UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB
DEPARTAMENTO DE INFORMÁTICA
ENGENHARIA DE COMPUTAÇÃO

Integrantes:
Andrea Brito do Nascimento - 11311923
Chaenne Carolina Pessoa-11406556
Jonas Antas da Silva-

Disciplina*: Introdução a Computação Gráfica*
Prof. *Christian Pagot*

Primeira Atividade

João Pessoa
Março/2018

Especificação técnica do projeto:

Nesta atividade iremos desenhar triângulos através da implementação de um algoritmo de rasterização de pontos e linhas , onde os mesmo devem ser as arestas dos triângulos.
Neste trabalho desenvolvemos 3 funções. São elas:
• **PutPixel:** Função que rasteriza um ponto na memória de vídeo, recebendo como parâmetros a posição do pixel na tela (x,y) e sua cor (RGBA).
• **DrawLine:** Função que rasteriza uma linha na tela, recebendo como parâmetros os seus vértices(inicial e final, representados respectivamente pelas tuplas (x0,y0) e (x1,y1)), e as cores (no formato RGBA) de cada vértice. As cores dos pixels ao longo da linha rasterizada devem ser obtidas através da interpolação linear das cores dos vértices. O algoritmo de rasterização a ser implementado deve ser o algoritmo de Bresenham!
• **DrawTriangle:** Função que desenha as arestas de um triângulo na tela, recebendo como parâmetros as posições dos três vértices (xa,ya), (xb,yb) e (xc,yc) bem como as cores (RGBA)de cada um dos vértices. As cores dos pixels das arestas do triângulo devem ser obtidas através da interpolação linear das cores de seus vértices.

Como os sistemas operacionais não permitem o acesso direto à memória usamos um framework a fim de simular operações em memória como colorbuffer e o framebuffer.

Rasterização

Rasterização é o processo onde se transforma uma imagem vetorial em uma imagem formada de pixel ou pontos impressos na tela .
framebuffer é uma memória especial capaz de armazenar e transferir para a tela os dados de um quadro de imagem completo. **Colorbuffer** é uma subdivisão do framebuffer, essa memória é responsável pela definição de cores dentro do sistema. As cores são representadas dentro do sistema da seguinte maneira. O RGBA que é definido por bytes e cada letra presente na sigla representa uma cor, **R**ed, **G**reen, **B**lue, **A**lpha(transparência)

Um pixel é o menor ponto que forma uma imagem digital, sendo que o conjunto de pixels formam a imagem inteira. Um pixel é capaz de armazenar sua posição em relação a tela do monitor e sua cor. Diante destas informações elaboramos as seguintes estruturas:
Onde Point armazena as coordenadas do ponto , Color a cor e pixel as informações referentes a cada pixel .

```
typedef struct {
    int x;
    int y;
} Point;

typedef struct {
    int r;
    int g;
    int b;
    int a;
} Color;

typedef struct {
    Point coord;
    Color color;
} Pixel;
```

PutPixel()

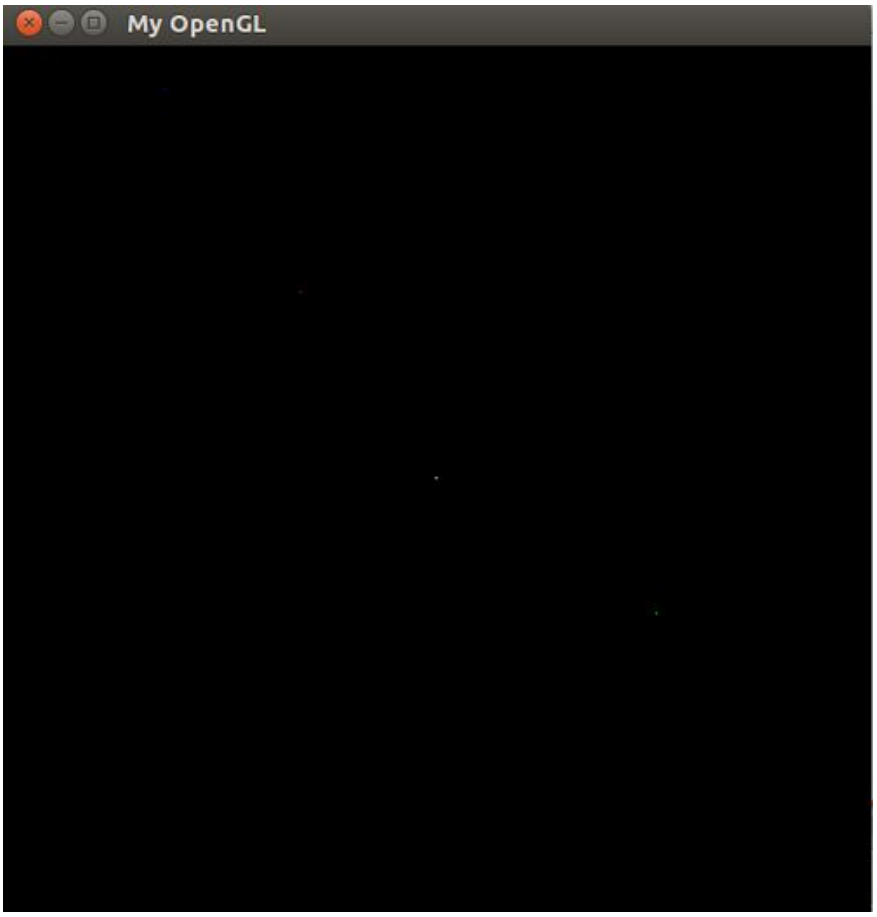
A função PutPixel é a função capaz de imprimir os vértices com as cores e no local desejado. Para descobrirmos em que local do framebuffer iremos imprimir usamos a variável **pos** que foi definida da seguinte forma:

```
unsigned int pos = (pixel.coord.y * IMAGE_WIDTH + pixel.coord.x) * 4;
```

Implementação a função PutPixel():

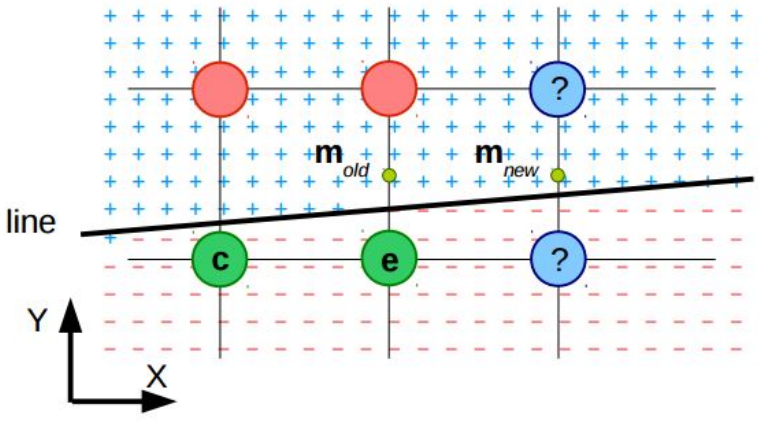
```
void PutPixel(Pixel pixel) {
    if(!(pixel.coord.x < 0 || pixel.coord.y < 0 || pixel.coord.x > IMAGE_WIDTH || pixel.coord.y > IMAGE_HEIGHT)) { // não permitir ultrapassar os limites tirar print sem essa proteção

        unsigned int pos = (pixel.coord.y * IMAGE_WIDTH + pixel.coord.x) * 4;

        FBptr[pos]     = pixel.color.r;
        FBptr[pos + 1] = pixel.color.g;
        FBptr[pos + 2] = pixel.color.b;
        FBptr[pos + 3] = pixel.color.a;
    }
}
```

Resultado obtido após a execução da função PutPixel():



Rasterização da linha:

Algoritmo de **Bresenham** é um algoritmo criado para o desenho de linhas, em dispositivos matriciais (como por exemplo, um monitor), que permite determinar quais os pontos numa matriz de base quadriculada que devem ser destacados para atender o grau de inclinação de um ângulo. Para essa seleção é feita uma verificação de se o ponto médio está acima ou abaixo da reta, chegando na decisão de qual pixel irá ser colorido. Mas dependendo da escolha feita, será diferente o tratamento para descobrir o próximo ponto médio.

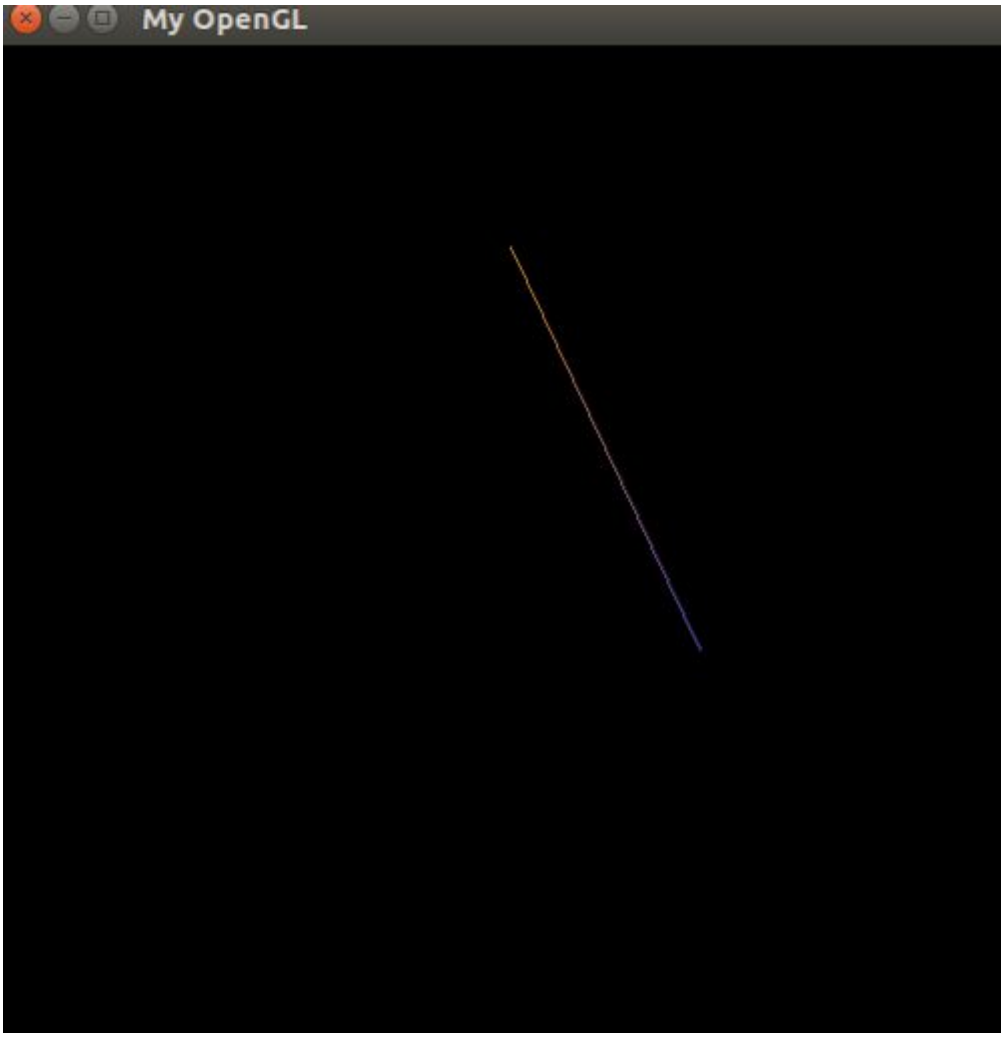


Função DrawLine()

Essa função recebe os dados presentes nos pixels que representam o primeiro e segundo pontos e faz a variação de cores ao longo da reta que está sendo desenhada na tela.

```
void DrawLine(Pixel pointA, Pixel pointB) {
    std::vector<Pixel> line;

    Pixel p = {.coord = {.x = pointA.coord.x, .y = pointA.coord.y},
    .color = {.r = pointA.color.r, .g = pointA.color.g, .b = pointA.color.b,
    .a = pointA.color.a}};

    int x = pointA.coord.x, y = pointA.coord.y;
    int * axis1 = &x;
    int * axis2 = &y;

    int last_x, last_y;

    if(pointB.coord.x < pointA.coord.x)
        last_x = (pointA.coord.x - pointB.coord.x) + pointA.coord.x;
    else last_x = pointB.coord.x;

    if(pointB.coord.y < pointA.coord.y)
        last_y = (pointA.coord.y - pointB.coord.y) + pointA.coord.y;
    else last_y = pointB.coord.y;

    int last_d = last_x;
    int dx = last_x - pointA.coord.x;
    int dy = last_y - pointA.coord.y;

    if(dx < dy) {
        int tmp = dx;
        dx = dy;
        dy = tmp;

        last_d = last_y;

        axis1 = &y;
        axis2 = &x;
    }

    int d = 2 * dy - dx;
    int incr_e = 2 * dy;
    int incr_ne = 2 * (dy - dx);
```

```
line.push_back(p);

while (*axis1 < last_d) {
    if (d <= 0) {
        d += incr_e;
        *axis1 += 1;
    } else {
        d += incr_ne;
        *axis1 += 1;
        *axis2 += 1;
    }

    p.coord.x = x;
    p.coord.y = y;

    if(last_x != pointB.coord.x)
        p.coord.x = pointA.coord.x - (p.coord.x - pointA.coord.x);

    if(last_y != pointB.coord.y)
        p.coord.y = pointA.coord.y - (p.coord.y - pointA.coord.y);

    line.push_back(p);
}

double incr_color[] = {
    (double) (pointB.color.r - pointA.color.r) / (line.size() - 1),
    (double) (pointB.color.g - pointA.color.g) / (line.size() - 1),
    (double) (pointB.color.b - pointA.color.b) / (line.size() - 1),
    (double) (pointB.color.a - pointA.color.a) / (line.size() - 1)
};

double color[] = {pointA.color.r, pointA.color.g, pointA.color.b, pointA.color.a};

for(int i = 0; i < line.size(); i++) {
    line.at(i).color.r = round(color[0]);
    line.at(i).color.g = round(color[1]);
    line.at(i).color.b = round(color[2]);
    line.at(i).color.a = round(color[3]);

    PutPixel(line.at(i));

    color[0] += incr_color[0];
```

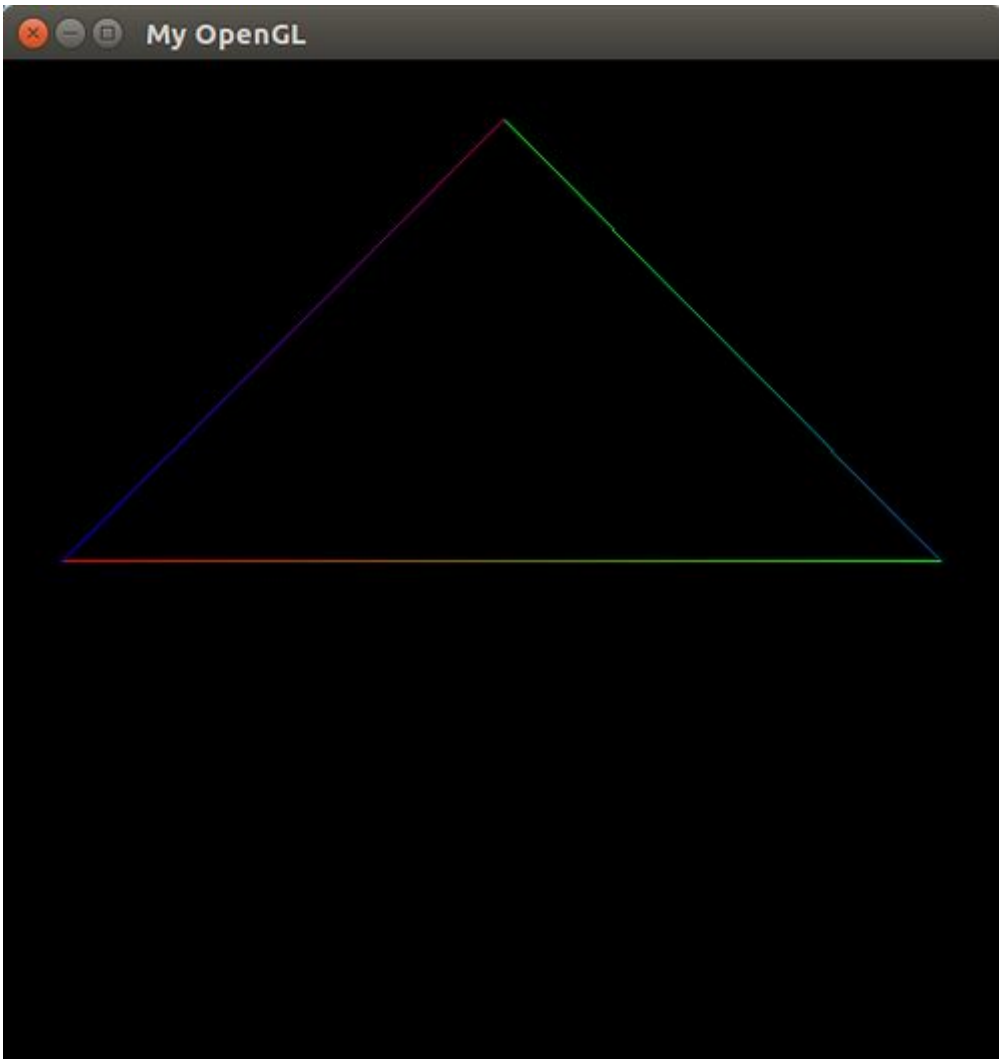


Os resultados obtidos pela função podem ser observados na figura acima.

Função DrawTriangle()

A definição desta função é muito simples, ela vai receber os parâmetros dos vértices que compõem o triângulo, com isso a função drawline desenhar as linhas que formam as arestas do triângulos

```
void DrawTriangle(Pixel pointA, Pixel pointB, Pixel pointC) {
    DrawLine(pointA, pointB);
    DrawLine(pointB, pointC);
    DrawLine(pointC, pointA);
}
```



Referências :
F. Lima , Lucas . Computação Gráfica UFPB .Disponível em :
https://icglima20152.wordpress.com/ . acessado em 16 mar. 2018.
WIKIPÉDIA. **Algoritmo de Bresenham.** Disponível em:
<https://pt.wikipedia.org/wiki/Algoritmo_de_Bresenham >. Acesso em: 15 mar. 2018.