Algorithms for Julia Sets

Augusto Butkewitsch, Alex Tu

PURDUE UNIVERSITY | John Martinson Honors College

Introduction

Rendering the infinitely complex figures of fractals presents many computational hurdles, especially for real-time display in non-dedicated hardware.

In our research, we focus on exploring and optimizing rendering algorithms and techniques for escape-time fractals, such as Julia and Mandelbrot sets. Our aim is to create an interactive web platform for students to easily and dynamically explore these structures, enhancing their mathematical engagement.

Background

Fractals are often observed in our natural world, ranging from snowflake patterns to lightning bolts and even galaxies. They exhibit self-similarity and infinite complexity.

One well-known class of fractals, Julia Sets- the visualization of which is our current focus- is generated by iterating complex functions of the form:

$$f_c(z) = z^n + c$$

Typical rendering approaches involve iterating through each pixel on the screen as a coordinate in the complex plane. This proved to be computationally expensive, especially at higher resolutions. This prompted our project to facilitate visualizing these shapes and generating greater interest in their creation.

Initial Approach

- Pixels on screen represent coordinates on the complex plane
- Pixel color is determined by escape rate: how fast that coordinate, under a function, diverges.

Algorithm 1 Mandelbrot Function

```
function MandelbrotFunction(x, y)
    iteration ← 0
    z ← (0, 0)
    for i ← 0 to maxIterations do
        z ← z × z + (x, y)
        if |z| > {threshold} then
            break
        iteration ← iteration + 1
    end for
    return iteration
end function
```

Example pseudocode of a Mandelbrot function

- Higher resolution ⇔ Degrading performance

Lifting Algorithms

- As opposed to the noisy pixel based coloring, using a lifting algorithm is an alternative.
- Takes a backward combinatorial approach, and builds a structure of pre-images.
- Reveals the hidden shape of the Julia set by tracing how points could have changed, layer by layer.

Algorithm 2 Combinatorial Julia Graph via Path-Lifting

Require: $c \in \mathbb{C}$, an integer $N > 0$

$R \leftarrow \frac{1+\sqrt{1+4|c|}}{2}$
Choose $\beta > R$ and simple paths $d\colon [\beta^2 + c] \to [\beta]$, e_1, e_2 the upper/lower half–circles on $|z| = \beta$, all avoiding the post–critical set P_c.
▷ Initialize level–0 vertices at the circle $|z| = \beta$
$V \leftarrow \{\beta, -\beta\}$
$E \leftarrow \emptyset$
for $n \leftarrow 0$ **to** $N - 1$ **do**
  $V_{\text{next}} \leftarrow \emptyset$
  for all $v \in V$ **do** ▷ $f_c^n(v) = \pm\beta$ by construction
    ▷ Lift the radial path d one more iterate
    $\delta v \leftarrow \textsc{LiftPath}(f_c^{n+1}, v, d)$
    $E \leftarrow E \cup \{(v, \delta v)\}$
    if $f_c^n(v) = \beta$ **then**
      ▷ Lift upper half-circle e_1
      $\eta v \leftarrow \textsc{LiftPath}(f_c^n, v, e_1)$
    else
      ▷ Lift lower half-circle e_2
      $\eta v \leftarrow \textsc{LiftPath}(f_c^n, v, e_2)$
    end if
    $E \leftarrow E \cup \{(v, \eta v)\}$
    $V_{\text{next}} \leftarrow V_{\text{next}} \cup \{\delta v, \eta v\}$
  end for
  $V \leftarrow V_{\text{next}}$
end for
return (V, E)

- $\texttt{LiftPath}(g, v, \gamma)$ denotes the endpoint of the unique lift of the path γ starting at v under the covering map g.
- After N iterations, (V, E) is a finite approximation of the infinite combinatorial graph whose infinite walks encode points of the Julia set.

Pseudocode for Lifting Algorithm Implementation

Professor Mummert's image for the Lifting algorithm

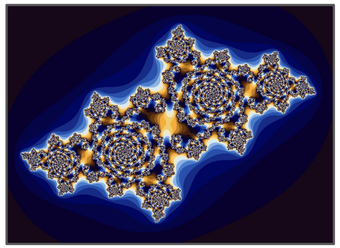
$$\begin{array}{ccccccc} v & \xrightarrow{f_c} & \cdots & \xrightarrow{f_c} & \pm\beta & \xrightarrow{f_c} & \beta^2 + c \\ \downarrow \tilde{\delta} & & & & \downarrow & & \downarrow d \\ \delta v & \xrightarrow{f_c} & \cdots & & \xrightarrow{f_c} & \beta & \end{array}$$

Lifting diagrams for δ (above) and η (below).

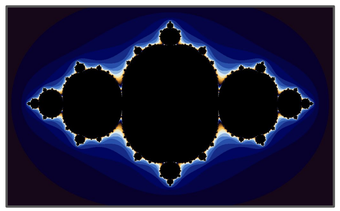
$$\begin{array}{ccccccc} v & \xrightarrow{f_c} & \cdots & \xrightarrow{f_c} & \pm\beta & \xrightarrow{f_c} & \beta^2 + c \\ \downarrow \tilde{\eta} & & & & \downarrow e_{1/2} & & \\ \eta v & \xrightarrow{f_c} & \cdots & \xrightarrow{f_c} & \mp\beta & \xrightarrow{f_c} & \beta^2 + c \end{array}$$

React JS

- 2D Julia Set visualizer using the iterative pixel coloring approach as mentioned above
- Used vanilla javascript for the computations, and React for the canvas visualizations
- UI allows for a varying complex parameter *c*, and the ability to pan and zoom in/out



c = -0.522 + 0.515i
maxIterations = 500, renders in 84ms



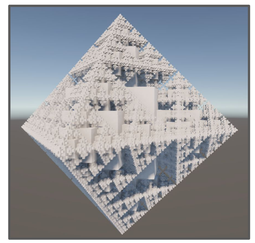
c = -0.75 + 0i
maxIterations = 500, renders in 278ms

- Pixel coloring algorithm based off of whether or not they escaped and how many iterations it took to "escape"
- Flaw: visualization through this method is finite; the computer loses precision at very small numbers

https://julia-set-visualizer.vercel.app/

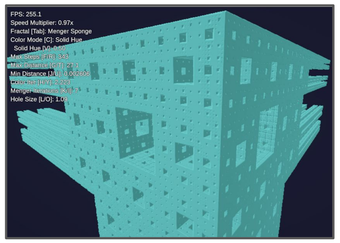
Unity C#

- The initial approach to rendering 3D fractals consisted of basic cube meshes placed and sized recursively.
- This yielded poor performance overall, alongside an expensive creation subroutine.
- Runs on local hardware, drawing polygons.

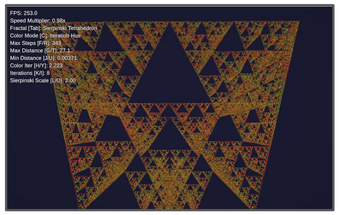


Fractal in Unity 3D: Cube Tetrahedron

Raymarching



Menger Sponge Fractal with Raymarching



Sierpiński Tetrahedron Fractal with Raymarching

- Casts a ray, iteratively "marches" a number of steps, within some distance of the nearest surface.
- More efficient for mathematical shapes such as fractals, with an undetermined polygon count.
- Runs on local hardware & web browser.
- Interactive online viewer, with control over different fractal parameters and custom viewports.

https://augusto-butkewitsch.github.io/raymarch

Conclusion

- Fractals provide an unique avenue to explore both the mathematical properties of recursion and several complex computer graphics & rendering techniques.
- There is a large variety of fractals that can be explored, both in two and three dimensions.
- Visualization approaches and their optimizations are largely case dependent.

Future Goals

- Our algorithms can surely be optimized further, allowing not only for faster rendering, but also for the visualization of more complex fractals.
- A greater, more interactive web interface for visualizing all our findings (and more fractals) in one place would aid us in our objective.

Acknowledgements

Our project would not have been possible without:

- Shengwei Qiu, our graduate TA
- Dr. Philip Mummert, our faculty mentor

References

[1] Johnson, A., Madden, K., & Sahin, A. (2013). Discovering Discrete Dynamical Systems. MAA Press.
[2] Mandelbrot, B.B. (2002). Chapter 3 Fractals , Graphics , and Mathematics Education 1.
[3] Churchill, M. (2004). Introduction to Fractal Geometry.