

# *Tricorn-Fractal Presentation*

*Group t111:*

- *Nikola Granitsky*
- *Cem Temizkan*
- *Yashar Alan Zenker*



# *The Multicorn Fractal Function*

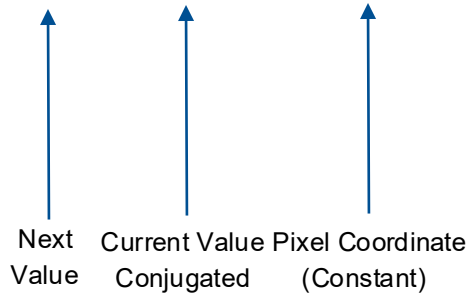


# How does the function work?

## Creating an image with our function

A Multicorn fractal is generated by repeatedly applying the following iteration rule:

$$z_{i+1} = \overline{z_i}^m + c \quad (i \geq 0)$$



- For the Tricorn fractal,  $m = 2$ .
- For every pixel, the sequence starts with  $z_0 = 0$

$$z_0 = 0 \longrightarrow z_1 = c \longrightarrow z_2 = \overline{z_1}^2 + c \longrightarrow \dots$$

# How does the function work?

## Creating an image with our function

A Multicorn fractal is generated by repeatedly applying the following iteration rule:

$$z_{i+1} = \overline{z_i}^m + c \quad (i \geq 0)$$

- The value  $c$  is determined by the following equations:

$$c_{\text{real}} = \text{Re}(\text{start}) + x \cdot \text{res}$$

$$c_{\text{imag}} = \text{Im}(\text{start}) \pm y \cdot \text{res}$$

Depends on  
the sign of  
the height



- Every pixel represents one coordinate  $c$ :

|           |           |           |
|-----------|-----------|-----------|
| $c_{0,0}$ | $c_{1,0}$ | $c_{2,0}$ |
| $c_{0,1}$ | $c_{1,1}$ | $c_{2,1}$ |
| $c_{0,2}$ | $c_{1,2}$ | $c_{2,2}$ |

# How does the function work?

## Creating an image with our function

A Multicorn fractal is generated by repeatedly applying the following iteration rule:

$$z_{i+1} = \overline{z_i}^m + c \quad (i \geq 0) \longrightarrow \begin{array}{l} z_i = x_i + y_i i \\ c = c_r + c_i i. \\ \overline{z_i} = x_i - y_i i. \\ \overline{z_i}^2 = (x_i - y_i i)^2. \\ (x_i - y_i i)^2 = x_i^2 - 2x_i y_i i + y_i^2 i^2. \\ i^2 = -1, \\ \overline{z_i}^2 = x_i^2 - y_i^2 - 2x_i y_i i. \\ z_{i+1} = (x_i^2 - y_i^2 + c_r) + (-2x_i y_i + c_i) i. \\ x_{i+1} = x_i^2 - y_i^2 + c_r \\ y_{i+1} = -2x_i y_i + c_i \end{array}$$

$$c = c_r + c_i i.$$

$$\overline{z_i} = x_i - y_i i.$$

$$\overline{z_i}^2 = (x_i - y_i i)^2.$$

$$(x_i - y_i i)^2 = x_i^2 - 2x_i y_i i + y_i^2 i^2.$$

$$i^2 = -1,$$

$$\overline{z_i}^2 = x_i^2 - y_i^2 - 2x_i y_i i.$$

$$z_{i+1} = (x_i^2 - y_i^2 + c_r) + (-2x_i y_i + c_i) i.$$

$$x_{i+1} = x_i^2 - y_i^2 + c_r$$

$$y_{i+1} = -2x_i y_i + c_i$$

- We separate  $z_i$  and  $C$  into their real and imaginary parts.

- Independent values  $\rightarrow$  Effective Calculations

# How does the function work?

## Creating an image with our function

A Multicorn fractal is generated by repeatedly applying the following iteration rule:

$$z_{i+1} = \overline{z_i}^m + c \quad (i \geq 0)$$

$$z = x + yi,$$

For each pixel, the iteration has 2 possible outcomes:

$$|z| = \sqrt{x^2 + y^2}.$$

### 1) *The orbit escapes.*

→ The orbit “escapes” when its magnitude satisfies:

$$|z_i| > 2$$

$$|z|^2 = x^2 + y^2.$$

$$|z| > 2$$

→ Magnitude grows without bound

$$|z|^2 > 4.$$

### 2) *The orbit reaches the maximum iteration limit (n).*

→ The orbit didn't escape in n iterations.

# How does the function work?

## Creating an image with our function

- We introduce a new variable  $\rightarrow i$   
Orbit *escapes*  $\rightarrow i = \text{index where the function first satisfies: } |z_i|^2 > 4.$   
Orbits *don't escape*  $\rightarrow i = \text{maximum iteration limit (n)}$

- Calculate grayscale intensity:

$$\text{value} = \begin{cases} 0, & i = n, \\ 255 - 4(i \bmod 64), & i < n. \end{cases}$$

- for  $i = n \rightarrow$  pixel becomes black,  
for  $i < n \rightarrow$  pixel is colored.

# How does the function work?

## Creating an image with our function

- If the color flag is set, a colored output is generated through mapping into RGB

$$\text{value} = \begin{cases} 0, & i = n, \\ 255 - 4(i \bmod 64), & i < n. \end{cases} \longrightarrow$$

$$R = (3 \cdot \text{value}) \bmod 255,$$

$$G = (2 \cdot \text{value}) \bmod 255,$$

$$B = (5 \cdot \text{value}) \bmod 255.$$

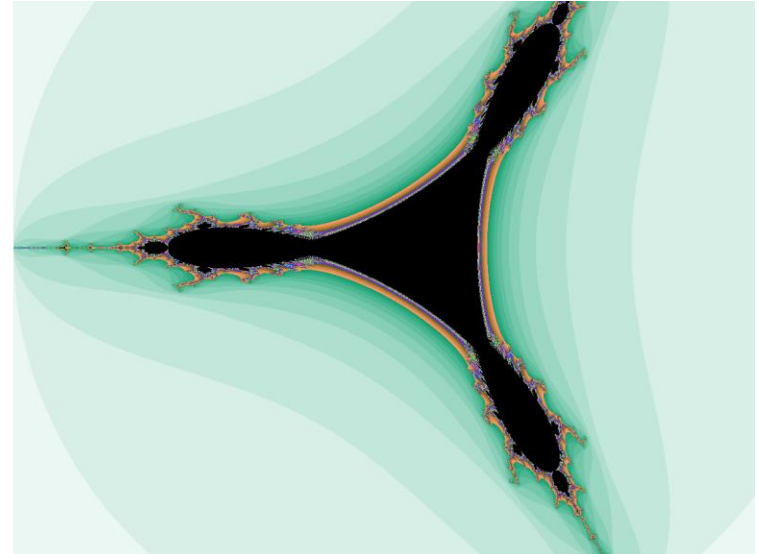
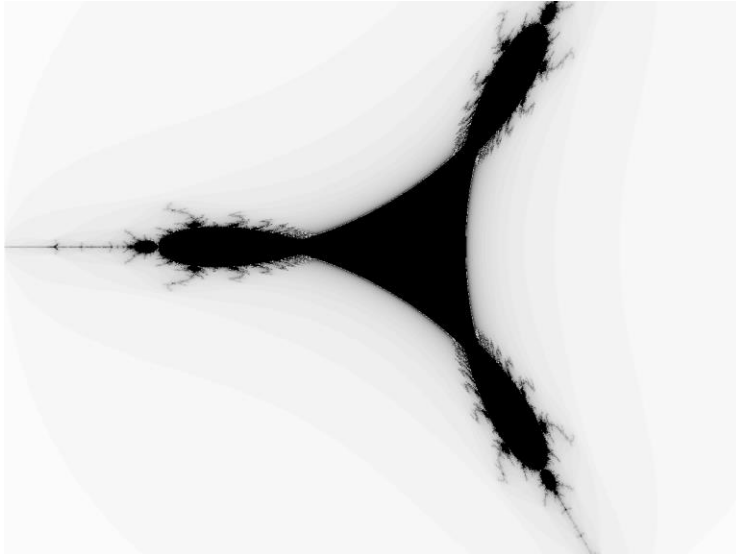
$$i \longrightarrow \text{value} \longrightarrow (R, G, B)$$

- Different escape iterations produce different colors,  
Non-escaping orbits remain black.

# How does the function work?

## Creating an image with our function

- Applying this to process to every pixel produces the **Tricorn image**.
- Examples of grayscale and colored Tricorn images:

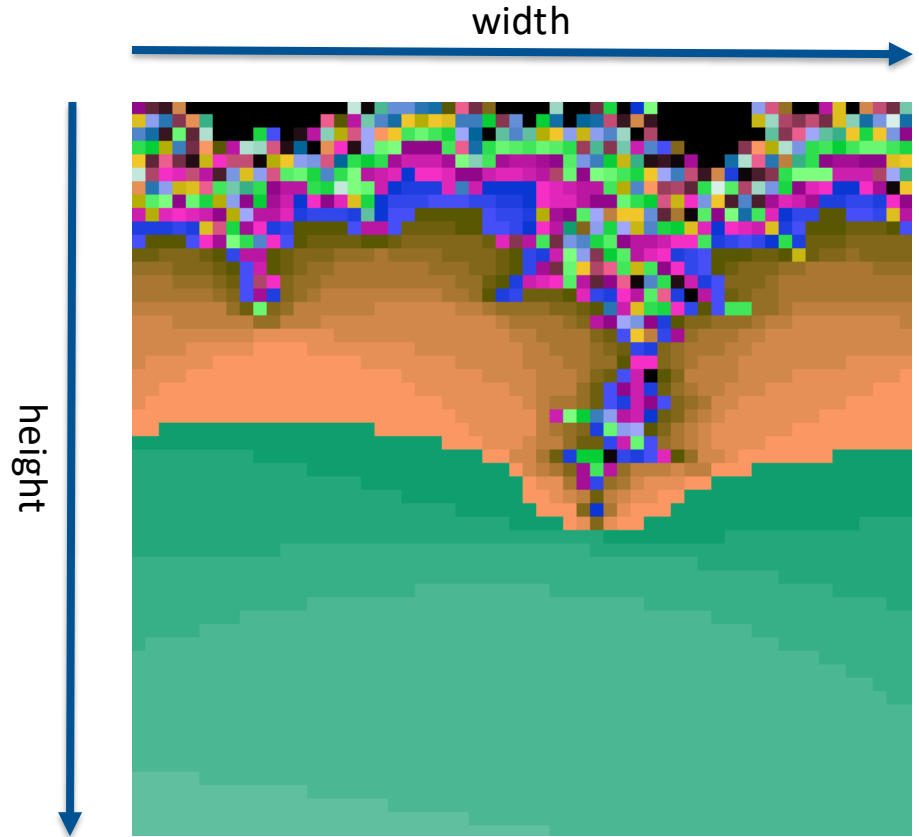


# Naive Implementation

## V1

- Looping over width and height
- $O(n^2)$  complexity

```
compute(){  
    for (y = 0; y < height; y++){  
        for(x = 0; x < width; x++){  
            converge();  
        }  
    }  
}
```



# Naive Implementation

## V1

- Looping over width and height
- $O(n^2)$  complexity
- With per-pixel divergence detection

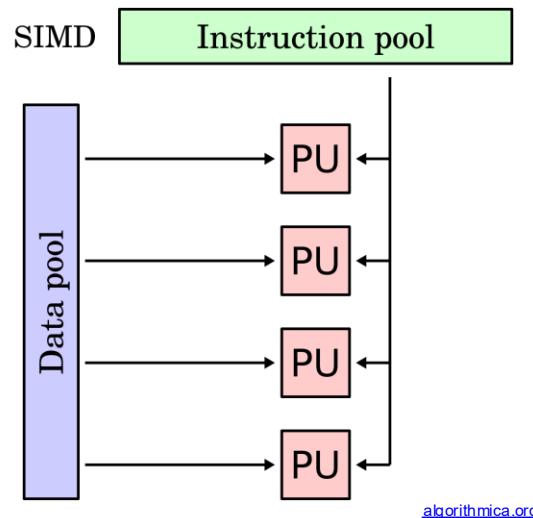
```
converge(){  
    resultVector = 0;  
    for(i = 0; i < maxIterations; i++){  
        if(|resultVector| > 2)  
            return 255 - 4(i mod 64);  
  
        resultVector =  $\overline{z_i}^m$   
    }  
    return 0;  
}
```



# SIMD Optimization

## V0+V2

- Four floats at a time
- AMD64 special instructions
- Special 128bit registers
- SSE 4.1



width

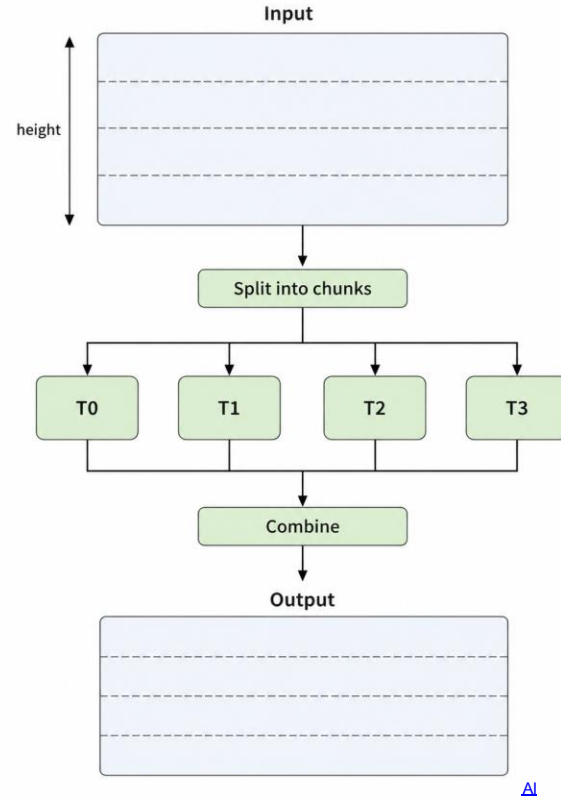
Calculated at once



# Multithreading Optimization

## V3

- Division of calculation in concurrent tasks
- Image split into Chunks
- Harmonizes with SIMD on width



height  
←



# Additional Optimizations

## All versions

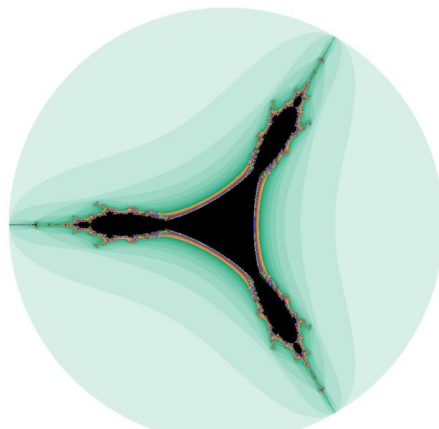
- ABI overhead
- Custom math
- Moving calculations outside of the hot path
- Lookup tables

$$\sqrt{a^2 + b^2} > 2,0$$

Convergence check

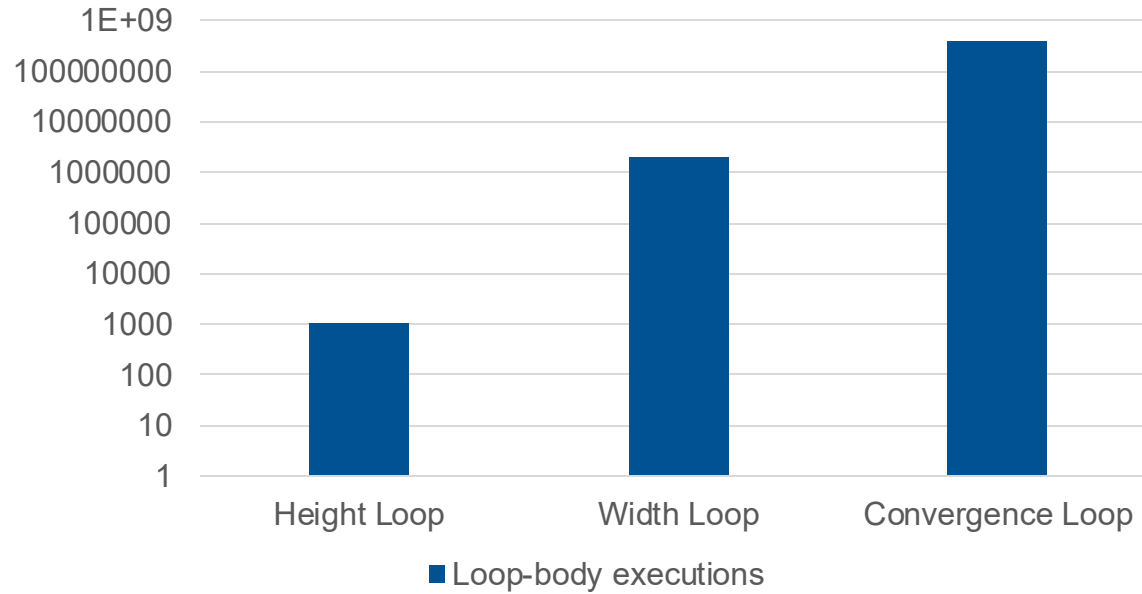

$$a^2 + b^2 > 4,0$$

```
compute(){  
    for (y = 0; y < height; y++){  
        for(x = 0; x < width; x++){  
            for(i = 0; i < maxIterations; i++){  
                // Small expenses accumulate  
            }  
        }  
    }  
}
```



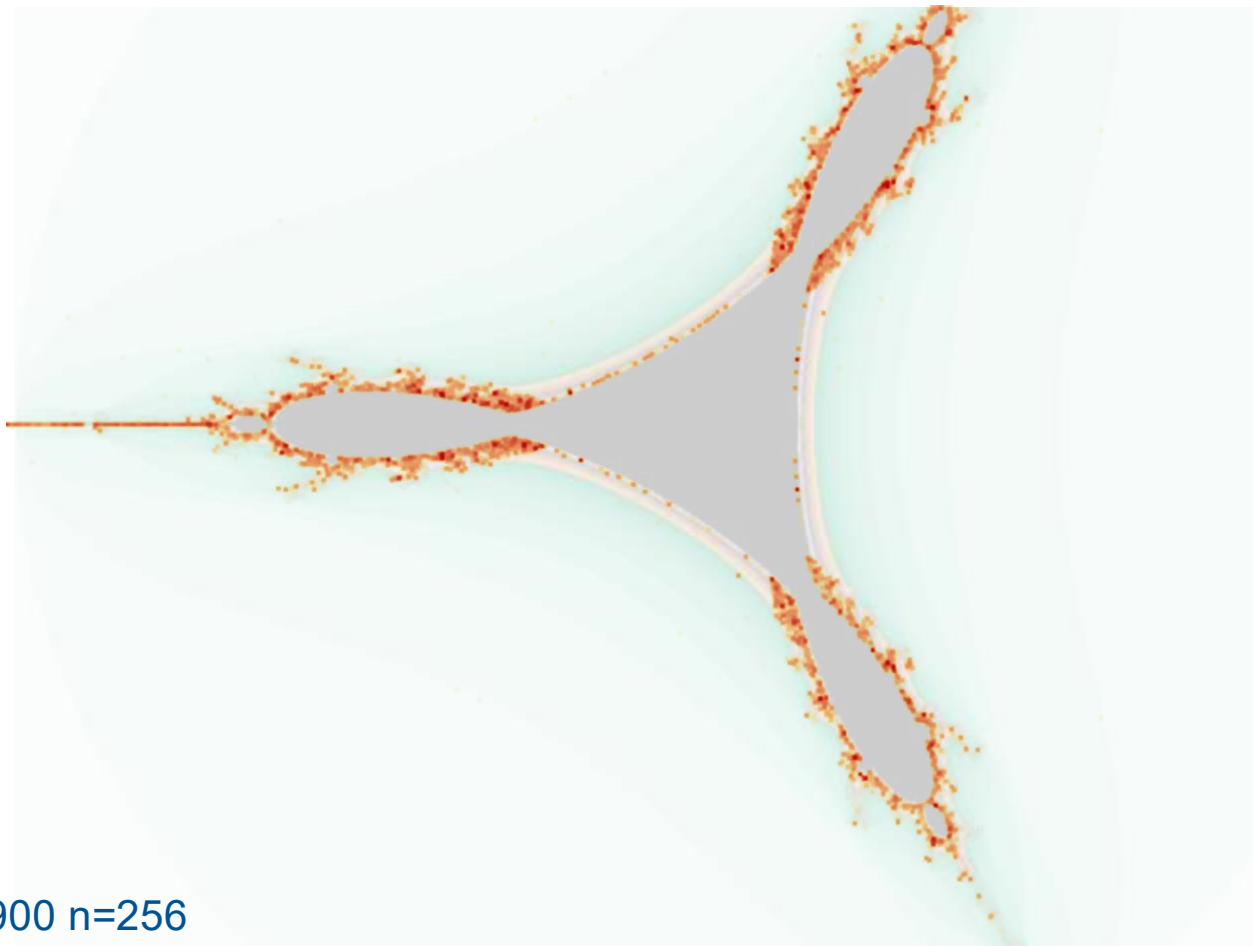
# Performance Impact Comparison

Example: 1920x1080 with  $n = 200$  (SISD)



# FP Error Visualized

V0 vs V2



# Benchmarking

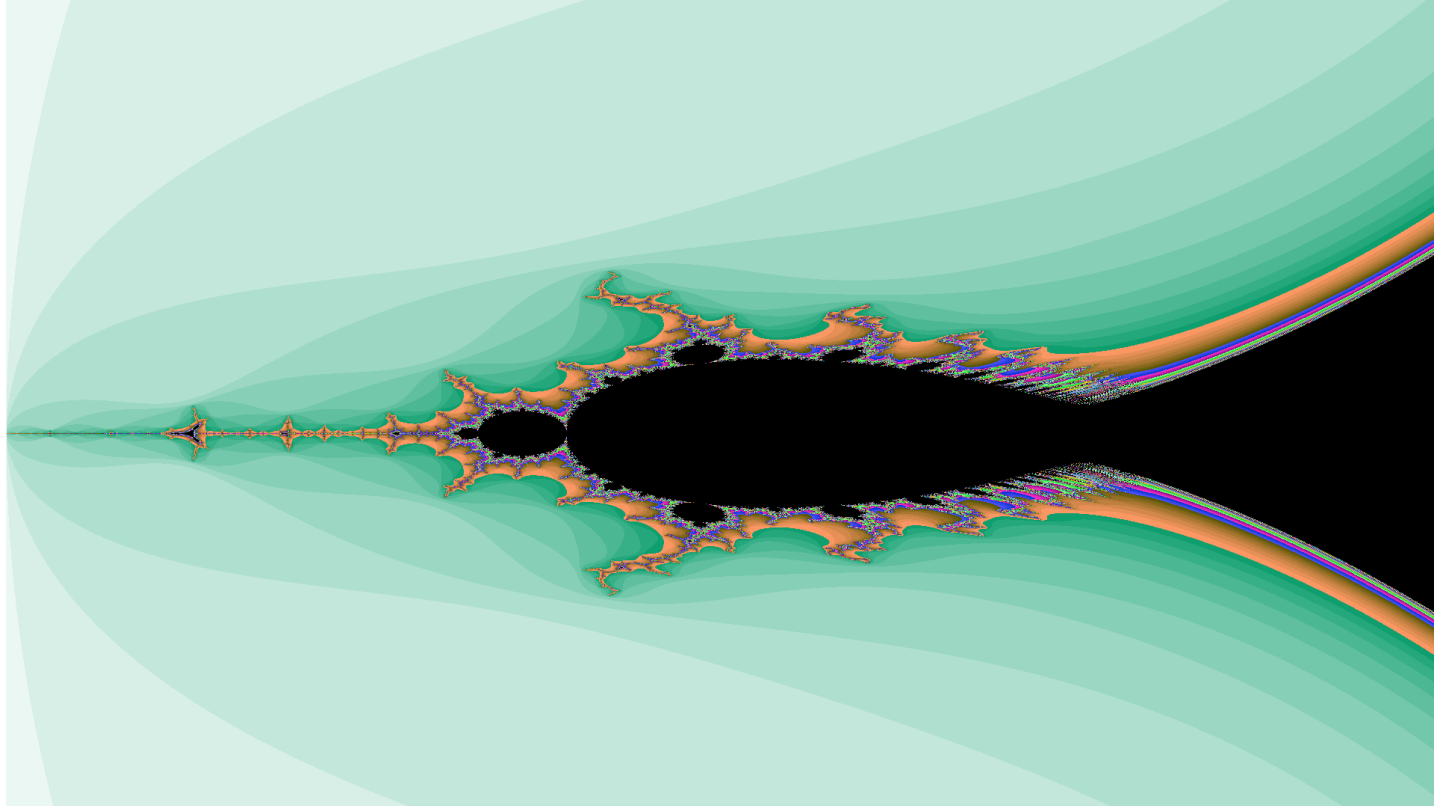
## What did we achieve with the different implementations?

To see how much of a speed up the different implementations yield we have prepared a couple of tests

- Test 1: Default 1920x1080 pixel image
- Test 2: Ultra high resolution image – 7680x4320 pixels, with very low iterations
- Test 3: Normal image, zoomed on the edge of the fractal
- Test 4: Fully black image with high iterations

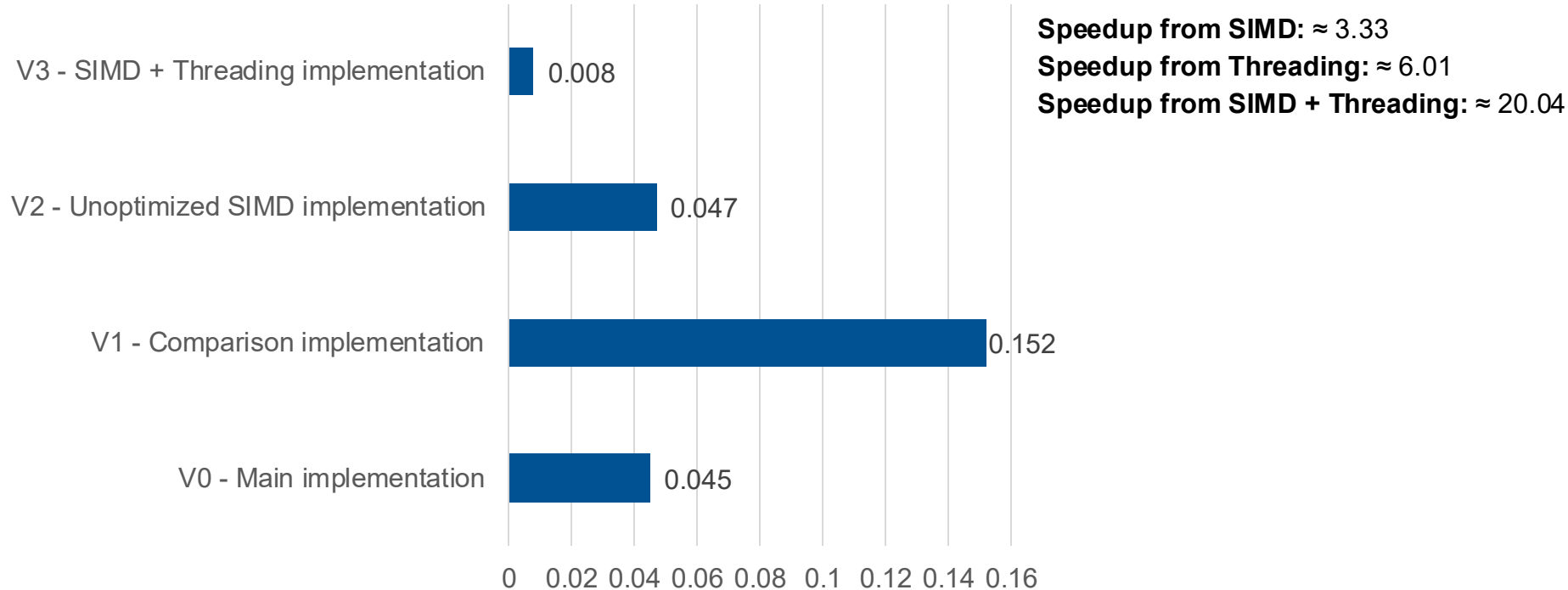
Test 1: Default 1920x1080 pixel image

What did we achieve with the different implementations?



## Test 1: Default 1920x1080 pixel image

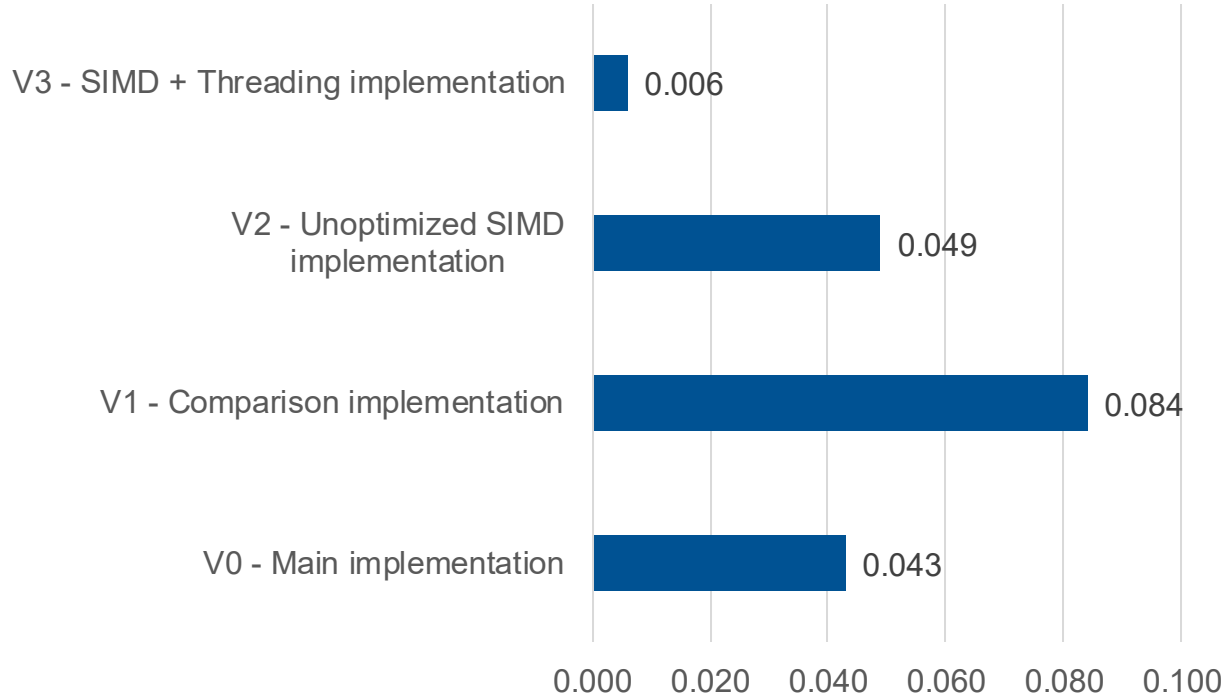
### What did we achieve with the different implementations?



## Test 2: Ultra high resolution image – 7680x4320 pixels, with very low iterations



### What did we achieve with the different implementations?



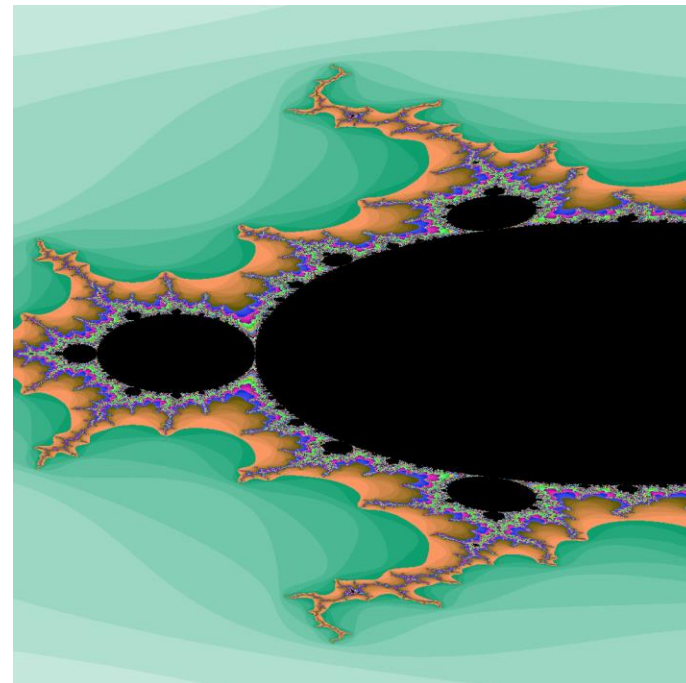
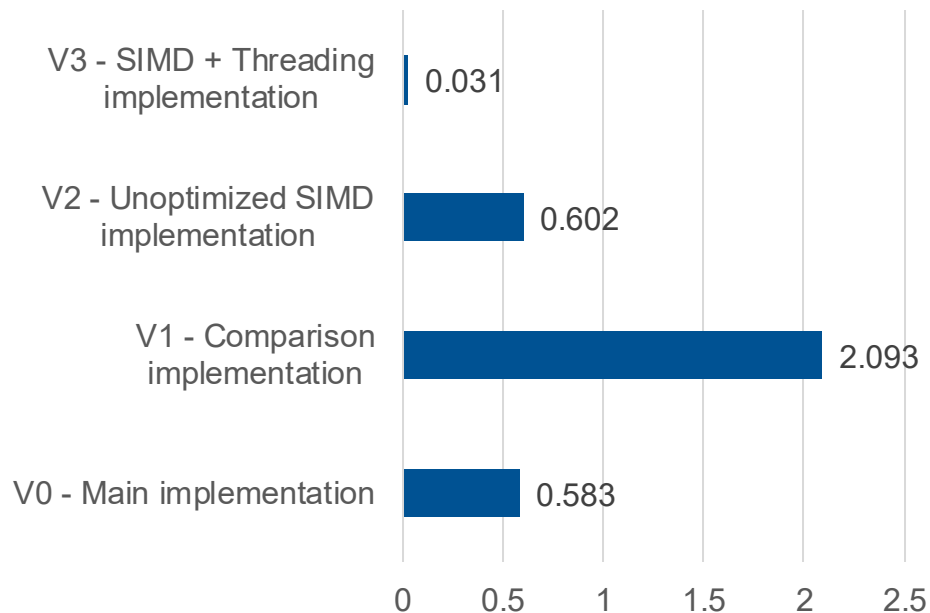
**Speedup from SIMD:  $\approx 1.96$**

**Speedup from Threading:  $\approx 7.19$**

**Speedup from SIMD + Threading:  $\approx 14.08$**

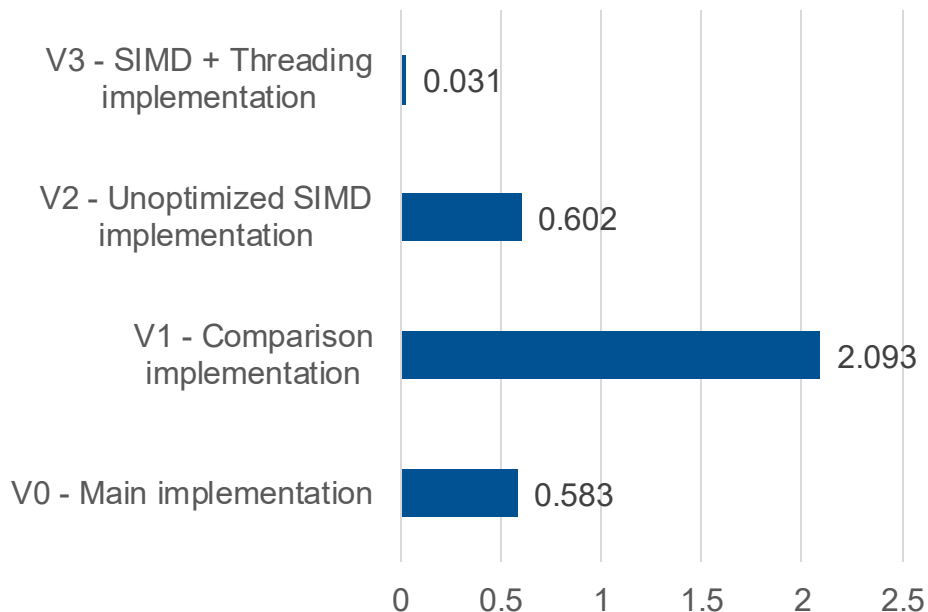
### Test 3: Normal image, zoomed on the edge of the fractal

## What did we achieve with the different implementations?



### Test 3: Normal image, zoomed on the edge of the fractal

## What did we achieve with the different implementations?



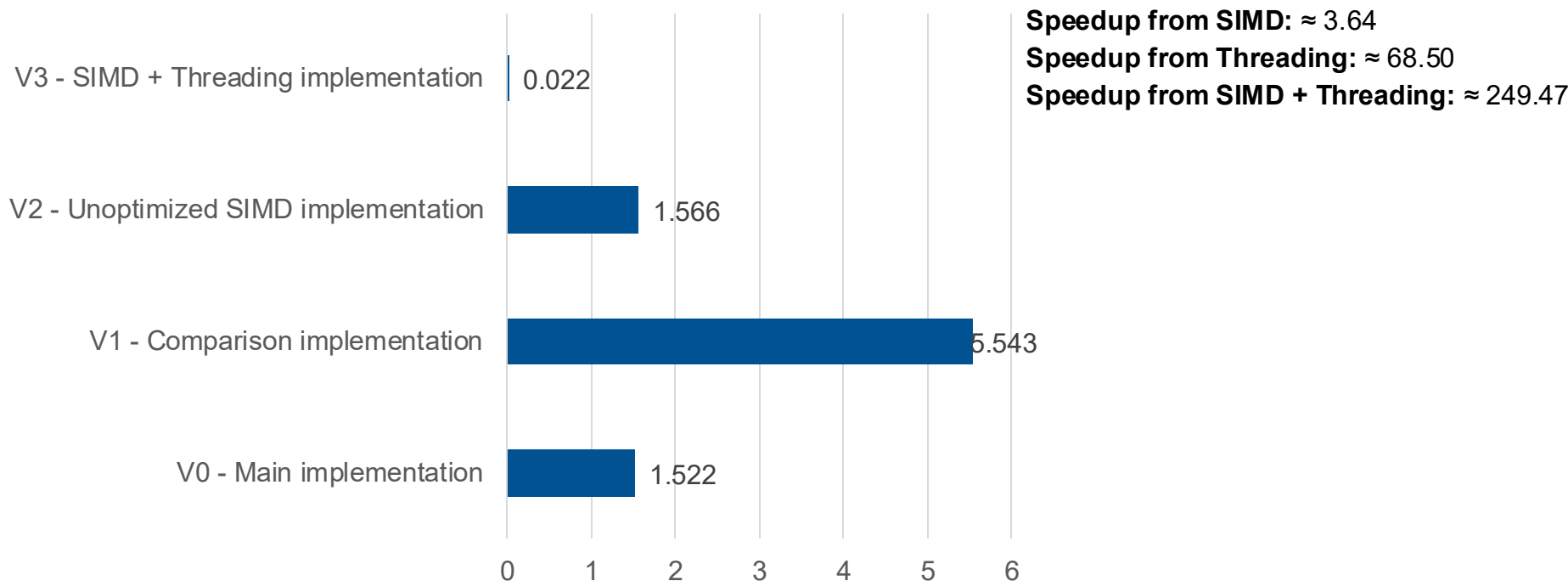
**Speedup from SIMD:**  $\approx 3.59$

**Speedup from Threading:**  $\approx 18.79$

**Speedup from SIMD + Threading:**  $\approx 67.46$

## Test 4: Fully black image with high iterations

### What did we achieve with the different implementations?



# ***GRA – Tricorn fractal***

Thank you for your attention!

*Group t111:*

- *Nikola Granitsky*
- *Cem Temizkan*
- *Yashar Alan Zenker*

25.08.2026

