Introducción al fractal Mandelbrot Set y su representación en OpenGL3+

Roger Ramos Paredes

Septiembre 2020

Resumen

La geometría fractal es una rama muy importante actualmente para la ciencia, el estudio de los fractales ha traído muchos beneficios a la vida del hombre y uno de los más importantes es el conjunto de Mandelbrot, el cual es el fractal más investigado hoy en día ya que ayuda a comprender la naturaleza mediante un lenguaje matemático y esto permite a la ciencia dar grandes avances. En este artículo se detalla el uso del algoritmo conocido como “Escape Time” para representar el fractal del conjunto de Mandelbrot en OpenGL3+ con C++.

Abstract

Fractal geometry is a very important branch of science today, the study of fractals has brought many benefits to the life of man and one of the most important is the Mandelbrot set, which is the most investigated fractal today. that helps to understand nature through a mathematical language and this allows science to make great advances. This article details the use of the algorithm known as "Escape Time" to represent the fractal of the Mandelbrot set in OpenGL3 + with C ++.

Introducción

El conjunto de Mandelbrot y los conjuntos de Julia son conjuntos de puntos en el plano complejo. Los conjuntos de Julia fueron estudiados por primera vez por los matemáticos franceses Pierre Fatou y Gaston Julia a principios del siglo XX. Sin embargo, en este momento no había computadoras, y esto hacía prácticamente imposible estudiar la estructura del conjunto más de cerca, ya que se necesitaba una gran cantidad de potencia computacional. (Fredriksson, 2015)

Es estudio de los fractales es un campo de las matemáticas muy importante hoy en día, ha contribuido a obtener importantes logros en áreas como la medicina, la geología, el procesamiento de gráficos por computador, entre otros. El estudio de sus propiedades ha brindado a la ciencia una nueva forma de observar y comprender la naturaleza y su composición.

El fractal más conocido y estudiado en la actualidad lleva el nombre de Mandelbrot Set, cual fue descubierto por Benoit Mandelbrot cuando este trabajaba en IBM (1961) resolviendo el problema del ruido blanco que perturbaba la transmisión de las líneas telefónicas (IBM, 2015).

Los fractales han estado siempre en la naturaleza pero permanecieron invisibles al ser humano, hasta que pudimos entender que gran parte de la naturaleza no tiene cabida en la geometría euclidiana convencional, es para lo cual nace la geometría fractal, que nos permite analizar en la actualidad formas complejas como las nubes, las montañas, el sistema circulatorio, las líneas costeras o los copos de nieve los cuales son fractales naturales.

En este articulo nos concierne la representación eficiente del conjunto de Mandelbrot mediante la especificación estándar de OpenGL (Open Graphics Library) usando el lenguaje de programación C++, así como estudiar su complejidad algorítmica que tiene el calculo de los puntos y colores para cada punto del plano complejo a representar.

1. Preliminares

Fractal

El término “fractal” fue acuñado por el matemático Benoit Mandelbrot cuando investigaba sobre estos patrones que ocurrían en la naturaleza con el objetivo de entender el problema del ruido blanco que perturbaba la transmisión por las líneas telefónicas cuando trabajaba en IBM. (Valdés, 2016)

Algunos ejemplos muy conocidos sobre fractales pueden se mencionan a continuación:

Fractales naturales

Encontrados en la naturaleza, es decir, plantas, nubes o en el organismo del ser humano.

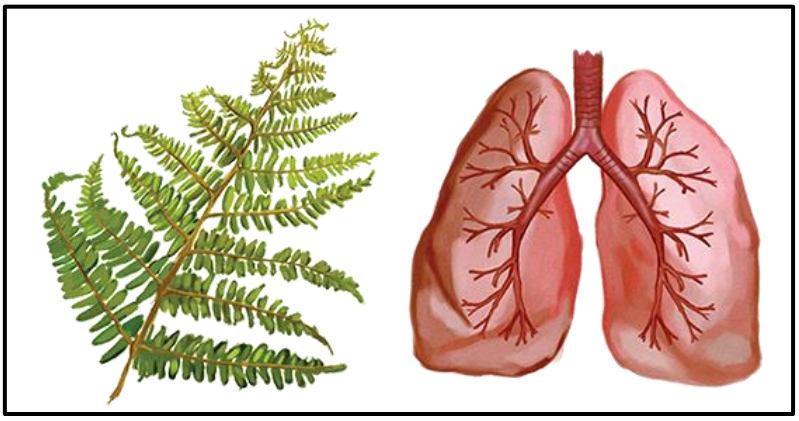


Figura 1. Naturaleza fractal de las ramificaciones de las plantas o dentro de los pulmones.

Fuente: Ricardo Armentano. Ingenierías de la Vida.

Fractales lineales

Generados a partir de iteraciones sobre figuras geométricas sencillas (rectas, triángulos, etc.). (Valdés, 2016)

La alfombra de Sierpinski

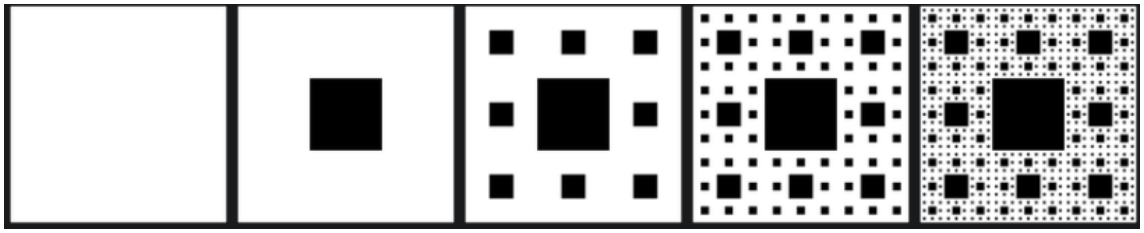


Figura 2. Iteraciones de la alfombra de Sierpinski

Fuente: Wikipedia.

Curva de Von Koch

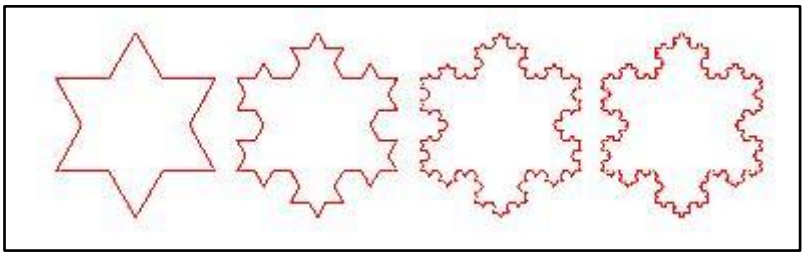


Figura 3. Iteraciones en la curva de Von Koch

Fuente: UPM.

Fractales no lineales

También denominados fractales 'caóticos' debido a que están formados por los números complejos. (Valdés, 2016)

Mandelbrot Set

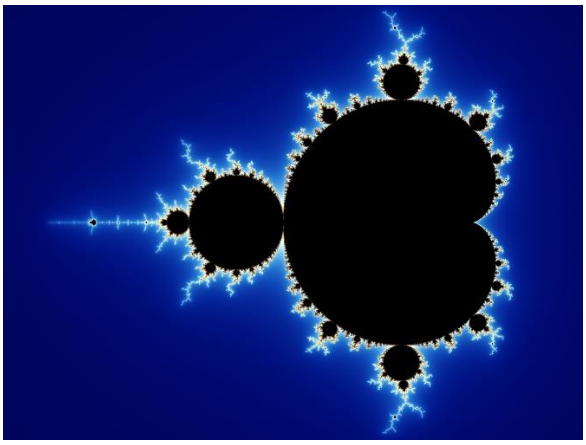


Figura 4. Fractal de Mandelbrot Set

Fuente: Wikipedia

Conjunto de Julia

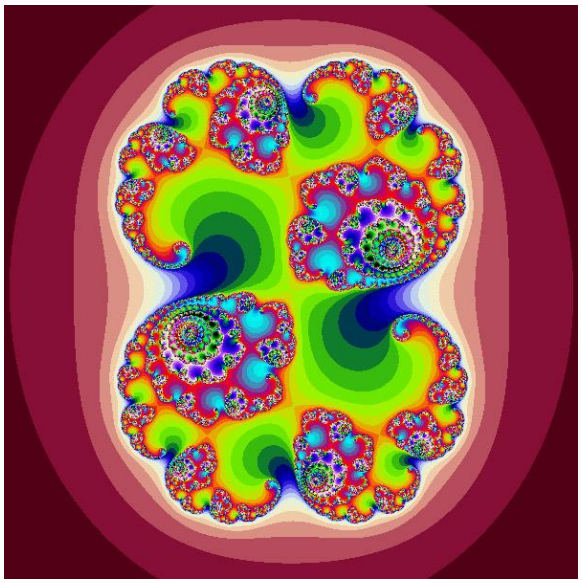


Figura 5. Fractal de Conjunto de Julia

Fuente: Wikipedia

Autosimilitud

Una propiedad muy importante para entender los fractales es la autosimilitud, la cual de forma sencilla se podría definir como una propiedad que tiene una forma la cual al ser acercada se encuentran formas similares a la anterior, repitiéndose esto por más que se acerque a la forma, en ese caso dicha forma podría denominarse fractal. (Valdés, 2016)

Podemos observar esta propiedad en el conjunto de Mandelbrot en las siguientes imágenes:

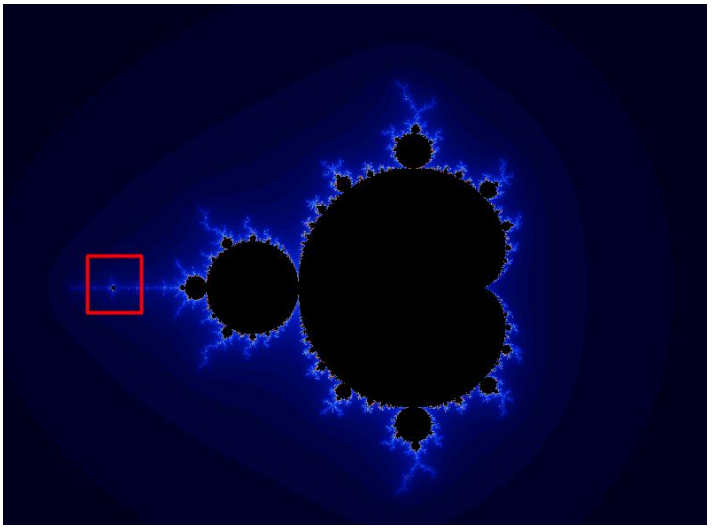


Figura 6. Mandelbrot Set, 150 iteraciones.

Fuente: Elaboración propia.

Si hacemos zoom en el recuadro rojo de la imagen anterior obtenemos una imagen muy similar a la anterior:

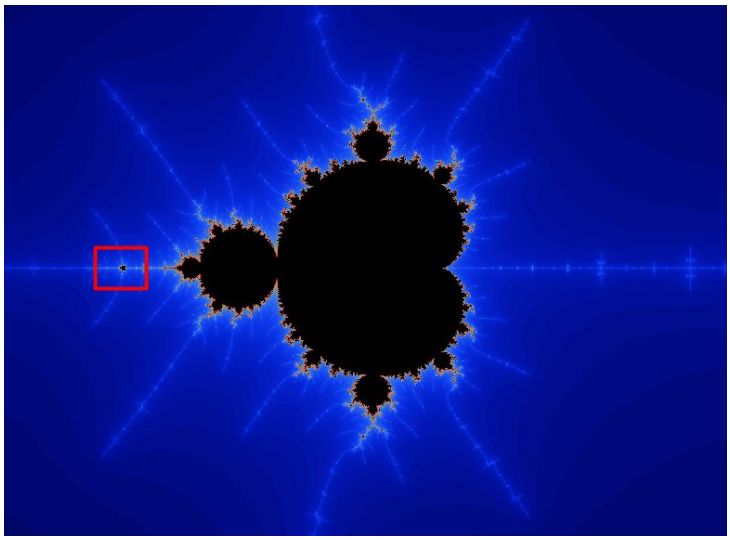


Figura 7. Mandelbrot Set (1° zoom), 150 iteraciones.

Fuente: Elaboración propia.

Si volvemos a hacer zoom en el recuadro rojo de la anterior imagen obtenemos la siguiente imagen:

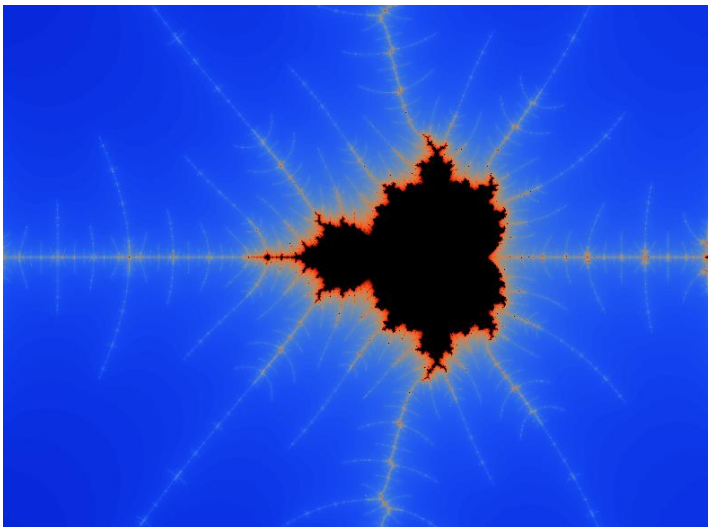


Figura 8. Mandelbrot Set (2° zoom), 150 iteraciones.

Fuente: Elaboración propia.

La imagen poco se parece a la anterior y la segundo, sin embargo aumentando la cantidad de iteraciones en el cálculo por computador podemos obtener un mejor resultado:

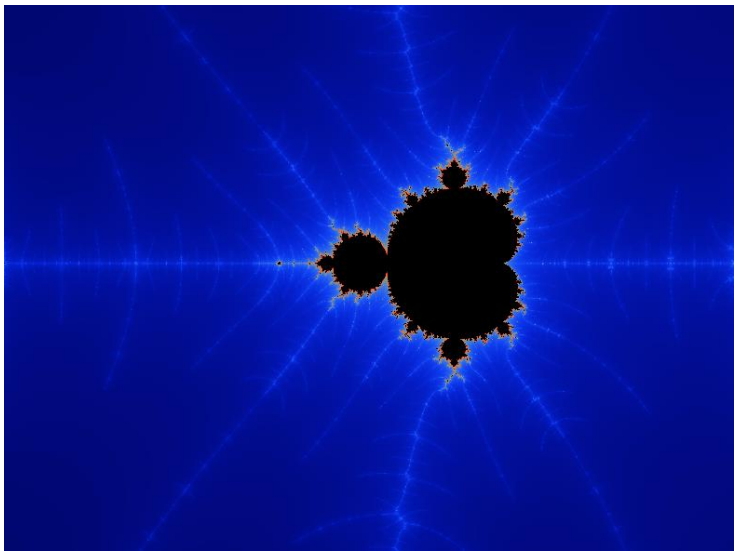


Figura 9. Mandelbrot Set (3° zoom), 400 iteraciones.

Fuente: Elaboración propia.

Esta es una pequeña visualización de la autosimilitud en el fractal de Mandelbrot Set, en donde claramente se observa que mientras hacemos zoom a la imagen esta contiene patrones que hacen que la misma imagen aparezca dentro de sí misma mientras hacemos más zoom, en el conjunto de Mandelbrot esto ocurre de forma indefinida, los computadores nos permiten representarlo hasta cierto límite lo cual está determinado por el número de iteraciones, lo que significa que podemos obtener aproximaciones de la representación del conjunto de Mandelbrot pero que sin embargo representarlo a su total exactitud conllevaría a considerar una cantidad infinita de iteraciones, lo cual sería imposible calcular por un computador debido a su gran pero limitada capacidad de cálculo.

2. El Conjunto de Mandelbrot

Para conocer si un número pertenece o no al conjunto de Mandelbrot hacemos uso de la siguiente ecuación recursiva:

- $Z_0 = 0 \in \mathbb{C}$ $\quad (término\ inicial)$
- $Z_{n+1} = Z_n^2 + c$ $\quad (sucesión\ recursiva)$

Decimos que un número complejo 'c' pertenece al conjunto de Mandelbrot set si demuestra que al iterarlo en la ecuación el resultado no diverge. Por ejemplo:

- Si c=1, obtenemos los resultados 0, 2, 5, 26, 677, ... de lo cual se nota claramente que tiende al infinito, entonces se demuestra que 1 no pertenece al conjunto de Mandelbrot
- Si c=-1 obtenemos los resultados 0, -1, 0, -1, ... lo cual sí está acotado por lo tanto se dice que -1 sí pertenece al conjunto de Mandelbrot.

Una propiedad es que para los números que se encuentren 2 o más unidades alejados del origen (0+0i) la sucesión diverge, por lo que ninguno de estos puede ser parte del conjunto de Mandelbrot. (Weisstein & W, s.f.)

3. Representación con OpenGL3+

Algoritmo de tiempo de escape

Para este algoritmo primero definimos una cantidad límite de iteraciones que calculará el computador para evaluar si el número pertenece o no al conjunto de Mandelbrot. Luego debemos establecer una condición la cual nos sirva de 'escape' la cual nos servirá para evitar cálculos innecesarios. Para establecer la condición de escape nos apoyaremos del enunciado anterior, es decir, nuestro algoritmo 'escapará' cuando el input sea un número que se encuentre 2 o más unidades alejado del origen.

Mediante OpenGL con uso del fragment shader accederemos a la posición de cada pixel, los cuales serán nuestro input para el algoritmo. Iteraremos el input en la función de Mandelbrot, está función es recursiva sin embargo la trabajaremos de forma iterativa ya que GLSL (el lenguaje de programación para shaders) no permite recursividad.

Además, tendremos una variable para guardar el número de iteraciones que se ejecutaron hasta que se dio la condición de escape o hasta que se realizaron el número máximo de iteraciones, esta variable nos servirá para elegir un color para el pixel el cual se tomó como input. Mientras más iteraciones se necesitaron hasta que se cumpla la condición de escape el color será más claro, mientras que los que con muy pocas iteraciones se dio la condición de escape tendrán un color más oscuro, así mismo los puntos alejados en 2 o más unidades del origen (los cuales cumplen la condición de escape de inmediato) tendrán un color negro.

Para el cálculo de las iteraciones debemos tener en cuenta que los valores que se recibirán como coordenadas de pixel representarán puntos en el plano complejo (eje-x para los números reales y eje-y para los números imaginarios), estos valores, siguiendo la ecuación de Mandelbrot, necesitaremos elevar al cuadrado, para no trabajar las operaciones con números complejos podemos hacer uso del siguiente artificio:

Sea el pixel de coordenadas (x, y), el valor de x representa la parte real y el valor de y la parte imaginaria de nuestro número complejo, entonces:

$$(x+y) \in \mathbb{C} \qquad (x: parte\ real, y: parte\ imaginaria)$$
$$x \in \mathbb{R}$$
$$y \in \mathbb{C} \rightarrow \exists\, y' \in \mathbb{R} \mid y = y' * i\ ;\ i\ =\ \sqrt{-1}$$
$$(x+y)\ =\ (x+y'.i)$$
$$(x+y)^2\ =\ (x+y'*i)^2 = x^2 + y'^2\ *\ i^2\ +\ 2*x*y'*i$$
$$(x+y)^2\ =\ x^2 + y'^2 * (-1)\ + 2*x*y'*i$$
$$(x+y)^2 =\ x^2 - y'^2\ +\ 2*x*y*i$$
$$(x+y)^2 =\ (k+w)\ , donde:$$

$$k \in \mathbb{R},\ k = x^2 - y'^2$$
$$w \in \mathbb{C},\ w = 2 * x * y * i$$

Entonces el resultado de iterar el número complejo $c=x+y$ en la función de Mandelbrot se puede calcular de la siguiente manera:

$$Z_0 = 0 \in \mathbb{C}$$
$$Z_1 = Z_0^2 + c = 0^2 + c = x + y$$
$$Z_2 = Z_1^2 + c = (x+y)^2 + x + y = (k+w) + x + y = (k+x) + (w+y)$$
...

Lo dicho anteriormente lo podemos representar en el siguiente pseudocódigo

```
CONSTANTE ENTERO MAX_ITER = 100
FUNCIÓN Mandelbrot(FLOTANTE x, FLOTANTE y): RETORNA ENTERO
    ENTERO contIter ← 0
    FLOTANTE k ← 0
    FLOTANTE w ← 0
    MIENTRAS(contIter<MAX_ITER y k*k + w*w < 4):
        FLOTANTE actK ← k
        k ← k*k – w*w + x
        w ← 2*actK*y + y
        contIter ← contIter + 1
    FIN MIENTRAS
    RETORNAR contIter
FIN FUNCIÓN
```

La complejidad de la función Mandelbrot estaría en O(n), dado que podemos considerar la multiplicación del dos números de tipo flotante como O(1).

Dado que esta función se llama para cada pixel de nuestra pantalla, la complejidad total del cálculo para un frame estaría en el orden de O(n*ANCHO*ALTO) = O(n^3) donde ANCHO y ALTO representan la altura y anchura de nuestra ventana (en pixeles). Debemos tener en cuenta que la complejidad el rendimiento de la aplicación puede variar mucho desde donde estemos enfocando con la cámara de OpenGL3+, por ejemplo si en nuestra ventana estamos enfocando un área en donde todos o la mayoría de puntos estén más de dos unidades del origen entonces el calculo sería muy simple porque todos estos puntos escaparían en la primera iteración por lo que en promedio la función Mandelbrot tendría una complejidad

de O(1) de esa manera el calculo total aproximado del frame sería casi del orden O(n^2) y dado que n representa las dimensiones de una pantalla que por lo general son pequeñas (800, 1600, etc) podemos notar que el cálculo es muy rápido, sin embargo mientras en nuestra pantalla aparezcan puntos en donde el algoritmo de tiempo de escape se tarde más en 'escapar' el cálculo del frame se volverá cada vez más lento, a continuación se puede ver una comparativa:

Las siguientes pruebas muestran resultados del algoritmo de tiempo de escape en una ventana de GLFW de resolución 800 x 800, y con OpenGL3+ para acceder a las funciones del controlador de video. Para los siguientes ejemplos se usó 300 iteraciones. (IEEE, 2008)

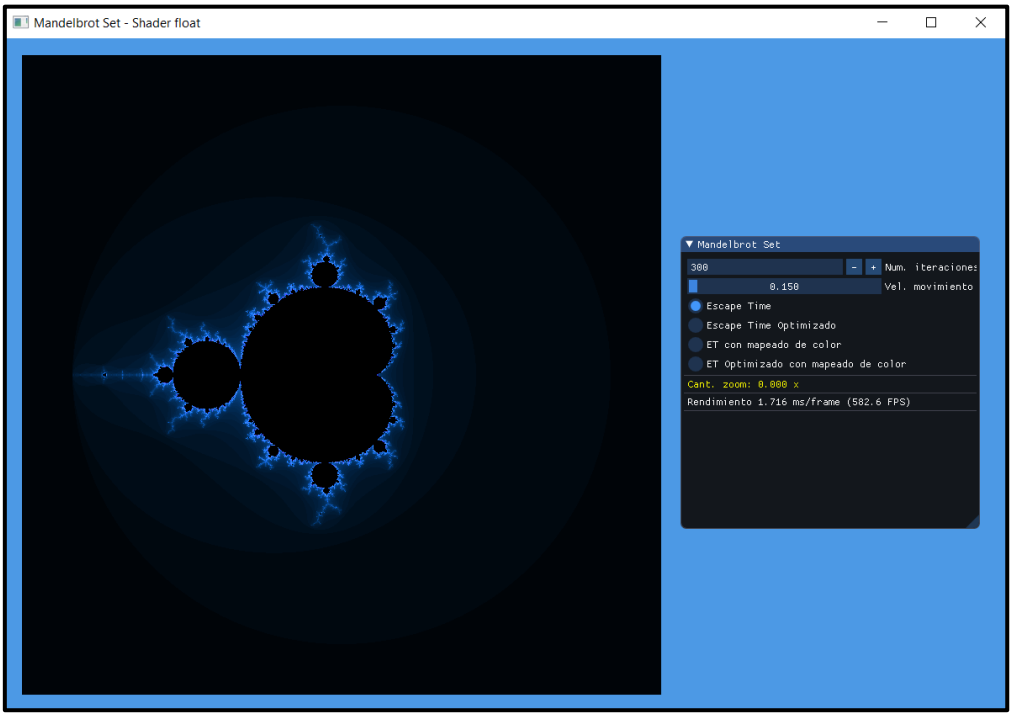


Figura 10. Enfocando totalidad del conjunto. 600 FPS (aprox.)

Fuente: Elaboración propia.

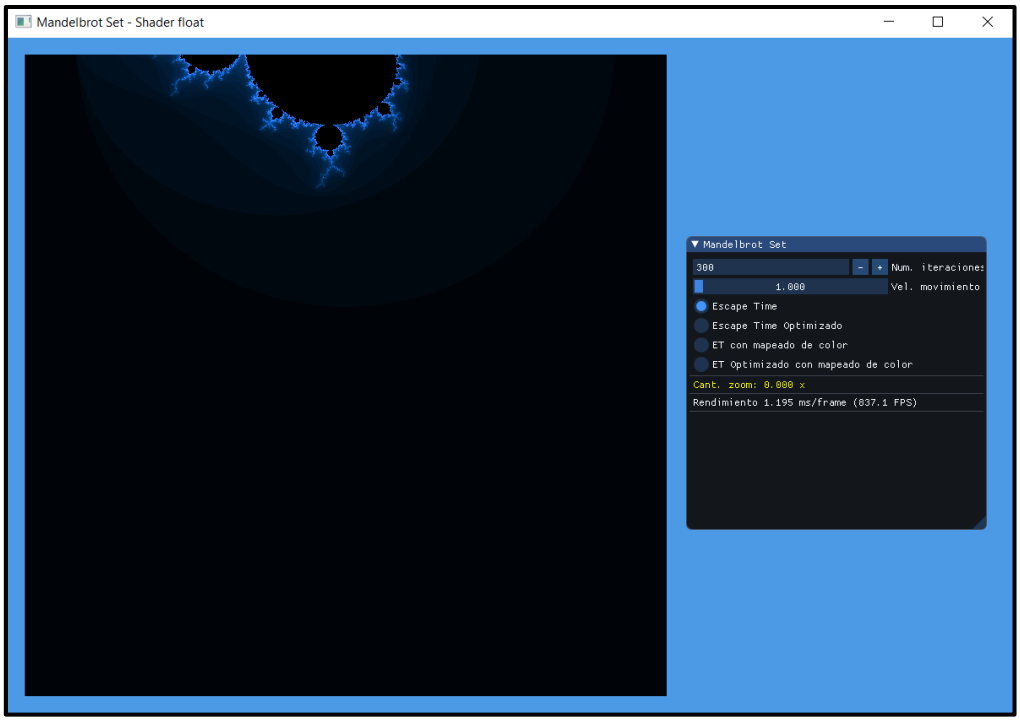


Figura 11. Enfocando 45% (aprox.) del conjunto. 850 FPS (aprox.)

Fuente: Elaboración propia.

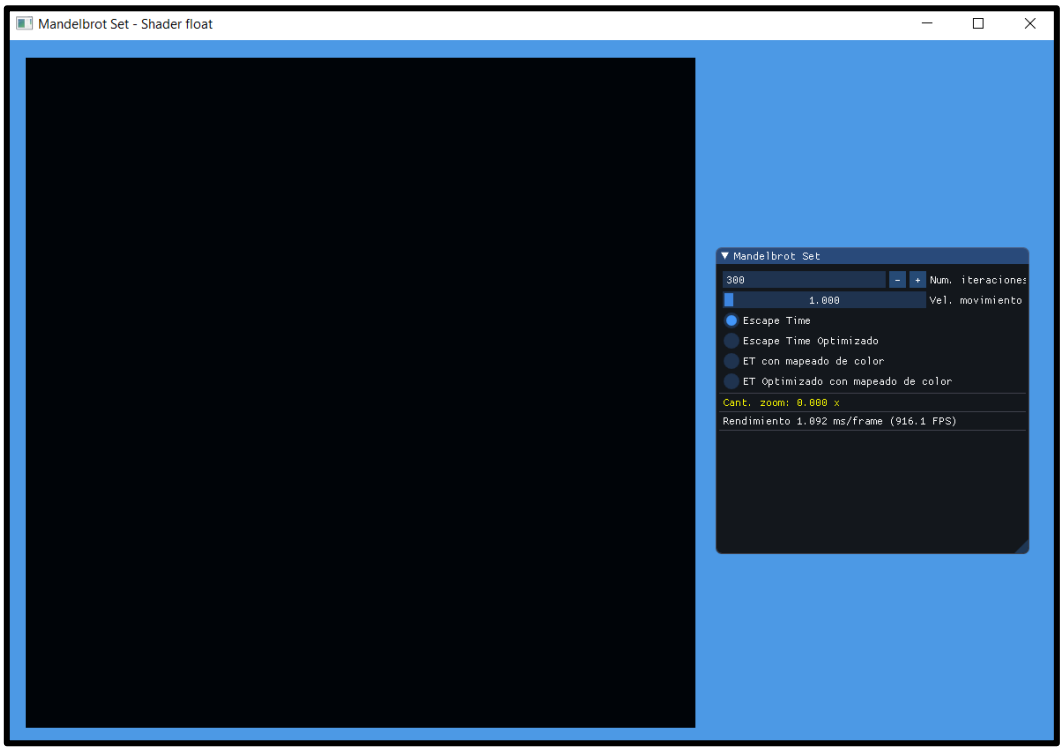


Figura 12. Enfocando 0% del conjunto (todos los puntos escapan con 1 iteración). 900 FPS (aprox.)

Fuente: Elaboración propia.

En el algoritmo anterior usamos para la precisión de los decimales el tipo de dato flotante según la documentación del GLSL siguen el estándar IEEE 754, es decir tienen capacidad de almacenamiento de 32 bits (8 bytes), esto es una limitante cuando para cuando se quiera profundizar mucho dentro de la vista del conjunto, ya que mientras más zoom las coordenadas contienen más y más decimales, y en el punto en que la capacidad del tipo flotante no sea suficiente el calculo de los puntos del conjunto de Mandelbrot se volverá más inexacto, por ejemplo para el mismo programa mostrado podemos explorar hasta un zoom de 10X aprox.

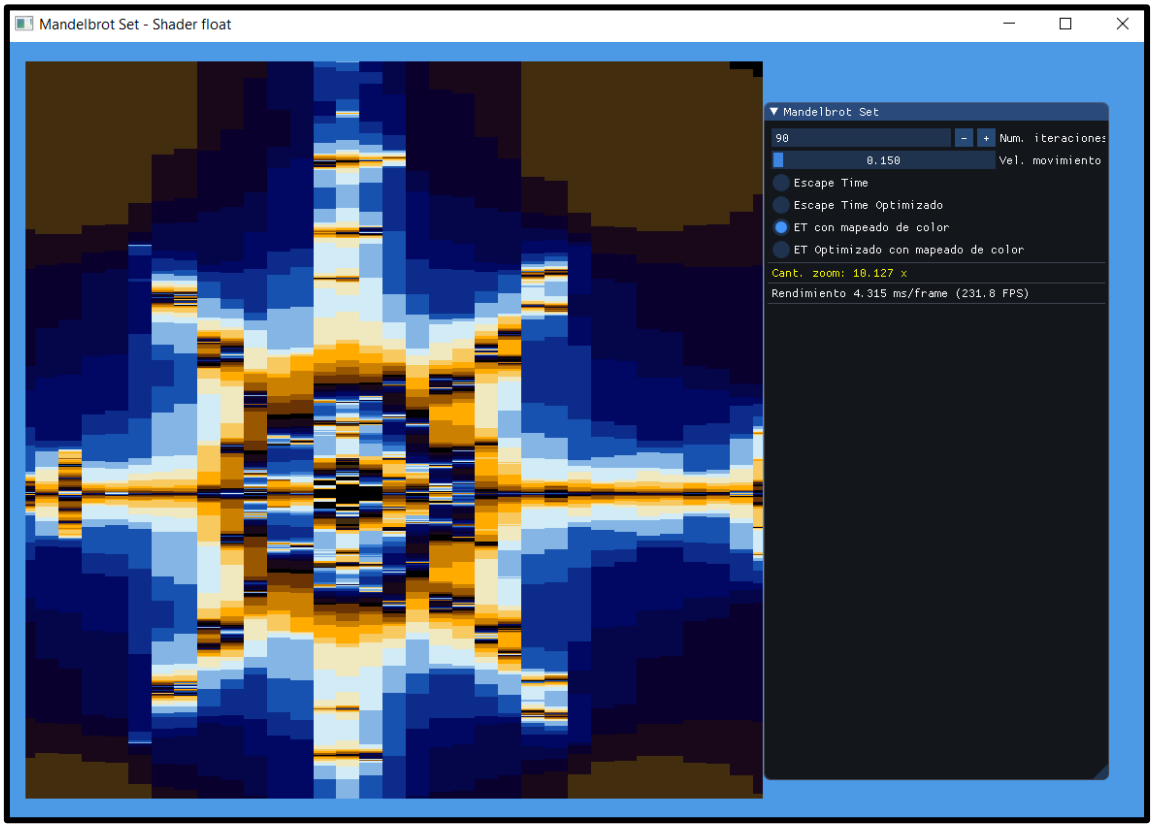


Figura 13. Zoom 10X usando precisión flotante para el cálculo.

Fuente: Elaboración propia.

Podemos notar a simple vista como comienza a tener un aspecto "pixelado" debido a que el tipo de dato flotante no tiene la capacidad de almacenar completamente los decimales hasta ese nivel, entonces el cálculo de los puntos del conjunto se vuelve muy impreciso justamente por esta perdida de decimales que el tipo flotante no pudo almacenar.

Podemos mejorar el cálculo anterior cambiando el tipo de dato de precisión flotante a precisión doble ("double") los cuales según la documentación de GLSL sigue el estándar IEEE-754, es decir tiene capacidad de 64 bits de almacenamiento, usar este tipo de dato nos permitirá poder calcular puntos que

se encuentren a más profundidad dentro del conjunto ya que nos permitirá almacenar más decimales que en tipo flotante, mejorando la precisión de cálculo, esto podremos verlo en la siguiente imagen:

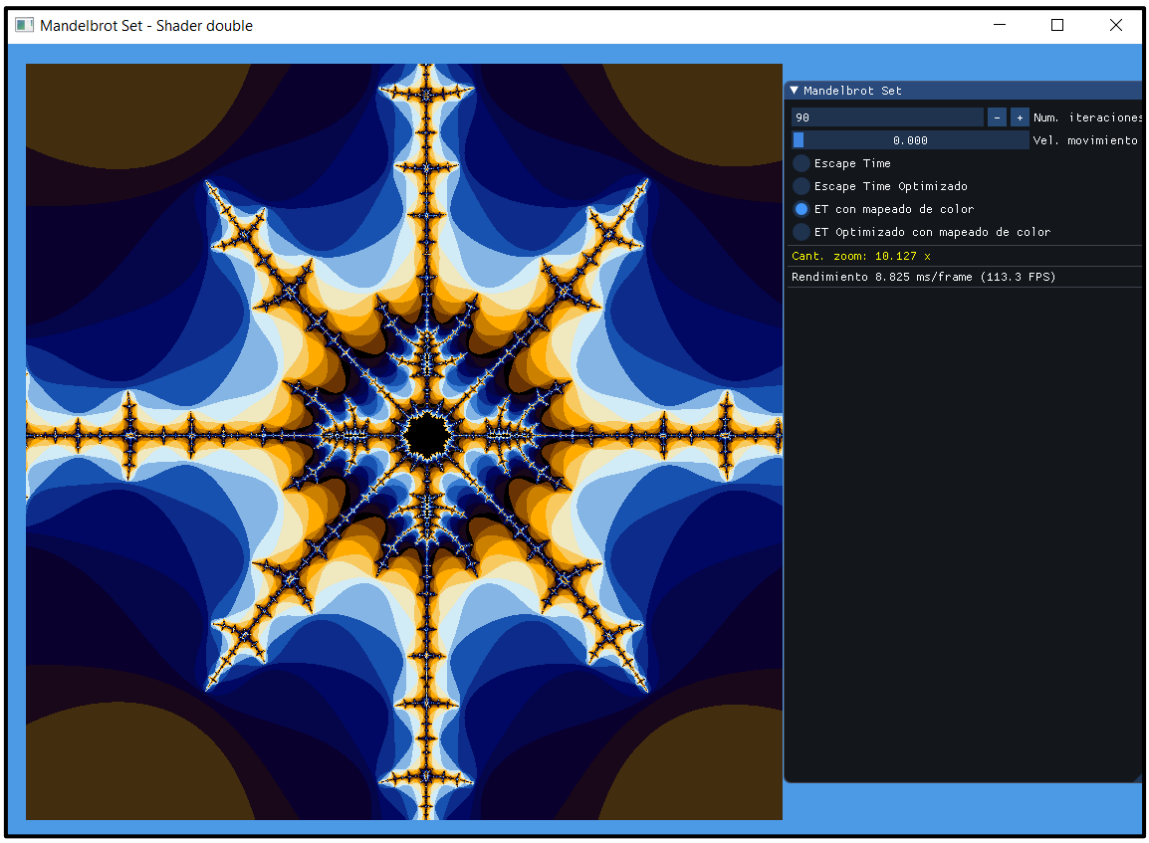


Figura 14. Zoom 10X usando precisión doble para el cálculo.

Fuente: Elaboración propia.

De todas maneras el tipo de doble precisión puede resultarnos muy limitado si tratamos de explorar con un zoom mayor a 24X, para lo cual el tipo de dato de doble precisión ya no es suficiente para un correcto cálculo de los puntos del conjunto de Mandelbrot. A continuación se muestra el resultado de explorar con un zoom de 24X:

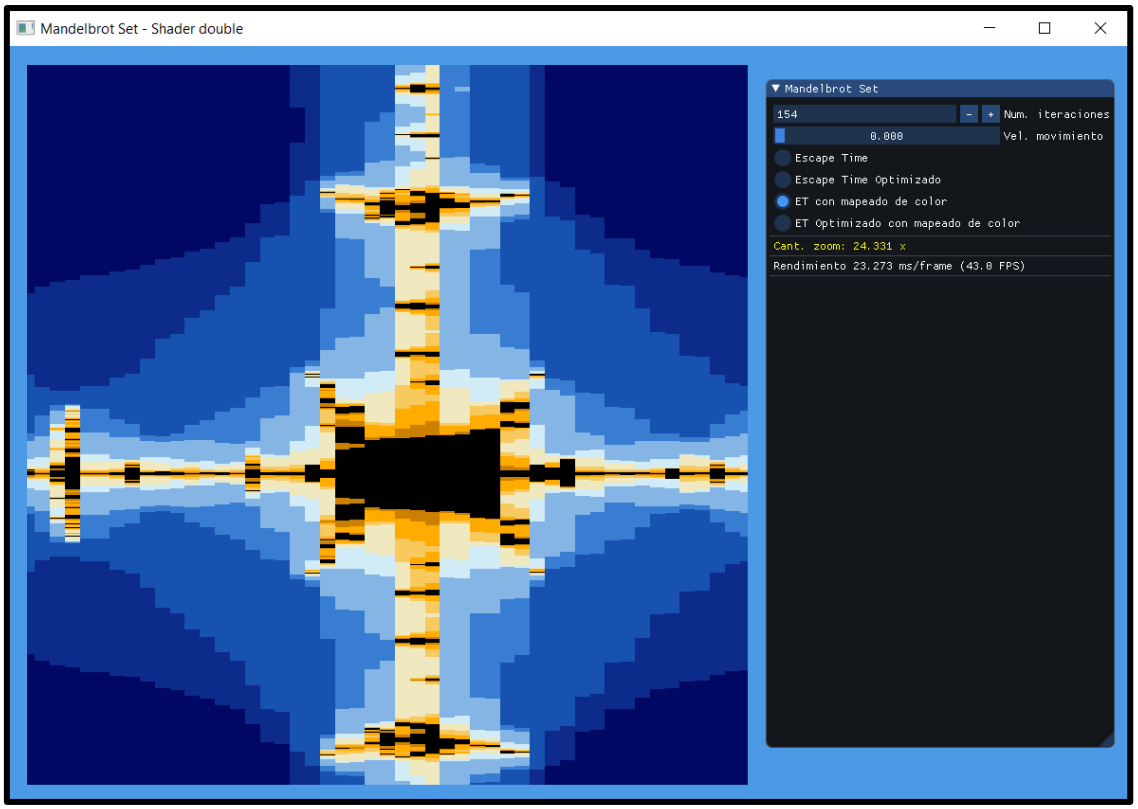


Figura 15. Zoom 24X usando precisión doble para el cálculo.

Fuente: Elaboración propia.

Una posible solución podría ser emplear una estructura que permita almacenar más números de forma eficiente y para la multiplicación podría usarse el algoritmo de Karatsuba que usa el enfoque "divide y vencerás" y tiene una complejidad temporal en el orden de $O(n^{log_2 3}) = O(n^{1.585})$ (Eyupoglu, 2015) pero esta solución es más compleja y no se desarrollada en este artículo.

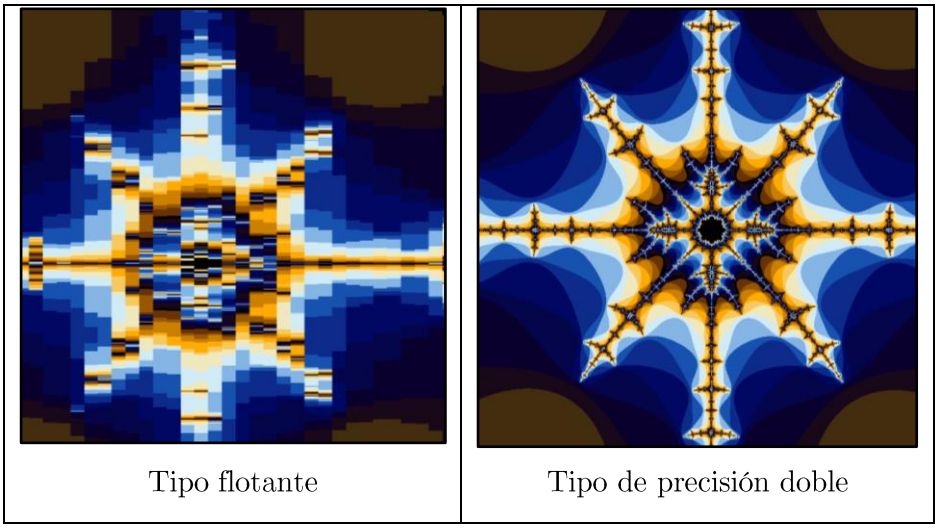
Hay una manera de volver más eficiente el algoritmo de tiempo de escape, aunque esta eficiencia es insignificante cuando trabajemos con precisión doble o flotante, sin embargo puede ser muy útil si se desea desarrollar la idea propuesta en el párrafo anterior. La idea es básicamente almacenar cálculos para no repetir innecesariamente la cantidad de cálculos matemáticos, en este caso podemos tratar de reducir la cantidad de multiplicaciones de la siguiente manera:

```
CONSTANTE ENTERO MAX_ITER = 100
FUNCIÓN Mandelbrot(FLOTANTE x, FLOTANTE y): RETORNA ENTERO
        ENTERO contIter ← 0
        FLOTANTE k ← 0
        FLOTANTE w ← 0
        FLOTANTE kCuad ← 0
        FLOTANTE wCuad ← 0
        MIENTRAS(contIter<MAX_ITER y kCuad + wCuad < 4):
                w ← (w+w)*k + y
                k ← kCuad – wCuad + x
                kCuad ← k*k
                wCuad ← w*w
                contIter ← contIter + 1
        FIN MIENTRAS
        RETORNAR contIter
FIN FUNCIÓN
```

Esta versión de tiempo de escape 'optimizada' se diferencia de el anterior algoritmo en la cantidad de multiplicaciones. El algoritmo anterior usaba 6 multiplicaciones mientras que la versión optimizada solo 3 multiplicaciones, para casos donde la multiplicación es O(1) como cuando usamos precisión flotante o doble esta eficiencia es insignificante, sin embargo cuando usa, por ejemplo, el algoritmo de Karatsuba donde la complejidad de la multiplicación está en el orden $O(n^{1.585})$ puede ayudarnos a acelerar los cálculos. Este enfoque usa más almacenamiento ya que necesita dos variables más para guardar los cuadrados de k y w, llamadas kCuad y wCuad, esto con el propósito de no repetir el proceso de multiplicación entre w y w o de k y k.

4. Resultados

El algoritmo de "Tiempo de escape" es un algoritmo eficiente para representar el conjunto de Mandelbrot con OpenGL3+, tanto si se usan como tipo de dato la precisión doble o flotante para los valores calculados en las iteraciones. La precisión doble permite explorar el conjunto de Mandelbrot hasta un zoom de 10X (aproximadamente) mientras que si usamos una precisión de tipo doble nos permite explorar el conjunto a una profundidad de 24X (aproximadamente 140% más que con precisión flotante). El tipo de dato flotante o de doble precisión solo influye en los resultados en el aspecto de la nitidez de la imagen, ambos más allá de los límites establecidos de zoom se obtendrán resultados con un aspecto "pixelado" debido a la incapacidad de estos de almacenar cantidad de decimales grandes. Sin embargo, ambos tipos de datos tienen el mismo rendimiento en fotogramas por segundo (FPS).



Tipo flotante | Tipo de precisión doble

Figura 16. Resultados con zoom 10X

El rendimiento en fotogramas por segundo (FPS) usando el algoritmo de tiempo de escape se muestra en la siguiente tabla, teniendo en cuenta que para las mediciones se enfocó en la totalidad del conjunto de Mandelbrot (-2<x, y<2).

Iteraciones	Rendimiento (ms/frame)	FPS
90	3.5	280
500	60.6	150
800	11.8	84
3000	28.0	36
7000	79.7	12.5
10000	112.4	9
30000	248.3	4

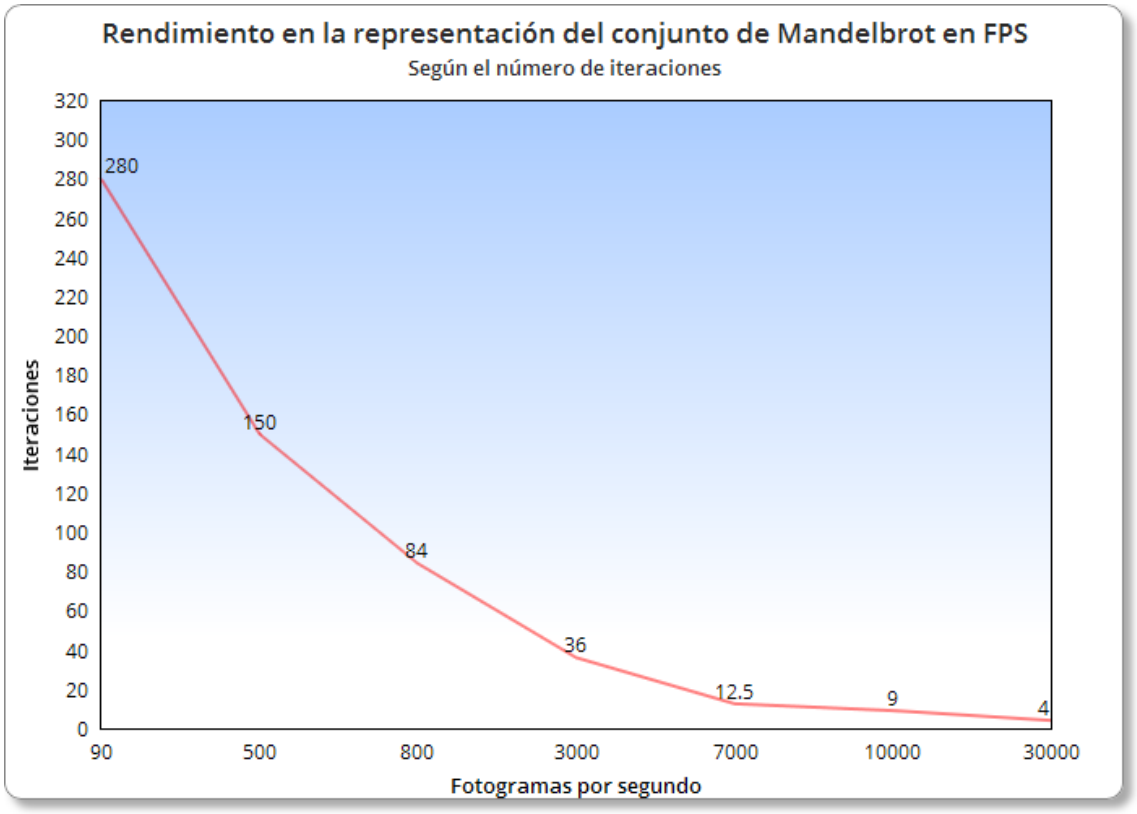


Figura 17: Gráfica de la cantidad de fotogramas por segundo según el número de iteraciones con el algoritmo de tiempo de escape.

Referencias

Alonso-Sanz, R. (2014). Scouting the Mandelbrot set with Memory. *Research Article*.

Armentano, R. (2019). *Fractales y pulmones*. Obtenido de https://losingenierosdelavida.blogspot.com/2019/04/fractales-y-pulmones.html

Chu, P. P. (2012). Mandelbrot Set Fractal Accelerator.

Eyupoglu, C. (2015). Performance Analysis of Karatsuba Multiplication Algorithm for Different Bit Lengths. *ScienceDirect*.

Falconer, K. (2005). Iteration of Complex Functions—Julia Sets.

Fredriksson, B. (2015). *An introduction to the Mandelbrot set.*

Hertling, P. (2004). Is the Mandelbrot set computable ?

IBM. (2015). *Icons of progress - Fractal geometry.*

IEEE. (2008). *IEEE 754 Standard for Floating-Point Arithmetic.* Obtenido de https://standards.ieee.org/content/ieee-standards/en/standard/754-2019.html

Khronos. (2014). *OpenGL*. Obtenido de Reference Pages: https://www.khronos.org/registry/OpenGL-Refpages/gl4//

Khronos. (2019). *OpenGL Wiki*. Obtenido de Data Type (GLSL): https://www.khronos.org/opengl/wiki/Data_Type_(GLSL)

UPM. (s.f.). La Alfombra de Sierpinski.

UPM. (s.f.). La curva de Koch.

Valdés, V. (2016). *Introducción a la geometría fractal.*

Weisstein, & W, E. (s.f.). *Mandelbrot Set*. Obtenido de MathWorld--A Wolfram Web Resource: https://mathworld.wolfram.com/MandelbrotSet.html