Grupo:
José Nilton Oliveira Neto (RGM: 27287467)
Danilo de Oliveira Marques (RGM: 26166101)
Ricardo de Oliveira Sardinha Junior (RGM: 27290441)

Atividade Prática – Rasterizando Linhas

1. Introdução da Atividade

A tarefa envolve a criação de um algoritmo utilizando C++ e OpenGL. Inicialmente, o algoritmo deve incluir uma função responsável por rasterizar um ponto na memória de vídeo, que receberá as coordenadas (x, y) do pixel na tela e sua cor (RGBA) como parâmetros. Além disso, é necessário implementar uma função para rasterizar uma linha na tela, a qual receberá as coordenadas dos pontos iniciais e finais da linha. Para realizar essa tarefa, será utilizado o algoritmo de Bresenham. Esta função também permitirá a aplicação de um degradê de cores, recebendo dois valores RGBA como referência. Por último, o algoritmo deverá contar com mais uma função para criar um triângulo, desenhando três linhas que se conectam.

2. Desenvolvimento do Algoritmo

Para começar, foi criada uma função simples que recebe uma coordenada no formato "x, y" e uma cor no formato RGBA. Essa função tem como objetivo definir os valores RGBA no frame buffer da imagem, utilizando a função FBptr. Esta função foi desenvolvida para ser posteriormente utilizada na função responsável por rasterizar uma linha reta na imagem final. Depois de implementar a função PutPixel, que tem a responsabilidade de receber coordenadas e aplicar cores a elas, iniciamos o desenvolvimento da função DrawLine, que tem como objetivo rasterizar uma linha reta na imagem.

```
void PutPixel(int x, int y, int R, int G, int B, int A) {
    int index = 4 * (x + y * IMAGE_WIDTH);
    FBptr[index + 0] = R;
    FBptr[index + 1] = G;
    FBptr[index + 2] = B;
    FBptr[index + 3] = A;
}
```

```
void DrawLine(int x0, int y0, int R0, int G0, int B0, int A0,
              int x1, int y1, int R1, int G1, int B1, int A1) {
    int dx = abs(x1 - x0), dy = abs(y1 - y0);
    int sx = (x0 < x1) ? 1 : -1, sy = (y0 < y1) ? 1 : -1;
    int err = dx - dy, e2;
    float t;
```

Para começar é necessário inicializar variáveis que permitam calcular a diferença entre os pontos de início e fim nas direções x e y. Também é essencial determinar em qual direção essas variáveis devem ser incrementadas, com valores de 1 para positivo e -1 para negativo. Além disso, é crucial calcular o erro inicial. Além disso,

uma variável de interpolação chamada 't' (com um intervalo de valores entre 0 e 1) deve ser criada para facilitar a mistura das cores dos pixels.

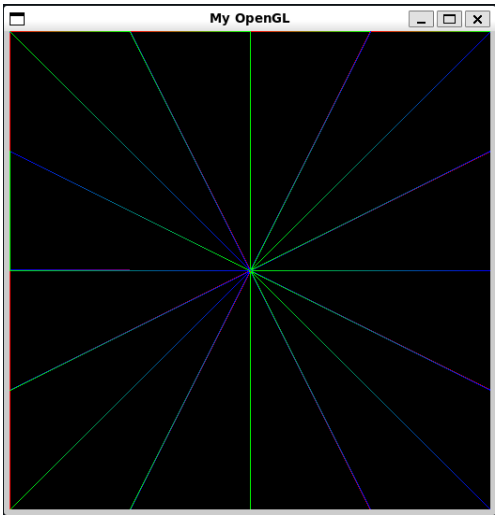
Após todas as declarações, um loop é iniciado para percorrer os pixels ao longo da linha, calculando o valor de 't' e invocando a função PutPixel para desenhar o pixel atual. Após desenhar o pixel atual, o algoritmo calcula o erro atual. Se o erro for maior que a diferença em 'y' (-dy), incrementa 'x0' e ajusta o erro. Enquanto, se o erro for menor que a diferença em 'x' (dx), incrementa 'y0' e ajusta o erro.

```
while (x0 != x1 || y0 != y1) {
    t = static_cast<float>(x0 - x1) / dx;
    PutPixel(
        x0, y0, R0 + static_cast<int>(t * (R1 - R0)),
        G0 + static_cast<int>(t * (G1 - G0)),
        B0 + static_cast<int>(t * (B1 - B0)),
        A0 + static_cast<int>(t * (A1 - A0))
    );
    e2 = 2 * err;
    if (e2 > -dy) {
        err -= dy;
        x0 += sx;
    }
    if (e2 < dx) {
        err += dx;
        y0 += sy;
    }
}
```

Dessa forma, a função DrawLine conclui sua execução após desenhar uma linha que se estende do ponto (x0, y0) até o ponto (x1, y1). E agora podemos criar a função DrawTriangle, que irá desenhar três linhas retas, formando assim um triângulo.

```
void DrawTriangle(int x0, int y0, int R0, int G0, int B0, int A0,
                  int x1, int y1, int R1, int G1, int B1, int A1,
                  int x2, int y2, int R2, int G2, int B2, int A2) {
    DrawLine(x0, y0, R0, G0, B0, A0, x1, y1, R1, G1, B1, A1);
    DrawLine(x1, y1, R1, G1, B1, A1, x2, y2, R2, G2, B2, A2);
    DrawLine(x2, y2, R2, G2, B2, A2, x0, y0, R0, G0, B0, A0);
}
```

Ao final do código, produzimos a imagem abaixo, na qual o algoritmo que desenvolvemos é empregado para criar diversos triângulos dentro de uma janela. Essa imagem é o resultado visual do processo em que os triângulos são renderizados usando as instruções e os parâmetros definidos no código.



3. Possíveis Melhorias

Uma maneira de otimizar o código seria aprimorar o processo de desenho do fractal, uma vez que atualmente estamos recorrendo a chamadas manuais para cada triângulo. No entanto, não conseguimos encontrar uma solução eficaz para implementar um loop que desenhasse os triângulos de forma mais concisa e legível

4. Referências

- http://orion.lcg.ufrj.br/cg/downloads/LCG_Rasterizacao.ppt
- http://disciplinas.ist.utl.pt/leic-cg/textos/livro/Rasterizacao.pdf
- http://wesnydyribeiro.blogspot.com/2017/02/rasterizacao-de-pontos-e-linhas.html
- https://medium.com/@filhojoseildo/implementa%C3%A7%C3%A3o-de-algoritmos-de-rasteriza%C3%A7%C3%A3o-ef413aaccf3d
- http://matheuspraxedescg.blogspot.com/2016/08/trabalho-1-rasterizacao-de-ponto-e-linha.html