Relatório - Rasterizando Linhas

Alunos: Caroliny Santos Arruda, João Claudino Francisco Neto e Vinicius Candeia Pereira Vieira

RESUMO

Foi desenvolvido um algoritmo de desenho de triângulos por meio da implementação para rasterização de pontos e linhas, baseado no Algoritmo de Bresenham, que utiliza um laço de repetição para mover-se entre os pixels nas coordenadas dos eixos X e Y e formar as arestas que compõem os triângulos.

DESENVOLVIMENTO

Inicialmente, no arquivo mygl.h, foi declarada a classe do pixel, um tipo de dado para criar objetos, com seus atributos de cor, com cada elemento do colorbuffer (RGBA) e posição (X,Y), e os métodos GET e SET para acessá-los. Além disso, foram criadas as funções gráficas: a função PutPixel: coloca um ponto no mapa com as cores e as posições do objeto Pixel passado, a função DrawLine: desenha uma reta entre dois pixels passados, um inicial e outro final, e, finalmente, a função DrawTriangle: desenha um triângulo recebendo como parâmetro os seus três vértices.

```
9   class Pixel
10  {
11
12  private:
13      int posX, posY;
14      int corR, corG, corB, corA;
15
16  public:
17      void definirCor(int _corR, int _corG, int _corB, int _corA);
18
19      void definirPos(int _posX, int _posY);
20
21      Pixel interLinear(Pixel pixelI, Pixel pixelF, Pixel pixel);
22
23      void setPosX(int _posX);
24      int getPosX();
25
26      void setPosY(int _posY);
27      int getPosY();
28
29      void setCorR(int _corR);
30      int getCorR();
31
32      void setCorG(int _corG);
33      int getCorG();
34
35      void setCorB(int _corB);
36      int getCorB();
37
38      void setCorA(int _corA);
39      int getCorA();
40  };
41
42  void PutPixel(Pixel pixel);
43  void DrawLine(Pixel pixelF, Pixel pixelI);
44  void DrawTriangle(Pixel pixel1, Pixel pixel2, Pixel pixel3);
```

Figura 1: Declaração do pixel e suas funções.

Já no arquivo mygl.cpp, ocorreu a implementação das funções anteriormente declaradas e a criação da função raiz_quadrada que será usada para pintar o pixel de acordo com o algoritmo de Interpolação Linear.

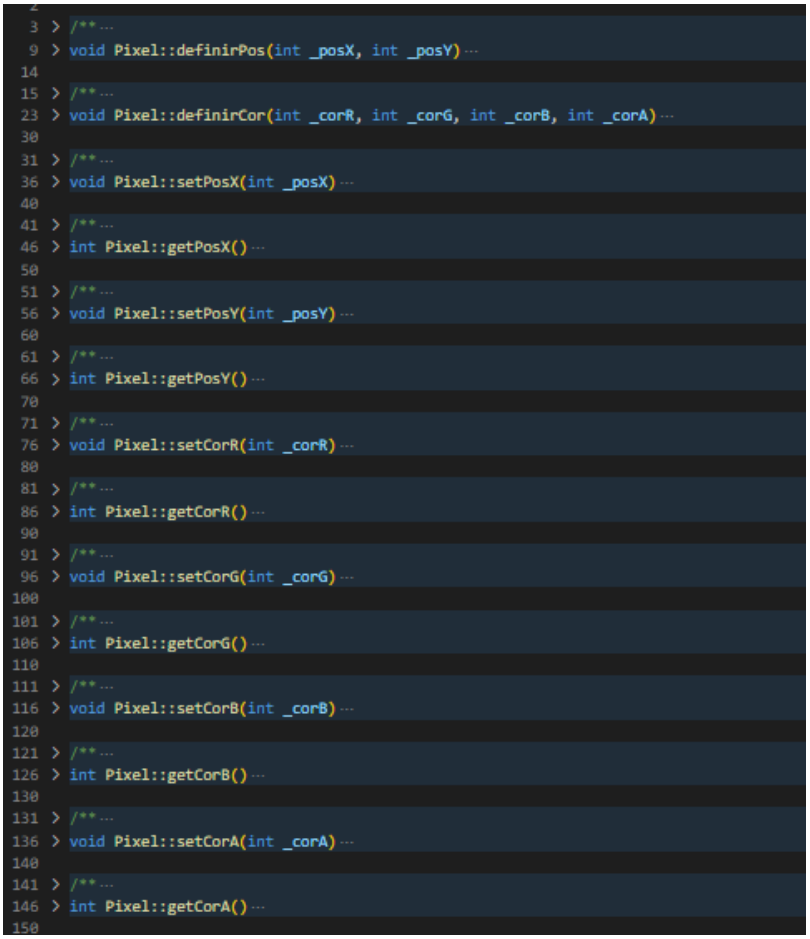


Figura 2: Funções de definição das cores do colorbuffer (RGBA) e da posição do pixel.

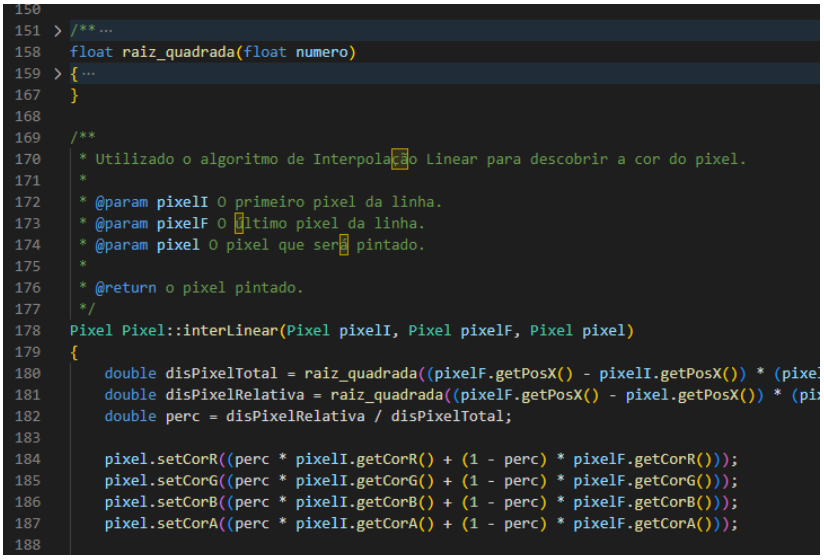


Figura 3: Função raiz_quadrada e uso da interpolação linear para definir as cores da reta

Na função PutPixel é feito o cálculo, por meio das posições passadas, do offset do colorbuffer.

```
156  void PutPixel(Pixel pixel)
157  {
158      int offset = 4 * pixel.getPosX() + 4 * pixel.getPosY() * IMAGE_WIDTH;
159      FBptr[offset + 0] = pixel.getCorR();
160      FBptr[offset + 1] = pixel.getCorG();
161      FBptr[offset + 2] = pixel.getCorB();
162      FBptr[offset + 3] = pixel.getCorA();
163  }
```

Figura 4: Função PutPixel

Na função DrawLine() se inicia criando um objeto Pixel e define através de métodos sua cor e sua posição. Em seguida, ele usa o Algoritmo de Bresenham para calcular, em X e em Y, a variação entre os objetos passados como parâmetro. Após isso, utiliza-se a variável “sel” para armazenar um valor positivo ou negativo a depender da posição do pixel. Com o valor do “sel”, calcula-se o valor da inclinação e direção da reta e insere o pixel criado atrás do PutPixel(). A partir desse ponto, utiliza-se de condicionais e laços/loops para completar a reta (através de inserções de pixel) a partir do primeiro pixel inserido.

```
/**
 * Desenha uma reta entre dois pixels passados.
 *
 * Há 12 cenários que são necessários criar:
 *  1) Reta vertical para cima.
 *  2) Reta vertical para baixo.
 *  3) Reta horizontal para direita.
 *  4) Reta horizontal para esquerda.
 *  5) Reta diagonal com o ângulo entre 0° e 45°.
 *  6) Reta diagonal com o ângulo entre 45° e 90°.
 *  7) Reta diagonal com o ângulo entre 90° e 135°.
 *  8) Reta diagonal com o ângulo entre 135° e 180°.
 *  9) Reta diagonal com o ângulo entre 180° e 225°.
 *  10) Reta diagonal com o ângulo entre 225° e 270°.
 *  11) Reta diagonal com o ângulo entre 270° e 315°.
 *  12) Reta diagonal com o ângulo entre 315° e 360° (0°).
 *  13) Reta diagonal com o ângulo de 45°.
 *  14) Reta diagonal com o ângulo de 135°.
 *  15) Reta diagonal com o ângulo de 225°.
 *  16) Reta diagonal com o ângulo de 315°.
 *
 * @param pixelF O pixel final.
 * @param pixelI O pixel inicial.
 */
> void DrawLine(Pixel pixelF, Pixel pixelI)…
```

Figura 5: Função DrawLine

Na função DrawTriangle são desenhadas as três arestas que vão compor o triângulo.

```
482  void DrawTriangle(Pixel pixel1, Pixel pixel2, Pixel pixel3)
483  {
484      DrawLine(pixel2, pixel1);
485      DrawLine(pixel3, pixel2);
486      DrawLine(pixel1, pixel3);
487  }
```

Figura 6: Função DrawTriangle

Finalmente, na função MyGlDraw são indicadas as posições e cores dos pixels e a formação do triângulo.

```
489  void MyGlDraw(void)
490  {
491
492      Pixel pixel1, pixel2, pixel3, pixel4, pixel5, pixel6;
493
494      pixel1.definirCor(255, 255, 255, 255);
495      pixel1.definirPos(256, 256);
496
497      pixel2.definirCor(255, 255, 255, 255);
498      pixel2.definirPos(322, 222);
499
500      pixel3.definirCor(255, 255, 255, 255);
501      pixel3.definirPos(456, 226);
502
503      pixel4.definirCor(255, 255, 255, 255);
504      pixel4.definirPos(400, 290);
505
506      pixel5.definirCor(255, 255, 255, 255);
507      pixel5.definirPos(226, 476);
508
509      pixel6.definirCor(255, 255, 255, 255);
510      pixel6.definirPos(226, 76);
511
512      DrawTriangle(pixel1, pixel5, pixel2);
513      DrawTriangle(pixel3, pixel6, pixel4);
514
515  }
```

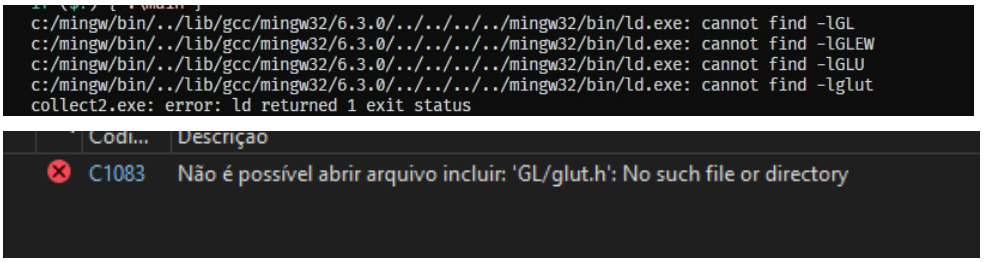
Figura 7: Função MyGlDraw

RESULTADOS

O grupo apresentou 3 grandes dificuldades no desenvolvimento do projeto:

1. Compilar no Linux: nenhum participante do grupo conseguiu fazer a compilação do projeto utilizando o sistema operacional Windows, pois, mesmo instalando tudo o que era necessário, as flags que são passadas na hora da compilação são de Linux, e não foi encontrada as equivalentes para

Windows. Dessa maneira, foi necessário realizar uma instalação do sistema operacional Linux, para o desenvolvimento desse projeto.



Figuras 8 e 9: Erros no Windows

2. Lógica necessária para o desenho das retas: mesmo entendendo o Algoritmo de Bresenham, tivemos dificuldade em generalizar para todos os octantes. Muitas vezes, a reta aparecia apenas horizontal ou, então, vertical, e não crescia na outra direção. Então, levou um tempo até encontrar a falha no raciocínio, em que, na maioria dos casos, era apenas um sinal utilizado no cálculo que estava errado. Por exemplo, a reta entre os pontos (456, 226) e (226, 76) deveria ser na diagonal, mas acabava saindo na horizontal (figura 10). Após a revisão, a reta saia como era desejado (figura 11).

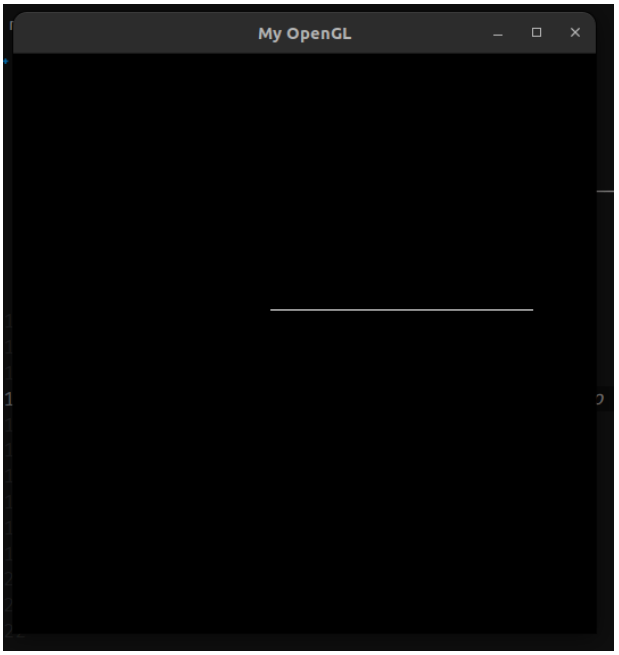


Figura 10: Reta errada.

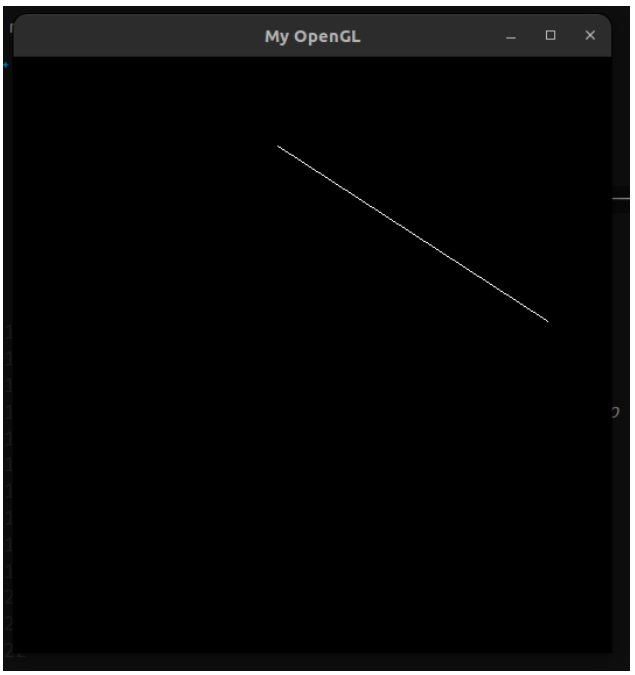


Figura 11: Reta correta.

3. Na descrição do projeto, não há determinação sobre a cor da reta e por isso, inicialmente, deixamos como branco (255, 255, 255, 255), a cor padrão para a reta. Depois, conversando sobre, percebemos que poderíamos fazer um degradê na reta, ou seja, uma transição gradual de cores entre os vértices da reta, o que leva o nome de Interpolação Linear.

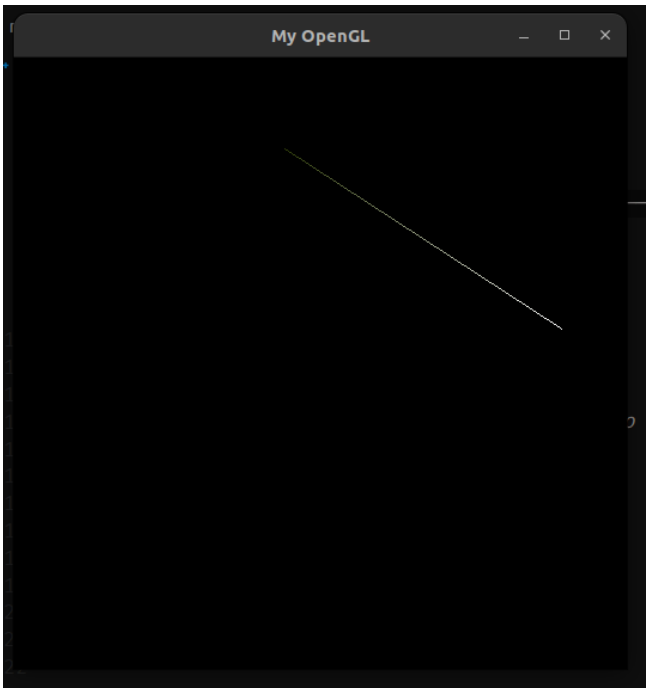


Figura 12: Reta com Interpolação Linear.

Dessa forma, o resultado final ficou:

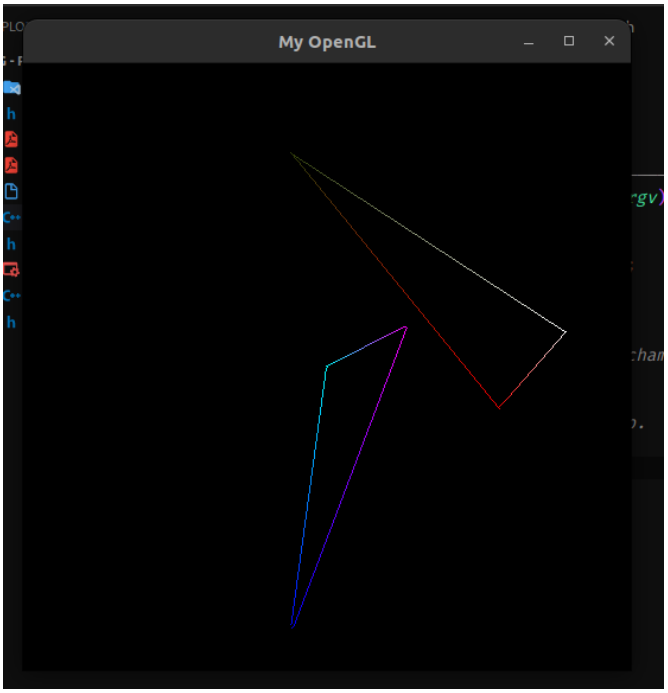


Figura 13: Resultado Final

Uma (possível) melhoria no código seria a refatoração dele, pois, a função DrawLine ficou muito extensa, já que em cada caso na criação da reta (totalizando 16), foi utilizado um while como o da figura 14.

```
while (pixel.getPosX() ≤ pixelF.getPosX())
{

    if (decPos ≤ 0)
    {
        decPos += incH;
        pixel.setPosY(pixel.getPosY() + 1);
    }
    else
    {
        decPos += incD;

        pixel.setPosX(pixel.getPosX() + 1);
        pixel.setPosY(pixel.getPosY() + 1);
    }

    pixel = pixel.interLinear(pixelI, pixelF, pixel);
    PutPixel(pixel);
}
```

Figura 14: Exemplo do while.

REFERÊNCIAS BIBLIOGRÁFICAS

https://materialpublic.imd.ufrn.br/curso/disciplina/5/69/7/4

http://wiki.icmc.usp.br/images/c/c2/Introdu%C3%A7%C3%A3oPr%C3%A1ticaOpenGL.pdf

https://pt.wikipedia.org/wiki/Interpola%C3%A7%C3%A3o_linear