Fractais

Gabriele Pinheiro Sá

Introdução

Fractais são objetos geométricos que são autossimilares, ou seja, contêm neles cópias exatas, porém menores, deles mesmos. Tais figuras podem ser geradas de forma recursiva ou iterativa, e suas equações são usadas para descrever vários fenômenos, que seguem determinadas regras.

1. Floco de neve onda senoidal 1 de von Koch

Axioma: F
Ângulo: π / 3
Regra: F $\rightarrow$ F-F++FF--F+F

O código deste fractal foi escrito de maneira recursiva. Para isso, foram definidas duas constantes, MAX_ITERACAO e MAX_STRING_TAM, que determinam o número máximo de estágios (4) e o tamanho máximo da sequência de caracteres (65000), respectivamente. Além disso, é necessário definir a constante _USE_MATH_DEFINES, para usar M_PI.

A função **geraFractal** é uma função recursiva, responsável por gerar o fractal. Ela recebe os parâmetros file (grava a saída em um arquivo), string (armazena o resultado), iteracao (nº de iterações), angulo, axioma e regra. Se o número de iterações for igual a zero, a função copia axioma para string e retorna, caso contrário, ela irá alocar memória para uma string temporária e chamará de forma recursiva a própria função, gerando a iteração anterior.

Depois disso, são inicializadas as variáveis tamanho e index, e o comprimento de stringTemp é calculado usando **strlen**. Em seguida, um loop verifica cada caractere de stringTemp, incrementando o index conforme o caractere correspondente ao caractere atual. Os segmentos são rotacionados para a esquerda e para a direita usando os símbolos (-) e (+), respectivamente.

A função **iteraString** recebe os parâmetros file, uma string de origem e outra de destino, e a regra de produção. Ela copia a string de origem para a string de destino usando a função **strcpy** e chama a função **geraFractal** para gerar a próxima iteração do fractal.

Na função **main** são definidas as variáveis axioma, que recebe “F”; regra, que recebe “F-F++FF--F+F”; e string, que armazena a sequência de caracteres do fractal em cada iteração. Depois, a função **strcpy** copia o axioma inicial para a string string. Em seguida, um loop gera os estágios do fractal, a começar pelo 1º, que é a própria regra de produção.

Equação de recorrência

Para montar a equação de recorrência que calcula a quantidade de segmentos F gerados em cada estágio do fractal, primeiro é necessário contar quantos deles existem na regra de produção. No caso do fractal floco de neve onda senoidal 1 de von koch, temos 6 F's. Dessa forma, podemos dizer que S(1) = 6, em que S(n) é a quantidade de segmentos F no estágio *n*. Para os próximos estágios, devemos utilizar a equação S(n) = 6 * S(n-1).

Para calcular a quantidade total de símbolos em cada estágio, é preciso considerar F, + e -. Semelhante ao processo anterior, devemos contar quantos símbolos temos na regra de produção, que no total são 12. Sendo assim, T(1) = 12, em que T(n) é a quantidade total de símbolos no estágio *n*. Para *n* > 1, usaremos a equação T(n) = 6 * T(n-1) + 6.

A tabela abaixo mostra a quantidade de segmentos F e a quantidade total de símbolos de cada estágio do fractal calculados usando as equações de recorrência encontradas.

n	#F	#Símbolos
1	6	12
2	36	78
3	216	474
4	1296	2850
⋮	⋮	⋮

Complexidade do algoritmo

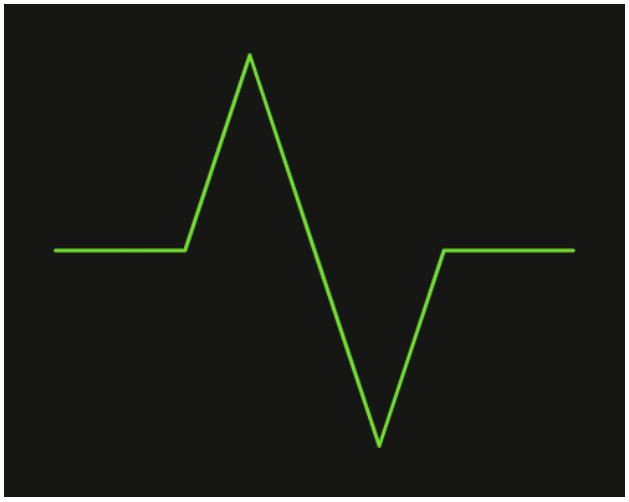
A notação assintótica do algoritmo irá depender do número de estágios do fractal. Sabemos que a cada estágio, o número de segmentos F é multiplicado por 6. Sendo assim, #F em um estágio específico *n* pode ser dado como S(n) = 6^n. Já a quantidade total de símbolos inclui a quantidade do número de segmentos F e também os símbolos + e - gerados pela regra de produção. No caso desse fractal, o número de símbolos é proporcional ao número de segmentos F, e ao fazer a expansão da equação, a recorrência pode ser expressa como T(n) = 12 * S(n-1) + 6 * (n - 1) = 12 * 6^{n-1} + 6 * (n - 1) = Θ(6^n).

Portanto, a complexidade do algoritmo é exponencial, com taxa de crescimento de O(6^n), o que significa que o tempo de execução do algoritmo aumenta exponencialmente à medida que *n* aumenta. A complexidade assintótica precisa é de Θ(6^n). Apesar de possuir alto custo assintótico em relação ao número de símbolos gerados, a quantidade de cálculos realizados por símbolo é relativamente pequena, tendo em vista que a regra de produção é simples. Assim, o tempo de execução total pode ser baixo para valores moderados de *n*.

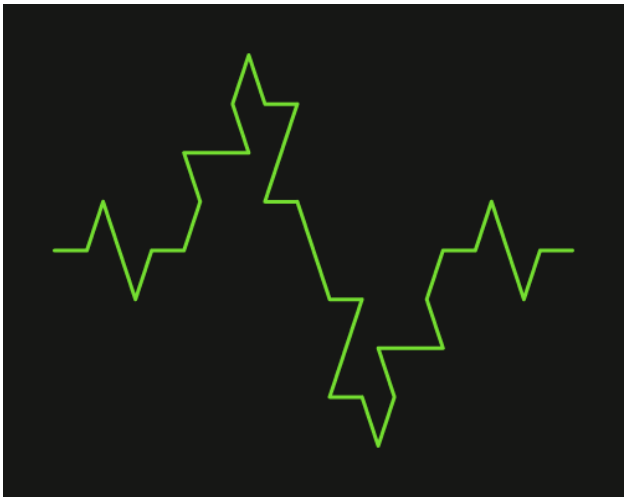
Imagens geradas pelos 4 estágios do fractal Floco de Neve Onda Senoidal 1 de von Koch

As imagens a seguir foram geradas utilizando o software gratuito Online Math Tools.

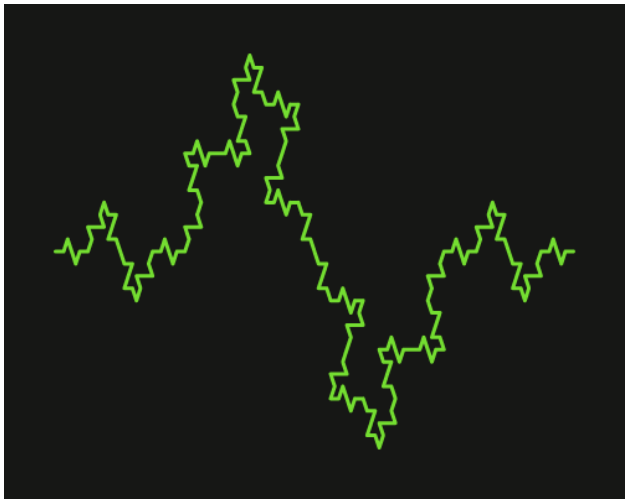
Estágio 1:



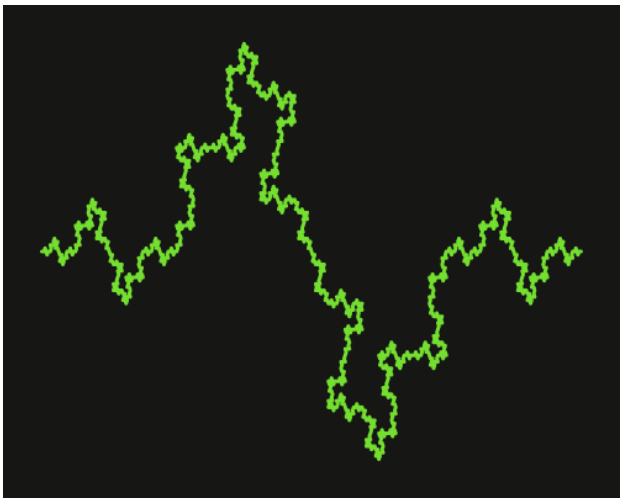
Estágio 2:



Estágio 3:



Estágio 4:



2. Preenchimento de Espaço de Hilbert

Axioma: X
Ângulo: π / 2
Regras: X → -YF+XFX+FY-
Y → +XF-YFY-FX+

O código desse fractal foi escrito de maneira iterativa, utilizando o sistema de substituição de Lindenmayer (L-system). Para isso, foram utilizadas as mesmas constantes do fractal anterior, e a função **iteraString** é declarada para aplicar as regras de substituição do L-system em uma string.

Na função **main**, são declaradas as variáveis angulo, axioma, regraX, regraY e string. A iteração acontece por meio de um loop **for** nessa função, onde as regras de substituição são aplicadas repetidamente a cada iteração para construir a sequência do fractal. Nesse loop, a função **iteraString** é chamada para substituir os caracteres na string.

De forma geral, o programa aplica as regras X e Y, que são substituídas em uma string inicial para gerar a sequência de caracteres do fractal Preenchimento de Espaço de Hilbert. A cada iteração, as regras são novamente aplicadas na string resultante da iteração anterior, gerando uma sequência mais longa e complexa. O processo é repetido 4 vezes, gerando a sequência dos primeiros 4 estágios, podendo esse valor pode ser alterado na constante MAX_ITERACAO para mostrar uma maior quantidade de estágios (tomar cuidado com a capacidade do compilador). Após isso, o resultado é armazenado em um arquivo de saída e também pode ser visto no próprio terminal.

Obs.: As sequências de caracteres geradas em cada estágio levam em conta todos os símbolos: X, Y, F, + e -.

Apesar de ter sido escrito para gerar o fractal Preenchimento de Espaço de Hilbert, esse código pode ser adaptado para gerar a sequência de caracteres de outros fractais, com regras diferentes, desde que não passe de duas regras. Para fazer essa adaptação, basta mudar o valor das variáveis angulo, axioma, regraX e regraY, conforme as especificações do fractal desejado, além de modificar o nome do arquivo de saída.

Dessa forma, esse mesmo programa será usado para gerar o próximo fractal.

Equação de recorrência

Para encontrar a equação de recorrência que calcula a quantidade de segmentos F gerados em cada estágio do fractal, precisamos primeiro definir o caso base, ou seja, a quantidade de símbolos F para o primeiro estágio. Com isso, temos que S(1) = 3. Agora, para n > 1, devemos somar a quantidade de F's do estágio anterior, somada a 3, com a quantidade de F's do estágio anterior multiplicada por 3. A equação ficará: S(n) = (3 * S(n-1)) + (S(n-1) + 3).

Para encontrar a quantidade total de símbolos em cada estágio do fractal, temos que o caso base é T(1) = 11 e para n > 1, somaremos o próprio caso base a 4 vezes a quantidade de símbolos totais do estágio anterior menos 1, ou seja, T(n) = 11 + (4 * (T(n-1) - 1)).

Para descobrir a quantidade de símbolos, excluindo o F, a equação é semelhante à anterior. Vamos mudar apenas o caso base, que agora é U(1) = 8, para n = 1. Para n > 1, usaremos a equação U(n) = 8 + (4 * (U(n-1) - 1)).

Segue abaixo a tabela com as quantidades de F, símbolos totais e símbolos sem o F para os primeiros estágios do fractal Preenchimento de Espaço de Hilbert.

n	#F	#Símbolos totais	#Sem o F
1	3	11	8
2	15	51	36
3	63	211	148
4	255	851	596
⋮	⋮	⋮	⋮

Complexidade do algoritmo

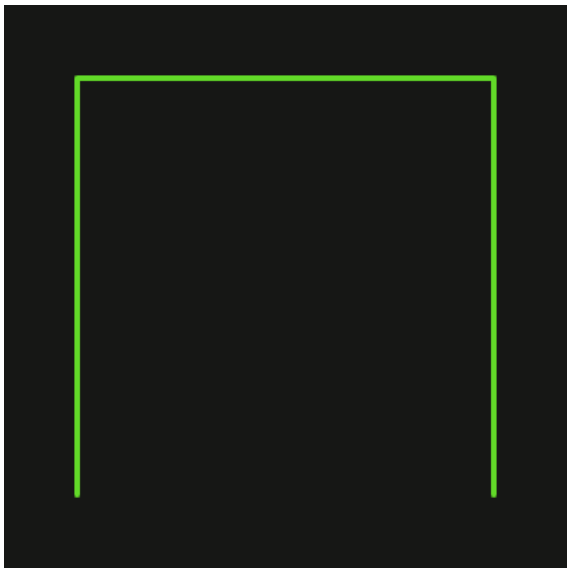
Ao expandir a equação T(n) = 11 + (4 * (T(n-1) - 1)), a recorrência pode ser expressa como T(n) = 10 * 4^{n-1} + 11 * (n - 1) = Θ(4^n). Sendo assim, a complexidade assintótica mais precisa do algoritmo, tendo em vista a quantidade total de símbolos do fractal para cada estágio, é Θ(4^n).

A complexidade do algoritmo desse fractal é semelhante à do primeiro, portanto, características como crescimento, custo assintótico e outros comportamentos são igualmente concluídas para esse.

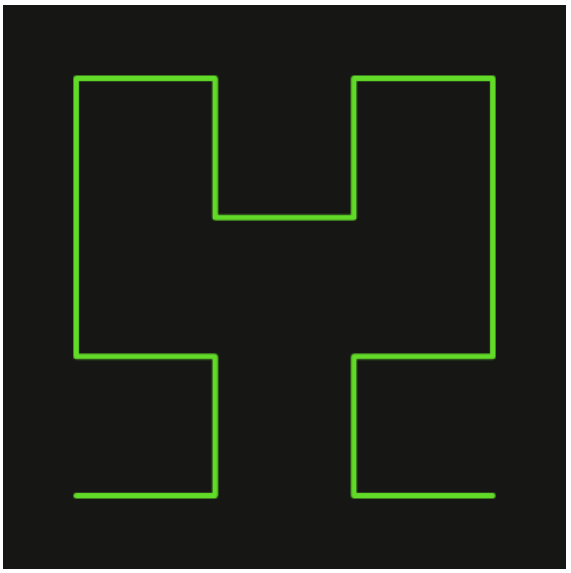
Imagens dos 4 estágios do fractal Preenchimento de espaço de Hilbert

As imagens a seguir foram geradas com o software gratuito Online Math Tools.

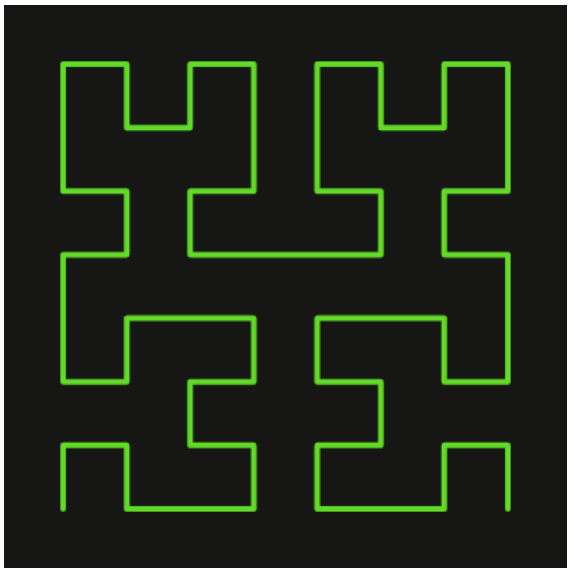
Estágio 1:



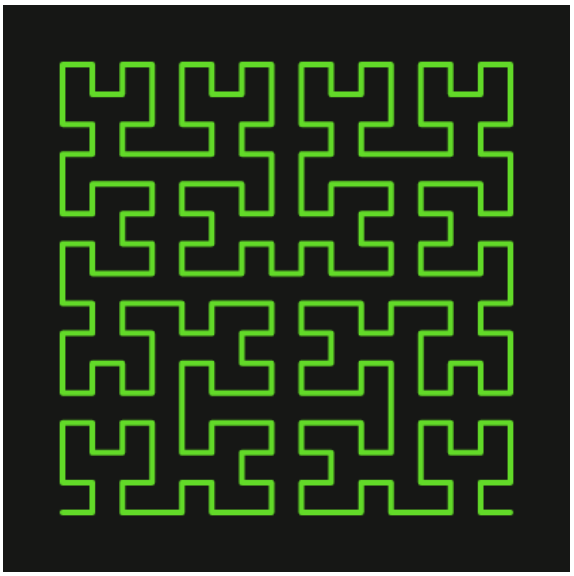
Estágio 2:



Estágio 3:



Estágio 4:



3. Fractal autoral definido por mim

Axioma: A
Ângulo: π / 2
Regras: A → +FB+FB+FB+FB
B → -AF+BF+FB-FA

O código usado para criar esse fractal foi o mesmo do fractal 2. Foram alteradas apenas as variáveis necessárias para caracterizar o novo fractal. Sendo assim, a explicação anterior a respeito do programa desenvolvido se aplica aqui também.

Equação de recorrência

Para encontrar a equação de recorrência que calcula a quantidade de segmentos F gerados em cada estágio do fractal, precisamos primeiro definir o caso base, ou seja, a quantidade de símbolos F para o primeiro estágio. Com isso, temos que S(1) = 4. E, para n > 1, a equação ficará: S(n) = 4 * S(n-1) + 4.

Para encontrar a quantidade total de símbolos em cada estágio do fractal, temos que o caso base é T(1) = 12 e, para n > 1, T(n) = 4 * T(n-1) + 8.

Para descobrir a quantidade de símbolos, excluindo o F, a equação é semelhante à anterior. U(1) = 8, para n = 1 e para n > 1, usaremos a equação U(n) = 4 * U(n-1) + 4.

n	#F	#Símbolos totais	#Sem o F
1	4	12	8
2	20	56	36
3	84	232	148
4	340	936	596
⋮	⋮	⋮	⋮

Complexidade do algoritmo

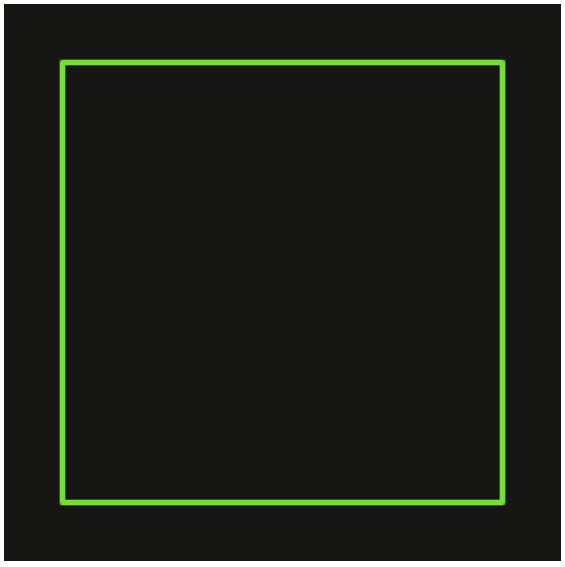
Ao expandir a equação para encontrar o total de símbolos, T(n) = 4 * T(n-1) + 8, a recorrência pode ser expressa como T(n) = 12 * 4^{n-1}+ 8 * (n - 1) = Θ(4^n). Portanto, a complexidade assintótica mais precisa do algoritmo é Θ(4^n).

As conclusões acerca da complexidade desse algoritmo são as mesmas levantadas para os fractais anteriores, tendo em vista que esse também possui crescimento exponencial.

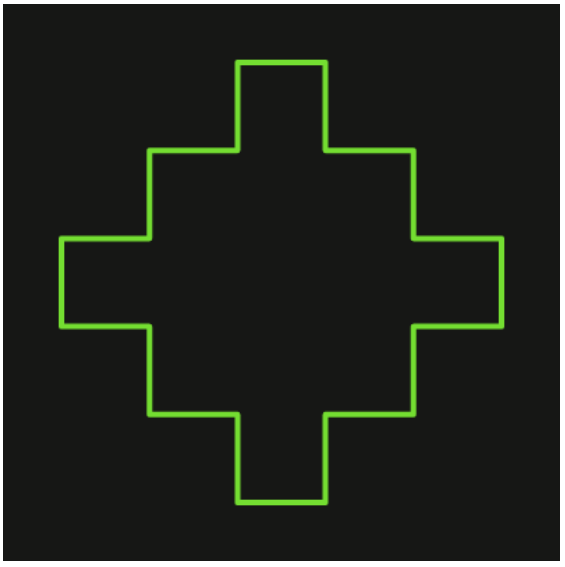
Imagens dos 4 estágios do fractal autoral

As imagens a seguir foram geradas com o software gratuito Online Math Tools.

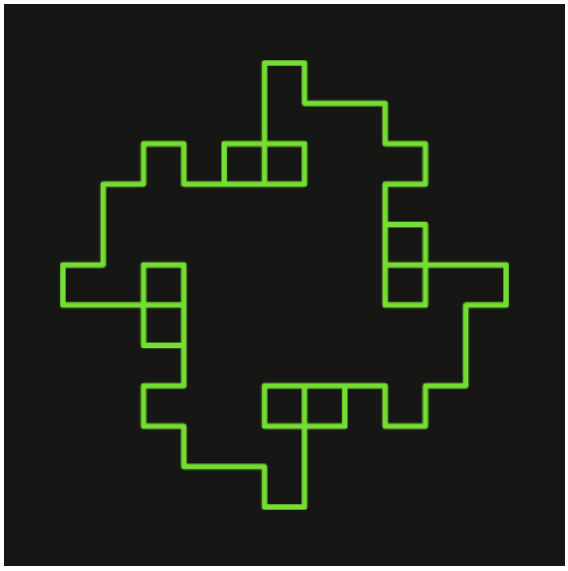
Estágio 1:



Estágio 2:



Estágio 3:



Estágio 4: