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 - 11228383

Disciplina: **Introdução a Computação Gráfica**

Professor: **Christian Pagot**

Primeira Atividade

João Pessoa

Mar/2018

Especificação Técnica do Projeto

Nesta atividade implementamos uma ferramenta capaz de desenhar triângulos através de um algoritmo de rasterização de pontos e linhas, e como produto final deve ser as arestas de um triângulo.

Separamos este desenvolvimento em três funções que 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

A 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 por RGBA que é definido por bytes e cada letra presente na sigla representa uma cor: Red, Green, Blue e Alpha(transparência).

Um pixel é a menor unidade em uma imagem digital, sendo assim o conjunto de pixels formam a imagem inteira. Um pixel é capaz de armazenar sua posição em relação à 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.

```c
// definindo o pixel

typedef struct
{
    int c_x;
    int c_y; // esta struct define as coordenadas x e y do ponto

} ponto;
typedef struct
{   /* esta estruct será capaz de definir a cor em um ponto dentro o
    framework */
    int r;
    int g;
    int b;
    int a;

} cor;
typedef struct
{ /*esta struct define o pixel*/
    ponto coordenada;
    cor 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:

```
/*determinando o local no framebuffer */

unsigned int p_p= 4 * (pixel.coordenada.c_y*IMAGE__WIDTH+ pixel.coordenada);
```

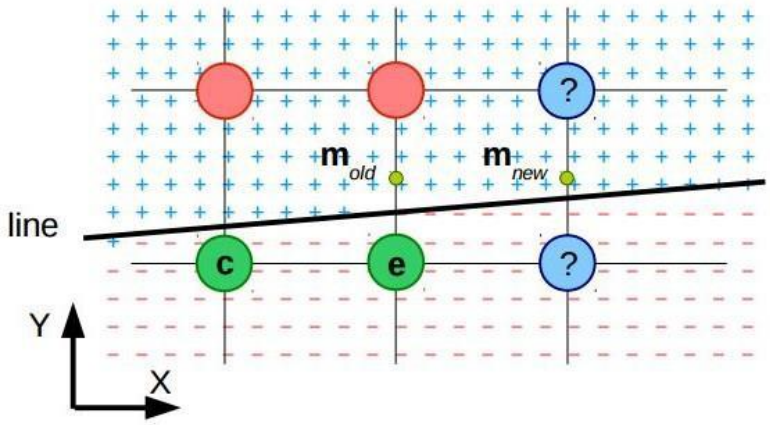
```
void putpixel(pixel quadrado)
{

    if(!(quadrado.coordenada.c_x<0 || quadrado.coordenada.c_y ||
     quadrado.coordenada.c_x> IMAGE__WIDTH ||
      quadrado.coordenada.c_y>IMAGE__WIDTH ))
    {
        unsigned int p_p= 4 * (pixel.coordenada.c_y*IMAGE__WIDTH+ pixel.coordenada);

        FBptr[p_p] = quadrado.color.r;
        FBptr[p_p+1] = quadrado.color.g;
        FBptr[p_p+2] = quadrado.color.b;
        FBptr[p_p+3] = quadrado.color.a;
    }

}
```

Implementação da função PutPixel():

Após a execução da função PutPixel() obtivemos os seguintes resultados:

Rasterização da Linha

O 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 verificado se o ponto médio está acima ou abaixo da reta, chegando na decisão de qual pixel irá ser colorido. Dependendo da escolha feita, será diferente o tratamento para descobrir o próximo ponto médio.



Função DrawLine()

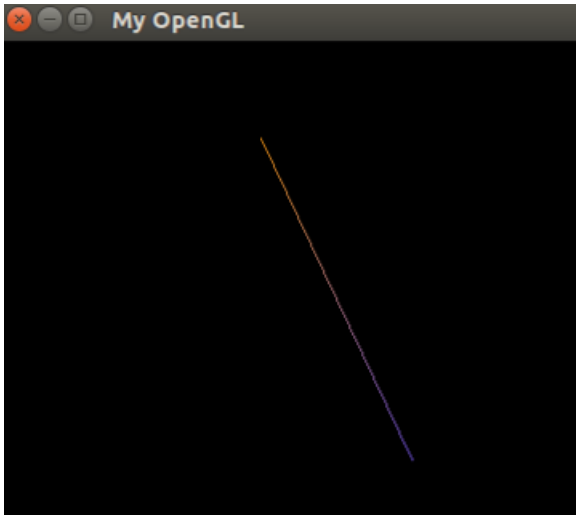
Essa função recebe os dados presentes nos pixels que representam o primeiro e segundo pontos e faz a variação de cores ao longo da reta que está sendo desenhada na tela.

```
void DrawLine (pixel ponto1, pixel ponto2)
{
    std:: vector<pixel> line;

    pixel ponto = {.coordenada = {.c_x=ponto1.coordenada.c_x, .c_y=ponto1.coordenada.c_y},
    .cor={.r=ponto1.cor.r, .g=ponto1.g, .b=ponto1.cor.b,
    .a=ponto1.cor.a}};

    int p_x = ponto1.coordenada.c_x, p_y =ponto1.coordenada.c_y;
    int * a_1 = &p_x;
    int * a_2 = &p_y;

    int ultimo_x, ultimo_y;

    if(ponto2.coordenada.c_x< ponto1.coordenada.c_x)
    {
        ultimo_x=(ponto1.coordenada.c_x - ponto2.coordenada.c_x) + ponto1.coordenada.c_x;

    }
    else
    {
        ultimo_x = ponto2.coordenada.c_x;
    }
    if(ponto2.coordenada.c_y < ponto1.coordenada.c_y)
    {
        ultimo_y = (ponto1.coordenada.c_y - ponto2.coordenada.c_y) + ponto1.coordenada.c_y;

    }
    else
    {
        ultimo_y = ponto2.coordenada.c_y;
    }

    /*ALGORITMO DE BRESENHAM CLASSICO
    IMPLEMENTADO EM SALA DE AULA*/
     int ultimoD= ultimo_x;

     int d_x = ultimo_x - ponto1.coordenada.c_x;
     int d_y = ultimo_y - ponto1.coordenada.c_y;

     if(d_x < d_y)
     {
        int temp= d_x;
```

```
if(d_x < d_y)
{
   int temp= d_x;
   d_x = d_y;
   d_y = temp;
   ultimoD = ultimo_y;

   a_1 = &c_y;
   a_2 = &c_x;
}

int dd = 2 * d_y - d_x;
int incrementa_e = 2 * d_y;
int incrementa_ne = 2 * (d_y - d_x);

line.push_back(ponto);

while (*a_1 < ultimoD)
{
   if( dd <= 0)
   {
       dd += incrementa_ne;
       *a_1 += 1;
   }
   else
   {
       dd += incrementa_ne;
       *a_1 += 1;
       *a_2 += 1;
   }
   ponto.coordenada.c_x = c_x;
   ponto.coordenada.c_y = c_y;

   if(ultimo_x != ponto2.coordenada.c_x)
   {
       ponto.coordenada.c_x= ponto1.coordenada.c_x - (ponto.coordenada.c_x - ponto1.coordenada.c_x);
   }
   if (ultimo_y != ponto2.coordenada.c_y)

   {
   ponto.coordenada.c_y = ponto1.coordenada.c_y - (ponto.coordenada.c_y - ponto1.coordenada.c_y);
   line.push_back(ponto);
   }
```

```
   {
   ponto.coordenada.c_y = ponto1.coordenada.c_y - (ponto.coordenada.c_y - ponto1.coordenada.c_y);
   line.push_back(ponto);
   }

   double incrementa_cor[] =
   { (double) (ponto2.cor.r - ponto1.cor.r)/ (line.size()) - 1),
       (double) (ponto2.cor.g - ponto1.cor.g)/ (line.size()) - 1),
       (double) (ponto2.cor.b - ponto1.cor.b)/ (line.size()) - 1),
       (double) (ponto2.cor.a - ponto1.cor.a)/ (line.size()) - 1)
   };
   double cor[] = {ponto1.cor.r, ponto1.cor.g,ponto1.cor.b, ponto1.color.a};
   for(int i = 0; i < line.size(), i++)
   {
       line.at(i).cor.r = round(cor[0]);
       line.at(i).cor.g = round(cor[1]);
       line.at(i).cor.b = round(cor[2]);
       line.at(i).cor.a = round(cor[3]);
   }
   putpixel(line.at(i));
   cor[0] += incrementa_cor[0];
```

Após a execução da função DrawLine obtivemos os seguintes resultados:

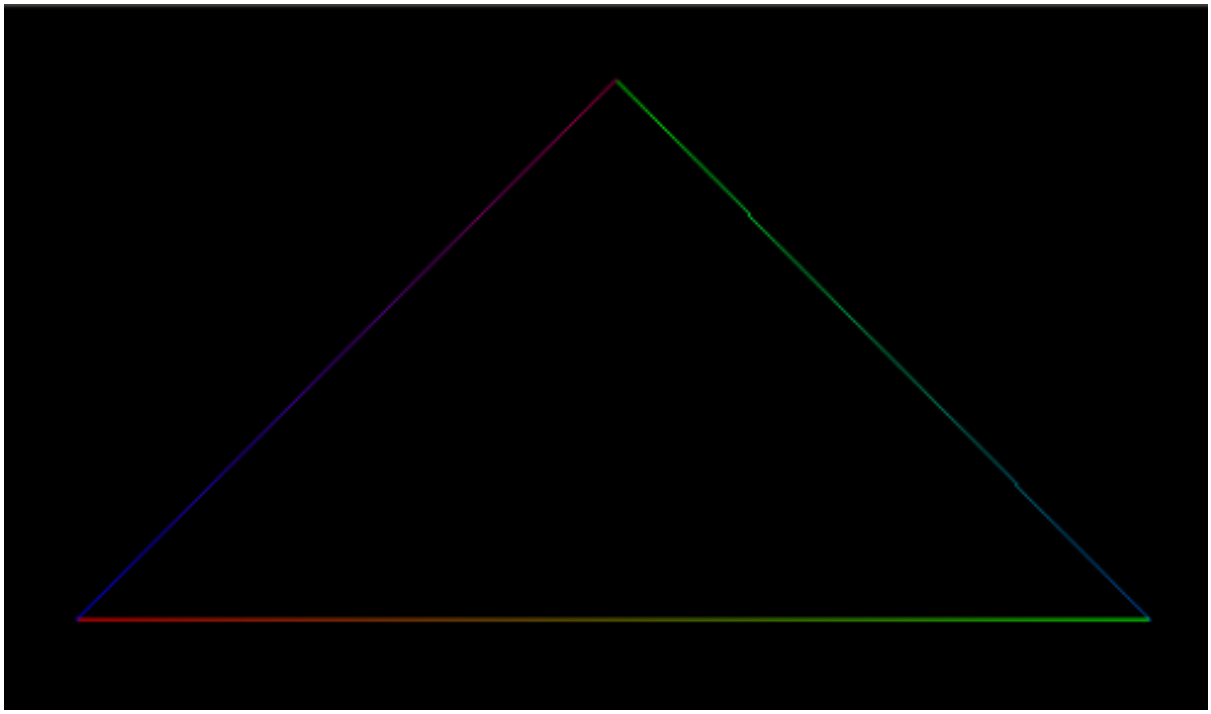


Função DrawTriangle()

A definição desta função é muito simples, pois receberá os parâmetros dos vértices que compõem o triângulo, com isso a função DrawLine irá desenhar as linhas que são as arestas do triângulo.

```
void DrawTriangule(pixel ponto1, pixel ponto2, pixel ponto3)
{
    /*CONECTA AS RETAS COM O OBJETIVO DE FORMAR UM
    TRIANGULO EQUILATERO*/
    DrawLine(ponto1,ponto2);
    DrawLine(ponto2,ponto3);
    DrawLine(ponto3,ponto1);
}
```

Após a execução da função DrawTriangule obtivemos os seguintes resultados:



Referências

- F. Lima, Lucas. Computação Gráfica UFPB. Disponível em: <https://icglima20152.wordpress.com/>. Acesso em: 01 Jun. 2018.

- WIKIPÉDIA. Algoritmo de Bresenham. Disponível em: <https://pt.wikipedia.org/wiki/Algoritmo_de_Bresenham >. Acesso em: 01 Jun. 2018.