UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Ciência da Computação

FRACTAL DE MANDELBROT COM C++ E PYTHON

Trabalho avaliativo para a disciplina de Conceitos de Linguagens de Programação, ministrada pelo Prof. Gerson Geraldo H. Cavalheiro, realizado pelos discentes Leonardo Braga Westphal e Rafael da Silva de Freitas.

Pelotas
2024

1. INTRODUÇÃO

Este projeto tem como objetivo desenvolver uma aplicação gráfica que renderiza o Fractal de Mandelbrot, utilizando C++ para realizar cálculos matemáticos e Python para a interface gráfica. Ele é um dos exemplos mais icônicos de sistemas dinâmicos e geometria fractal, conhecido por sua complexidade visual e beleza matemática. A combinação de C++ e Python permite alavancar a eficiência computacional de C++ com a versatilidade e simplicidade de Python na criação de interfaces gráficas.

1.1 OBJETIVO E JUSTIFICATIVA

O principal objetivo desta aplicação é permitir a visualização do Fractal de Mandelbrot, focando na integração de C++ e Python. Ele é um excelente exemplo de um sistema que, apesar de ser definido por regras simples, gera padrões de complexidade infinitos, proporcionando uma rica experiência visual. Além disso, a implementação deste fractal exige um bom entendimento de conceitos matemáticos e de otimização de código, tornando-o uma escolha ideal para demonstrar habilidades em programação e desenvolvimento de software gráfico.

2. FUNDAMENTAÇÃO TEÓRICA

O Fractal de Mandelbrot é definido no plano complexo e é formado pelo conjunto de pontos c para os quais a sequência de números complexos gerada pela iteração da função quadrática $f_c(z) = z^2 + c$ não tende ao infinito quando iterada a partir de $z = 0$. Em termos matemáticos, o conjunto de Mandelbrot M pode ser descrito como:

$$M = \{c \in \mathbb{C} : \limsup_{n\to\infty} |z_{n+1}| < \infty \quad \text{onde} \quad z_{n+1} = z_n^2 + c \text{ e } z_0 = 0\}$$

Figura 1 - Termo Matemático do Conjunto de Mandelbrot

Para cada valor de c, se a magnitude de z_n permanecer limitada (i.e., $|z_n| \leq 2$ para todos os n), então o ponto c pertence ao conjunto de Mandelbrot. Graficamente, os pontos que pertencem ao conjunto são coloridos de preto, enquanto os pontos que não pertencem são coloridos com base no número de iterações necessárias para que a magnitude de z_n exceda um certo limite.

Os fractais foram popularizados pelo matemático Benoît B. Mandelbrot na década de 1970, quando ele começou a explorar formas geométricas complexas que exibiam auto-similaridade em diferentes escalas. O conjunto de Mandelbrot, em particular, foi visualizado pela primeira vez em computadores na década de 1980, abrindo caminho para novas maneiras de entender estruturas matemáticas complexas e infinitas.

Fractais, como o de Mandelbrot, possuem propriedades únicas, como a auto-similaridade, onde uma pequena porção do fractal, quando ampliada, se assemelha ao todo. Isso os torna úteis em várias áreas, incluindo modelagem de fenômenos naturais (como montanhas, linhas costeiras e árvores), compressão de imagens, e até mesmo na criação de arte digital.

Na computação gráfica, fractais são utilizados não apenas pela sua beleza visual, mas também por suas propriedades matemáticas que permitem a geração de padrões infinitamente complexos com relativamente pouca informação inicial. A exploração dos fractais permite um profundo entendimento

da relação entre a matemática pura e suas aplicações práticas, tornando o estudo do conjunto de Mandelbrot uma excelente introdução ao mundo dos sistemas dinâmicos e da geometria fractal.

3. ARQUITETURA

A arquitetura do sistema desenvolvido para renderizar o Fractal de Mandelbrot é baseada na integração entre C++ e Python, onde C++ é utilizado para os cálculos matemáticos intensivos e Python para a interface gráfica e gerenciamento da aplicação. A seguir, são descritos os componentes principais dessa arquitetura.

3.1. CÁLCULO DO FRACTAL EM C++ (mandelbrot.cpp e mandelbrot.h)

O cálculo do Fractal de Mandelbrot é realizado em C++ devido à necessidade de desempenho e eficiência no processamento de operações matemáticas intensivas. O arquivo '$mandelbrot.cpp$' contém a função '$calculate\_mandelbrot$', responsável por gerar a imagem do fractal. Essa função recebe as dimensões da imagem, o número máximo de iterações e as coordenadas que definem a área do plano complexo a ser renderizada.

- **Função** '$calculate\_mandelbrot$'**:** Implementada como uma função exportada através de '$\_\_declspec(dllexport)$' para ser acessível a partir de Python. Ela itera sobre cada pixel da imagem, calculando o valor correspondente no plano complexo e determinando a cor com base no número de iterações até que o valor escape de um determinado limite (normalmente 2).

- **Parâmetros e Otimização:** A função aceita como parâmetros a largura e altura da imagem, o número máximo de iterações ('$max\_iteration$'), as coordenadas do plano complexo ('p_x1', 'p_y1', 'p_x2', 'p_y2') e um ponteiro para o array de saída ('$output$'), onde as cores são armazenadas.

- **Bibliotecas e Complexidade:** A biblioteca ' $< complex >$ ' do C++ é utilizada para facilitar as operações com números complexos, e o uso de uma DLL permite a integração eficiente com a interface Python.

O arquivo '$mandelbrot.h$' define as macros para exportação da função, tornando possível a compilação do código em formato de DLL, que pode ser carregada e usada em Python.

3.2 INTERFACE GRÁFICA E INTEGRAÇÃO COM PYTHON (gui.py)

A interface gráfica é implementada em Python, utilizando bibliotecas como '$ctypes$', '$numpy$', '$pillow$' e '$tkinter$'. A escolha do Python para essa parte do sistema se dá pela sua simplicidade e poder na criação de GUIs interativas.

- **Integração com C++:** O arquivo '$gui.py$' usa a biblioteca '$ctypes$' para carregar a DLL gerada a partir do código C++ (mandelbrot.dll) e configurar os tipos de argumentos necessários para a chamada da função '$calculate\_mandelbrot$'.

- **Função 'generate_mandelbrot_image':** Esta função em Python chama a função C++ passando os parâmetros necessários e recebe o array de pixels gerado. Os dados são então organizados em uma matriz tridimensional usando '*numpy*', representando a imagem RGB.

- **Exibição da Imagem:** A função '*display_image*' utiliza a biblioteca '*pillow*' para converter o array de pixels em uma imagem, que pode ser visualizada usando o método '*show*()'.

- **Interface com Tkinter:** A GUI é construída com '*tkinter*', onde o usuário pode gerar o fractal clicando em um botão. A interface captura os parâmetros para a renderização (dimensões, número de iterações e coordenadas) e passa esses valores para a função de geração da imagem.

3.3 COMPILAÇÃO E SOLUÇÃO NO VISUAL STUDIO (mandelbrot.sln)

A solução '*mandelbrot.sln*' configura o projeto no Visual Studio, onde o código C++ é compilado em uma DLL para diferentes configurações (Debug, Release) e arquiteturas (x64, x86). Essa configuração permite a compilação flexível e a integração fácil com a interface Python.

- **Compilação da DLL:** O arquivo de solução define as configurações de compilação necessárias para gerar a DLL ('*mandelbrot.dll*'), que é então utilizada pelo script Python para realizar os cálculos do fractal.

3.4 EXECUÇÃO

Após a compilação, a DLL gerada ('*mandelbrot.dll*') é armazenada na pasta x64/Release/. O script Python ('*gui.py*') faz referência a esse caminho para carregar a DLL e executar as funções necessárias.

- **Execução da Aplicação:** Com a DLL carregada, o usuário pode executar o script '*gui.py*', que apresentará a interface gráfica, permitindo a exploração do Fractal de Mandelbrot através da interação com a GUI.

Essa arquitetura modular, que separa os cálculos de alto desempenho em C++ da interface gráfica em Python, resulta em uma aplicação eficiente e intuitiva, combinando o poder do processamento em C++ com a flexibilidade e facilidade de desenvolvimento de Python.

4. RESULTADOS

A aplicação desenvolvida conseguiu gerar com sucesso o Fractal de Mandelbrot, permitindo a visualização da estrutura fractal. A combinação de C++ e Python resultou em uma aplicação eficiente, capaz de processar e renderizar o fractal rapidamente. A seguir vemos alguns dos resultados dos casos de estudo:

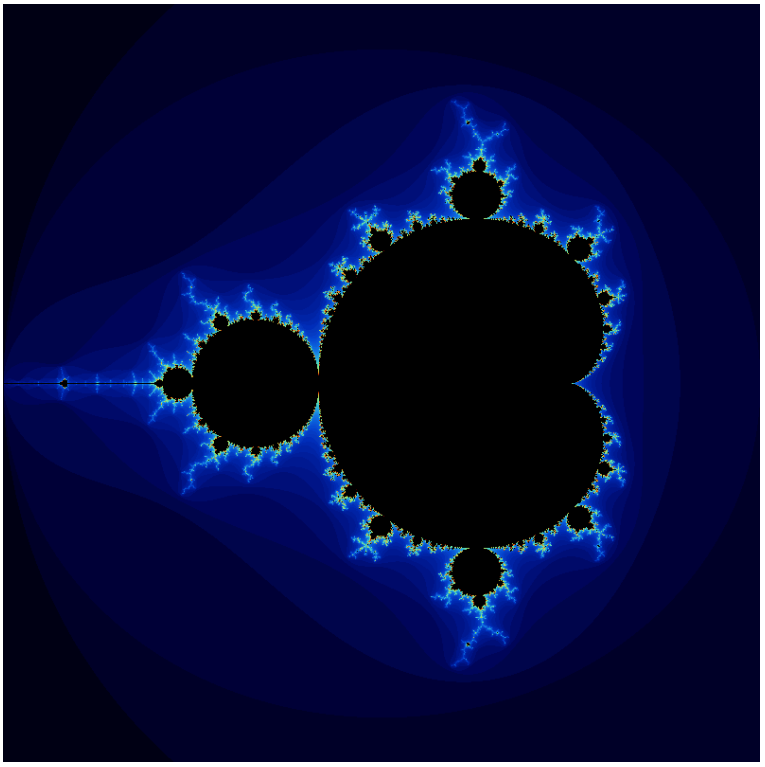

Figura 2 - Visão Ampla Padrão

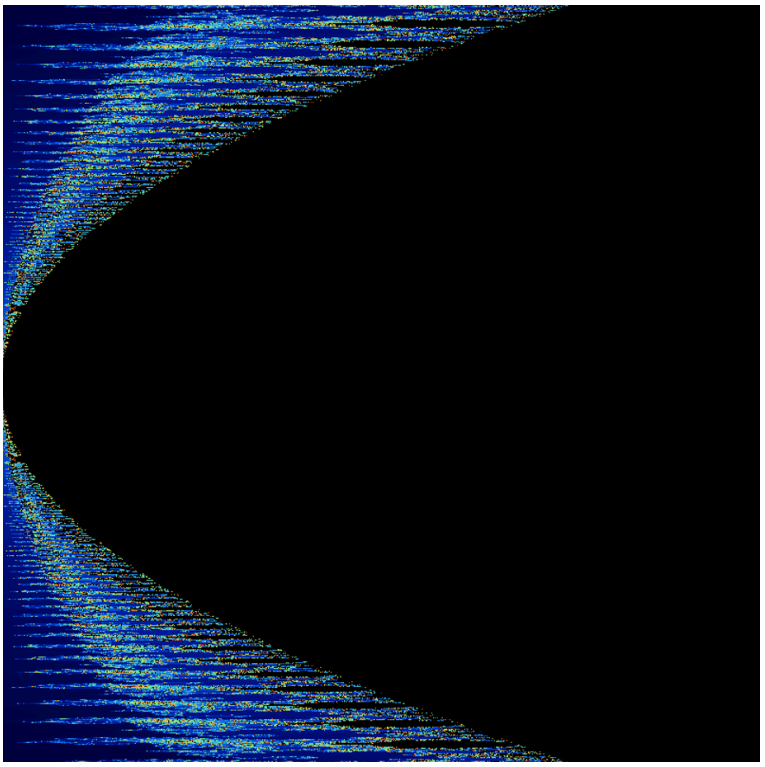

Figura 3 - Zoom em um Ramificação

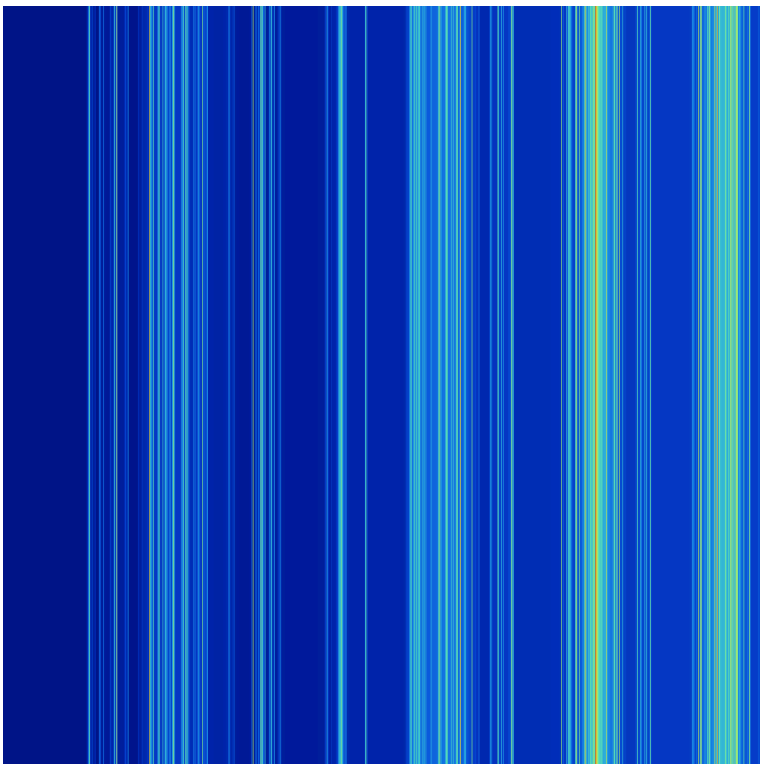

Figura 4 - Zoom Extremo em uma Característica Específica

O sistema também demonstrou a flexibilidade da interface gráfica em Python, permitindo aos usuários ajustar parâmetros, oferecendo uma experiência intuitiva e interativa. Os resultados validaram a escolha das tecnologias e destacaram a precisão e a beleza dos padrões gerados, evidenciando o potencial do Fractal de Mandelbrot tanto em termos matemáticos quanto visuais.

5. CONCLUSÃO

Ao longo do projeto, foi possível observar a complexidade e a beleza do Fractal de Mandelbrot, uma estrutura que, embora baseada em regras matemáticas simples, gera padrões infinitamente detalhados. A implementação permitiu explorar conceitos importantes como o mapeamento de coordenadas no plano.

Além disso, a integração entre C++ e Python mostrou-se uma solução viável e poderosa, combinando o desempenho de uma linguagem de baixo nível com a flexibilidade e facilidade de uso de uma linguagem de alto nível. Essa abordagem também destacou a importância do planejamento cuidadoso na definição das interfaces entre os módulos de código, garantindo a eficiência e a funcionalidade da aplicação final.

Em resumo, este projeto cumpriu com sucesso seus objetivos técnicos, que era realizar o funcionamento da integração entre duas linguagens de programação de forma bem estruturada.

6. REFERÊNCIAS

IBM. *Benoît B. Mandelbrot: Fractals and the Field of Complex Dynamics*. Disponível em: https://www.ibm.com/history/benoit-mandelbrot. Acesso em: 05 ago. 2024.

Fractal Foundation. *The Mandelbrot Set: Fractals and the Beauty of Mathematics*. Disponível em: https://fractalfoundation.org/OFC/OFC-5-5.html. Acesso em: 05 ago. 2024.

7. APÊNDICES

Códigos e repositório do projeto.

Repositório: https://github.com/raff08/2024-CLP-MandelbrotFractal

Códigos:

mandelbrot.cpp

```cpp
#include <complex>

extern "C"
{
    __declspec(dllexport) void calculate_mandelbrot(int width, int height, int max_iteration,
float p_x1, float p_y1, float p_x2, float p_y2, int *output)
    {
        for (int y = 0; y < height; ++y)
        {
            for (int x = 0; x < width; ++x)
            {
                std::complex<float> z, c = {p_x1 + ((float)x / width) * (p_x2 - p_x1),
                                            p_y1 + ((float)y / height) * (p_y2 - p_y1)};
                int i = 0;
                while (abs(z) < 2 && ++i < max_iteration)
                    z = z * z + c;

                double t = (double)i / max_iteration;
                int r = (int)(9 * (1 - t) * t * t * t * 255);
                int g = (int)(15 * (1 - t) * (1 - t) * t * t * 255);
                int b = (int)(8.5 * (1 - t) * (1 - t) * (1 - t) * t * 255);

                output[(y * width + x) * 3 + 0] = r;
                output[(y * width + x) * 3 + 1] = g;
                output[(y * width + x) * 3 + 2] = b;
            }
        }
    }
}
```

mandelbrot.h

```cpp
#ifdef MANDELBROT_EXPORTS
#define DLL __declspec(dllexport)
#else
#define DLL __declspec(dllimport)
#endif // MANDELBROT_EXPORTS
```

gui.py

```python
import ctypes
import numpy as np
from PIL import Image
import tkinter as tk
import os
import argparse

# Configurações de argparse
parser = argparse.ArgumentParser(description='Gera uma imagem do Conjunto de Mandelbrot.')
parser.add_argument('--width', type=int, default=800, help='Largura da imagem')
parser.add_argument('--height', type=int, default=800, help='Altura da imagem')
parser.add_argument('--max_iter', type=int, default=100, help='Número máximo de iterações')
parser.add_argument('--x1', type=float, default=-2.0, help='Coordenada x1')
parser.add_argument('--y1', type=float, default=-2.0, help='Coordenada y1')
parser.add_argument('--x2', type=float, default=2.0, help='Coordenada x2')
parser.add_argument('--y2', type=float, default=2.0, help='Coordenada y2')

args = parser.parse_args()

current_dir = os.path.dirname(os.path.abspath(__file__))

dll_path = os.path.join(current_dir, 'mandelbrot', 'x64', 'Release', 'mandelbrot.dll')

mandelbrot_lib = ctypes.CDLL(dll_path)

mandelbrot_lib.calculate_mandelbrot.argtypes = [
    ctypes.c_int,   # width
    ctypes.c_int,   # height
    ctypes.c_int,   # max_iteration
    ctypes.c_float, # p_x1
    ctypes.c_float, # p_y1
    ctypes.c_float, # p_x2
    ctypes.c_float, # p_y2
    ctypes.POINTER(ctypes.c_int) # output array
]

def generate_mandelbrot_image(width, height, max_iteration, p_x1, p_y1, p_x2, p_y2):
    output = np.zeros((width * height * 3), dtype=np.int32)
    mandelbrot_lib.calculate_mandelbrot(width, height, max_iteration, p_x1, p_y1, p_x2, p_y2,
output.ctypes.data_as(ctypes.POINTER(ctypes.c_int)))
    return output.reshape((height, width, 3))

def display_image(image_data):
    img = Image.fromarray(image_data.astype('uint8'))
    img.show()

def main():
    width, height = args.width, args.height
    max_iteration = args.max_iter
    p_x1, p_y1 = args.x1, args.y1
    p_x2, p_y2 = args.x2, args.y2

    root = tk.Tk()
    root.title("Mandelbrot Set")

    def on_generate():
        image_data = generate_mandelbrot_image(width, height, max_iteration, p_x1, p_y1, p_x2, p_y2)
        display_image(image_data)

    generate_button = tk.Button(root, text="Generate Mandelbrot", command=on_generate)
    generate_button.pack()

    root.mainloop()

if __name__ == "__main__":
    main()
```