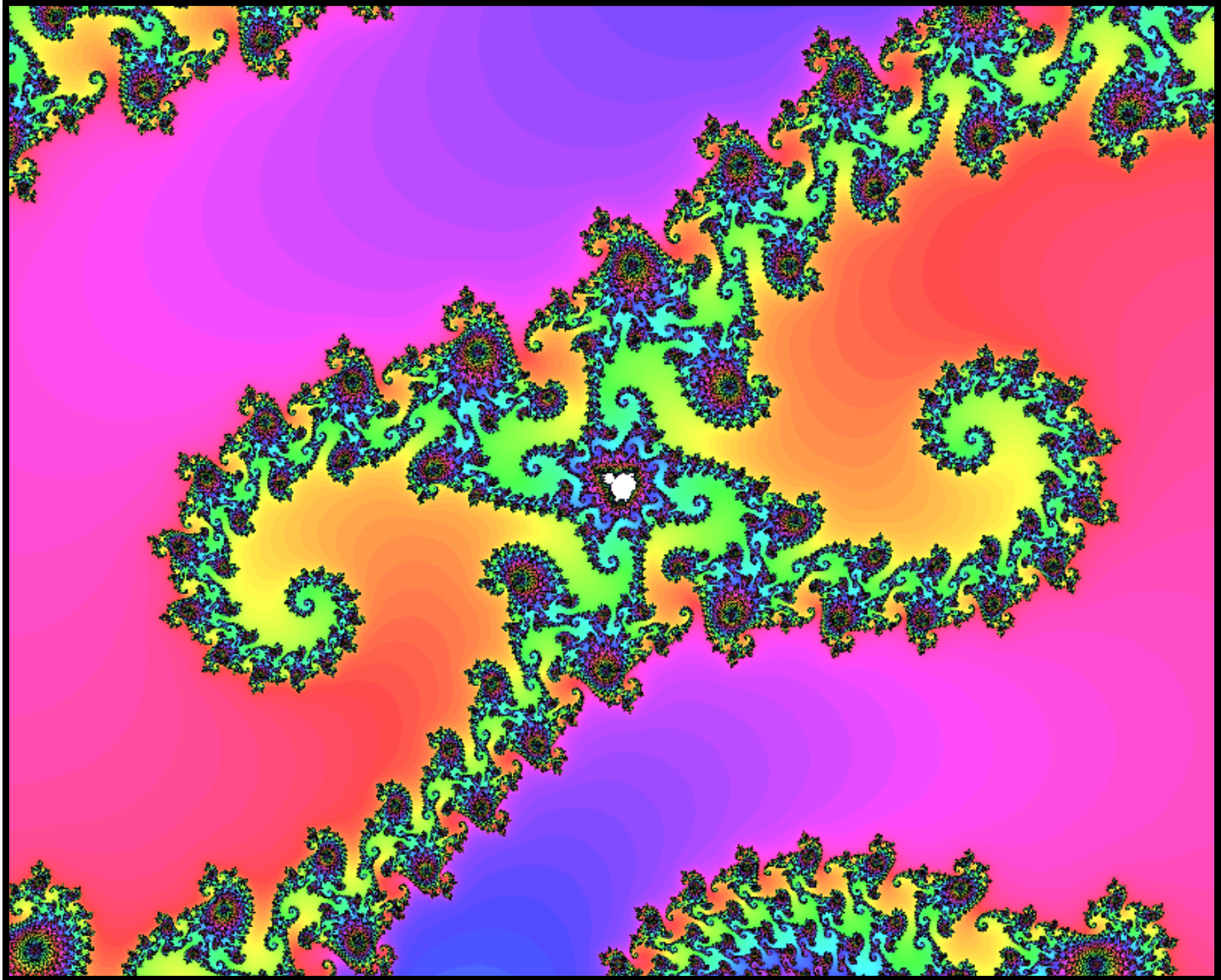
$$R, G, B = (R_1 + m, G_1 + m, B_1 + m)$$

return  $R, G, B$

# App: Distance Estimator Color



# App: Distance Estimator Color



# App: Distance Estimator Smooth Color

## Smoothly Coloring Hue

Due to the finite number of iterations banding appears in the Hue. To get a smooth continuous Hue variation we can define the continuous iteration value.

*Color(distance, iterations, radius):*

*if iterations*  $\geq$  *iter<sub>max</sub>*  
*return white*

*if distance*  $< \frac{1}{2} * \text{pixel}_{size}$   
*Value* = *pow* $\left(\frac{\text{distance}}{\frac{1}{2} * \text{pixel}_{size}}, \frac{1}{3}\right)$   
*else*

*Value* = 1.0

*Saturation* = 0.7

*iter<sub>continuous</sub>* = *iterations* -  $\log_2\left(\frac{\ln(\text{radius})}{\ln(\text{radius}_{max})}\right)$

*Hue* =  $\frac{\ln(\text{iter}_{continuous})}{\ln(\text{iter}_{max})}$

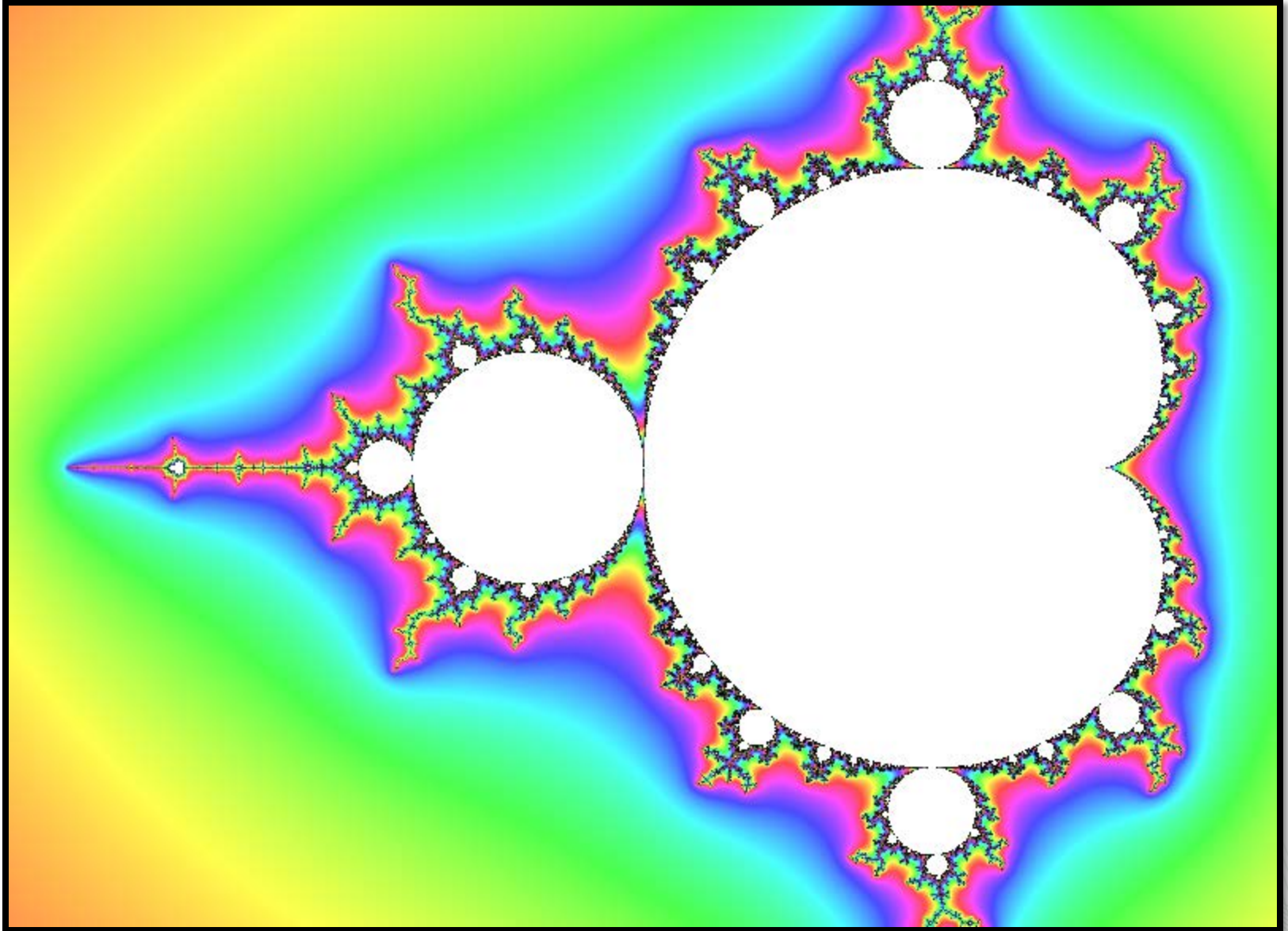
*Hue* = *Hue* \* 10

*Hue* = *Hue* - *floor*(*Hue*)

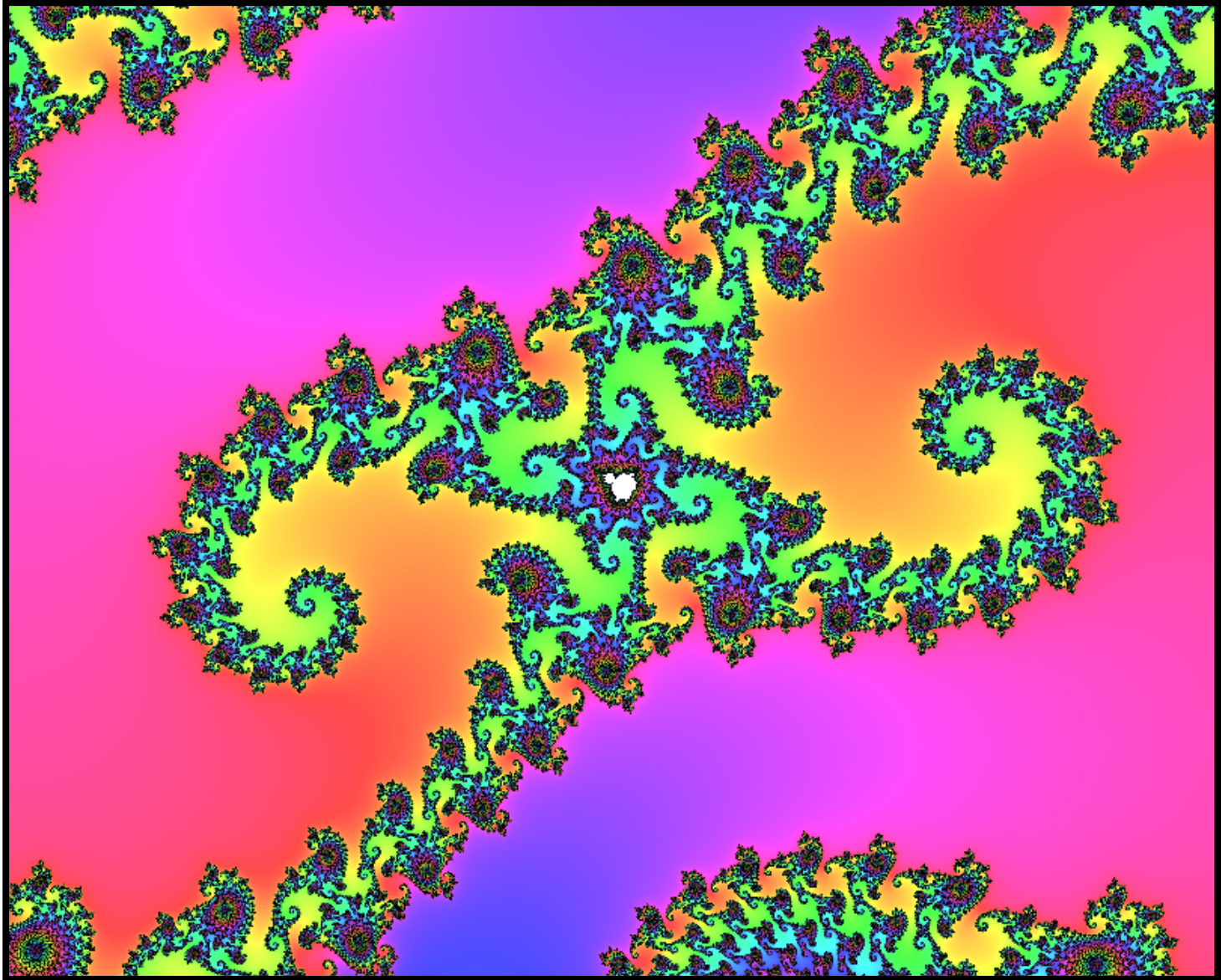
*return Hue, Saturation, Value*



# App: Distance Estimator Smooth Color



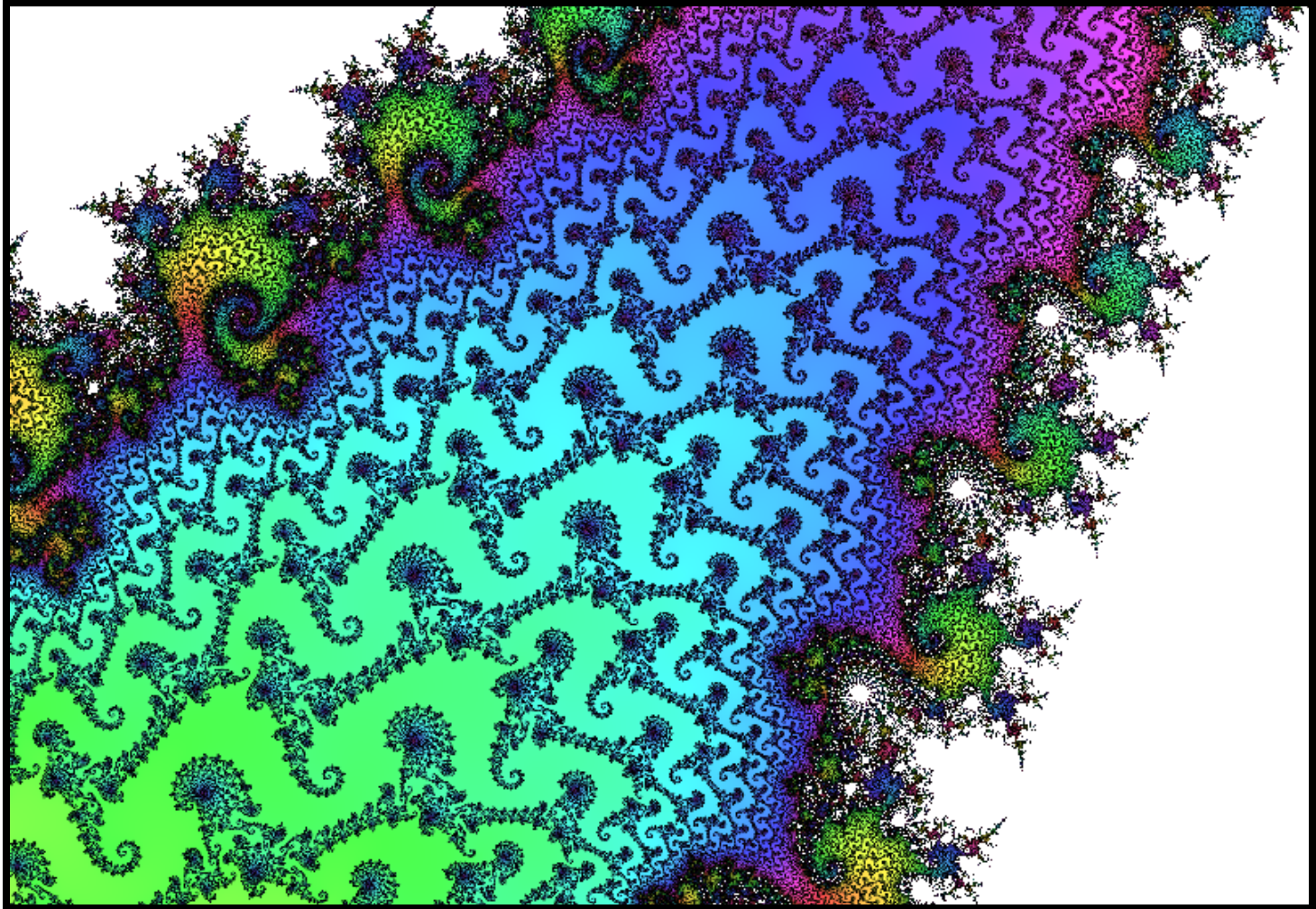
# App: Distance Estimator Smooth Color





# App: Aliasing artifacts

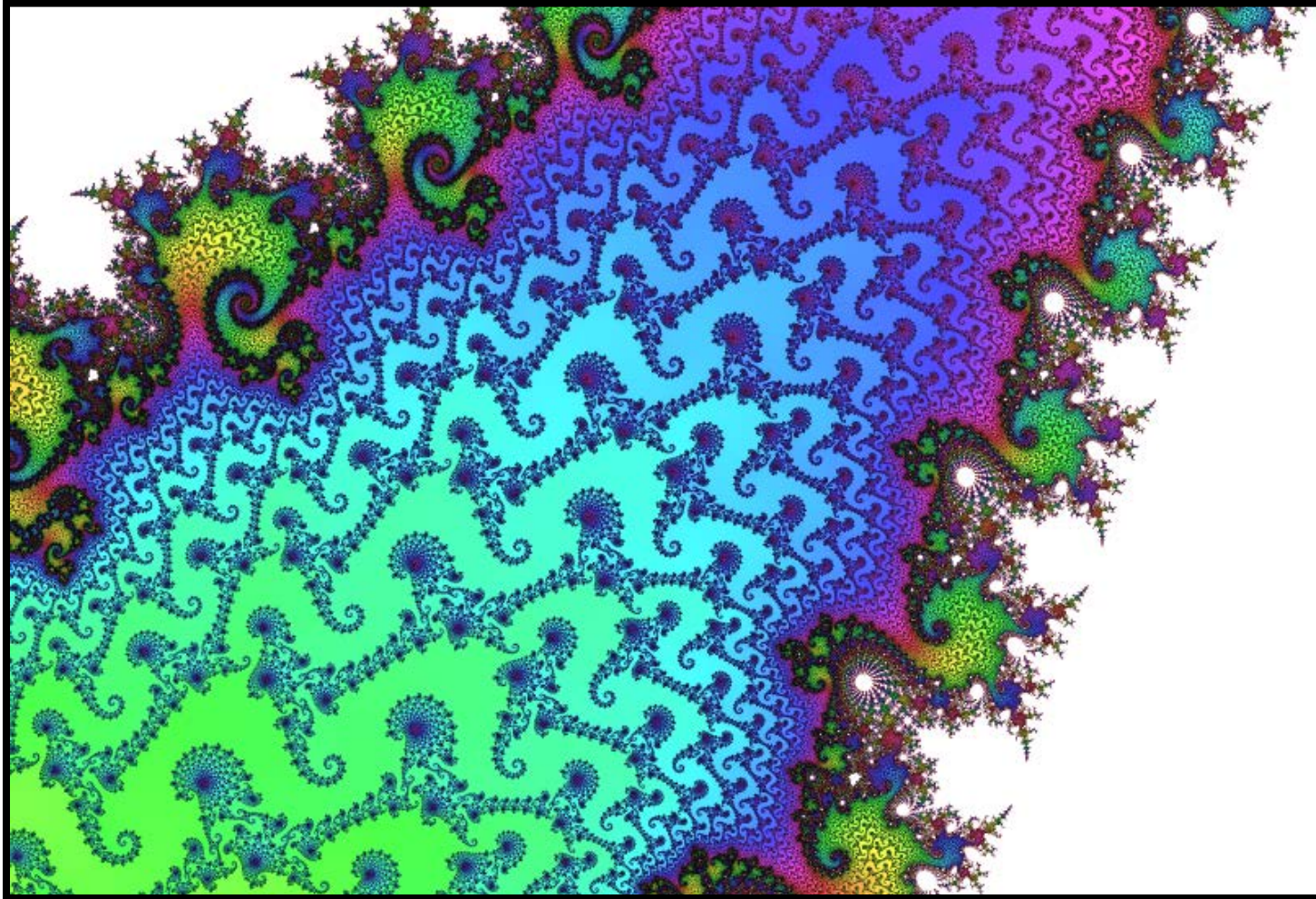
The Mandelbrot set exhibits extreme aliasing artifacts near the boundaries as the escape time iterations tend to infinity near the border of the set.





# App: Aliasing artifacts

Supersampling can be used quite effectively as a form of **anti-aliasing**. This method splits the pixel into multiple sub-pixels, calculates their value, and averages the result to determine the overall pixels color. Many options exist as to where the sub-pixel points are located and how they are averaged. Below shows 4x4 uniform subsampling



# Thank you!

## Questions?

**Contact the OLCF at:**

**[help@olcf.ornl.gov](mailto:help@olcf.ornl.gov)**

**(865) 241 - 6536**

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.