Larissa Gabriela Barrozo da Silva Moura

Documentação de Implementação: Visualizador do fractal de Mandelbrot

1. Introdução

Este documento descreve a implementação do Mandelbrot , um projeto interativo de visualização de fractal. 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 - Interoperabilidade

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

4.3.1. No C (main.c) :

Python.h no main.c

O arquivo main.c que usa #include <Python.h> é uma **Extensão Nativa**.

- **Tradução de Tipos:** PyArg_ParseTuple é uma conversa onde o C "pergunta" ao interpretador Python: *"O que tem dentro desse objeto Python?"* e o converte para double ou int.
- **Criação de Objetos:** Com Py_BuildValue, o C está ativamente criando um objeto que o Python entende. O C está "falando" a linguagem do coletor de lixo (Garbage Collector) do Python

4.3.2No Python (main.py):

ctypes no main.py

No main.py, usa a biblioteca ctypes., permitindo o **compartilhamento de memória binária**.

- **Espelhamento de Estruturas:** se define class DadosMandelbrot(ctypes.Structure), do qual se diz ao Python para reservar um bloco de memória que segue exatamente o layout da struct do C.
- **Conversa via Ponteiros:** O comando ponteiro.contents acessa a memória RAM que o C alocou. O Python não "copia" os dados nesse momento, ele olha para o mesmo endereço de memória que o C acabou de escrever.
- **Leitura de Bytes Brutos:** O uso de ctypes.string_at(dados.pixels, tamanho_buffer) é o ponto alto da conversa. O Python entra "dentro" do ponteiro de pixels gerado pelo C e lê os dados brutos para montar a imagem

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.

6. Análise de Dependência Mútua

O projeto foi arquitetado sob o conceito de **Programação Híbrida**, onde existe uma simbiose técnica entre o Python e o C, tornando-os codependentes para o funcionamento do sistema:

6.1. Indispensabilidade do C:

Aritmética de Ponto Flutuante: O Python não possui a lógica de cálculo do fractal internamente. Sem a biblioteca compilada em C, a aplicação é incapaz de gerar a matriz de pixels.

Otimização de Memória: O motor em C realiza o pós-processamento de cores via **Histogram Coloring** diretamente ao nível de bytes (uint8_t), uma tarefa que em Python puro resultaria em latência inaceitável para uma aplicação em tempo real.

Binário Específico: O arquivo mandelbrot.c é compilado como uma biblioteca dinâmica (.so ou .dll). Ele é um "motor sem chassi": possui a força bruta, mas não possui interface, dependendo do Python para receber os parâmetros de entrada.

6.2 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.

O C é "cego" quanto à interação do usuário. É o Python que executa a **Matemática da Câmera**, convertendo cliques no Canvas em coordenadas do plano complexo e aplicando **Zoom Logarítmico** antes de despachar os dados para o C.

Garantia de Ciclo de Vida (Memory Safety): Como o C não possui *Garbage Collector*, o Python atua como o **Gestor de Memória**. Via blocos try/finally, o Python garante que a memória alocada no *Heap* do C seja obrigatoriamente liberada, prevenindo *Memory Leaks* que travariam o sistema operacional.

Ponte de Dados (FFI): O Python utiliza a biblioteca ctypes para realizar a tradução de tipos de dados. Sem essa camada, o resultado binário do C seria apenas um bloco de memória bruto e ilegível para o usuário final.

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), 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.*

O main.py garante Robustez. Se o C falhar, o Python captura a exceção; se o Python for removido, o código C permanece inerte por ser uma biblioteca estritamente passiva.

7. 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. Essa arquitetura demonstra o domínio de Interoperabilidade entre Linguagens, requisito central da disciplina. O trabalho destaca conceitos fundamentais de linguagens de programação, como gerenciamento de memória, tipos de dados e interoperabilidade.

8 Referências

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