Trabalho Prático - Espirais

João Correia Costa

07 de Setembro de 2023

1 Introdução

As espirais quadradas e triangulares são figuras geométricas que combinam características de uma espiral com as propriedades de quadrados e triângulos respectivamente. São formadas por uma série de quadrados ou triângulos que se conectam uns aos outros de maneira que cada ciclo seja ligeiramente maior ao anterior, criando uma curva contínua que se estende indefinidamente. Trata-se de uma figura geométrica fractal, pois exibe autossimilaridade, o que significa que a forma da figura se repete em diferentes escalas.

Neste trabalho prático, iremos desenvolver um algoritmo simples que mapeia o index ($i \geq 0$) de um ponto qualquer na espiral para coordenadas cartesianas (x, y).

2 Algoritmo dos Quadrados Perfeitos

A inpersão visual da espiral revela padrões lineares que desempenham um papel fundamental na formação da figura como um todo. Por exemplo, a espiral quadrada pode ser construída usando exclusivamente segmentos horizontais e verticais, correspondentes às direções de $0°$ e $90°$. Da mesma forma, a espiral triangular pode ser composta por segmentos nas direções de $45°$ e $-45°$, juntamente com segmentos horizontais.

Além disso, um outro alinhamento notável surge ao considerar os grupos de pontos quadrados perfeitos, tanto pares (QP) quanto ímpares (QI). Esses padrões oferecem uma intuição valiosa para navegar eficientemente pelas espirais.

2.1 Intuição

Pontos que estão alinhados na espiral podem ser acessados em um único passo, se sourbermos das seguintes informações *a priori*:

1. Coordenadas (x_a, y_a) de um ponto P_a qualquer, pertencente ao alinhamento que conecta P_a e P_b.
2. A posição relativa α do ponto objetivo P_b em relação a P_a.
3. A direção ou vetor $\overrightarrow{V}$ do alinhamento.

Uma vez obtidas essas três informações, calcular as coordenadas de P_b é trivial. Basta aplicar a equação 1.

$$P_b = P_a + alpha \cdot \overrightarrow{V} \tag{1}$$

O método para determinar o ponto de destino, denotado como P_b, envolve uma abordagem que consiste em saltar para um ponto conhecido, P_a, com coordenadas previamente estabelecidas e que esteja próximo a P_b. Em seguida, é necessário determinar a posição relativa, representada por α, de P_b em relação a P_a, bem como identificar a direção representada por $\overrightarrow{V}$ que conecta esses dois pontos. Posteriormente, aplicamos a equação 1 para calcular as coordenadas do ponto-objetivo.

Os grupos QI e QP possuem um papel especial, pois oferecem a capacidade de acessar facilmente qualquer ciclo da espiral. Podemos utilizar esses dois grupos para identificar pontos P_a que estejam próximos ao ponto de destino P_b. O método descrito a seguir possibilita o cálculo das coordenadas de pontos que são quadrados perfeitos.

2.2 Coordenadas de Quadrados Perfeitos

Utilizamos o mesmo método, mas com parâmetros distintos, para determinar as coordenadas de um ponto pertencente ao grupo QI ou QP.

O alinhamento do grupo QI inicia-se com o ponto P_1 e segue na direção $odd\_direction$. O ponto subsequente no grupo é P_9, com uma posição relativa denotada por $\alpha = 1$. Em seguida, encontramos P_{25}, com $\alpha = 2$, e assim por diante. Analogamente, o grupo QP possui um ponto inicial em P_0 e segue na direção $even\_direction$.

Para calcular a posição relativa α, utilizamos as seguintes equações:

1. QI:
$$\alpha = \frac{\sqrt{index} - 1}{2} \tag{2}$$

2. QP:
$$\alpha = \frac{\sqrt{index}}{2} \tag{3}$$

Uma vez calculada a posição relativa do quadrado perfeito, aplicamos a equação (1) com os hiperparâmetros de cada grupo.

1. QI:
$$(x, y) = P_1 + \alpha \cdot odd\_direction \tag{4}$$

2. QP:
$$(x, y) = P_0 + \alpha \cdot even\_direction \tag{5}$$

É importante observar que P_1 corresponde a (0, 1) na espiral quadrada e a (1, 0) na espiral triangular. Da mesma forma, a direção $odd\_direction$ corresponde a (1, 1) na espiral quadrada e a (2, -1) na espiral triangular. A direção $even\_direction$ corresponde a (-1, -1) na espiral quadrada e a (-2, -1) na espiral triangular. Ao aplicarmos corretamente as fórmulas 4 e 5, podemos obter as coordenadas de qualquer ponto que seja um quadrado perfeito. Ver *algorithm 2*.

2.3 Coordenadas de quaiquer pontos

Para determinar as coordenadas de um ponto P_b em ambas as espirais, o processo envolve vários passos. Inicialmente, procuramos o quadrado perfeito P_a mais próximo. Isso é alcançado calculando a raiz quadrada do índice fornecido e arredondando para o número inteiro mais próximo, resultando na raiz r, ou seja, $P_a = r^2$. As coordenadas de P_a são então obtidas aplicando a equação 4 ou 5, dependendo da paridade da espiral em questão.

Em seguida, determinamos a posição relativa α de P_b em relação a P_a e identificamos a direção $\overrightarrow{V}$ do alinhamento. Em seguida, aplicamos a equação 1 utilizando os parâmetros apropriados.

Esse processo permite identificar as coordenadas de qualquer ponto na espiral quadrada ou triangular. Ver *algorithm 2*.

3 Análise de Complexidade

3.1 Função `nearest_root(unsigned index)`:

- **Complexidade de Tempo:**
  1. A função calcula a raiz quadrada arredondada do índice dado, o que é uma operação de complexidade constante, pois envolve apenas um cálculo matemático simples.
  2. Portanto, a complexidade de tempo é $O(1)$.
- **Complexidade de Espaço:**
  1. A função não alocará memória adicional com base no valor de entrada, portanto, a complexidade de espaço também é $O(1)$.

3.2 Função `square_coordinates(unsigned root)`:

- **Complexidade de Tempo:**
  1. A função determina as coordenadas de um ponto em uma espiral com base na raiz quadrada passada como entrada.
  2. O cálculo das coordenadas envolve operações matemáticas simples e atribuições, todas executadas em tempo constante.
  3. Portanto, a complexidade de tempo é $O(1)$.
- **Complexidade de Espaço:**
  1. A função não aloca memória adicional com base no valor de entrada, então a complexidade de espaço é $O(1)$.

3.3 Função `main(int argc, char* argv[])`:

- **Complexidade de Tempo:**
  1. O tempo de execução desta função depende da entrada, isto é, do valor de `index`.
  2. As operações iniciais para verificar o número de argumentos de linha de comando, converter o índice para um número inteiro e verificar se o índice é 0 ou 1 são todas operações de complexidade constante, ou seja, $O(1)$.
  3. Em seguida, o código chama `nearest_root(index)` para calcular a raiz quadrada do índice, que é uma operação de complexidade constante.
  4. A função `square_coordinates(root)` também é uma operação de complexidade constante.
  5. Portanto, a maior parte do tempo é gasta em operações de complexidade constante, e a complexidade de tempo total é dominada por elas.
- **Complexidade de Espaço:**
  1. O espaço alocado nesta função não depende do valor de entrada, apenas das variáveis locais e das chamadas de função.
  2. Portanto, a complexidade de espaço é $O(1)$.

4 Conclusão

Neste trabalho prático, exploramos o mapeamento de índices de pontos em espirais quadradas e triangulares para suas coordenadas cartesianas correspondentes. Essas espirais são estruturas geométricas fascinantes, exibindo propriedades autossimilares em diferentes escalas e constituindo figuras fractais.

Para realizar o mapeamento, desenvolvemos um algoritmo eficiente que se baseia na identificação de quadrados perfeitos e alinhamentos específicos. Ao aproveitar os grupos de quadrados perfeitos pares (QP) e ímpares (QI), conseguimos determinar as coordenadas de pontos em qualquer ciclo da espiral.

As equações de cálculo das coordenadas são simples e versáteis, permitindo que o algoritmo funcione para ambas as espirais quadradas e triangulares. Além disso, a abordagem é eficiente, uma vez que evita cálculos dispendiosos e se concentra na identificação de alinhamentos e quadrados perfeitos próximos ao ponto de destino.

Em resumo, o algoritmo desenvolvido neste trabalho demonstra como a geometria e os padrões nas espirais quadradas e triangulares podem ser explorados para mapear índices de pontos para coordenadas cartesianas de forma eficaz.

A Instruções para Compilação e Execução

Observação: Certifique-se de que você tenha o compilador GCC (g++) instalado em seu sistema para a compilação.

A.1 Compilação do Projeto

Para compilar o projeto, siga as instruções abaixo:

1. Abra um terminal e navegue até o diretório raiz do projeto.
2. Certifique-se de que o projeto contenha a seguinte estrutura de diretórios:

   ```
   - src/
   - bin/
   ```

3. Utilize o seguinte comando para compilar o projeto:

   ```
   make
   ```

   Isso irá compilar o projeto e gerar os executáveis `bin/esptriangular` e `bin/espquadrada`.

A.2 Execução do Projeto

Para executar os programas compilados, utilize os seguintes comandos:

A.2.1 Executar o programa `esptriangular`

```
./bin/esptriangular <point-index>
```

Este comando executará o programa `esptriangular`.

A.2.2 Executar o programa `espquadrada`

```
./bin/espquadrada <point-index>
```

Este comando executará o programa `espquadrada`.

A.3 Ajuda e Limpeza

A.3.1 Ajuda

Para obter ajuda sobre como executar o projeto, utilize o seguinte comando:

```
make help
```

Isso fornecerá informações sobre como executar os programas com argumentos.

A.3.2 Limpeza dos Arquivos Compilados

Para limpar os arquivos compilados e executáveis, utilize o seguinte comando:

```
make clean
```

Isso removerá os executáveis gerados e quaisquer arquivos temporários.

B Algoritmos

Algorithm 1: Calcula as coordenadas do ponto na espiral triangular

1. Verifique o número de argumentos na linha de comando.
2. Se o número de argumentos for diferente de 2, imprima a mensagem "Uso: [executável] [índice do ponto]" e encerre o programa.
3. Converta o índice do ponto de uma string para um valor unsigned.
4. Verifique se o índice é trivial (0 ou 1).
5. Se o índice for 0, imprima o resultado usando as coordenadas de origem e retorne 0.
6. Se o índice for 1, imprima o resultado usando as coordenadas (1, 0) e retorne 0.
7. Calcule a raiz do índice fornecido e arredonde o valor para o inteiro mais próximo: root.
8. Calcule o índice do quadrado perfeito mais próximo elevando root ao quadrado
9. Determine o grupo do quadrado perfeito (ímpar ou par).
10. Calcule as coordenadas do quadrado perfeito mais próximo usando a função *square_coordinates*.
11. Calcule a posição relativa do ponto em relação ao índice do quadrado perfeito.
12. Inicialize as coordenadas de destino como as coordenadas do quadrado perfeito mais próximo.
13. Se a posição relativa for maior que 0, calcule as coordenadas de destino usando a função *linear_combination* com os vetores apropriados (*_135_direction* ou *right_direction*) e atualize as coordenadas de destino.
14. Caso contrário, calcule as coordenadas de destino usando a função *linear_combination* com os vetores apropriados (*left_direction* ou *_45_direction*) e atualize as coordenadas de destino.
15. Imprima o resultado das coordenadas do ponto.
16. Retorne 0 para encerrar o programa.

Algorithm 2: Calcula as coordenadas do quadrado perfeito

1. Inicialize uma estrutura de dados Vector2 chamada result para armazenar as coordenadas do quadrado perfeito.
2. Identifique o grupo do quadrado perfeito: ímpar (1) ou par (0) com base no índice *root*.
3. Calcule a posição relativa do quadrado perfeito em seu grupo respectivo: Ímpar(QI) ou Par (QP) usando a fórmula:

$$\alpha = \begin{cases} \frac{\sqrt{\text{root}}-1}{2} & \text{se for ímpar} \\ \frac{\sqrt{\text{root}}}{2} & \text{se for par} \end{cases}$$

4. Determine as coordenadas do quadrado perfeito usando a fórmula:

$$(x, y) = \begin{cases} (1, 0) + \alpha \cdot (-1, 1) & \text{se for ímpar} \\ (0, 0) + \alpha \cdot (2, -1) & \text{se for par} \end{cases}$$

5. O result agora contém as coordenadas do quadrado perfeito.
6. Retorne o result.