



**CENTRO UNIVERSITÁRIO DE JOÃO PESSOA**  
**BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO**

COMPUTAÇÃO GRÁFICA  
ATIVIDADE PRÁTICA - RASTERIZANDO LINHAS  
DAVI CLEMENTINO CARNEIRO  
PROF: MAELSO B. PACHECO  
25/09/2023

## **RESUMO**

A atividade em questão representa uma implementação simples de uma biblioteca gráfica em C++, onde as classes Point e Color são usadas para representar pontos 2D e cores RGBA, respectivamente. A função PutPixel é responsável por definir a cor de pixels em uma matriz de imagem, enquanto a função DrawLine utiliza o algoritmo de Bresenham para desenhar linhas entre pontos, e a função DrawTriangle conecta três pontos para desenhar triângulos.

## **INTRODUÇÃO**

A rasterização é um conceito fundamental no mundo da computação gráfica, sendo o processo de converter objetos geométricos descritos em coordenadas vetoriais em uma representação de imagem baseada em pixels. Essa técnica é amplamente utilizada na renderização de gráficos 2D e 3D, permitindo a exibição de imagens detalhadas e realistas em telas de computador, dispositivos móveis e muito mais.

O OpenGL (Open Graphics Library) é uma API (Application Programming Interface) de gráficos amplamente utilizada na indústria de jogos, simulações, visualizações científicas e muitas outras aplicações gráficas. O OpenGL fornece um conjunto de funções e recursos que permitem criar ambientes gráficos interativos e renderizar objetos tridimensionais em tempo real.

O algoritmo de Bresenham, por sua vez, é uma técnica de rasterização desenvolvida por Jack E. Bresenham em 1962, amplamente aplicada na computação gráfica e no desenho de linhas e curvas em dispositivos de exibição. Sua principal vantagem é a eficiência, pois evita a necessidade de cálculos de ponto flutuante complexos, trabalhando com números inteiros para determinar quais pixels devem ser coloridos ao longo de uma linha.

Em resumo, a rasterização desempenha um papel crucial na renderização de gráficos computacionais, com o OpenGL e o algoritmo de Bresenham sendo componentes fundamentais na criação de ambientes gráficos avançados e eficientes. Essas tecnologias permitem a visualização de imagens complexas e interativas, proporcionando experiências visuais impressionantes em uma ampla variedade de aplicações modernas.

## METODOLOGIA

Para o desenvolvimento deste projeto, foi adotado o sistema operacional Linux Ubuntu 22.04.3 LTS em uma máquina virtual. O software UTM foi empregado para a criação da máquina virtual, permitindo a execução do código e testes de renderização sem obstáculos, garantindo um ambiente de desenvolvimento eficiente e funcional. Essa escolha foi motivada pelo fato de que a máquina original utilizada é um Macbook Air M1 2020, no qual o suporte ao OpenGL foi descontinuado e adaptado para outras bibliotecas gráficas. Portanto, a utilização de uma máquina virtual com Linux proporcionou um ambiente de desenvolvimento estável e compatível com as ferramentas necessárias.

No código, foram adotadas as seguintes estratégias:

- Uso de Classes e Structs: Classes como Point e Color são usadas para organizar dados e funcionalidades relacionadas.
- Funções Modulares: Funcionalidades são implementadas as funções independentes PutPixel, DrawLine, e DrawTriangle, para facilitar a reutilização e legibilidade.
- Passagem de Parâmetros: As funções aceitam parâmetros, como Point e Color, permitindo personalização das posições e cores das formas gráficas.
- Estrutura de Guardas de Inclusão: A definição de classes é protegida por guardas de inclusão em um arquivo de cabeçalho (mygl.h) para evitar conflitos.

Conforme solicitado, o algoritmo utiliza três funções:

- PutPixel(Point p, Color c): Esta função recebe um ponto p e uma cor c como parâmetros e define a cor de um pixel na matriz de imagem com base nas coordenadas do ponto. Isso permite a renderização de pixels coloridos em uma tela.
- DrawLine(Point p0, Point p1, Color color): A função DrawLine utiliza o algoritmo de Bresenham para desenhar uma linha entre dois pontos p0 e p1 com a cor especificada por color. Esse algoritmo calcula os pixels individuais que compõem a linha, proporcionando um desenho de linha preciso e eficiente.
- DrawTriangle(Point p0, Point p1, Point p2, Color color): A função DrawTriangle é responsável por desenhar um triângulo conectando três pontos p0, p1 e p2 com a cor especificada por color. Ela utiliza a função DrawLine para traçar as três arestas do triângulo, resultando na representação gráfica do triângulo.

```
56
57 void MyGldraw(void) {
58     // Desenha uma linha azul de (50, 50) a (200, 200)
59     Point lineStart(50, 50);
60     Point lineEnd(200, 200);
61     Color blue(0, 0, 255, 255);
62     DrawLine(lineStart, lineEnd, blue);
63
64     // Desenha um triângulo vermelho com vértices em (300, 50), (200, 1
65     Point vertex1(300, 50);
66     Point vertex2(200, 175);
67     Point vertex3(300, 150);
68     Color red(255, 0, 0, 255);
69     DrawTriangle(vertex1, vertex2, vertex3, red);
70 }
```

*Figura 1 - Trecho de código onde os vértices da linha e do triângulo são desenhados e coloridos, no software Sublime Text 4.*

## DISCUSSÃO

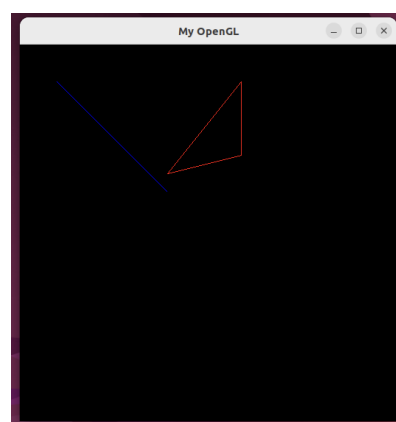
Na computação gráfica, esse código serve como um ponto de partida para desenvolvedores que desejam criar suas próprias aplicações gráficas personalizadas. Ele demonstra os princípios fundamentais da rasterização, que é o processo de conversão de objetos geométricos em pixels para exibição em uma tela.

Alternativamente, em outros frameworks e bibliotecas gráficas mais robustos, como DirectX ou Vulkan, você encontrará funcionalidades avançadas e otimizações de hardware que podem lidar com gráficos 2D e 3D complexos em tempo real. Essas alternativas oferecem suporte a recursos avançados, como shaders, texturas e iluminação, que são essenciais para a criação de jogos e aplicativos gráficos mais sofisticados.

No entanto, o código apresentado aqui tem limitações significativas, incluindo a falta de suporte para recursos 3D, falta de otimizações de hardware e recursos avançados, como shaders e texturas. Também é menos eficiente em comparação com frameworks gráficos especializados. Portanto, enquanto é um ótimo recurso para aprendizado e experimentação, ele não é adequado para projetos gráficos de grande escala e de alto desempenho.

Dentre as principais dificuldades encontradas, explico em primeiro lugar a configuração do Framework em minha máquina, que já foi descontinuado e necessitaria de maiores adaptações no código para funcionar.

Infelizmente, o uso de tal código tem suas limitações e possibilidades de melhoramento. Dentre eles, listo o suporte a renderização 3D, o uso de algoritmos mais eficientes (como em cenários com muitos pixels), shaders e uma interface gráfica para o usuário. Apesar de suas limitações, este código simples fornece um alicerce fundamental para compreender o processo de rasterização em computação gráfica. Ao implementar conceitos básicos, como a representação de pontos, cores e a aplicação do algoritmo de Bresenham para desenho de linhas, ele oferece uma base sólida para aqueles que desejam aprofundar seus conhecimentos na renderização de gráficos. Esse entendimento é essencial para construir aplicações gráficas mais avançadas e explorar as capacidades de frameworks e bibliotecas gráficas mais robustos.

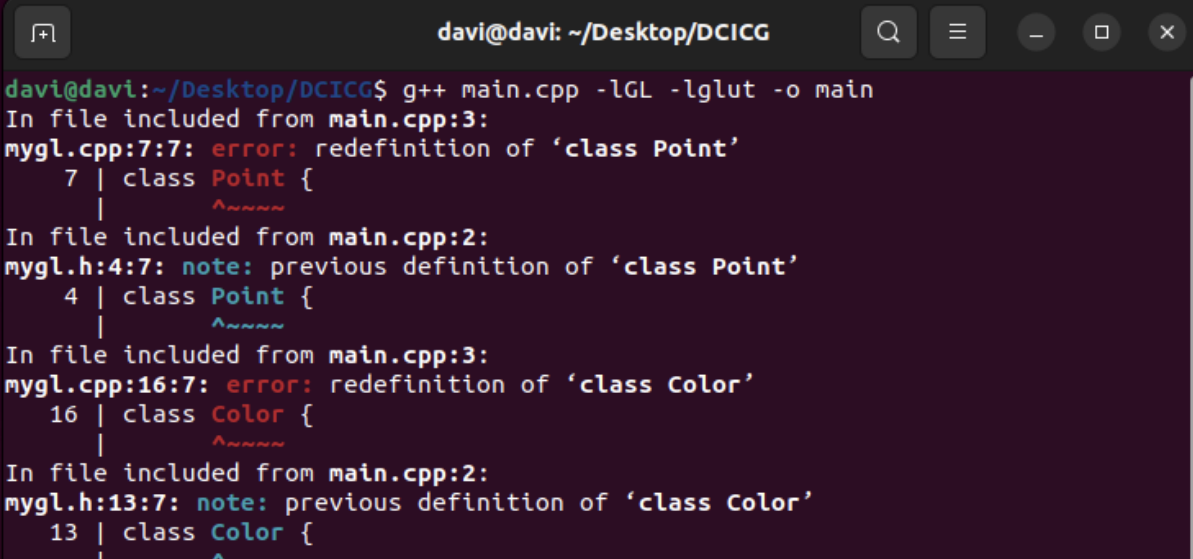


*Figura 2 - Rasterização realizada.*

## REFERÊNCIAS

1. SHREINER, Dave et al. "OpenGL Programming Guide." 7ª edição. Addison-Wesley Professional, 2016.
2. MANSSOUR, Isabel H. "Introdução ao OpenGL." PUC-RS. Disponível em: <https://www.inf.pucrs.br/~manssour/OpenGL/Tutorial.html>. Acesso em: 25 de setembro de 2023.
3. CACERES, Sheila. "Rasterização de Linhas." UNIP. Disponível em: <http://sheilacaceres.com/courses/cg/CG3-Reta-DDA-Bresenham.pdf>. Acesso em: 25 de setembro de 2023.
4. "Conceitos Iniciais do Algoritmo de Bresenham." MetrÓpole Digital. Disponível em: <https://materialpublic.imd.ufrn.br/curso/disciplina/5/69/7/4>. Acesso em: 25 de setembro de 2023.

## APÊNDICE

A terminal window with a dark background and light-colored text. The window title is 'davi@davi: ~/Desktop/DCICG'. The command 'g++ main.cpp -lGL -lglut -o main' has been executed. The output shows two errors: 'redefinition of 'class Point'' and 'redefinition of 'class Color'', each preceded by a 'note: previous definition of' message. The errors point to lines in 'mygl.cpp' and 'mygl.h'.

```
davi@davi:~/Desktop/DCICG$ g++ main.cpp -lGL -lglut -o main
In file included from main.cpp:3:
mygl.cpp:7:7: error: redefinition of 'class Point'
    7 | class Point {
      |         ^~~~~
In file included from main.cpp:2:
mygl.h:4:7: note: previous definition of 'class Point'
    4 | class Point {
      |         ^~~~~
In file included from main.cpp:3:
mygl.cpp:16:7: error: redefinition of 'class Color'
   16 | class Color {
      |         ^~~~~
In file included from main.cpp:2:
mygl.h:13:7: note: previous definition of 'class Color'
   13 | class Color {
      |         ^~~~~
```

Figura 3 - Um dos erros encontrados: declaração de classe *Point* e *Color* repetida nos arquivos *mygl.h* e *mygl.cpp*.