Universidade Federal da Paraíba

Centro de Informática

Engenharia da Computação

Introdução à Computação Gráfica

Atividade Prática 2

Rasterizando Linhas

Aléxandros Augustus

11501517

João Pessoa

2020

1. Objetivo

Neste trabalho implementamos 3 funções:

- PutPixel: Função que dados uma coordenada e uma cor, pinta um pixel na tela com a posição e cor especificadas;
- DrawLine: Função que dadas duas coordenadas e duas cores, desenha uma linha que vai da primeira à segunda coordenada, interpolando as cores de cada vértice;
- DrawTriangle: Função que dados os vértices de um triângulo e a cor de cada vértice, desenha o triângulo na tela, interpolando a cor entre os vértices.

2. PutPixel

2.1. Implementação

Para implementar esta função, utilizamos os seguintes conceitos:

- Pixel: um pixel corresponde à uma posição na matriz da tela;
- *Color buffer*: É o espaço de memória onde estão armazenadas as informações de cor de cada pixel. Cada pixel possui 4 *bytes* para si, cada um para uma banda de cor (RGBA)

Pensando no *color buffer* como um *array*, e sabemos que as bandas estão organizadas neste na ordem vermelho, verde, azul e alfa, utilizamos a API fornecida pelo professor para percorrer esta estrutura utilizando a seguinte fórmula para, dadas as coordenadas x e y de um pixel, definir o *offset* que deve ser aplicado ao ponteiro que percorre o *color buffer* utilizando a fórmula a seguir, que foi apresentada em sala de aula:

$$offset(x, y) = x \times 4 + y \times 4 \times W$$

Onde (x, y) é a coordenada do pixel, e W é a largura da tela.

Após calcular este valor, fazemos o ponteiro apontar para o índice *offset*, que corresponde à banda vermelha do pixel desejado, e incrementamos a posição para as bandas seguintes na ordem verde, azul e alfa.

2.2. Testes

Para verificar o bom funcionamento desta função realizamos dois testes, primeiramente desenhamos um “arco-íris” horizontal na tela:

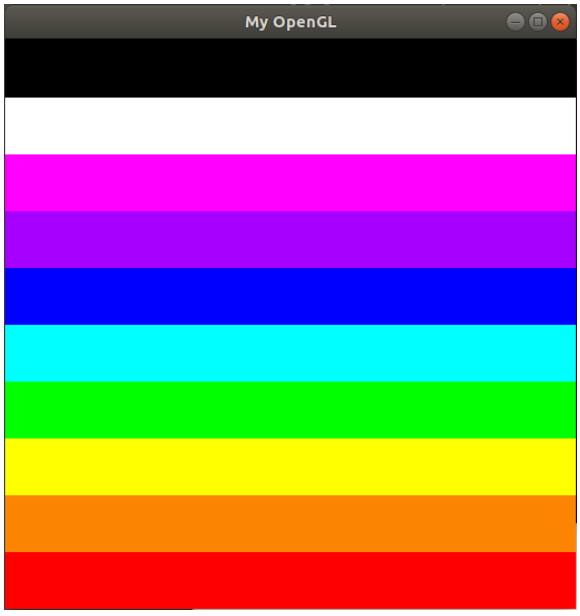

Figura 1 - Teste da função PutPixel pintando a tela em intervalos horizontais

Em seguida, pintamos um quadrado azul de tamanho 100x100 pixels, no centro da tela:

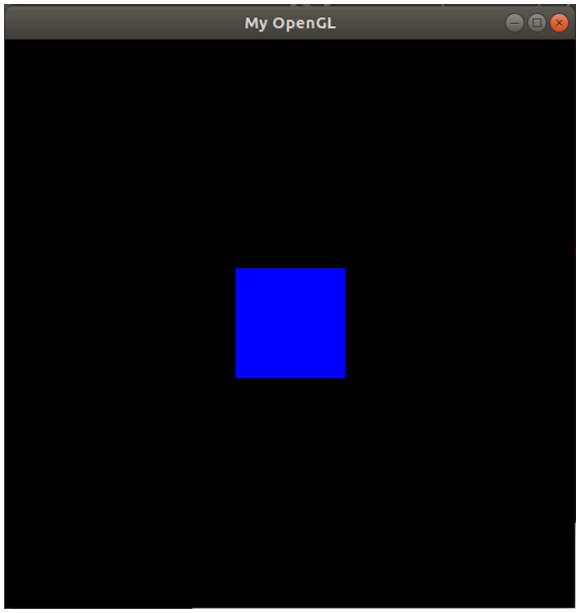

Figura 2 - Teste da função PutPixel pintando um quadrado na tela

3. DrawLine

3.1. Implementação

Para implementar esta função utilizamos o Algoritmo do Ponto Médio como solicitado pelo professor, nas aulas vimos uma versão simplificada deste algoritmo para retas pertencentes ao primeiro octante, assim algumas modificações se fizeram necessárias para generalizar este algoritmo.

Primeiro acrescentamos 2 variáveis auxiliares, chamadas “aux” e “auy”, “aux” ajusta os valores de x quando a coordenada x do pixel final é menor que a coordenada x do pixel inicial, já a variável “auy” faz o mesmo ajuste, só que para a coordenada y quando a coordenada y do pixel final é menor que a coordenada y do pixel inicial.

Outra alteração é uma condição “if” que verifica se o módulo do coeficiente angular é maior ou menor que 1, pois caso seja menor que 1, podemos utilizar o algoritmo normalmente, caso seja maior que 1 devemos inverter as coordenadas x e y, ou seja, em vez de calcular um valor de y para cada x, calcularemos um valor de x para cada y.

A última alteração que não estava presente na aula é a parte da interpolação linear, para isso utilizamos o algoritmo de Newton, tal algoritmo dita que, para o caso de termos os pontos (x_0, R_0) e (x_n, R_n), a reta aproximada que passa por esses pontos pode ser calculada com a seguinte fórmula:

$$R(x) = \frac{\Delta R}{\Delta x}x - \frac{\Delta R}{\Delta x}x_0 + R_0$$

Sabendo que $R(x_0) = R_0$ e $R(x_n) = R_n$, podemos calcular $R(x_i)$, dado por:

$$R(x_i) = \frac{\Delta R}{\Delta x}x_i - \frac{\Delta R}{\Delta x}x_0 + R_0$$

Como estamos em intervalos discretos, sabemos que cada x aumenta o anterior em uma unidade, temos que $x_{i+1} = x_i + 1$, então:

$$R(x_{i+1}) = \frac{\Delta R}{\Delta x}(x_i + 1) - \frac{\Delta R}{\Delta x}x_0 + R_0 \therefore R(x_{i+1}) = \frac{\Delta R}{\Delta x} + \frac{\Delta R}{\Delta x}x_i - \frac{\Delta R}{\Delta x}x_0 + R_0 \therefore$$

$$\therefore R(x_{i+1}) = \frac{\Delta R}{\Delta x} + R(x_i)$$

Assim, podemos generalizar sabendo que a cada novo termo, seu valor será acrescido de um fator $\frac{\Delta R}{\Delta x}$. Abstraindo R como o valor da banda vermelha e x como a coordenada x dos pixels, podemos calcular a cor de todos os pixels ao longo da reta, assim, calculamos o fator de interpolação a ser somado e, a cada novo termo, somamos este valor ao anterior para cada uma das bandas.

3.2. Testes

Para verificar que a função estava funcionando corretamente, desenhamos uma reta completamente branca indo do canto inferior esquerdo ao canto superior direito da tela:

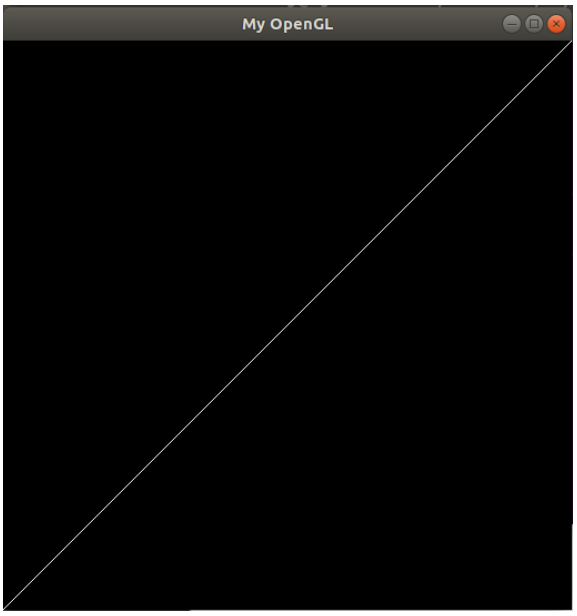

Figura 3 - Teste da função DrawLine desenhando uma reta unicromática percorrendo a tela

Por fim, para termos uma prova definitiva, desenhamos um conjunto de retas partindo do centro, uma para testar cada octante da tela:

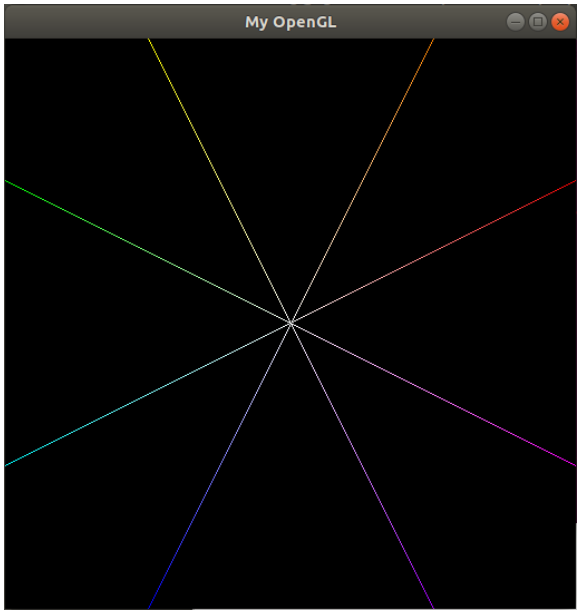

Figura 4 - Teste da função DrawLine desenhando um conjunto de linhas na tela

Um detalhe interessante da interpolação é que esta não funciona corretamente caso os valores das bandas sejam definidos como inteiros, apenas apresentando o comportamento esperado quando declarados como tipo ponto flutuante (float).

4. DrawTriangle

4.1. Implementação

Por fim esta função apenas chama a função anterior 3 vezes, sendo uma para cada aresta.

4.2. Testes

Para testar a função desenhamos um triângulo qualquer na tela, ao escrever o código de teste, tivemos dificuldade em gerar triângulos que não tenham a coordenada (0, 0) como um de seus vértices.

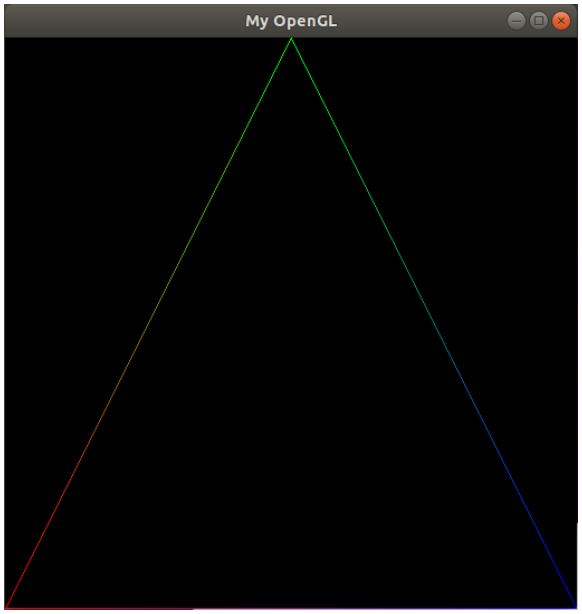

Figura 5 - Teste da função DrawTriangle desenhando um triângulo na tela

5. Conclusão

Concluímos que as funções foram todas implementadas com êxito, apesar de algumas particularidades para as quais não temos explicação, o conteúdo coberto pelo escopo das aulas foi compreendido com sucesso.

6. Referências

Vídeo-aulas disponibilizadas pelo professor Christian