Trabalho 3: Fractal Mandelbrot em OpenGL 4.0

Guilherme Meneghetti Einloft

1. Introdução

O conjunto de Mandelbrot é um conjunto dos números complexos c em que a sequência
$z_0 = 0$
$z_{n+1} = z_n^2 + c$
não diverge. Reescrevendo a sequência em termos de suas partes real e imaginária, temos:
$x_{n+1} = x_n^2 - y_n^2 + a$
$y_{n+1} = 2 x_n y_n + b.$
Essa representação é útil para visualização do gráfico do conjunto.

O conjunto de Mandelbrot, enquanto disposto no plano 2D, apresenta diversas cópias do conjunto, que podem ser vistas caso seja dado cada vez mais zoom, portanto, é considerado um Fractal.

2. Implementação

A implementação da demonstração foi feita usando OpenGL 4.0 e GLSL, usando como base a demonstração feita por **John Tsiombikas** e **Brennen Green**, que são baseadas no algoritmo de **escape time**, que será explicado na seção sobre o shader de fragmento. A demonstração se divide em três partes:

2.1. Código principal

No código principal estão:

- Carregamento e inicialização das bibliotecas usadas (OpenGL, GLEW, GLFW);
- Criação do VBO e VAO do quadrado onde será mostrado o fractal;
- Carregamento dos shaders de vértice e fragmento;
- Funções de deslocamento da tela e escala, a partir do teclado.

2.2. Shader de vértice

O shader de vértice é responsável apenas por mostrar o quadrado na tela. Não é necessária nenhuma transformação, pois estamos apenas mostrando um quadrado posicionado entre as coordenadas (-1, -1) e (1, 1).

2.3. Shader de fragmento

O shader de fragmento é mais complexo, pois é o responsável pelo cálculo dos pontos que estão presentes no fractal, e portanto exige uma compreensão maior.

As variáveis uniformes que o shader recebe são:

- **scale (float):** escala das coordenadas
- **center (vec2):** deslocamento do centro do conjunto
- **aspect (float):** largura da tela dividida pela altura

Agora, vamos entrar um pouco mais a fundo no código do shader:

```
c.x = aspect * scale * (TexCoord.x - 0.5) - center.x;
c.y = scale * (TexCoord.y - 0.5) – center.y;
```

O ponto c é gerado através da coordenada da textura, multiplicado pela escala e deslocado em relação ao centro.

```
z = vec2(0);
z2 = vec2(0);
```

Aqui, os pontos **z** e **z2** (referentes aos pontos z_n e z_{n+1} da sequência) são criados e inicializados com as coordenadas (0, 0).

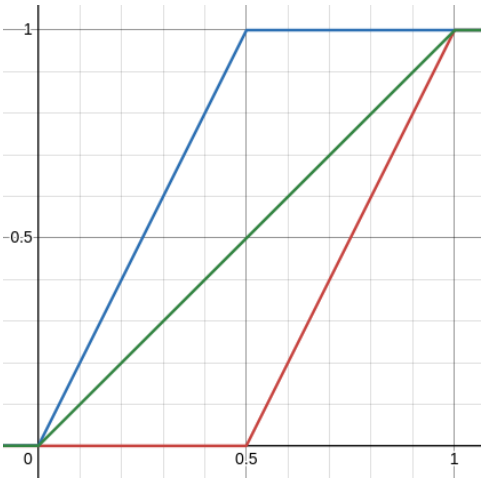
```
for (i = 0; i < iter; i++) {
        z2.x = (z.x * z.x - z.y * z.y) + c.x;
        z2.y = (z.x + z.x) * z.y + c.y;
        if (z2.x * z2.x + z2.y * z2.y > 4.0)
                break;
        z = z2;
}
```

Esse trecho de código é baseado no algoritmo de **escape time**. Nesse algoritmo, o código é executado um número fixo de vezes (no caso do programa, o número é definido no shader) e, caso o comprimento do vetor ultrapasse um valor de escape, nesse caso 2, o algoritmo assume que ele tende ao infinito e para a execução. Caso contrário, o algoritmo assume que a recorrência converge. No programa, o vetor **z2** é utilizado para guardar temporariamente o próximo valor da recorrência.

```
float c_temp = (i == iter) ? 0.0 : float(i) / iter;
FragColor = vec4(c_temp * 2.0 - 1.0, c_temp, c_temp * 2.0, 1.0);
```

Aqui é definida a cor do pixel. Caso o algoritmo execute todas as vezes, ou seja, a sequência converge de acordo com o algoritmo, o pixel é pintado com a cor preta. Caso contrário, o

pixel é pintado com uma cor baseada no número de iterações necessárias para atingir o valor de escape.



Função usada para cada uma das cores (RGB) para definir a cor final do pixel.

3. Uso do programa

O programa é controlado pelas seguintes teclas:

- W: desloca o gráfico para cima (diminui center.y)
- S: desloca o gráfico para baixo (aumenta center.y)
- A: desloca o gráfico para esquerda (aumenta center.x)
- D: desloca o gráfico para direita (diminui center.x)
- Q: diminui o zoom (aumenta scale)
- E: aumenta o zoom (diminui scale)

O programa pode receber zoom infinito (ou até o scale atingir 1.96182e-44), entretanto a partir de certo ponto o programa começa a apresentar artefatos, devido ao zoom alto.

4. Tempo de processamento

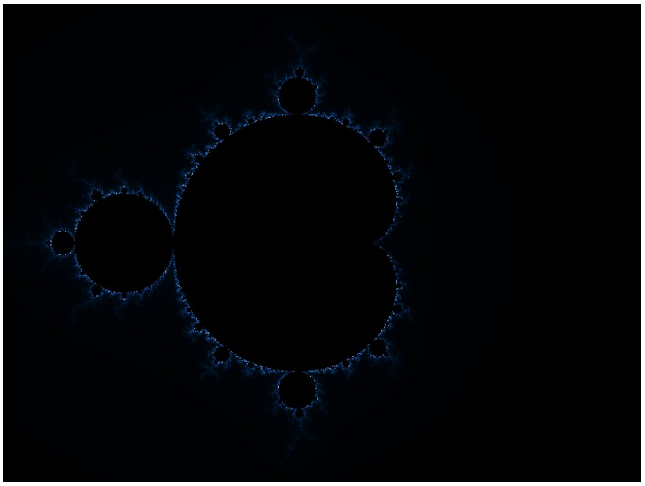
O tempo de processamento do programa é dependente do número de iterações máximo do shader de fragmento. O programa foi executado por 10s em uma AMD Radeon 610M, com resolução de 800x600, sem nenhum input, e apresentou os seguintes resultados:

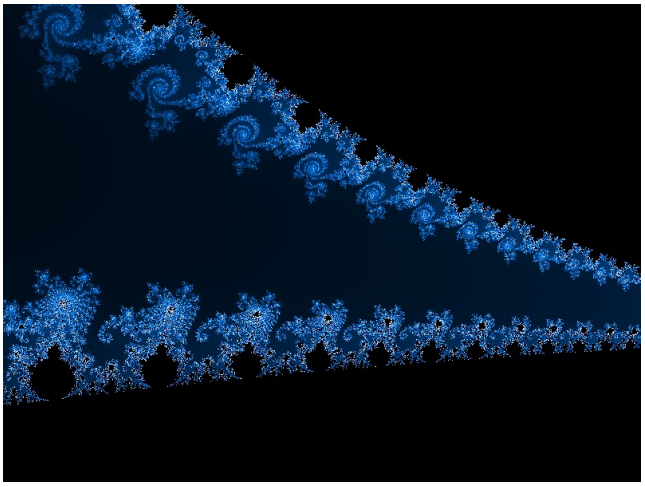
Número de iterações	Menor FPS	FPS médio
256	57.736721	60.259174
512	47.107594	60.286545
1024	32.438046	54.367578
2048	18.746601	45.691036

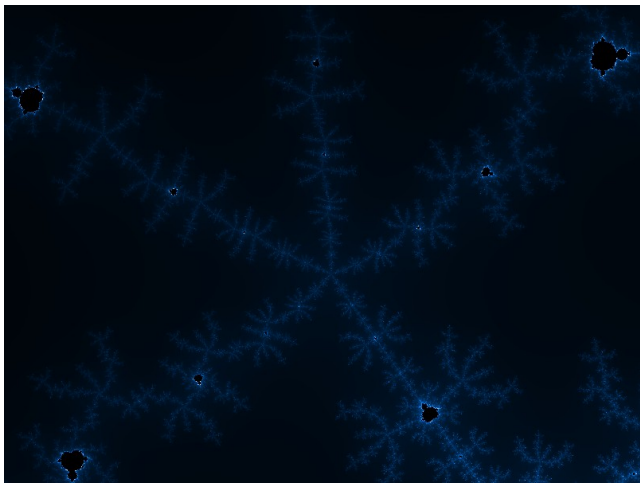
4096	10.524322	24.263832

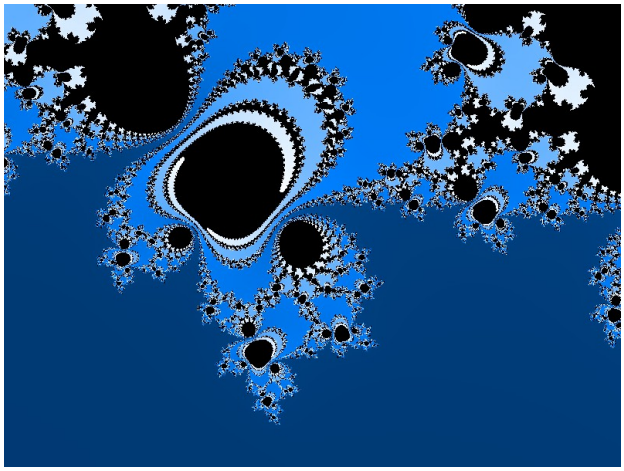
5. Imagens geradas

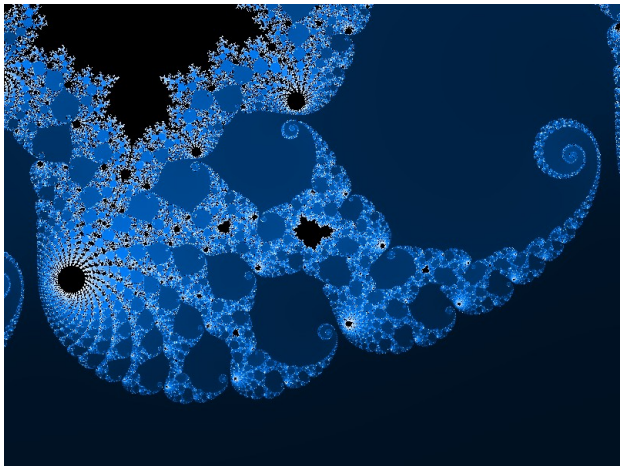
As seguintes imagens foram geradas com resolução de 800x600, com o algoritmo rodando no máximo 512 iterações.











6. Referências

Wikipedia. Conjunto de Mandelbrot. Disponível em: <https://pt.wikipedia.org/wiki/Conjunto_de_Mandelbrot>.

Tsiombikas, J. Fast and easy high resolution fractals with a pixel shader. Disponível em: <http://nuclear.mutantstargoat.com/articles/sdr_fract/>. Acesso em: 27 out. 2025.

Green, B. Mandelbrot Fractals in OpenGL. Disponível em: <https://www.brennengreen.dev/blog/posts/1/>.