wxChaos

1.0.0

Manual de usuario

Índice general

Capítulo 1

Introducción

wxChaos es un programa generador de fractales cuyos objetivos son proporcionar al usuario la capacidad de explorar el mundo de la geometría fractal, y también proveer una interfaz para crear nuevos fractales bien mediante fórmulas o bien mediante un lenguaje de programación. La intención es proveer una herramienta para aprender qué son los fractales, cómo se dibujan y darle al usuario la capacidad de crear sus propios fractales.

Existen ya muchos programas en la red para dibujar fractales, programas que también son de código abierto. Ante eso podríamos preguntarnos, ¿Qué tiene *wxChaos* que ofrecer? Una de las características es la gran cantidad de fractales disponibles para dibujar junto con una variedad de algoritmos para explorar sus diferentes representaciones. Otra de las características es la inclusión de un lenguaje de programación para que el mismo usuario pueda aprender a programarlos.

wxChaos es una herramienta gratuita que se distribuye bajo una licencia de código abierto, lo cual quiere decir que cualquier persona puede inspeccionar, usar y modificar el código fuente del programa. Para más información véase la sección de licencia.

Capítulo 2

¿Qué es un fractal?

De MathWorld:

> "Un fractal es un objeto o cantidad que muestra la propiedad de auto-similaridad en todas las escalas. El objeto no necesita mostrar la misma estructura en todas las escalas, sino el mismo "tipo" de estructura."
>
> Un ejemplo sencillo e ilustrativo de fractal es el triángulo de Sierpinski, que se forma dividiendo sucesivamente un triángulo.

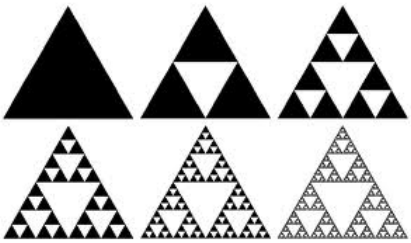

Figura 2.1: Triangulo de Sierpinsky

Al hacer un procedimiento como este infinitamente se está haciendo un fractal, ya que que si se hace un aumento al triángulo tiene la misma forma que el triángulo original, es decir, es auto-similar. Ejemplos similares se podrían hacer con cuadrados, líneas y otros tipos de figuras.

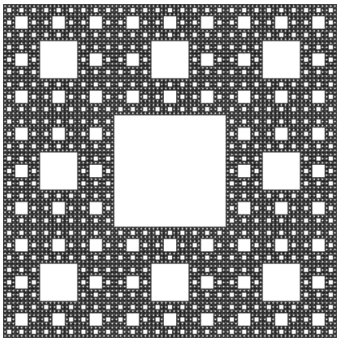

Figura 2.2: Alfombra de Sierpinsky

Al analizar el objeto se puede observar un hecho un tanto paradójico: ¡El objeto no tiene área alguna! Y sin embargo es un objeto que se representa en un plano bidimensional (como la pantalla de la computadora). Se dice que el objeto no posee área por que en el caso de que se hicieran infinitas subdivisiones no quedaría ningún triángulo, y por lo tanto nada que dote al objeto de área. Ahora imagínese que toma el resultado de hacer infinitas divisiones en el

triángulo, lo agarra por dos de sus lados y lo estira. Ocurriría algo más paradójico aún: Su longitud sería infinita.

La razón de esto subyace en el hecho de que un fractal no necesariamente tiene dimensiones enteras como las que conocemos (1 dimensión para la línea, 2 dimensiones para el área y 3 para el espacio), sino que puede tener dimensiones fraccionarias. En el caso del triángulo de Sierpinski el objeto tiene una dimensión 1,58496, razón por la cual no se puede describir el triángulo en términos de área (no alcanza las 2 dimensiones), pero tampoco en términos de longitud (tiene más de 1 dimensión). A este número se le conoce como dimensión fractal.

Es por el mismo hecho de que los fractales tienen dimensiones fraccionarias que se pueden realizar infinitos aumentos sobre estos y aún así ver una variedad de figuras autosemejantes.

Otro tipo más complejo de fractal surge al graficar una ecuación en un plano complejo, que muestra estructuras mucho más complejas e imprevisibles. Este es el tipo de fractal con el que trabaja *wxChaos*. Ejemplos de fractal graficado en el plano complejo son el conjunto de Mandelbrot y el conjunto de Julia. En siguientes capítulos se dará una explicación acerca de las fórmulas complejas.

Capítulo 3

Interfaz del programa

El programa tiene una interfaz simple donde se muestra principalmente el área del trazado de fractal. En la esquina superior izquierda se muestra el número de iteraciones que utiliza el fractal. Las iteraciones podrían pensarse como los pasos que realiza el programa para trazar el fractal. En el ejemplo anterior del triángulo de Sierpinsky cada vez que se dividía el triángulo se estaba realizando una iteración. En el caso de un fractal complejo cada iteración significa un cálculo. Cuantas más iteraciones se utilicen más detallada será la imagen generada, pero tardará más tiempo en crearse ya que aumentarán los cálculos que tiene que hacer la computadora. Otra cosa que también puede afectar el rendimiento es el tamaño de la ventana. Cuanto más grande sea la ventana más operaciones tendrá que realizar, ya que el fractal tendrá mayor resolución.

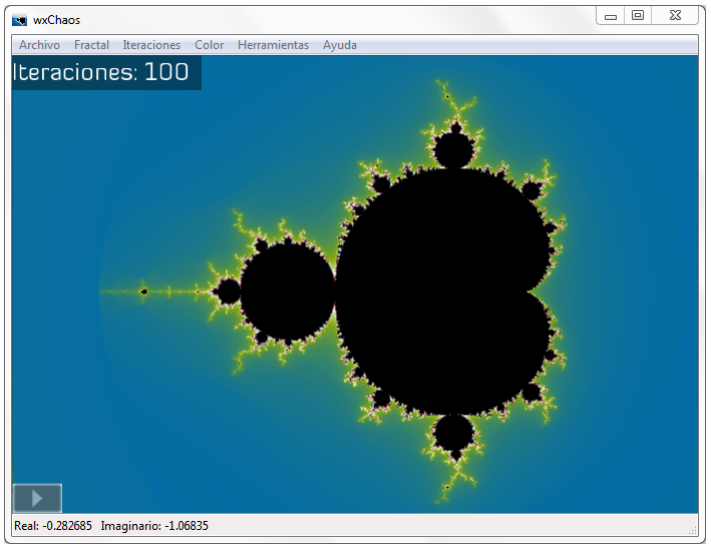

Figura 3.1: Interfaz del programa

En la parte inferior de la ventana principal se muestran dos etiquetas, *real* e *imaginario*. Esto se refiere a la posición del puntero dentro del plano del fractal. El plano del fractal es un plano complejo, donde su eje horizontal se llama eje real y el eje vertical se llama eje imaginario. No todos los fractales usan este plano complejo, así que las etiquetas pueden variar según el fractal.

Para realizar un aumento en la imagen basta con seleccionar un rectángulo donde se desee aumentar la imagen. Para anular el aumento hágase un click derecho. A continuación ser dará una explicación detallada de cada uno de los elementos del menú.

3.0.1. Fractal

Fórmula Cada fractal esta descrito por una fórmula para trazarlo. En este menú se pueden seleccionar diferentes fórmulas que dan lugar a fractales con formas diferentes.

Introducir constante Julia Algunos fractales que son de la variedad Julia (esto se explicará posteriormente) dependen de un número que es una constante k. En este elemento se proveen menús para introducir esta constante. El método manual es útil cuando se desea introducir un número con mucha precisión, y el slider para cuando se desea tener una vista rápida de como cambia el fractal al variar la constante. El slider funciona tomando la constante k del punto al que apunta dentro del plano complejo del área de dibujado del fractal.

Modo Julia Algunos fractales de la variedad Mandelbrot como Mandelbrot $z = z^2 + c$, Mandelbrot $z = z^n + c$, Manowar y Burning Ship tiene la opción de poder crear una ventana con su variante Julia. Al hacer esto aparecerá una cruz de selección en la ventana principal que permite escoger la constante k, exactamente igual que con el deslizador.

Mostrar órbita Esta opción permite visualizar la órbita que realiza el punto al ser iterado. Esto tiene que ver con la forma en la que se trazan los fractales que se explicará posteriormente. La órbita tendrá un color rojo si diverge, es decir, que no pertenece al conjunto y órbita verde si converge.

Fórmula de usuario Permite al usuario introducir sus propias fórmulas.

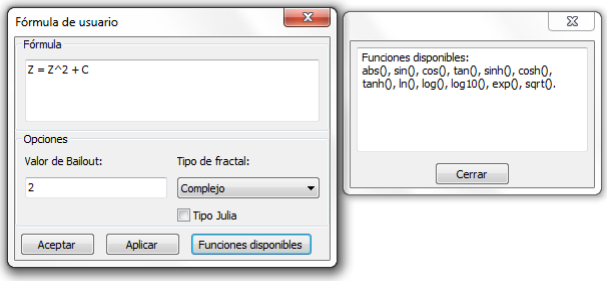

Figura 3.2: Fórmula de usuario

Las fórmulas de usuario permiten crear dos tipos de fractales: Fractales complejos y fractales creados a partir del método numérico conocido como *Punto fijo*. El *Bailout* se refiere a la posición que debe adquirir un punto de la órbita para determinar que no pertenece al conjunto fractal (y asi pintarlo de negro). Este menú también permite seleccionar si el fractal será de la variedad Julia. Las fórmulas deberán ser escritas utilizando las variables Z y C o z y c. En caso de que exista algún error en la fórmula este será descrito en la consola.

Opciones del fractal Algunos fractales poseen parámetros modificables que pueden cambiarse en este panel.

3.0.2. Iteraciones

Cambiar iteraciones Permite al usuario establecer un número especifico de iteraciones o aumentar y disminuir unidades de iteraciones.

Aumentar iteraciones Aumenta un número predeterminado de iteraciones. El número estándar son 20 iteraciones, pero algunos fractales como *Péndulo doble* pueden cambiar más iteraciones.

Disminuir iteraciones Disminuye un número predeterminado de iteraciones.

3.0.3. Opciones de color

A la hora de trazar el fractal este no contiene ninguna información específica de como se deben dibujar los colores, más bien se utiliza un algoritmo para interpretar qué colores deben utilizarse en base a alguna operación matemática.

wxChaos utiliza dos métodos para crear la paleta de colores. La forma estándar es utilizando un gradiente. En este menú el usuario puede escoger los colores que desee utilizar para crear un gradiente personalizado. Otra forma de crear la paleta es utilizar el color *EST*, que también es capaz de crear resultados interesantes.

Para poder utilizar las paletas de color el fractal necesita un algoritmo para poder seleccionar cada color. A continuación se muestran los algoritmos implementados.

Tiempo de escape El algoritmo de tiempo de escape es el algoritmo más tradicional y sin duda el más rápido en la mayoría de los casos. Este algoritmo mide cuantas iteraciones se tardó en determinar si un punto pertenece al conjunto fractal o no, y colorea según el número de iteraciones. Es por esto que por defecto produce colores discontinuos. Para evitar esto se puede seleccionar la opción de *smooth render*, la cual trata de reducir este efecto. Otra de las opciones son las trampas orbitales; estas miden cuanto se ha acercado cada punto a los ejes verticales y horizontales durante su órbita y colorea acorde a la distancia mínima a la cual llegaron.

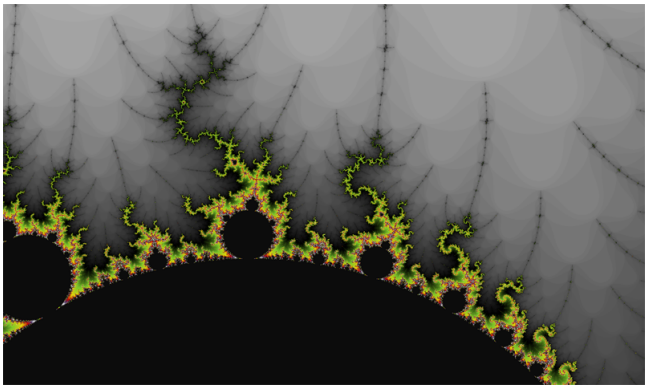

Figura 3.3: Algoritmo de tiempo de escape

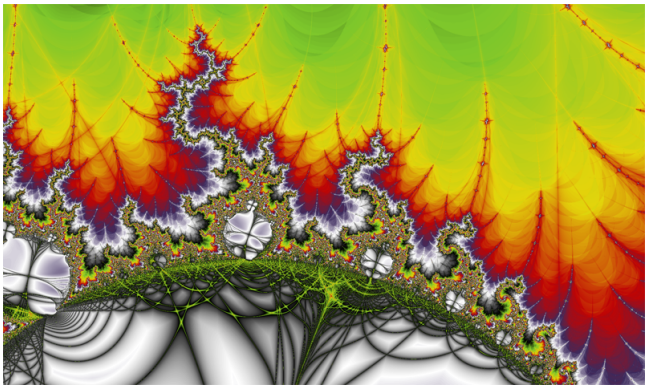

Figura 3.4: Algoritmo de tiempo de escape con trampas orbitales

Entero Gaussiano Los enteros gaussianos son enteros dentro del plano complejo, es decir números cuya parte real e imaginaria son enteros. Este algoritmo es más bien una trampa orbital repartida por todo el plano complejo midiendo la distancia de los enteros gaussianos. Este algoritmo tiene mejores resultados si se usa visualizándose con colores relativos.

Buddhabrot Este algoritmo lanza puntos aleatorios repartidos por toda la sección visible del plano complejo y los itera, de manera que cada punto marca su trayectoria al iterarse.

Ángulo de escape El algoritmo de ángulo de escape es similar al algoritmo de tiempo de escape, pero en vez de medir el número de iteraciones mide el ángulo del último punto iterado. Para proveer resultados más estéticos también utiliza en menor grado el tiempo de escape.

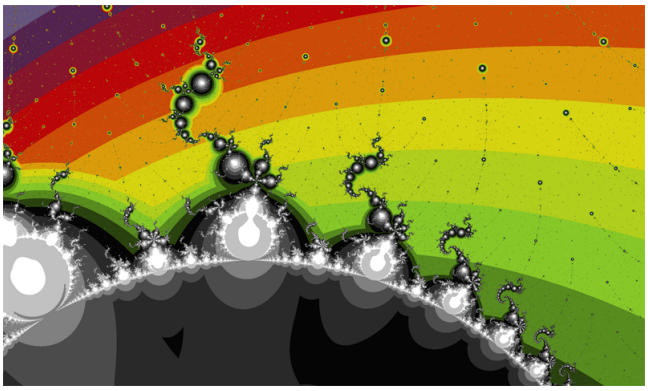

Figura 3.5: Algoritmo de enteros gaussianos

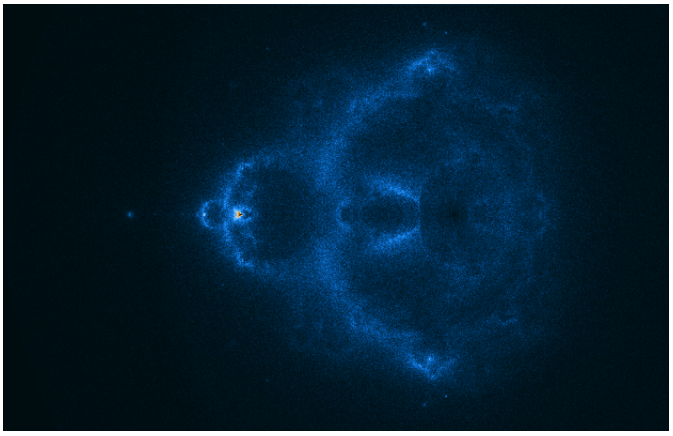

Figura 3.6: Buddhabrot

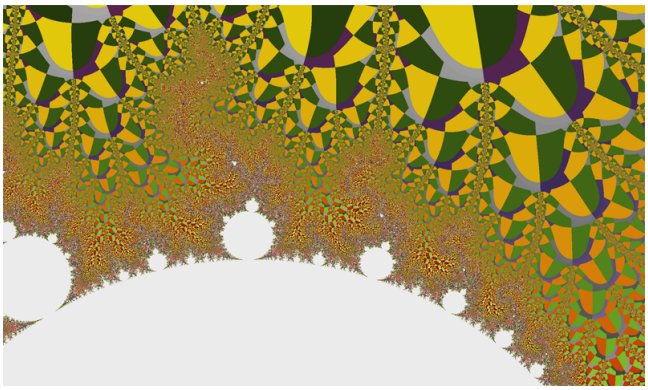

Figura 3.7: Ángulo de escape

Desigualdad del triángulo Este algoritmo se basa en un teorema de la geometría euclidiana conocido como *desigualdad del triángulo*. Este algoritmo obtiene mejores resultados usando colores relativos.

3.0.4. Gradiente

Un gradiente es una progresión de color a partir de varios colores especificados en diferentes posiciones. Para obtener mejores resultados se recomienda que el primer y último color del gradiente especificado coincidan.

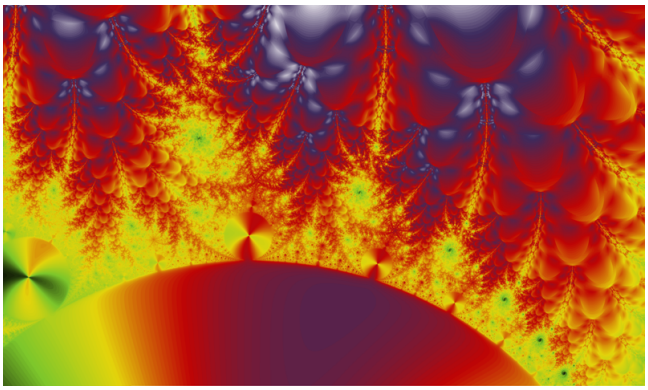

Figura 3.8: Desigualdad del triángulo

3.0.5. Color EST

El algoritmo EST genera una paleta de colores a partir de una distribución normal. A cada número de iteraciones se le asigna un color que es la suma de la distribución normal en cada canal. En el panel se puede modificar la intensidad (coeficiente), la posición (media de la distribución) y su desviación estándar.

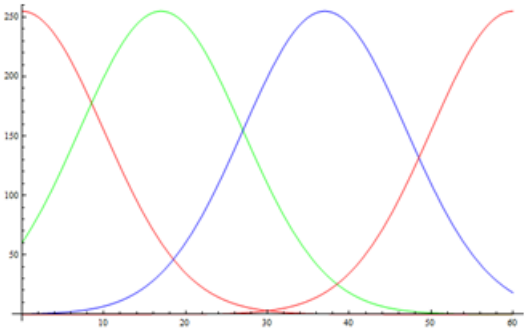

Figura 3.9: Distribución de colores

Este tipo de coloreado no resulta tan fácil de configurar como el gradiente, por lo que no siempre es recomendado.

Capítulo 4

Fractal de usuario

Una de las capacidades de *wxChaos* es que le permite al usuario introducir fórmulas personalizadas. Las fórmulas de usuario pueden ser una serie compleja o bien fractales creados a partir del método numérico conocido como *método del punto fijo*. Para ahondar en esta sección es necesario un mínimo de conocimiento en matemáticas elementales.

4.0.6. Números complejos

Antes de empezar a crear fractales es necesario definir el concepto de número complejo. Los números complejos son una extensión al concepto de número, dotando a este de una parte real y otra parte imaginaria. Cabe decir que todas las operaciones aritméticas como la suma, resta, multiplicación, división están definidas para este tipo de números. Una forma de representar estos números complejos es visualizando la parte real como un eje horizontal del número y la parte imaginaria como un eje vertical, de manera que ahora el número complejo es representable en un plano, a este se le llama el plano de *Argand*.

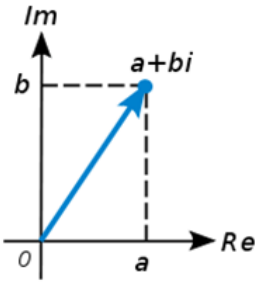

Figura 4.1: Plano complejo

Este plano de *Argand* es lo que se dibuja en la pantalla a la hora de trazar un fractal.

4.0.7. Trazando fractales

Matemáticamente hablando a la hora de trazar un fractal complejo estaremos realizando una serie de números complejos, y verificaremos la convergencia de esta serie ante una condición inicial. Entonces, ¿qué significa tener una formula como $Z_{n+1} = Z_n^2 + C$ (conjunto de Mandelbrot)? A la hora de trazar el fractal se recorre la pantalla tomando las coordenadas del plano de Argand que corresponden a cada pixel. Este punto será la constante C. Entonces se iterará este punto a partir de la fórmula anterior.

$$Z_0 = C$$
$$Z_1 = C^2 + C$$

$$Z_2 = (C^2 + C)^2 + C$$
$$Z_3 = ((C^2 + C)^2 + C)^2 + C$$
$$Z_4 = (((C^2 + C)^2 + C)^2 + C)^2 + C$$

Siguiendo este procedimiento durante N iteraciones. Después de realizar las iteraciones se verificará si la norma del punto final ha superado el valor de *bailout*. Esto se hará para todos los puntos que aparecen en la pantalla. Si los puntos superan el *bailout* querrá decir que la serie diverge, y por lo tanto se pintará el pixel correspondiente de un color, el cual viene dado por el número de iteraciones necesarias para determinar si divergió o no (en caso de usar el algoritmo de tiempo de escape). En caso de que no diverja se pintará el punto de color negro, lo cual significa que el punto pertenece al conjunto dado por la fórmula.

Para poder tener una mejor visualización de esto *wxChaos* cuenta con el modo de *show orbit* que sirve para mostrar la órbita.

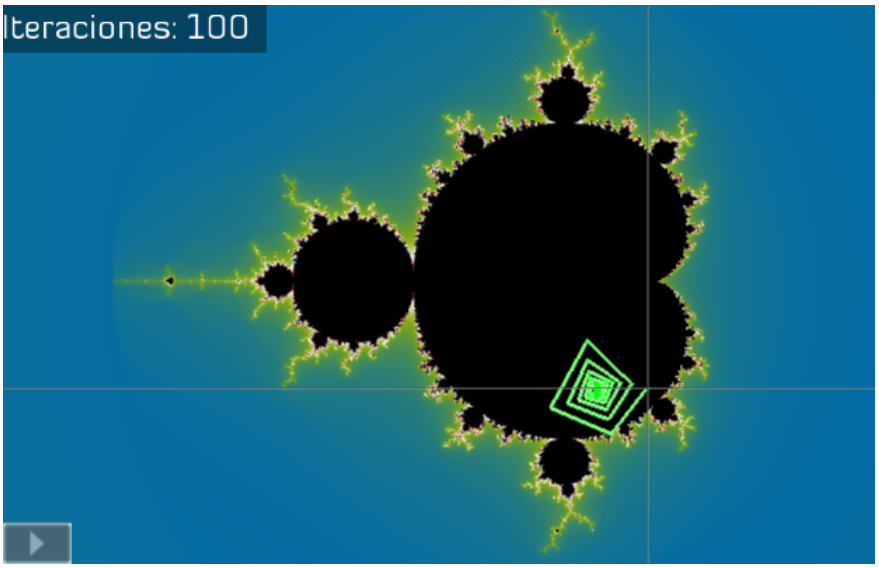

Figura 4.2: Órbita convergente

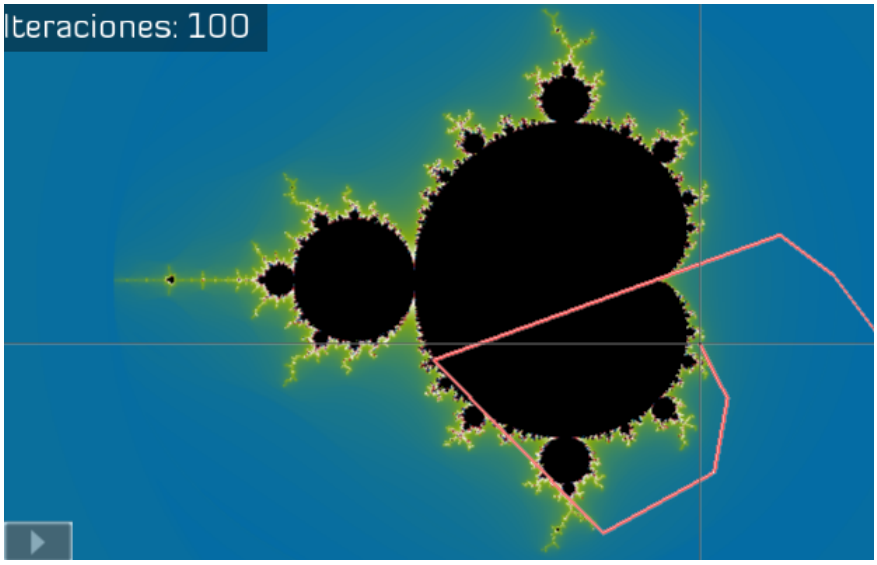

Figura 4.3: Órbita divergente

Este modo permite seleccionar un punto del plano complejo y observar su órbita al ser iterado. Se dibujará la órbita de verde si la serie converge, y de rojo si diverge.

Los fractales trazados a partir del método del punto fijo difieren en el método de trazado. Estos se basan en dicho método numérico, el cual tiene la sorprendente cualidad de que su convergencia resulta caótica. Este tipo de fractales miden la posición del último punto iterado para determinar el color. El parámetro opcional *min step* es el movimiento mínimo que debe tener un punto en alguna iteración para determinar que ya ha llegado a la solución.

4.0.8. Métodos numéricos

En esta sección se pretenderá explicar qué son los métodos numéricos y como se pueden trazar fractales a partir de estos. A la hora de resolver una ecuación sabemos (o deberíamos saber) encontrar sus soluciones a partir de procedimientos algebraicos elementales. Para un humano es algo relativamente fácil, pero dotar de las mismas capacidades a una máquina resulta una tarea algo difícil ya que enseñarle a realizar operaciones simbólicas a un aparato que sólo es capaz de realizar cálculos y lógica simple no es nada sencillo. Es por esto que para encontrar soluciones a las ecuaciones utilizando la computadora son muy populares los métodos numéricos. Estos utilizan un algoritmo para aproximarse sucesivamente a la solución de una ecuación sin necesidad de realizar ninguna operación algebraica. Es más, son capaces de llegar a la solución sin importar la dificultad de la ecuación, lo cual es ya una gran ventaja. Ejemplos de estos métodos son el *método de Newton-Rhapson*, *Método del punto fijo*, *Método de la secante*, *Método de la bisección*, etc. Lo sorprendente es que si se desea resolver una ecuación compleja y se utiliza alguno de estos métodos ¡llegan a tener un comportamiento caótico!, lo cual hace que al representarlos tengan forma fractal.

Para trazar los fractales lo que se hace es tomar del plano complejo una condición inicial, luego se aplica el método numérico a este punto para hacer que converja hacia alguna solución y por último se determina hacia que solución llego. Nótese que el método numérico sólo puede llegar a una solución dependiendo de la condición inicial, sin importar que la ecuación tenga n soluciones.

Capítulo 5

Herramientas

wxChaos viene con algunas herramientas para explorar las características caóticas de los fractales. Una es la calculadora de dimensión y la otra es el explorador de doble péndulo (Explorador DP).

5.0.9. Dimension Calculator

Una cosa bastante interesante de los fractales es que no tienen dimensión entera como las formas geométricas clásicas. Por ejemplo, la línea tiene 1 dimensión, el cuadrado 2, y el cubo es tridimensional. Con los fractales no se puede aplicar la misma regla. ¿Por qué? Con las formas geométricas regulares como el cuadrado si se mide el área y luego se vuelve a medir con una regla más pequeña no habrá cambio en el resultado. No importa la escala, no cambiará nunca el resultado. ¡Con los fractales es todo al revés! Dependiendo del tamaño de la regla se tendrán diferentes resultados.

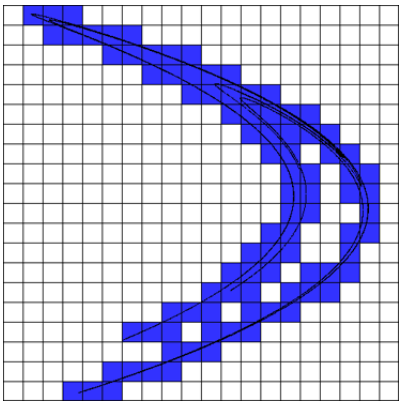

Figura 5.1: Conteo de cajas

Ahora, el método formal de hacer esto es dividir la imagen en múltiples secciones (cajas) y contar cuantas cajas hay dependiendo de su tamaño. La relación entre el número de cajas y su tamaño nos da la dimensión fractal. Más exactamente la dimensión de Minkowski-Bouligand (hay muchas definiciones de dimensión fractal).

Denotemos el tamaño de la caja por epsilon. Para tener resultados más precisos necesitamos muchos puntos para realizar el cálculo. Se puede ajustar esto en los parámetros del conteo de cajas. El primer camino para seleccionar los puntos (tamaños de epsilon) es generarlos a partir de una función. Por ejemplo, si la función es $f(x) = 2x$ y x va de 1 a 4 los tamaños de epsilon serán 2, 4, 6, 8. También se pueden seleccionar los tamaños a partir de una lista. Sólo escríbalos manualmente.

También puede graficar los resultados del cálculo. Cuando todos los puntos han sido contados, el programa calculara la dimensión usando la siguiente fórmula:

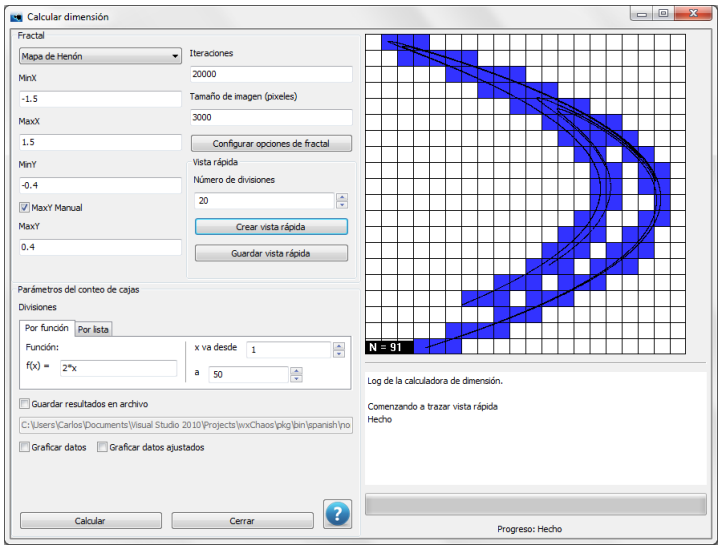

Figura 5.2: Ventana de la calculadora de dimensión

$$\dim_{\text{box}}(S) := \lim_{\varepsilon \to 0} \frac{\log N(\varepsilon)}{\log(1/\varepsilon)}$$

La gráfica de datos ajustados es solo una gráfica logarítmica, útil para visualizar la convergencia.

5.0.10. Explorador DP

El explorador DP es una pequeña herramienta para explorar el caos en un sistema de péndulo doble. *wxChaos* le permite generar fractales de péndulo doble, pero al ver el fractal por si mismo no le da ninguna pista de como este sistema puede ser caótico.

Para empezar, ¿qué es un péndulo doble?

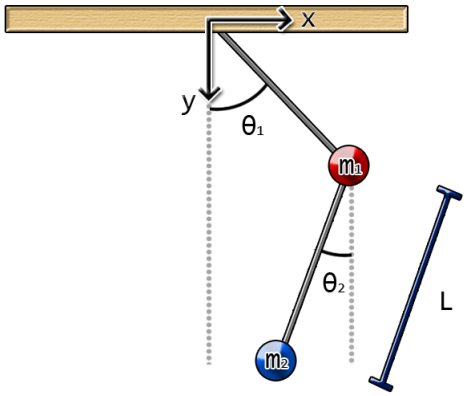

Figura 5.3: Diagrama del péndulo doble

En esencia es un sistema como el que se muestra en la figura 5.3. Dos masas unidas por dos varas rígidas. En este caso las varas no tienen masa. Cuando se simula este sistema se vuelve claro que dependiendo de las posiciones iniciales de las masas del sistema este se comporta de una manera totalmente diferente. Esto lo hace caótico, y también un sistema bastante interesante para analizar. El fractal del péndulo doble se crea a partir de un plano que contiene las posiciones iniciales de los ángulos θ_1 y θ_2. Luego se lanza una simulación por cada punto de este plano y el fractal mide si la posición de los ángulos ha llegado a alguna

condición predefinida. La condición por defecto es que alguna de las masas de la vuelta (ángulo mayor que 180°).

Esto le dará una imagen como la que viene pre-cargada en la venta del Explorador DP.

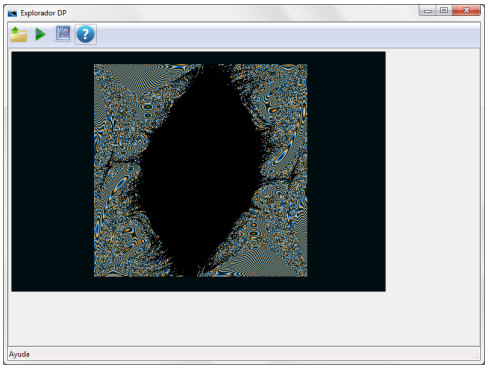

Figura 5.4: Ventana del explorador DP

Puede entonces simular el sistema haciendo click en cualquier punto de la imagen del explorador DP y luego haciendo click en el botón de Lanzar simulación.

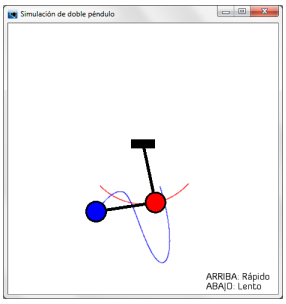

Figura 5.5: Simulación DP

También puede agregar una gráfica con el botón con el mismo nombre. ¿Para qué podría querer agregar una gráfica al fractal? Resulta que a pesar que el fractal es caótico aún es un sistema físico y es facil encontrar sus límites energéticos [1]. Estos están dados por:

$$3\cos\theta_1 + \cos\theta_2 = 2$$

5.0.11. Consola

Esto es una consola de comandos que se utiliza principalmente para la depuración de scripts de usuario, y acceder a opciones avanzadas del programa.

Para una explicación más detallada de los comandos disponibles en la consola véase el capítulo de Consola y scripts de usuario.

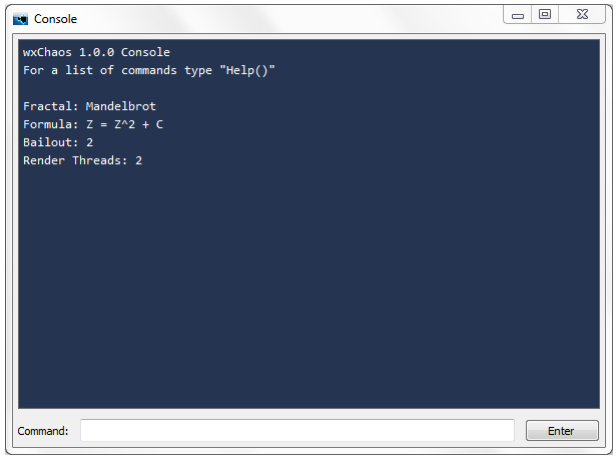

Figura 5.6: Consola de comandos

Capítulo 6

Consola y scripts de usuario

wxChaos le da al usuario la capacidad de crear sus propios fractales a través de scripts. Los scripts son creados en un lenguaje muy similar a C++ llamado *Angelscript*. Basta con tener nociones muy básicas de C++ para poder crear un fractal sin dificultad.

Todos los scripts de usuario deben ser guardados en la carpeta *UserScripts* donde se encuentra el programa. Este automáticamente cargará los scripts al iniciarse de manera que estarán disponibles en el menú de fórmulas. Si se ha guardado un script durante la ejecución del programa bastará con introducir en la consola el comando *ReloadScripts()*. Los script aquí mostrados se encuentran dentro de la carpeta *ScriptSamples*. Si se desean ejecutar basta con copiarlos a la carpeta *UserScripts*.

El programa hace uso de la consola para comunicar cualquier error durante la compilación o ejecución del script. También el mismo script puede enviar una salida de texto a través de la misma lo cual puede ser útil por ejemplo para detectar errores en el mismo script.

6.0.12. Consola de comandos

La consola de comandos cumple la función de mostrar errores de ejecución del script de usuario, mostrar texto que se envíe desde el mismo script y acceder a funciones avanzadas del programa. Tiene las capacidades que se pueden esperar de una consola de comandos: *tab* para completar algún comando y *flecha arriba o abajo* para desplazarse entre el historial de comandos.

A continuación se muestra una lista de los comandos disponibles.

AskInfo(realNum, imagNum, maxIter) Obtiene información acerca del número complejo especificado.

SetBoundaries(minX, maxX, minY, maxY) Selecciona área de aumento manualmente.

Redraw() Redibuja la pantalla. Igual que la opción que se encuentra en el menú.

Abort() Aborta script. Igual que la opción que se encuentra en el menú.

SaveOrbit(realNum, imagNum, maxIter, filepath) Guarda la órbita de un punto especificado en un archivo *.csv*. Actualmente sólo soportado con el conjunto de Mandelbrot.

SetGMPRender(mode) Si el usuario realiza sucesivos aumentos a cualquier fractal acabará llegando el momento en que este se vea completamente pixelado. Esto es debido a la precisión con la que el sistema maneja las variables de números reales. Para solucionar esto en algunos fractales esta disponible el modo **GMP** que utiliza variables de precisión arbitraria. La desventaja de esto es que el trazado del fractal es excesivamente lento.

SetGMPPrecision(precision) Establece la precisión (en bits) del modo **GMP**.

SetThreadNumber(threadNumber) Cambia manualmente el número de threads que se lanzan al trazar el fractal.

GetThreadNumber() Obtiene el número de threads que se lanzan al trazar un fractal.

ReloadScripts() Si se realiza algún cambio dentro de la carpeta *UserScripts* será necesario ejecutar este comando para que el programa vuelva a revisar los scripts disponibles. Cerrar el programa y volver abrirlo tiene el mismo efecto.

DrawCircle(x, y, radius) Añade la figura de un círculo a la pantalla.

DeleteFigures() Borra todas las figuras añadidas.

Clc() Borra el texto de la consola de comandos.

6.0.13. Hola mundo fractal

A la hora de aprender un nuevo lenguaje de programación el primer ejercicio suele ser el programa *Hola mundo*, el cual su única función es mostrar las palabras "Hola mundo.[en] la pantalla. Esto es con el fin de mostrar la estructura general del lenguaje. Para los habituados a C y C++ el *Hola mundo fractal* presenta algunas diferencias respecto a su contraparte clásica. Para empezar en C/C++ existe un punto de entrada; la función *main*. En el caso de *Hola mundo fractal* Son necesarias dos puntos de entrada. Uno es la función **SetParams** cuyo objetivo es establecer las opciones del fractal y la información que se mostrará en el menú, y la función **Render** la cual se encarga de dibujar el fractal. En este caso el programa no dibujará ningún fractal, solamente enviará texto y números a la consola.

```
1  void SetParams()
2  {
3          SetFractalName("Hola mundo");
4          SetCategory("Other");
5  }
6
7  void Render()
8  {
9          if(threadIndex == 0)
10         {
11                 PrintString("Hola mundo fractal\n");
12                 PrintInt(34);
13                 PrintString("\n");
14                 PrintFloat(0.02);
15                 PrintString("\n");
16                 complex z(0.23, 389);
17                 PrintComplex(z);
18         }
19 }
```

Al ejecutar el programa se genera la siguiente salida:

```
1  Hola mundo fractal
2  34
3  0.02
4  0.23+i389
```

Nótese que a diferencia de C/C++ en este script no ha sido necesario añadir ningún encabezado. Esto es por que todas las funciones y variables necesarias ya estan definidas dentro de *wxChaos*.

6.0.14. Funciones y variables en script

A continuación se describirán todas las funciones y variables disponibles dentro de los scripts.

Funciones:

SetFractalName(string) Especifica el nombre del fractal dentro del menú.

SetCategory(string) Especifica la categoría del fractal. Están disponibles: *Complex, NumMet, Physic, Other*.

SetMinX(float) Límite izquierdo.

SetMaxX(float) Límite derecho.

SetMinY(float) Límite inferior.

SetJuliaVariety(bool) Especifica si el fractal de script será de la variedad Julia.

SetDefaultIter(int) Especifica el valor por defecto del número de iteraciones.

SetRedrawAlways(bool) Poner en **true** si necesita redibujar todo el fractal cada vez que se cambia la posición de la vista.

NoSetMap(bool) poner en **true** si el fractal no utiliza el mapa del conjunto. Esto es para evitar que aparezca en la calculadora de dimensión.

PrintString(string) Envía texto a la consola.

PrintInt(int) Envía número entero a la consola.

PrintFloat(float) Envía número de coma flotante (número real) a la consola.

PrintComplex(complex) Envía un número complejo a la consola.

SetPoint(int x, int y, bool inSet, int iter) Dibuja punto en las coordenadas (x, y) del plano. El parámetro **inSet** se usa para determinar si el punto marcado pertenece al conjunto o no. Si se marca como *true* se pintará el punto de negro. Si se marca como *false* no se pintará el punto. **iter** es el número de iteraciones necesarias para determinar si ese punto pertenece al conjunto o no. Se usa para escoger un color de la paleta de colores.

Variables:

minX Límite izquierdo seleccionado.

maxX Límite derecho seleccionado.

minY Límite inferior seleccionado.

maxY Límite superior.

xFactor Factor de conversión longitud ventana -¿eje horizontal.

yFactor Factor de conversión altura ventana -¿eje vertical.

kReal Constante K real para modo Julia.

kImaginary Constante K imaginaria para modo Julia.

ho Límite superior de la sección a dibujar por la thread.

hf Límite inferior de la sección a dibujar por la thread.

wo Límite izquierdo de la sección a dibujar por la thread.

wf Límite derecho de la sección a dibujar por la thread.

maxIter Número máximo de iteraciones del fractal.

threadIndex Índice de la thread en la que se encuentra el script.

screenWidth Tamaño horizontal de la pantalla.

screenHeight Tamaño vertical de la pantalla.

paletteSize Tamaño de la paleta de colores.

Clase complex:
Esta clase se utiliza para realizar operaciones entre números complejos. Sus funciones miembros son:

real Devuelve la parte real del número complejo.

imag Devuelve la parte imaginaria del número complejo.

norm Devuelve la norma del número complejo.

Sus constructores tienen la forma:

Complex() Constructor vacío.

Complex(Complex in) Constructor de copia.

Complex(real, imag) Construye a partir de dos números *float*.

Además de que están definidas todas las operaciones algebraicas elementales como suma, resta, multiplicación y división, también están definidas las funciones elementales. En la izquierda están las funciones para los números complejos y a la derecha para los números reales:

pow(z, exp)	pow_r(x, exp)
sqrt(z)	sqrt_r(x)
sin(z)	sin_r(x)
cos(z)	cos_r(x)
tan(z)	tan_r(x)
csc(z)	csc_r(x)
sec(z)	sec_r(x)
cot(z)	cot_r(x)
sinh(z)	sinh_r(x)
cosh(z)	cosh_r(x)
tanh(z)	tanh_r(x)
exp(z)	exp_r(x)
log(z)	log_r(x)
log10(z)	log10_r(x)

6.0.15. Dibujando en la pantalla

Con las funciones y variables declaradas anteriormente queda resuelta toda la mecánica para trazar un fractal, ahora se explicará el procedimiento. A la hora de trazarlo estaremos recorriendo todos los puntos del área de la pantalla donde se dibujará. Normalmente el procedimiento se resume en

1. Recorrer la pantalla.
2. Calcular coordenadas en el píxel en que se encuentra.
3. Realizar operaciones con estas coordenadas.
4. Poner el resultado en el píxel.

Un ejemplo simple de la mecánica del primer paso es dibujar un gradiente:

```
1  void SetParams()
2  {
3          SetFractalName("Gradient");
4          SetCategory("Other");
5  }
6
7  void Render()
8  {
9          int color;
10         if(threadIndex == 0)
11         {
12                 for(int y=0; y<screenHeight; y++)
13                 {
14                         color = (float(y)/screenHeight)*paletteSize;
15                         for(int x=0; x<screenWidth; x++)
16                         {
17                                 SetPoint(x, y, false, color);
18                         }
19                 }
20         }
21 }
```

Nótese el uso que se hace de las variables predefinidas. **threadIndex** se utiliza debido a que a la hora de ejecutar el script el programa lanza un número de threads igual al número de procesadores en el sistema para aprovechar mejor la potencia del CPU. En algunos casos donde el dibujo sea demasiado simple (como este) no convendrá usar más de una thread, es por eso que en este script sólo se ejecutará la thread con índice 0. En este ejemplo se recorre la pantalla en dos bucles, uno horizontal que recorre desde 0 a **screenWidth** y otro vertical que recorre desde 0 a **screenHeight**. Como se desea dibujar un gradiente se requiere información del tamaño de la paleta en **paletteSize**. El lenguaje de scripts difiere sin embargo de C en la manera en la que se realizan las conversiones de tipo. En este caso para realizar una conversión de una variable *int* a una *float* se utiliza **float(**y**)**. Un procedimiento similar se aplica para el resto de las conversiones. Por último para asignarle un color al píxel se utiliza **SetPoint**.

Ahora bien, cuando se desea aprovechar la capacidad multinúcleo del sistema cada thread se encarga de diferentes secciones de la pantalla. En este caso se hace uso de las variables **ho**, **hf**, **wo**, y **wf**.

```
1  void SetParams()
2  {
3          SetFractalName("Gradient");
4          SetCategory("Other");
5  }
6
7  void Render()
8  {
9          int color;
10         for(int y=ho; y<hf; y++)
11         {
12                 color = (float(y-ho)/(hf-ho))*paletteSize;
13                 for(int x=wo; x<wf; x++)
14                 {
15                         SetPoint(x, y, false, color);
16                 }
17         }
18 }
```

En este caso cada thread dibuja un gradiente en su área de trazado. Con estos ejemplos ya se puede proceder a dibujar un fractal, se empezará por el conjunto de Mandelbrot.

```
1  void SetParams()
2  {
3          SetFractalName("ScriptMandelbrot");
4          SetCategory("Complex");
5          SetMinX(-2.52);
6          SetMaxX(1.16);
7          SetMinY(-1.16464);
8          SetJuliaVariety(false);
9  }
10
11 void Render()
12 {
13         complex z, c;
14         float c_im;
15         int n;
16         int i;
17         bool insideSet;
18         for(int y=ho; y<hf; y++)
19         {
20                 c_im = maxY - y*yFactor;
21                 for(int x=wo; x<wf; x++)
22                 {
23                         c = z = complex(minX + x*xFactor, c_im);
24                         insideSet = true;
25
26                         for(n=0; n<maxIter; n++)
27                         {
28                                 z = pow(z,2) + c;
29
30                                 if(z.real()*z.real() + z.imag()*z.imag() > 4)
31                                 {
32                                         insideSet = false;
33                                         break;
34                                 }
35                         }
36                         SetPoint(x, y, insideSet, n);
37                 }
38         }
39 }
```

Aquí primero se declaran más parámetros. Por defecto cada fractal de usuario comienza dibujándose en **minX** = -2, **maxX** = 2 y **minY** = -2. Para ajustar estos parámetros es recomendable ejecutar el script primero y ajustar posteriormente. Ahora para trazar el conjunto de Mandelbrot se hace uso de los números complejos **z** y **c**. Para asignar sus valores iniciales hace falta realizar una conversión de posición en pixeles a valor numérico en el plano, para esto son las variables **xFactor** y **yFactor**. De aquí que se pueda crear un número complejo

con las coordenadas del plano en el pixel seleccionado con el constructor **complex(minX + x*xFactor, maxY - y*yFactor)**; esta será la mecánica a seguir en todos los fractales. Acto seguido es necesario iterar este punto ejecutando **z = pow(z,2) + c** que es el equivalente a la formula $z_{n+1} = z_n^2 + c$. Luego se revisa si la norma al cuadrado del número ha superado el bailout al cuadrado (esto se hace así para aumentar el rendimiento), y dependiendo si supera el bailout o no se especifica si el punto pertenece al conjunto o no.

Este ejemplo que se acaba de mostrar corresponde al caso más simple del algoritmo de tiempo de escape, aunque también se pueden implementar fácilmente otros algoritmos como por ejemplo el conjunto del Mandelbrot con el algoritmo de ángulo de escape.

```
1  void SetParams()
2  {
3          SetFractalName("Escape_angle");
4          SetCategory("Complex");
5          SetMinX(-2.52);
6          SetMaxX(1.16);
7          SetMinY(-1.121);
8          SetJuliaVariety(false);
9  }
10
11 void Render()
12 {
13         complex z, c;
14         float c_im;
15         int n;
16         int i;
17         bool insideSet;
18         for(int y=ho; y<hf; y++)
19         {
20                 c_im = maxY - y*yFactor;
21                 for(int x=wo; x<wf; x++)
22                 {
23                         c = z = complex(minX + x*xFactor, c_im);
24                         insideSet = true;
25
26                         for(n=0; n<maxIter; n++)
27                         {
28                                 z = pow(z,2) + c;
29
30                                 if(z.real()*z.real() + z.imag()*z.imag() > 4)
31                                 {
32                                         insideSet = false;
33                                         break;
34                                 }
35                         }
36                         if(z.real() > 0 && z.imag() > 0)
37                         {
38                                 SetPoint(x, y, false, n + 1);
39                         }
40                         else if(z.real() <= 0 && z.imag() > 0)
41                         {
42                                 SetPoint(x, y, false, n + 27);
43                         }
44                         else if(z.real() <= 0 && z.imag() < 0)
45                         {
46                                 SetPoint(x, y, false, n + 37);
47                         }
48                         else
49                         {
50                                 SetPoint(x, y, false, n + 47);
51                         }
52                 }
53         }
54 }
```

También con una simple modificación se puede mostrar la versión Julia del conjunto de Mandelbrot. Basta con obtener los valores de la constante **k** a partir de las variables **kReal** y **kImaginary** y de ahí realizar las cuentas pertinentes.

```
1  void SetParams()
2  {
```

```
3           SetFractalName("ScriptJulia");
4           SetCategory("Complex");
5           SetMinX(-1.81818);
6           SetMaxX(1.8441);
7           SetMinY(-1.16464);
8           SetJuliaVariety(true);
9   }
10
11  void Render()
12  {
13          complex z, k;
14          float c_im;
15          int n;
16          int i;
17          bool insideSet;
18          k = complex(kReal, kImaginary);
19          for(int y=ho; y<hf; y++)
20          {
21                  c_im = maxY - y*yFactor;
22                  for(int x=wo; x<wf; x++)
23                  {
24                          z = complex(minX + x*xFactor, c_im);
25                          insideSet = true;
26
27                          for(n=0; n<maxIter; n++)
28                          {
29                                  z = pow(z,2) + k;
30
31                                  if(z.real()*z.real() + z.imag()*z.imag() > 4)
32                                  {
33                                          insideSet = false;
34                                          break;
35                                  }
36                          }
37                          SetPoint(x, y, insideSet, n);
38                  }
39          }
40  }
```

6.0.16. Avanzado

Al principio de este capítulo se mencionó que el lenguaje de scripts *Angelscript* es similar a C++, pero hasta ahora sólo se han utilizado características muy básicas del lenguaje. Surge la pregunta ¿hasta qué punto son similares?

Angelscript soporta las características orientadas a objetos de C++, siendo la única limitando que sólo soporta la herencia simple.

Una diferencia fundamental estriba en los tipos de datos soportados. A continuación se muestra una tabla comparativa entre los tipos de datos, extraída de la documentación de *Angelscript*.

AngelScript	**C++**	**Tamaño (bits)**
void	void	0
int8	signed char	8
int16	signed short	16
int	signed int	32
int64	signed int64_t	64
uint8	unsigned char	8
uint16	unsigned short	16
uint	unsigned int	32
uint64	unsigned uint64_t	64
float	float	32
double	double	64
bool	bool	8

Para más información acerca de *Angelscript* y su síntaxis visite *http://www.angelcode.com/angelscript/*.

Con esto se cubre todo lo necesario para que el usuario pueda comenzar a implementar sus propios scripts.

Capítulo 7

Agradecimientos

Se agradece el trabajo a los desarrolladores de las siguientes librerías que fueron utilizadas en la creación de *wxChaos*.

wxWidgets Librería de interfaz gráfica usada en la creación de casi todas las ventanas.

SFML Librería orientada principalmente a la creación de videojuegos, utilizada para dibujar los fractales.

MPIR Fork de GMP para manejar números de precisión arbitraria en Windows.

muParserX Parser con capacidad de manejar números complejos.

Angelscript Lenguaje de scripts utilizado para ejecutar los scripts de usuario.

Capítulo 8

Licencia

Este software se distribuye bajo la licencia GPLv3, que básicamente la da la capacidad al usuario de:

- Realizar un número ilimitado de copias al software.
- Redistribuir el software sin ningún tipo de restricción.
- Modificar el software, añadir o eliminar funcionalidad como se desee. Siempre y cuando al ser modificado vuelva a distribuirse bajo la misma licencia.

Para más información véase el archivo *license* que se incluye junto al programa.

Bibliografía

[1] Jeremy S. Heyl (2008), The Double Pendulum Fractal.