Trabalho de Implementação

Juliano S. Fonseca[1], **Fábio C. Cardoso**[1], **João Vitor M. Lago**[1]

[1]Centro de Desenvolvimento Tecnológico – Universidade Federal de Pelotas (UFPel)
Pelotas – RS – Brasil

1. Introdução

Este trabalho descreve a implementação de um visualizador do conjunto de Mandelbrot, desenvolvido como trabalho de implementação disciplina de Conceitos de Linguagens de Programação. O projeto vai combinar duas linguagens: C++ para o cálculo e Python para a interface gráfica e também a geração de imagens.

O conjunto de Mandelbrot é definido no plano complexo. A implementação é bastante simples, para cada ponto c, é repetida a operação $z_{n+1} = z_n^2 + c$ a partir de $z_0 = 0$. Se o valor de z não diverge após um número máximo de iterações, o ponto vai ser considerado parte do conjunto.

2. Estrutura do projeto

O repositório contém os seguintes arquivos:

- `main.cpp` — biblioteca compartilhada em C++ que calcula as iterações do conjunto de Mandelbrot para cada pixel de uma grade.
- `mandelbrotUI.py` — interface gráfica em Python (Tkinter com Pillow) que permite visualizar o fractal, aplicar zoom com o mouse e ajustar o número máximo de iterações.
- `mandelbrot_case.py` — script Python que gera imagens PNG de três regiões pré-definidas do conjunto, sem abrir a interface gráfica.
- `Makefile` — compilação da biblioteca e execução dos casos de estudo.
- `README.md` — instruções de compilação e execução.

3. Papel de cada linguagem

3.1. C++ — Cálculo numérico

O arquivo `main.cpp` implementa a função `calculate_mandelbrot`, que recebe as dimensões da imagem, os limites do plano complexo e o número máximo de iterações. Para cada pixel, a função mapeia a posição (x, y) para um número complexo $c = a + bi$ e executa o laço $z_{n+1} = z_n^2 + c$ até que $|z| > 2$ ou o limite de iterações seja atingido. O resultado (número de iterações de cada pixel) é escrito em um buffer de inteiros.

A função é compilada como biblioteca compartilhada (`main.so` no Linux, `main.dll` no Windows) e exportada com `extern "C"` para evitar a *name mangling* do C++.

3.2. Python — Interface e visualização

O arquivo `mandelbrotUI.py` cuida de toda a parte visual:

- Cria uma janela com Tkinter onde o fractal é desenhado.

- Permite zoom com seleção retangular pelo mouse.
- Possui controles para ajustar o número máximo de iterações e resetar a vista.
- Converte a matriz de iterações em uma imagem colorida usando NumPy e Pillow.

O arquivo `mandelbrot_case.py` funciona de forma semelhante, mas sem interface gráfica. Ele gera três imagens PNG correspondentes a diferentes regiões do conjunto:

1. **Visão geral** — o conjunto inteiro, com 256 iterações.
2. **Seahorse Valley** — zoom na região entre o cardioide principal e o bulbo de período 2, onde surgem padrões que lembram cavalos-marinhos (512 iterações).
3. **Espiral** — zoom na fronteira superior do cardioide, onde aparecem espirais formadas por minicópias do conjunto (1024 iterações).

4. Interface entre C++ e Python

A comunicação entre as duas linguagens é feita por meio da biblioteca `ctypes`, que faz parte da biblioteca padrão do Python. O mecanismo funciona assim:

1. O código C++ é compilado como biblioteca compartilhada com `extern "C"`, o que garante que o nome da função no binário seja simples e previsível.
2. No lado Python, `ctypes.CDLL` carrega a biblioteca compartilhada em tempo de execução.
3. Os tipos dos argumentos e do retorno da função são declarados manualmente com `argtypes` e `restype`, para que o `ctypes` saiba como converter os dados entre Python e C.
4. Um buffer de inteiros é alocado em Python com `ctypes.c_int * (width * height)` e passado como ponteiro para a função C++. A função preenche esse buffer diretamente na memória.
5. Após a chamada, o buffer é convertido em um array NumPy com `np.frombuffer` para facilitar a manipulação e colorização.

Essa abordagem não exige nenhuma dependência extra além do compilador C++ e do Python com NumPy e Pillow. Não é necessário escrever *bindings* manuais nem usar ferramentas como SWIG ou pybind11.

5. Conclusão

O projeto demonstra uma forma direta de combinar C++ e Python: o cálculo pesado (iteração pixel a pixel) fica em C++, que é compilado e significativamente mais rápido, enquanto a interface gráfica e a manipulação de imagens ficam em Python, onde essas tarefas são mais simples de implementar. A ponte entre as duas linguagens é feita pelo `ctypes`, sem dependências externas além da compilação da biblioteca compartilhada.