

Larissa Gabriela Barrozo da Silva Moura

## Documentação de Implementação: Visualizador do fractal de Mandelbrot

### 1. Introdução

Este documento descreve a implementação do Mandelbrot Explorer, um projeto interativo de visualização de fractais. O objetivo central do projeto é demonstrar a interoperabilidade entre duas linguagens de programação com paradigmas e vocações distintas:

1. **Python: Utilizado para a camada de interface e gerenciamento de estado.**
2. **C: Utilizado para o motor de cálculo de alta performance.**

A aplicação utiliza uma arquitetura onde o Python atua como se fosse o *frontend* gráfico e o C como o *backend* de cálculo, comunicando-se através de uma biblioteca compartilhada.

### 2. Visão Geral da Aplicação

A aplicação renderiza o conjunto de Mandelbrot definido pela função iterativa complexa:

$$z_{n+1} = z_n^2 + c$$

Onde:

- $z$  é um número complexo iniciando em (0,0).
- $c$  é a constante complexa correspondente ao ponto (pixel) no plano.

O número de iterações necessárias para a sequência divergir determina a cor do pixel. A aplicação suporta:

- Zoom Interativo: Aproximação e distanciamento via cliques do mouse.
- Navegação (Panning): Centralização do fractal no ponto clicado.
- Ajuste de Iterações: Controle em tempo real da precisão do cálculo através de um slider.
- Estatísticas: Exibição do tempo de processamento e dados do plano complexo.

### 3. Responsabilidades das Linguagens

#### 3.1 O Python é responsável por:

- Criar a janela gráfica e elementos de controle (via Tkinter).

- Manipular entradas do usuário (cliques de mouse e ajuste de parâmetros).
- Gerenciar o estado do programa (coordenadas do plano, nível de zoom).
- Orquestrar a execução via threads para manter a interface responsiva durante o cálculo.
- Converter os dados brutos de pixels em imagens visíveis (via Pillow).

### 3.2 O C é responsável por:

- Implementar o algoritmo de escape de Mandelbrot de forma otimizada.
- Gerenciar buffers de memória para armazenamento de pixels e dados de iteração.
- Implementar o algoritmo de coloração por histograma, que garante uma transição suave de cores e melhor percepção visual do fractal.
- Exportar as funções através de uma biblioteca dinâmica (.so ou .dll).

## 4. Interface entre Python e C

A comunicação entre as linguagens é implementada através da biblioteca `ctypes`, permitindo chamadas de funções nativas e compartilhamento de memória.

### 4.1 Carregamento da Biblioteca Compartilhada

O objeto compartilhado compilado é carregado dinamicamente:

**Python:**

```
lib = ctypes.CDLL("./mandelbrot.so") # ou .dll no Windows
```

### 4.2 Declaração da Assinatura da Função

Os tipos de argumentos e o tipo de retorno são declarados para garantir a conversão correta de tipos entre as linguagens:

```
lib.calcular_mandelbrot.restype = ctypes.POINTER(DadosMandelbrot)
```

```
lib.calcular_mandelbrot.argtypes = [
```

```
    ctypes.c_int, ctypes.c_int,    # largura, altura
```

```
    ctypes.c_double, ctypes.c_double, # x_min, x_max
```

```
    ctypes.c_double, ctypes.c_double, # y_min, y_max
```

```
    ctypes.c_int                # max_iteracoes
```

### 4.3 Estratégia de Compartilhamento de Memória

Para garantir que ambas as linguagens "entendam" o mesmo bloco de memória, definimos uma struct idêntica em ambos os lados:

**No C (mandelbrot.h):**

**No Python (main.py):**

A comunicação utiliza uma estrutura de dados (struct) definida em C e espelhada em Python. O C aloca um buffer de memória para os pixels:

```
dados->pixels = (uint8_t*)malloc(largura * altura * 3);
```

O Python acessa essa memória diretamente através de um ponteiro, evitando cópias desnecessárias e garantindo alta performance na transferência de dados da imagem:

1. O Python chama a função em C.
2. O C aloca o buffer de pixels via malloc.
3. O C preenche o buffer e retorna o ponteiro da estrutura.
4. O Python lê diretamente desse endereço de memória usando PIL.Image.frombytes:
5. O Python ordena a liberação da memória no C para evitar *memory leaks*.

### 5. Método de Interface (FFI via ctypes)

A integração entre as duas linguagens ocorre através uma **Foreign Function Interface (FFI)** utilizando o módulo ctypes do Python.

O método de comunicação baseia-se em **Compartilhamento de Memória via Estruturas**:

1. **Mapeamento de Tipos:** No C, define-se uma struct contendo as dimensões da imagem e o ponteiro para o array de pixels (uint8\_t\*). No Python, uma classe equivalente é criada herdando de ctypes.Structure.
2. **Carregamento Dinâmico:** O Python carrega a biblioteca compilada (.dll ou .so) em tempo de execução.
3. **Ponteiros de Memória:** O Python chama a função em C passando parâmetros de configuração. O C aloca a memória necessária, preenche o buffer com os cálculos e retorna o ponteiro da estrutura para o Python. O Python então lê diretamente esse endereço de memória para gerar a imagem.

#### 4. Análise de Dependência Mútua

O projeto foi construído de modo que as linguagens sejam codependentes:

3. **Indispensabilidade do C:** O Python não possui a lógica do fractal. Sem a biblioteca compilada em C, o interpretador dispara uma exceção de carregamento e a aplicação é interrompida.
  - Além de que o mandelbrot.c foi compilado como uma biblioteca dinâmica (.so ou .dll). Ele não possui interface própria e depende do Python para receber as coordenadas (x\_min, x\_max, etc.) e para exibir o resultado.
4. **Indispensabilidade do Python:** O código C foi desenvolvido estritamente como uma biblioteca dinâmica. Ele é incapaz de ser executado como um programa autônomo, dependendo do Python para o fornecimento de parâmetros (coordenadas, dimensões, iterações), que retorna um ponteiro para os pixels, e o controle de fluxo e exibição gráfica.
  - Além de que a dependência do Python também vem de se não encontrar mandelbrot.dll/.so, o programa dá erro e não funciona. Ou seja, O Python **não funciona** sem a biblioteca C (dá erro se a DLL não existir), do qual faz via *ctypes*.
  - *O main.py gerencia o ciclo de vida, a interface gráfica (Tkinter) e a manipulação de imagens (Pillow). Sem o C, ele não possui o algoritmo de cálculo e não consegue renderizar nada.*

#### 5. Conclusão

O projeto demonstra com sucesso a integração entre C e Python, aplicando cada linguagem onde ela é mais eficiente. O uso de **FFI (Foreign Function Interface)** via *ctypes* permitiu o compartilhamento direto de memória, resultando em uma aplicação gráfica fluida e de alta performance. O trabalho destaca conceitos fundamentais de linguagens de programação, como gerenciamento de memória, tipos de dados e interoperabilidade.

#### 6 Referências

- Documentação oficial do Python (*ctypes*).
- Algoritmo de Coloração por Histograma para Mandelbrot (Wikipedia).
- Código base de fractais em C (recursos da disciplina).