Centro de Informática
UFPE

Universidade Federal de Pernambuco
Centro de Informática
Ciência da Computação

Fractais no Plano Projetivo: Um estudo de caso utilizando o Conjunto de Mandelbrot

Adson Paulo da Silva Ramos

Recife, Outubro de 2022

Universidade Federal de Pernambuco
Centro de Informática
Ciência da Computação

Fractais no Plano Projetivo: Um estudo de caso utilizando o Conjunto de Mandelbrot

Aluno: Adson Paulo da Silva Ramos

Orientador: Silvio de Barros Melo

Trabalho de graduação apresentado no Centro de Informática da Universidade Federal de Pernambuco como parte dos requisitos finais para obtenção do grau de Bacharel em Ciência da Computação

Recife, Outubro de 2022

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Ramos, Adson Paulo da Silva.
Fractais no plano projetivo: um estudo de caso utilizando o conjunto de Mandelbrot / Adson Paulo da Silva Ramos. - Recife, 2022.
33p

Orientador(a): Silvio de Barros Melo
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2022.

1. Fractais. 2. Geometria Projetiva. 3. Geometria Computacional. 4. Conjunto de Mandelbrot. I. Melo, Silvio de Barros. (Orientação). II. Título.

000 CDD (22.ed.)

ADSON PAULO DA SILVA RAMOS

Fractais no Plano Projetivo: Um estudo de caso utilizando o Conjunto de Mandelbrot

Trabalho de graduação apresentado no Centro de Informática da Universidade Federal de Pernambuco como parte dos requisitos finais para obtenção do grau de Bacharel em Ciência da Computação

Recife, 27 de Outubro de 2022

BANCA EXAMINADORA

Prof. Silvio de Barros Melo

Prof. Nivan Roberto Ferreira Junior

Agradecimentos

A Deus, pelo dom da vida.

Aos meus pais. Sem eles, certamente, eu não teria chegado até aqui.

Ao Centro de Informática que me proporcionou estar em contato com excelentes professores e alunos, me fornecendo uma excelência profissional.

Ao meu orientador, Silvio Melo, que me ajudou na elaboração desse trabalho.

E aos meus amigos que fiz ao longo dessa trajetória, tornando a caminhada mais leve.

As the world fell, each of us in our own way it was broken. It was hard to know who was more crazy... me... or everyone else.
- Max Rockatansky (Mad Max: Fury Road)

RESUMO

Um dos postulados mais elementar da Geometria afirma que retas paralelas nunca se encontram. De fato, se restringimos a definição de paralelismo à Geometria Euclidiana, a afirmação dada se mantém verídica invariavelmente. No entanto, ao elevarmos o conceito de paralelismo à Geometria Projetiva, os paradigmas antes apresentados começam a se ressignificar. Nessa perspectiva, imagens formadas antes pelo Plano Euclidiano exibem configurações diferentes no Plano Projetivo, inclusive os Fractais, que são, essencialmente, figuras que repetem seus padrões geométricos, ou que suas partes separadas repetem os traços do todo completo [1]. O presente trabalho se propõe a estudar o comportamento dos fractais ao combiná-los com a Geometria Projetiva. Uma atenção especial é dada ao Conjunto de Mandelbrot.

ABSTRACT

One of the most elementary postulates of geometry asserts that parallel straight lines never find each other. Indeed, if we restrict the parallelism definition to Euclidean Geometry, the given proposition keeps truly invariably. Nonetheless, if we raise the concept of parallelism to the Projective Geometry, the paradigms shown before begin to resignify. In this perspective, images formed before in the Euclidean Plane show different shape in the Projective Plane, including the Fractals, that are, essentially, figures that repeat your geometric patterns, or that your parts separately repeat your trace as a whole [1]. The present research work proposes to study the behavior of the fractals when combining them with the Projective Geometry. An special attention is given to the Mandelbrot Set.

Lista de Figuras

Sumário

1 Introdução

Fractal é um objeto geométrico que pode ser dividido em partes, cada uma das quais semelhante ao objeto original. É uma figura da geometria não clássica muito frequente na natureza e no cotidiano, nas mais variadas formas, como em plantas, mapas, comidas e no mercado financeiro. Alguns Fractais podem ser definidos matematicamente. Seu precursor, Benoît Mandelbrot, definiu o termo a partir do adjetivo em latim *fractus*, que significa quebrar [2].

A Geometria Projetiva, por sua vez, nasce da necessidade dos artistas renascentistas em retratarem a realidade em suas pinturas, visando dar aos seus espectadores uma representação naturalista das imagens e não somente uma figura plana, ampliando as fronteiras impostas pela Geometria Euclidiana. [3]

Combinando-se essas duas áreas da matemática, o objetivo principal deste trabalho é visualizar Fractais no Plano Projetivo, explorando conceitos, definições e axiomas da Geometria Projetiva com a complexidade dos Fractais.

2 Referencial Teórico

2.1 Geometria Projetiva

Intuitivamente, pode-se entender a Geometria Projetiva como o ramo da Matemática que estuda as relações entre um objeto do mundo real e sua imagem projetada. É a geometria que descreve a forma como enxergamos e, com isso, observa-se que os limites da Geometria Projetiva transcendem a Geometria Euclidiana e seus postulados.

Sua essência nasce no século XVII, período em que matemáticos necessitam embasar cientificamente as técnicas utilizadas por artistas do Renascimento na construção de seus desenhos.[3]

Formalmente, a Geometria Projetiva é definida como o estudo das propriedades geométricas que são invariantes com respeito às transformações projetivas. [3]

2.1.1 Extensão do Plano Euclidiano e Reta no infinito

Antes de mergulharmos nas transformações projetivas que nos permitem projetar as imagens no infinito, precisamos entender alguns conceitos básicos.

Seja a seguinte definição do Plano Euclidiano Estendido[5]:

- Considere o Plano Euclidiano $\mathbb{R}^2$
- Dada uma reta r do $\mathbb{R}^2$, o conjunto consistindo de r e todas as retas paralelas a r é chamado de um **feixe** de retas paralelas. Para cada feixe de retas paralelas, adicione uma entidade abstrata P_∞, chamada **ponto no infinito** do feixe. As retas do feixe são ditas se interceptar em P_∞. Uma reta r unida ao seu ponto no infinito é chamada de **reta estendida**, e é denotada por r^*.
- Estipule que distintos feixe de retas têm distintos pontos no infinito.
- O conjunto de todos os pontos pontos no infinito é chamado de *reta no infinito*, e é denotado por r_∞.

Então, segue que o **Plano Euclidiano estendido** é uma tripla (P, R, I) com:

- P pontos que são os pontos de $\mathbb{R}^2$ e os pontos P_∞ do infinito.
- R retas que são as retas estendidas e a reta no infinito r_∞.
- Relação de incidência I herdada, ou seja:
  - um ponto P, que não está no infinito, pertence a r^* se, e somente se, P está em r.
  - P_∞ está em r^* se, e somente se, P_∞ é o ponto no infinito do feixe retas paralelas determinados por r^*.
  - Todos os pontos no infinito estão em r_∞.

2.1.2 Plano Projetivo

Seja a definição de Planos Projetivos.

Um Plano Projetivo π é um conjunto P de pontos e um conjunto L de subconjuntos de P, chamado de retas, satisfazendo as seguintes propriedades:

- $P1$. Existe uma única reta unindo dois pontos distintos.
- $P2$. Existe um único ponto de intersecção entre duas retas distintas.
- $P3$. Existem pelo menos três pontos não colineares
- $P4$. Existem pelo menos três pontos em cada linha.

Observe que o Plano Euclidiano Estendido é um Plano Projetivo, uma vez que todos os axiomas acima são satisfeitos. Entretanto, o Plano Euclidiano não é um plano projetivo desde que o axioma $P2$ não é satisfeita por nenhum par de retas paralelas.

2.2 Coordenadas Homogêneas

Um dos principais conceitos que se deve dominar quando estamos trabalhando com projetividade é o de Coordenadas Homogêneas e sua associação com a Geometria Projetiva.

"Coordenadas homogêneas, ou coordenadas projetivas, são um sistema de coordenadas usados em Geometria Projetiva, assim como as coordenadas cartesianas são usadas na Geometria Euclidiana. Sua principal vantagem consiste na possibilidade de que pontos, incluindo pontos no infinito, podem ser representados utilizando coordenadas finitas. (...) Em geral, transformações projetivas podem ser facilmente representadas por uma matriz."[6]

As afirmações a seguir são definidas e demonstradas em [4].

2.2.1 Pontos

Representação Euclidiana
No espaço Euclidiano padrão, utilizando notação vetorial, normalmente nós representaríamos pontos no $\mathbb{R}^2$ como segue:

$$P = \begin{bmatrix} x \\ y \end{bmatrix}$$

Representação Homogênea
Em coordenadas homogêneas, simplesmente adicionamos uma dimensão extra ao vetor, com valor unitário. Então, multiplicamos o novo vetor por um arbitrário fator escalar k_p, temos:

$$P = k_p \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} k_p x \\ k_p y \\ k_p \end{bmatrix} \tag{1}$$

Se quisermos voltar aos valores cartersianos, simplesmente dividimos as coordenadas pelo escalar escolhido.

2.2.2 Retas

Em Coordenadas Homogêneas, uma reta no espaço 2D, é simplesmente representada como segue:

$$r = \begin{bmatrix} a \\ b \\ c \end{bmatrix}.$$

O que corresponde à equação $ax + by + c = 0$.

2.2.3 Propriedades de Pontos e Retas

Agora, nós sabemos como representar pontos e retas em Coordenadas Homogêneas. Iremos explorar algumas propriedades oriundas das definições destes elementos.

Veficando se um ponto pertence a uma reta
Um ponto pertence a uma reta se, e somente se,:

$$\mathbf{r} \cdot \mathbf{P} = 0.$$

Ao expandirmos a equação, podemos encontrar:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}^T \cdot \begin{bmatrix} k_p x \\ k_p y \\ k_p \end{bmatrix} \therefore$$

$$ax + by + c = 0 \tag{2}$$

Que nos traz a equação da reta na forma cartesiana.

Intersecção de retas
Duas retas r e s se interceptam em

$$x = r \times s \tag{3}$$

Sejam $r = \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix}$ e $s = \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix}$ retas em coordenadas homogêneas. Segue, da Geometria Analítica sobre a interseção de dois planos no espaço tridimensional, que:

$$r \times s = \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix}$$

Expandido o produto vetorial, temos:

$$r \times s = (b_1 c_2 - b_2 c_1).\mathbf{i} + (a_2 c_1 - a_1 c_2).\mathbf{j} + (a_1 b_2 - a_2 b_1).\mathbf{k}$$

Escrevendo na forma de vetor coluna, temos:

$$P = \begin{bmatrix} b_1 c_2 - b_2 c_1 \\ a_2 c_1 - a_1 c_2 \\ a_1 b_2 - a_2 b_1 \end{bmatrix}$$

Observe que P é da forma

$$\begin{bmatrix} k_p x \\ k_p y \\ k_p \end{bmatrix}.$$

Para converter em Coordenadas Euclidianas, basta dividirmos pelo último termo.

$$x = \frac{k_p x}{k_p} = \frac{b_1 c_2 - b_2 c_1}{a_1 b_2 - a_2 b_1}$$

$$y = \frac{k_p y}{k_p} = \frac{a_2 c_1 - a_1 c_2}{a_1 b_2 - a_2 b_1}$$

Nesse caso, dizemos que os pontos estão normalizados.

Retas passando por dois pontos

Uma reta passa por dois pontos P e Q se, somente se:

$$r = P \times Q \tag{4}$$

isto é, a reta unindo dois pontos P e Q é dada pelo produto vetorial da representação homogênea desses pontos. Logo, vem:

Sejam os pontos $P_1 = \begin{bmatrix} k_{p1} x_1 \\ k_{p1} y_1 \\ k_{p1} \end{bmatrix}$ e $P_2 = \begin{bmatrix} k_{p2} x_2 \\ k_{p2} y_2 \\ k_{p2} \end{bmatrix}$, temos o seguinte produto vetorial:

$$P_1 \times P_2 = \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ k_{p1} x_1 & k_{p1} y_1 & k_{p1} \\ k_{p2} x_2 & k_{p2} y_2 & k_{p2} \end{bmatrix}$$

Expandido o produto vetorial, temos:

$$P_1 \times P_2 = k_{p1} k_{p2} (y_1 - y_2).\mathbf{i} + k_{p1} k_{p2} (x_2 - x_1).\mathbf{j} + k_{p1} k_{p2} (x_1 y_2 - x_2 y_1).\mathbf{k}$$

Escrevendo na forma do vetor coluna, temos:

$$r = k_{p1} k_{p2} \begin{bmatrix} y_1 - y_2 \\ x_2 - x_1 \\ x_1 y_2 - x_2 y_1 \end{bmatrix}$$

Observe que se chamarmos $a = k_{p1} k_{p2} (y_1 - y_2)$, $b = k_{p1} k_{p2} (x_2 - x_1)$ e $c = k_{p1} k_{p2} (x_1 y_2 - x_2 y_1)$, a equação se resume a:

$$r = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

descrevendo a equação da reta, como definimos acima.

2.2.4 Pontos ideais e Reta no infinito

Intersecção de Retas Paralelas
Considere duas retas $r : ax + by + c_1 = 0$ e $s : ax + by + c_2 = 0$. Essas retas são representadas, em coordenadas homogêneas, pelos vetores $r = (a, b, c_1)^T$ e $s = (a, b, c_2)$. Note que como as duas coordenadas são iguais, estas são paralelas. Vamos agora, aplicar a equação 3 para encontrar os pontos de intersecção:

$$r \times s = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a & b & c_1 \\ a & b & c_2 \end{vmatrix} \therefore \tag{5}$$

$$(c_2 - c_1)(\mathbf{i}b + \mathbf{j}(-a))$$

Logo, o ponto de intersecção entre as retas é $P = (c_2 - c_1)(b, -a, 0)^T$. Ignorando o fator escalar $(c_2 - c_1)$, ficamos com o ponto $(b, -a, 0)^T$.

Note que se quisermos a representação euclidiana do ponto, obeteríamos (b/0, -a/0), uma divisão não definida matematicamente, sugerindo apenas que possuem coordenadas infinitamente grandes. Pontos da forma homogênea $(x, y, 0)^T$ não correspondem a ponto finito no $\mathbb{R}^2$. Esse fato vai ao encontro da ideia tradicional de que retas paralelas se encontram no infinito.

Pontos ideais

Pontos de coordenadas homogêneas $x = (x_1, x_2, x_3)^T$ tais que $x_3 \neq 0$ correspondem a pontos finitos no $\mathbb{R}^2$. Os pontos com coordenada $x_3 = 0$ são conhecidos como pontos *ideais* ou *pontos no infinito*. Eles são da forma:

$$x_{ideal} = \begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix} \tag{6}$$

Reta no infinito
Observe que a união de todos os pontos ideais estão contidos numa única reta, chamada reta no infinito, denotada pelo vetor $r_\infty = (0, 0, 1)^T$, uma vez que o produto escalar entre os pontos ideais e a reta no infinito é igual a 0. Isto é,

$$x_{ideal} \cdot r_\infty = \begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix}^T \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0$$

2.3 Transformações Projetivas

Uma transformação projetiva é uma transformação linear de um vetor em coordenadas homogêneas representado por uma matriz não-singular 3×3, tal que:

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{7}$$

A transformação projetiva desenha cada imagem em um plano projetivamente equivalente, deixando todas as suas propriedades projetivas invariantes.

2.3.1 Reta no infinito

Sob uma transformação projetiva, pontos ideais podem ser mapeados para pontos finitos e, consequentemente, r_∞ pode ser mapeada por uma reta finita. Esse mapeamento é dado por:

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & c \end{bmatrix}, c \neq 0. \tag{8}$$

Seja o ponto que se quer projetar no infinito $P = (x, y, 1)^T$. Temos que:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ ax + by + c \end{bmatrix}$$

Dividindo as duas primeiras coordenadas, a fim de se obter os pontos em coordenadas euclidianas, temos:

$$x' = \frac{x}{ax + by + c}$$

$$y' = \frac{y}{ax + by + c}$$

2.4 Fractais

2.4.1 Definição

O termo *Fractal* foi definido inicialmente pelo matemático polonês - com nacionalidade francesa e americana - Benoit B. Mandelbrot em 1975 em seu livro *The Fractal Geometry of Nature*. O termo faz referência ao adjetivo em Latin *fractus*, o verbo correspondente em Latim é *frangere* e significa "quebrar"para criar fragmentos regulares [2]. Um fractal é uma figura geométrica em que cada parte é semelhante ao objeto como um todo, os padrões da figura inteira são repetidos em cada parte em uma escala menor.

2.4.2 Exemplos

Os Fractais são encontrados com facilidade na natureza como podemos observar em flocos de neve, plantas, árvores.

Figura 1: Folha de Samambaia

Figura 2: Floco de neve

Paralelamente, alguns fractais são definidos por equações matemáticas e podem exibir formas inusitadas e belas.

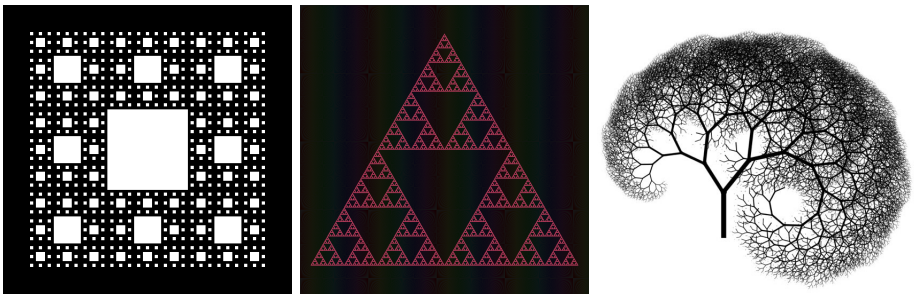

Figura 3: Tapete de Sierpinsky, Triângulo de Sierpinsky, Árvore Recursiva

2.4.3 O Conjunto de Mandelbrot

Um dos fractais que ganha mais destaque é o conjunto de Mandelbrot. Foi definido inicialmente por Robert W. Brooks e Peter Matelski em 1978 (Fig. 4), como parte de um estudo sobre grupos Kleinianos[8]. Benoit Mandelbrot, enquanto pesquisador da IBM, obteve visualizações do conjunto com maiores qualidades(Fig. 5). Por esse motivo, e por ser precursor do tema, tal conjunto leva seu nome.

É definido como segue:

Conjunto dos pontos de c no Plano Complexo para os quais a sequência definida por:

$$z_n = \begin{cases} 0, & \text{n=0} \\ z_{n-1}^2 + c, & \text{n>0} \end{cases} \tag{9}$$

não tende ao infinito.

Observe que c pode assumir qualquer valor dentro do conjunto dos Números Complexos. No entanto, para valores de c que se afastam da origem, a sequência tende ao infinito e c "escapa"do conjunto. Mais precisamente, o conjunto é limitado por uma circunferência de raio 2. Isto é, um ponto c pertence ao

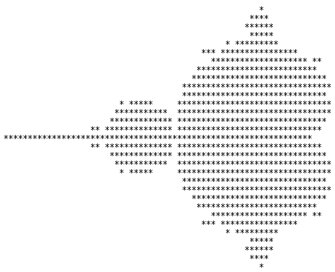

Figura 4: Conjunto de Mandelbrot obtido por Robert W. Brooks e Peter Matelski em 1978

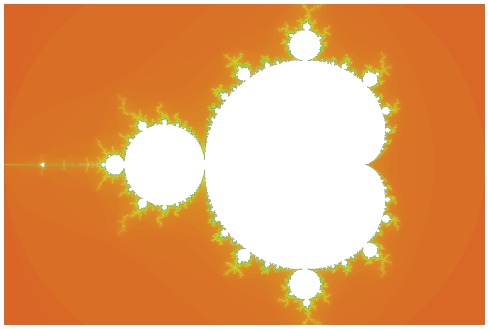

Figura 5: Conjunto de Mandelbrot

Conjunto de Mandelbrot se, e somente se, $|z_n| \geq 2$ para todo $n \geq 0$.[8]

Para desenhar o conjunto de mandelbrot em tela, descreveremos o pseudo-codigo abaixo. A ideia é simples. Basta percorrer todos os pixels da região que se quer desenhar, mapear os pontos para o intervalo pertencente ao conjunto de mandelbrot, iterar sobre os pontos e verificar se estes estão fora ou dentro do conjunto, e a depender do caso atribui-se uma cor ao ponto.

```
para cada pixel (px, py) da tela, faça:
    x0 = coordenada x mapeada dentro do eixo real que se quer observar
    y0 = coordenada y mapeada dentro do eixo imaginário que se quer observar
    c(x0, y0) // número complexo associado aos pontos x0 e y0
    iterações = 0
    max_iterações = 1000
    z = (0,0) // número complexo auxiliar
    enquanto(|c| < 2 && iterações < max_iterações), faça:
        z = z * z + c;
        iterações++;

    cor = pallete[iterações]
    plot(px, py, cor)
```

Observe que ao se escolher adequadamente os valores ao qual quer se mapear, dentro do Conjunto de Mandelbrot - isto é, definir as fronteiras do eixo real ou imaginário -, podemos deslocar o conjunto vertical ou horizontalmente, também é possível "dar zoom"no conjunto.

2.4.4 O Conjunto de Julia

O Conjunto de Julia é um conjunto que surge no estudo da Dinâmica Complexa[9]. O nome desse conjunto é em homenagem ao matemático francês Gaston Julia (1893-1978), quem descobriu esse conjunto e explorou suas propriedades.

Exploraremos a forma polinomial quadrática desse conjunto e suas renderizações. O Polinômio Quadrático desse conjunto é similar a definição recursiva do Conjunto de Mandelbrot:

$$f_c(z) = z^2 + c \tag{10}$$

No entanto, o parâmetro c é fixo. Diferentemente do Conjunto de Mandelbrot, onde se iterava variando-se os valores de c.

O pseudocódigo para o algoritmo da geração do Conjunto de Julia é:

```
escolha um número complexo c
para cada pixel (px, py) da tela, faça:
    x0 = coordenada x mapeada dentro do eixo real que se quer observar
    y0 = coordenada y mapeada dentro do eixo imaginário que se quer observar
    z = (x0, y0) // número complexo associado aos pontos x0 e y0
    iterações = 0
    max_iterações = 1000
    enquanto(|c| < 2 && iterações < max_iterações), faça:
        z = z * z + c;
        iterações++;

    cor = pallete[iterações]
    plot(px, py, cor)
```

3 Metodologia

3.1 OpenGL

A fim de se obter as imagens para este trabalho, foi utilizada a biblioteca OpenGL na versão 3.3.[7]

3.2 Projetividade

Como descrito na seção 2.3.1, para obter uma imagem projetada, deve-se multiplicar as coordenadas de seus pixels por uma matriz não-singular 3×3 que leve os pontos originais aos projetados seguidos por uma normalização.

A matriz deve ser da forma:

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & c \end{bmatrix}$$

Note que, como a, b e c representam os coeficientes de uma reta, podemos dividir toda a terceira linha por c, sem perda de generalidade, e a matriz é simplificada a:

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ v_1 & v_2 & 1 \end{bmatrix} \tag{11}$$

Essa matriz é definida como variável global no projeto e os valores de v1 e v2 podem ser alterados em tempo real via teclado.

```
float v1 = 0.00;
float v2 = 0.00;
float projectivity[3][3] = {{1, 0, 0}, {0, 1, 0}, {v1, v2, 1}};
```

O resultado dessa aplicação da projetividade é descrito na seção 3.4.

3.3 Conjunto de Mandelbrot no OpenGL

Como descrito no item 2.4.3, o conjunto é descrito por uma função recursiva, para os quais a sequencia definida pela equação 9 não tende ao infinito. Além disso, deve se fazer o mapeamento dos valores do conjunto de Mandelbrot para os valores em tela.

Inicialmente definiremos algumas variáveis globais:

- minRe, menor valor do eixo real a ser observado.
- maxRe, maior valor do eixo real a ser observado.
- minIm, menor valor do eixo imaginário a ser observado.
- maxRe, maior valor do eixo imaginário a ser observado.
- maxIterations, máxima quantidade de iterações, para evitar que haja loop infinito.

- width e height, largura e altura da tela, respectivamente.

Para fins de modularidade e organização do código, foi criada uma função "belongs"para saber, apenas, se o ponto está no conjunto ou não.

```
#include <complex>

int belongs(std::complex<double> c, int iterations) {
  std::complex<double> z(0, 0);
  int i = 0;
  while (abs(z) < 2 && i < iterations)  {
    z = z * z + c;
    i++;
  }
  return i;
}
```

A função principal fica:

```
for (int i = -width; i < width; i++) {
 for (int j = -height; j < height; j++) {
    xPosition = (float)(i) / width;
    yPosition = (float)(j) / height;
    xMandelbrot = minRe + (maxRe - minRe) * ((xPosition + 1) / 2);
    yMandelbrot = minIm + (maxIm - minIm) * ((yPosition + 1) / 2);
    result = belongs(std::complex<double>(xMandelbrot, yMandelbrot), maxIterations);
    if (result != maxIterations) {
      h = 40 + round(120 * result * 1.0 / maxIterations);
      rgb = hsl2rgb((float)h / 100, 0.6, 0.7);
      glColor3ub(rgb.r, rgb.g, rgb.b);
      glVertex3f(xPosition, yPosition, 0);
    }
  }
}
```

O resultado é exibido na figura 6.

Figura 6: Conjunto de Mandelbrot gerado utilizando OpenGL

Como foi dito também no item 2.4.3 ao mudar as variaveis de fronteiras, altera-se a imagem gerada. Ao escolher os valores $minRe = -1.5$, $maxRe = -1.3$, $minIm = -0.2$, $maxIm = 0.2$, obtemos a Fig. 7.

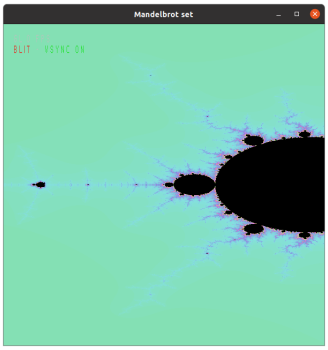

Figura 7: Conjunto de Mandelbrot no zoom

Também é possível alterar a paleta de cores através da definição da cor em HSL. Note que o valor de h (hue ou matiz) depende da quantidade de iterações necessárias para validar se o ponto está dentro ou fora do conjunto. Ao se escolher a base de h como 100 e os valores de s (saturação) como 0.5 e de l (luz) como 0.6, obtem-se a Fig. 8.

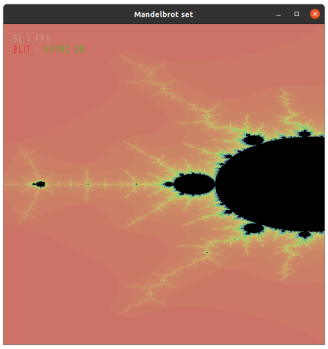

Figura 8: Conjunto de Mandelbrot variando-se as cores

3.4 Projetividade no Conjunto de Mandelbrot

Para obter os pixels projetados, devemos multiplicar a matriz da equação 11 pelo vetor coluna de coordenadas homogêneas dos pontos.

Para $v_1 = 0.23$ e $v_2 = 0.8$, temos:

```
float xScreen;
float yScreen;
float xMandelbrot;
```

```
float yMandelbrot;
float coord[3][3] = {{}, {}, {}};

float xProjected;
float yProjected;

    glBegin(GL_POINTS);

    for (int i = -width; i < width; i++)
    {
      for (int j = -height; j < height; j++)
      {
        xScreen = (float)(i) / width;
        yScreen = (float)(j) / height;
        xMandelbrot = minRe + (maxRe - minRe) * ((xScreen + 1) / 2);
        yMandelbrot = minIm + (maxIm - minIm) * ((yScreen + 1) / 2);
        coord[0][0] = xMandelbrot;
        coord[1][0] = yMandelbrot;
        coord[2][0] = 1;
        multiply(projectivity, coord, M4);
        xProjected = M4[0][0] / M4[2][0];
        yProjected = M4[1][0] / M4[2][0];
        result = belongs(std::complex<double>(xScreen, yScreen), maxIterations);
        if (result != maxIterations)
        {
          h = 100 + round(120 * result * 1.0 / maxIterations);
          rgb = hsl2rgb((float)h / 100, 0.5, 0.6);
          glColor3ub(rgb.r, rgb.g, rgb.b);
          glVertex3f(xProjected, yProjected, 0);
        }
      }
    }

    glEnd();
```

A figura 9 exibe o resultado dessa operação.

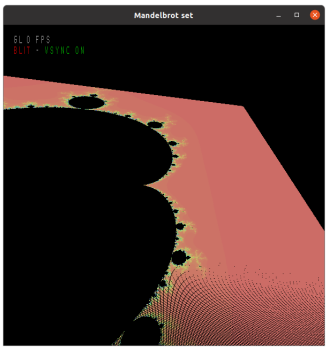

Figura 9: Conjunto de Mandelbrot renderizado com descontinuações na tela

Note, no entanto, que há espaços vazios e descontinuações na tela proveniente da renderização. Para corrigir essa falha, devemos aplicar a função *belongs* aos pontos projetados, desenhar os pontos $xScreen$ e $yScreen$ e a matriz a ser considerada é a matriz inversa, isto é, H^{-1}. Nesse caso, o código fica:

```
glBegin(GL_POINTS);

for (int i = -width; i < width; i++)
{
  for (int j = -height; j < height; j++)
  {
    xScreen = (float)(i) / width;
    yScreen = (float)(j) / height;
    xMandelbrot = minRe + (maxRe - minRe) * ((xScreen + 1) / 2);
    yMandelbrot = minIm + (maxIm - minIm) * ((yScreen + 1) / 2);
    coord[0][0] = xMandelbrot;
    coord[1][0] = yMandelbrot;
    coord[2][0] = 1;
    multiply(projectivityInverse, coord, M4);
    xProjected = M4[0][0] / M4[2][0];
    yProjected = M4[1][0] / M4[2][0];
    result = belongs(std::complex<double>(xProjected, yProjected), maxIterations);
    if (result != maxIterations)
    {
      h = 100 + round(120 * result * 1.0 / maxIterations);
      rgb = hsl2rgb((float)h / 100, 0.5, 0.6);
      glColor3ub(rgb.r, rgb.g, rgb.b);
      glVertex3f(xScreen, yScreen, 0);
    }
  }
}

glEnd();
```

O resultado da imagem corrigida é o que é exibido na figura 10.

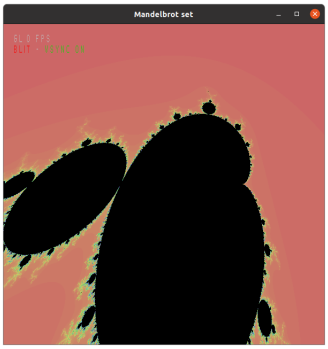

Figura 10: Conjunto de Mandelbrot sem descontinuações na tela

Adicionalmente, pode-se "elevar"a ordem do Conjunto de Mandelbrot, aumentando o grau do polinômio.

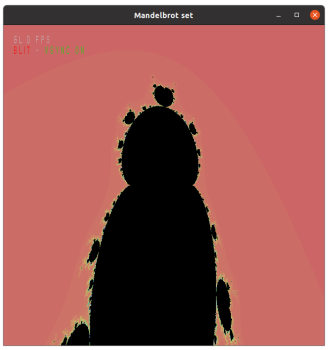

Figura 11: Conjunto de Mandelbrot para $f_c(z) = z^3 + c$

Figura 12: Conjunto de Mandelbrot para $f_c(z) = z^4 + c$

3.4.1 Conjunto de Julia

Como observado na seção 2.4.4, o Conjunto de Julia para polinômios quadráticos se aproxima da sequência recursiva do Conjunto de Mandelbrot, mudando-se apenas o fato de que fixa-se o número complexo c e variam-se os valores de z pelo plano complexo. Definiremos uma nova função chamada *belongsToJulia* como segue:

```
int belongsToJulia(std::complex<double> c, std::complex<double> z, int iterations)
{
  int i = 0;
  while (abs(z) < 2 && i < iterations)
  {
    z = z * z + c;
    i++;
  }
  return i;
}
```

E a função completa fica:

```
std::complex<double> c(-0.715,0.200);

    glBegin(GL_POINTS);

    for (int i = -width; i < width; i++)
    {
      for (int j = -height; j < height; j++)
      {
        xPosition = (float)(i) / width;
        yPosition = (float)(j) / height;
        xMandelbrot = minRe + (maxRe - minRe) * ((xPosition + 1) / 2);
        yMandelbrot = minIm + (maxIm - minIm) * ((yPosition + 1) / 2);
        coord[0][0] = xMandelbrot;
        coord[1][0] = yMandelbrot;
        coord[2][0] = 1;
        multiply(projectivityInverse, coord, M4);
        xProjected = M4[0][0] / M4[2][0];
        yProjected = M4[1][0] / M4[2][0];
        result = belongsToJulia(c, std::complex<double>(xProjected, yProjected), maxIterat
        if (result != maxIterations)
        {
          h = 10 + round(250 * result * 1.0 / maxIterations);
          rgb = hsl2rgb((float)h / 50, 0.8, 0.4);
          glColor3ub(rgb.r, rgb.g, rgb.b);
          glVertex3f(xPosition, yPosition, 0);
        }
      }
    }

    glEnd();
```

Note que os parâmetros de projetividade estão presentes pois são equivalentes ao do Conjunto de Mandelbrot.

Temos os seguintes resultados na figura 13 para diferentes números complexos escolhidos.

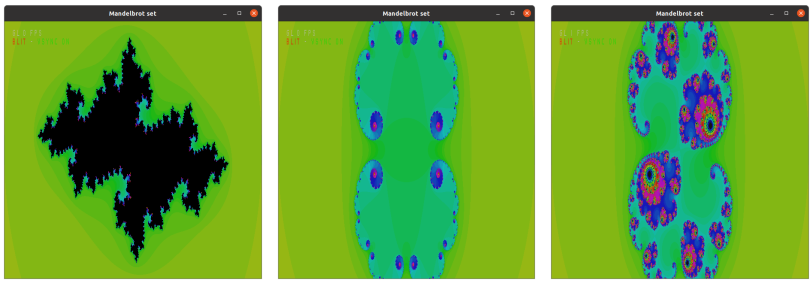

Figura 13: Conjunto de Julia para $c = -0.715 + 0.2i$, $c = 0.285 + 0i$ e $c = 0.285 + 0.01i$, respectivamente

Também é possível visualizar zoom e projetividade para a reta no infinito r_∞.

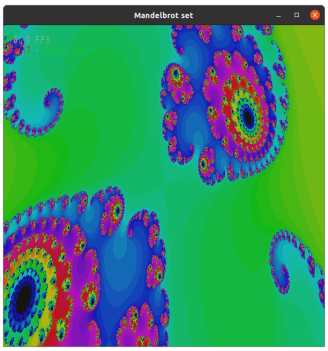

Figura 14: Conjunto de Julia com zoom e para valores de $v1 = 0.23$ e $v2 = 0.74$

3.5 Utilizando Curvas de Bezier para observar a mudança de cores

Curva de Bézier é uma curva polinomial paramétrica expressa como a interpolação linear entre pontos representativos, pontos de controle. Foi definida em 1962 por Pierre Bézier, então funcionário da fábrica de automóveis Renault.

3.5.1 Definição Matemática

Sejam $B_1, B_2, B_3, ..., B_n$ pontos de controle, um ponto pertencente a Curva de Bézier é dado por:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i B_i \tag{12}$$

O resultado da Curva de Bezier para 4 pontos de controle é mostrado na figura 15

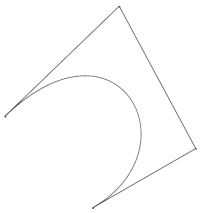

Figura 15: Curva de Bézier para 4 pontos de controle em 2 dimensões

Note que os pontos definidos na equação 12 não são restritos a 2 dimensões. Se estendemos os pontos para 3 dimensões, podemos obter, para cada

valor de t, um vetor $v = (B_x(t), B_y(t), B_z(t))$ e podemos definir estes valores como parâmetros de entrada para a renderização do Conjunto de Mandelbrot ou do Conjunto de Julia definido anteriormente, dentro de uma faixa de valores apropriada.

A implementação segue como:

Combinação de n elementos tomados r a r adicinada a função auxiliar *productRange*, que calcula o produto de todos os números inteiros entre números inteiros dados:

```
int combination(int n, int r)
{
  if (n == r || r == 0)
  {
    return 1;
  }
  else
  {
    r = (r < n - r) ? n - r : r;
    return productRange(r + 1, n) / productRange(1, n - r);
  }
}

int productRange(int a, int b)
{
  int prd = a, i = a;

  while (i++ < b)
  {
    prd *= i;
  }
  return prd;
}
```

Curva de Bézier:

```
point bezierPoint(float t, float xPoints[], float yPoints[], float zPoints[], int nPoints)
{
  float x = 0;
  float y = 0;
  float z = 0;

  struct point p;

  for (int i = 0; i <= nPoints; i++)
  {
    x += combination(nPoints, i) * pow(1 - t, nPoints - i) * pow(t, i) * xPoints[i];
    y += combination(nPoints, i) * pow(1 - t, nPoints - i) * pow(t, i) * yPoints[i];
    z += combination(nPoints, i) * pow(1 - t, nPoints - i) * pow(t, i) * zPoints[i];
  }
```

```
  p.x = x;
  p.y = y;
  p.z = z;

  return p;
}
```

Perceba que o retorno da função *bezierPoint* retorna um ponto com 3 coordenadas. Podemos atribuir essas coordenadas a cada uma das variáveis do nosso sistema de cor HSL.

O trecho de código modificado será:

```
h = 10 + round(360 * p.x * result * 1.0 / maxIterations);
rgb = hsl2rgb((float)h / 50, p.y, p.z);
```

onde p é um ponto da Curva de Bézier para pontos de controle arbitrários.

Para os pontos de controle $B_0 = (0.3, 0.7, 0.2)$, $B_1 = (0.1, 0.4, 0.63)$ e $B_2 = (0.8, 0.12, 0.67)$ e $t = 0.55$, temos como resultado a figura 16.

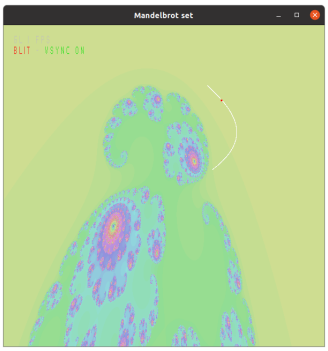

Figura 16: Utilizando a Curva de Bezier para variar as cores do Conjunto de Julia

3.6 Velocidade de Iterações do Conjunto de Mandelbrot

Um resultado natural que se poderia esperar para fractais projetivamente transformados (além da deformação geométrica em si) seria a apresentação de magnificações de diferentes níveis ("zooms") ao mesmo tempo em regiões distintas da figura. Magnificações de níveis muito díspares poderiam produzir fractais verdadeiramente novos, ao invés de fractais já conhecidos mas deformados projetivamente. Mas há uma questão de velocidade de avaliação em operação.

Observe a equação 9. Perceba que a sequência por ela definida evolui seus termos complexos quadraticamente. Se analisarmos o termo z_n^2 utilizando a fórmula de De Moivre[11], temos:

$$z_n^2 = [\rho^2(\cos\phi + i\sin\phi)] \tag{13}$$

Isto é, a cada vez que a sequência é iterada, seu módulo aumenta ao quadrado. E como descrito na seção 2.4.3, se o módulo de $z_n \geq 2$, o número

complexo z_n não pertence ao Conjunto de Mandelbrot. Como esta sequência converge para o infinito rapidamente, a deformação projetiva não contrabalanceia a velocidade da geração de fractais suficientemente para se observar novas estruturas de fronteira, visto que tal transformação, descrita em 2.3.1 e em 3.2, é uma operação linear seguida de uma normalização.

4 Conclusão

Esse trabalho se propôs a estudar a formação de fractais no Plano Projetivo, dando uma atenção especial ao Conjunto de Mandelbrot e suas derivações. Foram apresentados e discutidos diversos conceitos de Fractais e Geometria Projetiva, a fim de fornecer um embasamento teórico necessário ao entendimento do tema.

Os axiomas, postulados e definições apresentados na seção 2.1 dispõe de todo aparato matemático que alicerça a implementação. Caso o leitor sinta-se convidado a uma visão mais densa sobre o assunto, consultar [3], [4] e [5].

Para obter o resultado esperado, utilizou-se o OpenGL como motor gráfico para renderizar as imagens apresentadas nas seções 3.3, 3.4 e 3.5. O que resultou em imagens estáticas com boa resolução. No entanto, devido ao alto número de operações provenientes da implementação do algoritmo de renderização, ao se fazer ações interativas como zoom ou alterar a reta no infinito, o sistema apresentou alguns travamentos devido a limitação física do equipamento. Caso os parâmetros fossem apresentados previamente, a imagem era renderizada de forma fluida.

4.1 Trabalhos Futuros

Durante a elaboração deste trabalho, novas ideias surgiram de assuntos que transcendem, tangenciam ou complementam o tema. São eles:

- Utilizar GPU para renderização dos conjuntos.
- Analisar outros fractais.
- Investigar projetividade em curvas bidimensionais clássicas, tais como Lemniscata, Limaçons, Figuras de Lissajous.

5 Referências Bibliográficas

[1] Fractals, WIKIPEDIA https://en.wikipedia.org/wiki/Fractal Acesso em 01/10/2022.

[2] Mandelbrot, Benoit B., *The fractal geometry of nature*, H. B. Fenn and Company Ltd, 1st edition, 1977.

[3] H.S.M. Coxeter. *Projective Geometry*, Springer, 1st edition, 1987.

[4] Richard Hartley and Andrew Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2nd edition, 2004.

[5] Rey Casse *Projective Geometry: An Introduction*, Oxford University Press 1st edition, 2006.

[6] Homogeneous Coordinates: WIKIPEDIA. Disponível em: https://en.wikipedia.org/wiki/Homogeneous_coordinates . Acesso em 01/10/2022.

[7] OpenGL Documentation https://www.khronos.org/api/opengl .

[8] Mandelbrot Set, WIKIPEDIA https://en.wikipedia.org/wiki/Mandelbrot_set Acesso em 02/09/2022.

[9] Julia Set, WIKIPEDIA https://en.wikipedia.org/wiki/Julia_set Acesso em 08/09/2022.

[10] Bézier's Curve, WIKIPEDIA https://en.wikipedia.org/wiki/BÃľzier_curve Acesso em 26/09/2022.

[11] De Moivre Formula, Encyclopedia of Mathematics, Springer https://encyclopediaofmath.org/index.php?title=De_Moivre_formula Acesso em 04/10/2022.

[12] Richter-Gebert, Jürgen. Perspective on Projective Geometry: A Guided Tour Through Real and Complex Geometry, 2011.