Agrupamento de Escolas de Benfica

Escola Secundária José Gomes Ferreira

Grupo Disciplinar 510

Análise e Implementação Computacional da Geometria dos Favos de Mel

Adrian Dias

Nº 1, T 12º 3ª

– Disciplina de Física –

2025 - 2026

Resumo

Este relatório apresenta o desenvolvimento de um modelo matemático e a respectiva implementação computacional em `Python/turtle`, com vista à representação de uma forma geométrica presente na natureza. Seleccionou-se como objeto de estudo a estrutura hexagonal dos favos de mel, dada a sua relevância matemática e propriedades de optimização. São descritos os fundamentos teóricos, a metodologia de implementação e os principais resultados obtidos, com particular ênfase na modelação matemática subjacente.

1 Introdução

1.1 Enquadramento e Objectivos

O presente trabalho tem como objectivo principal a modelação matemática e implementação computacional de formas naturais, com particular enfoque nos padrões hexagonais observáveis em favos de mel. A selecção desta forma específica justifica-se pela sua fundamentação matemática robusta e pelas propriedades de optimização que exibe na natureza.

A abordagem adoptada integra conceitos de geometria euclidiana, teoria de tesselações e algoritmos computacionais, visando não apenas a reprodução gráfica da forma, mas também a compreensão dos princípios matemáticos que governam a sua estrutura.

1.2 Fundamentos Matemáticos dos Favos de Mel

1.2.1 Conjectura do Favo de Mel e Princípios de Optimização

A estrutura hexagonal dos favos de mel materializa a **Conjectura do Favo de Mel** (Honeycomb Conjecture), demonstrada matematicamente por Thomas Hales em 1999. Este teorema estabelece que, entre todas as partições do plano em regiões de área igual, a configuração hexagonal regular minimiza o perímetro total. A razão óptima perímetro-área é dada por:

$$\min \frac{P}{A} = \sqrt[4]{12} \approx 1,8612 \tag{1}$$

onde P representa o perímetro e A a área. Para o hexágono regular com lado s, obtém-se:

$$\frac{P}{A} = \frac{6s}{\frac{3\sqrt{3}}{2}s^2} = \frac{4}{\sqrt{3}s} \tag{2}$$

1.2.2 Tesselações Regulares do Plano

A geometria euclidiana impõe restrições rigorosas às tesselações regulares do plano. Conforme demonstrado matematicamente, apenas três polígonos regulares preenchem completamente o espaço bidimensional sem sobreposições ou espaços vazios:

- **Triângulos equiláteros** (notação de Schläfli: $\{3, 6\}$)
- **Quadrados** (notação de Schläfli: $\{4, 4\}$)
- **Hexágonos regulares** (notação de Schläfli: $\{6, 3\}$)

A condição matemática necessária e suficiente para tesselação regular é expressa por:

$$\frac{1}{m} + \frac{1}{n} = \frac{1}{2} \tag{3}$$

onde m e n são inteiros que satisfazem a equação para os três casos mencionados.

1.2.3 Limitações ao Refinamento Hierárquico

Os hexágonos regulares apresentam constrangimentos matemáticos inerentes no que concerne a sistemas de múltiplas escalas. A impossibilidade de subdivisão hierárquica perfeita decorre de três axiomas fundamentais:

a. **Conservação de área**: $A_{\text{parental}} = \sum A_{\text{filhas}}$

b. **Hierarquia simples**: Apenas um elemento parental por célula

c. **Cobertura perfeita**: Ausência de espaços vazios ou sobreposições

Estes princípios matemáticos explicam a preferência evolutiva por favos de dimensão uniforme em detrimento de sistemas com múltiplas escalas.

2 Modelo Computacional Implementado

2.1 Arquitetura do Sistema de Coordenadas

No script `Python/turtle`, a estrutura hexagonal é implementada através de um sistema de coordenadas baseado em vectores de rede hexagonal. A posição de cada hexágono é determinada por:

$$\vec{r}_{ij} = i \cdot \vec{a}_1 + j \cdot \vec{a}_2 \tag{4}$$

onde os vectores base da rede hexagonal são definidos como:

$$\vec{a}_1 = (1.5s, 0) \tag{5}$$

$$\vec{a}_2 = \left(0.75s, \frac{\sqrt{3}}{2}s\right) \tag{6}$$

sendo s o comprimento do lado do hexágono.

2.2 Especificações e Parâmetros do Modelo

Para a implementação computacional, foram estabelecidos os seguintes parâmetros matemáticos:

- **Dimensão do hexágono**: $s = 25$ unidades turtle
- **Número de camadas**: $n = 7$ camadas concêntricas
- **Total de hexágonos**: $N = 1 + \sum_{k=1}^{7} 6k = 127$ células
- **Ângulo de rotação**: $\theta = 60^\circ$ entre lados consecutivos
- **Área individual**: $A_{\text{hex}} = \frac{3\sqrt{3}}{2}s^2$

2.3 Algoritmo de Construção Progressiva

O modelo implementado simula o processo de construção natural através de um algoritmo de expansão radial, iniciando em pontos centrais e expandindo concentricamente. A progressão obedece à sequência:

$$N_k = 6k \quad \text{para} \quad k = 1, 2, \ldots, n \tag{7}$$

onde N_k representa o número de hexágonos na camada k-ésima, garantindo crescimento simétrico e geometricamente consistente.

2.4 Implementação Computacional

```python
import turtle
import math

def draw_hexagon(t, size, depth=0.3):
    """Desenha um hexágono com Turtle, simulando profundidade."""
    t.pensize(2)  # Bordas mais grossas como na imagem
    for _ in range(6):
        t.forward(size)
        t.left(60)

    # Simula profundidade desenhando um hexágono interno menor
    t.penup()
    t.forward(size * depth)  # Move para dentro
    t.left(30)  # Ajusta para alinhar com o centro
    t.pendown()
```

```
    t.pensize(1)  # Bordas internas mais finas
    for _ in range(6):
        t.forward(size * (1 - depth))
        t.left(60)
    t.penup()
    t.goto(t.xcor() - size * depth, t.ycor())  # Volta ao ponto
       inicial
    t.setheading(t.heading() - 30)  # Corrige orientação

def generate_honeycomb(rows=8, cols=10, hex_size=20):
    """
    Gera uma estrutura de favos de mel hexagonal usando Turtle.

    Parâmetros:
    - rows: Número de linhas de hexágonos.
    - cols: Número de colunas de hexágonos.
    - hex_size: Tamanho do lado do hexágono.

    Baseado na tesselação {6,3} (Schläfli) e otimizado para parecer
       com a imagem.
    """
    t = turtle.Turtle()
    t.speed(0)  # Máxima velocidade
    t.hideturtle()  # Esconde o Turtle para melhor visualização

    # Configurações de cor baseadas na imagem (tons de mel)
    t.fillcolor("#FFD700")  # Dourado mel
    t.pencolor("#DAA520")  # Bege acastanhado para bordas

    # Distâncias para posicionamento hexagonal
    hex_width = math.sqrt(3) * hex_size
    hex_height = 2 * hex_size

    for row in range(rows):
        y = row * (3/2 * hex_size)  # Offset vertical
        for col in range(cols):
```

```
            # Offset horizontal para linhas pares/ímpar
            x_offset = (col * hex_width) + (row % 2) * (hex_width /
                2)

            # Move para a posição inicial
            t.penup()
            t.goto(x_offset - (cols * hex_width / 2), -y - (rows *
                hex_height / 4))  # Centraliza
            t.pendown()

            # Desenha e preenche o hexágono
            t.begin_fill()
            draw_hexagon(t, hex_size, depth=0.3)
            t.end_fill()

    # Mantém a janela aberta
    turtle.done()

# Exemplo de uso
if __name__ == "__main__":
    generate_honeycomb(rows=8, cols=10, hex_size=20)
```

3 Parte Experimental

3.1 Contexto e Abordagem Metodológica

A componente experimental deste trabalho corresponde à elaboração e explicação do código para análise computacional da geometria hexagonal em favos de mel. Tal como num procedimento laboratorial, importa detalhar a lógica implementada, os algoritmos utilizados e as opções tomadas em cada etapa, de forma a permitir a replicação do processo.

A investigação parte da **Honeycomb Conjecture**, que estabelece o padrão hexagonal como a forma mais eficiente para particionar o plano em regiões de igual área com menor perímetro total. Esta fundamentação teórica justifica a escolha do hexágono regular como objeto central de estudo.

3.2 Implementação Computacional

A implementação foi realizada utilizando a biblioteca `turtle` do Python, escolhida pela sua capacidade de produzir representações gráficas bidimensionais de maneira simples e eficaz. O desenvolvimento seguiu uma abordagem baseada em tesselações [**mathstackexchange_tiling_notations_2022**], considerando notações padronizadas para empacotamento de polígonos.

3.2.1 Algoritmo Principal

O algoritmo central, implementado na função `generate_honeycomb`, organiza uma estrutura hexagonal que reflete os padrões naturais dos favos de mel. A lógica de posicionamento utiliza um sistema de coordenadas baseado em deslocamentos regulares, assegurando uma disposição simétrica e contínua das células:

$$x_{\text{offset}} = \sqrt{3} \times \text{hex\_size} \times \text{col} \tag{8}$$

$$y_{\text{offset}} = 1.5 \times \text{hex\_size} \times \text{row} \tag{9}$$

Esta abordagem considera as propriedades de subpavimentação com hexágonos [**mathstackexchange_hexagon_subpaving_2023**], garantindo cobertura total do plano sem sobreposições.

3.2.2 Desenho de Hexágonos Individuais

A função `draw_hexagon` foi desenvolvida para traçar cada hexágono individual, incorporando um elemento interno menor com um fator de profundidade de $0,3$, visando simular a tridimensionalidade observada na natureza. A generalização para outros polígonos regulares foi considerada com base em estudos sobre tesselações com pentágonos [**mathstackexchange_pentagon_tiling_2023**].

3.3 Parâmetros e Configurações

Foram estabelecidos parâmetros específicos para otimização da visualização:

- Número de linhas: rows = 8

- Número de colunas: cols = 10
- Tamanho do hexágono: hex_size = 20
- Cores: Preenchimento dourado (`#FFD700`) e bordas em bege (`#DAA520`)
- Espessura das bordas: pensize = 2

3.4 Processo de Construção e Validação

O processo de construção segue uma abordagem iterativa, simulando um crescimento progressivo a partir de um ponto central. A centralização da grade é assegurada por deslocamentos proporcionais às dimensões totais da estrutura:

$$x_{\text{start}} = -\frac{\text{cols} \times \sqrt{3} \times \text{hex\_size}}{2} \tag{10}$$

$$y_{\text{start}} = -\frac{\text{rows} \times 1.5 \times \text{hex\_size}}{2} \tag{11}$$

A validação geométrica considerou também aspectos de tesselações hiperbólicas [**mathstackexchange_hyperbolic_tessellations_2013**], embora o foco tenha permanecido no plano euclidiano.

3.5 Replicabilidade e Extensões

Esta metodologia permite que o código seja replicado em qualquer ambiente Python com a biblioteca `turtle` devidamente instalada. A arquitetura modular do sistema permite futuras extensões para:

- Análise de eficiência de empacotamento
- Simulação de deformações estruturais
- Estudo de transições para outros polígonos regulares
- Aplicação em contextos de geometria hiperbólica

A implementação serve assim como ferramenta versátil para exploração computacional de padrões geométricos na natureza.

4 Discussão dos Resultados

Apresentam-se e analisam-se, nesta secção, as imagens geradas automaticamente pelo código. Não foram utilizadas capturas de ecrã, mas sim exportações directas produzidas pelo programa. Discutem-se as semelhanças e diferenças entre os resultados e a imagem de referência, identificando as causas dos desvios e avaliando a qualidade da aproximação obtida.

5 Conclusões

As conclusões são redigidas a partir da análise dos resultados. Evitam-se afirmações superficiais ou subjectivas; privilegiam-se observações fundamentadas, como, por exemplo:

- O modelo reproduz com fidelidade parcial a forma natural seleccionada.
- As limitações decorrem de aproximações matemáticas ou restrições do ambiente de programação.
- Futuras melhorias poderão incluir optimizações algorítmicas ou refinamentos gráficos.

6 Bibliografia