Universidade Federal da Paraíba
Centro de Informática

Aluno: Rosivaldo Lucas da Silva
Disciplina: Introdução a Computação Gráfica
Professor: Mailson B. Pacheco

Atividade Prática - Rasterizando Linhas

- **Estrutura utilizada**

  Para a representação de um ponto na tela, foi utilizada uma **struct Ponto**, que tem como dados as coordenadas x e y e também a informação de RGB do ponto.
  O objetivo dessa estrutura é encapsular e simplificar a passagem desses valores para os métodos implementados, então para mostrar um pixel na tela até desenhar um triângulo, deve ser utilizada a **struct Ponto**.

```c
// estrutura para encapsular os dados da coordenada e cor do ponto
typedef struct Ponto Ponto;
struct Ponto {
  int x;
  int y;
  int R;
  int G;
  int B;
};
```

- **PutPixel**

  Implementação do método que realiza a adição de um pixel na tela. Esse método recebe as coordenadas x e y que indicam onde o pixel deve ser colocado na tela e os dados RGB de cor do pixel. Para passar esses dados ao método, deve ser utilizada a **struct Ponto**. Para essa implementação a maior dificuldade foi entender a lógica para encontrar o índice do ponteiro **FBptr** que representa a coordenada (x, y) a ser marcada na tela.

Abaixo segue o print com pixels marcados na tela, com as cores RGB, cada pixel em uma cor. Também é mostrado o print da implementação do método **myPutPixel**.

```
void myPutPixel(Ponto p) {
  /**
   * Calcula o índice no array FBptr com base nas coordenadas x e y,
   * assim acessando na coordenada (p.x, p.y).
  */
  int indiceInicioPixel = (p.y * IMAGE_WIDTH + p.x) * 4;

  // Escreve os valores das componentes de cor no array FBptr
  FBptr[indiceInicioPixel] = p.R;       // componente R
  FBptr[indiceInicioPixel + 1] = p.G;   // componente G
  FBptr[indiceInicioPixel + 2] = p.B;   // componente B
  FBptr[indiceInicioPixel + 3] = 255;   // componente A
}
```

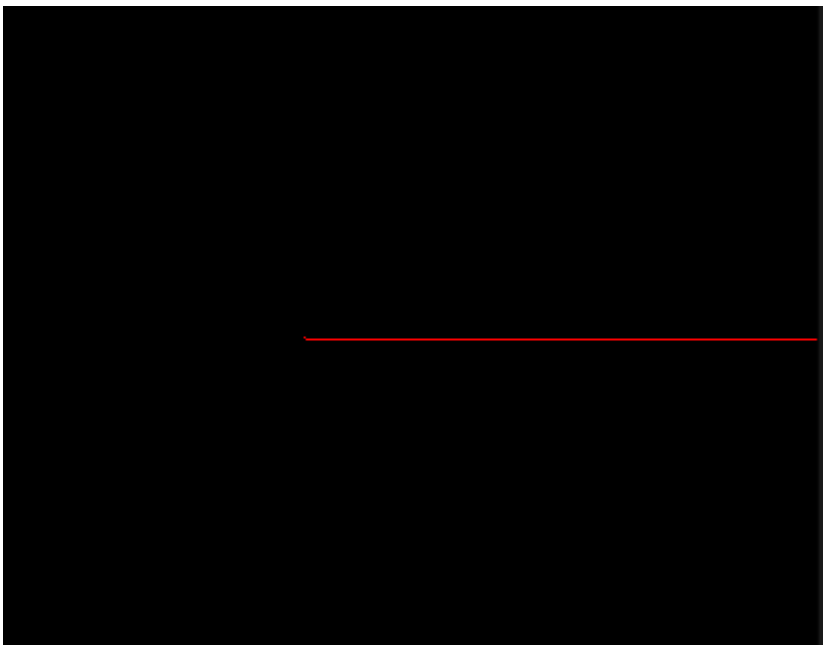
- **DrawLine**

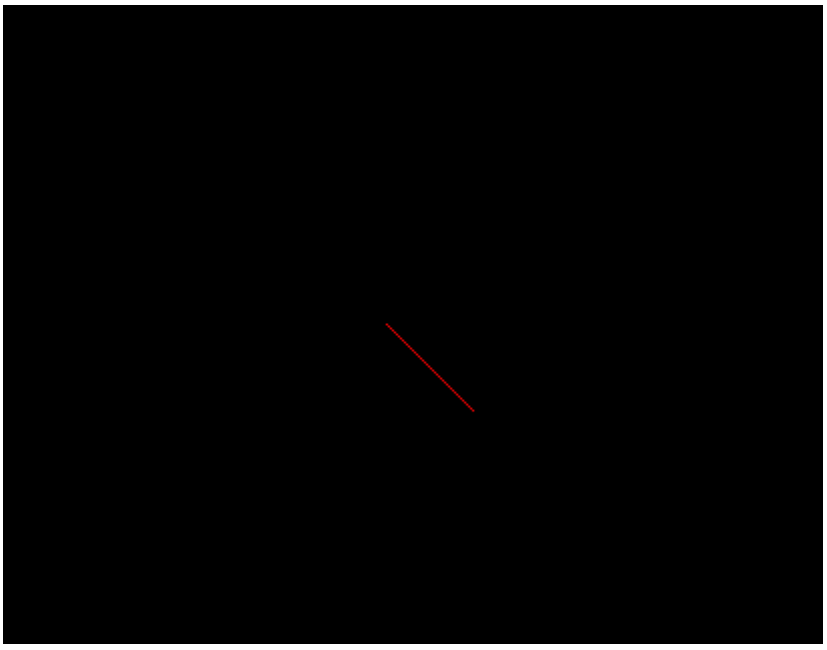
  Implementação do método **myDrawLine** que realiza a rasterização de uma linha na tela, com base no ponto inicial e ponto final, o método encontra os pontos (pixels) intermediários a serem adicionados para que a linha seja exibida na tela. O algoritmo implementado foi o **Algoritmo de Bresenham**.

  A primeira implementação do algoritmo tem a capacidade de rasterizar linhas no 1 octante do plano da tela.

  Segue o teste com os valores dos pontos inicial e final, pi(256, 256) e pf(512, 256), ponto na horizontal cobrindo toda a parte do primeiro octante da tela de 512x512 pixels. E outro teste com as coordenadas pi(256, 256) e pf(300, 512) que rasterizam uma linha no segundo octante da tela.

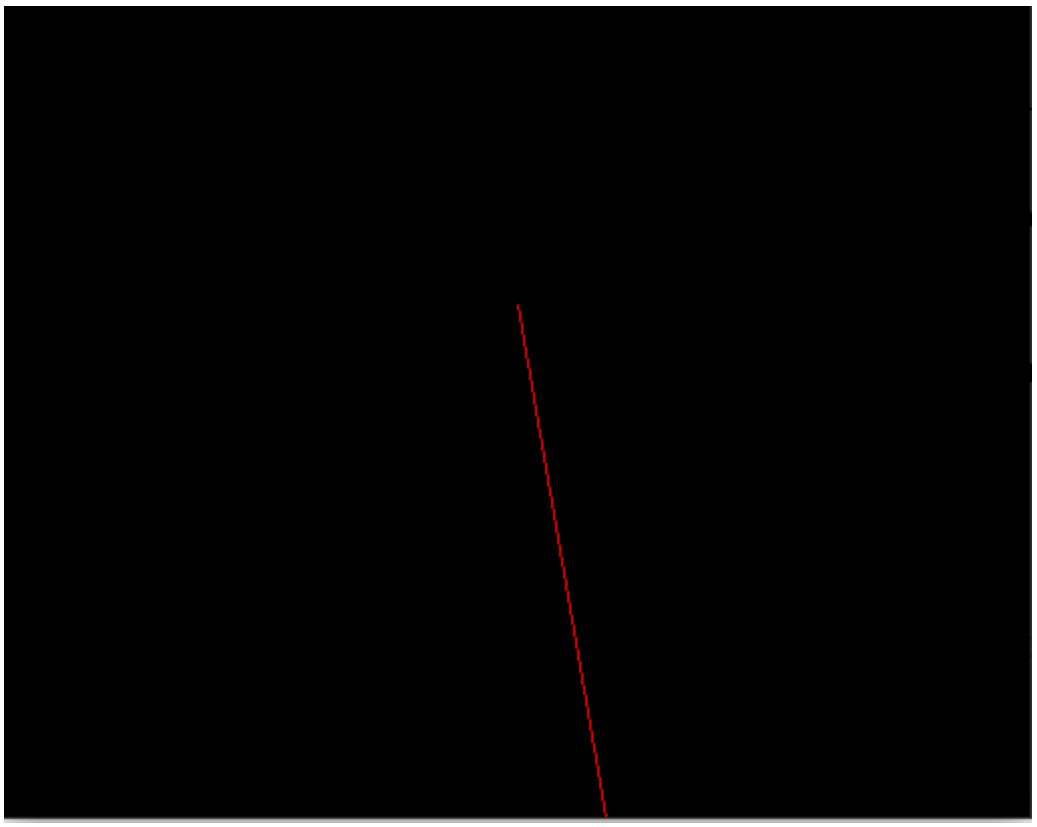
  A segunda imagem mostra a tentativa de rasterizar a linha para o segundo octante, mas o algoritmo não foi capaz de realizar a rasterização, pois ele apenas faz rasterização para o primeiro octante.



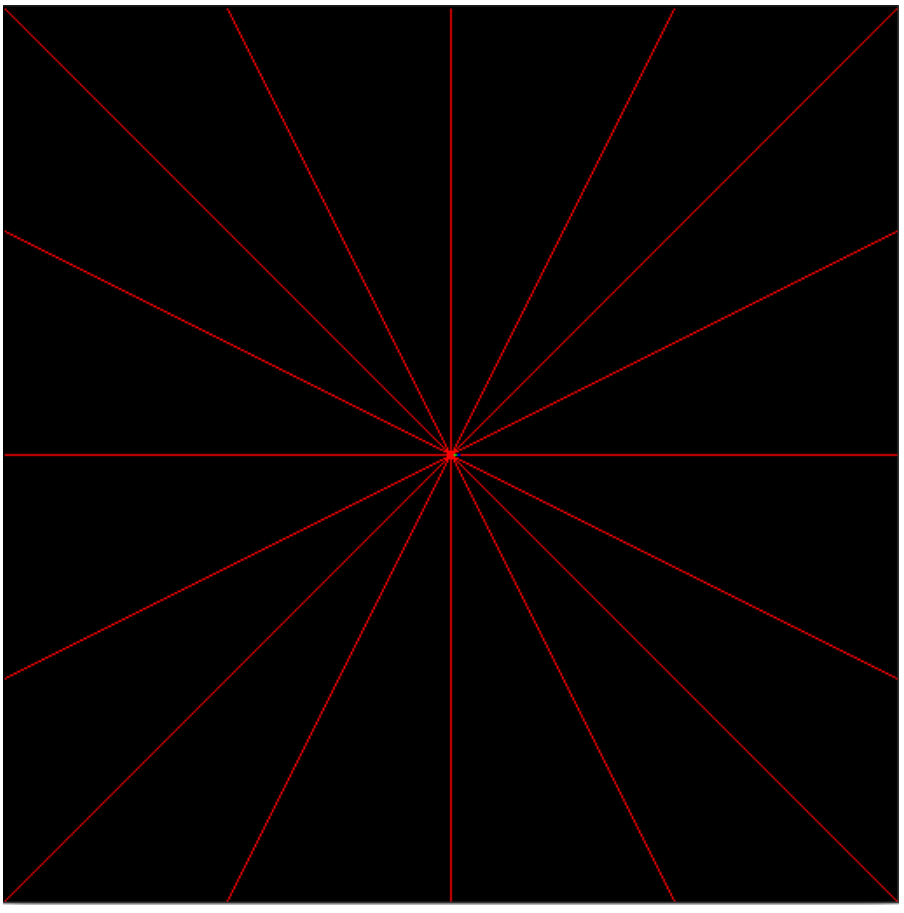


Implementação do método **myDrawline** que realiza a rasterização em todos os 8 octantes do plano da tela. Para essa implementação também foi utilizado o **Algoritmo de Bresenham**, mas com algumas alterações para que seja possível rasterizar todos os octantes. Foi adicionado as variáveis **stepX** e **stepY** que indica o passo de x e y, podendo ser definido como 1 e -1, os passos dependem do valor da diferença entre os valores final e inicial dos pontos inicial e final passados. Para o loop de rasterização foi utilizado a lógica de andar tanto na coordenada x quanto na y, assim podendo rasterizar em todos os octantes da tela.

Segue os testes de rasterização para dois pontos marcados nas coordenadas pi(256, 256) e pf(300, 512) que antes não era possível rasterizar a linha, esse ponto se encontra no segundo octante do plano da tela.



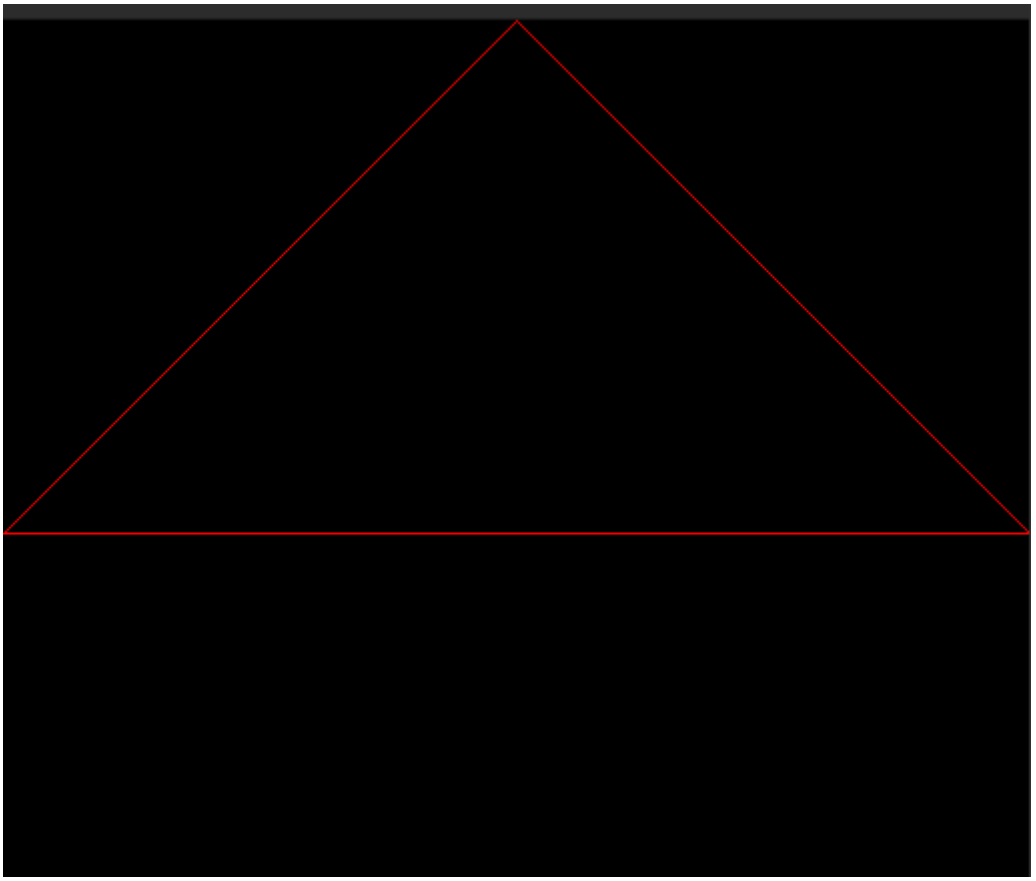
Segue o print solicitado, com retas passando pelos 8 octantes da tela.



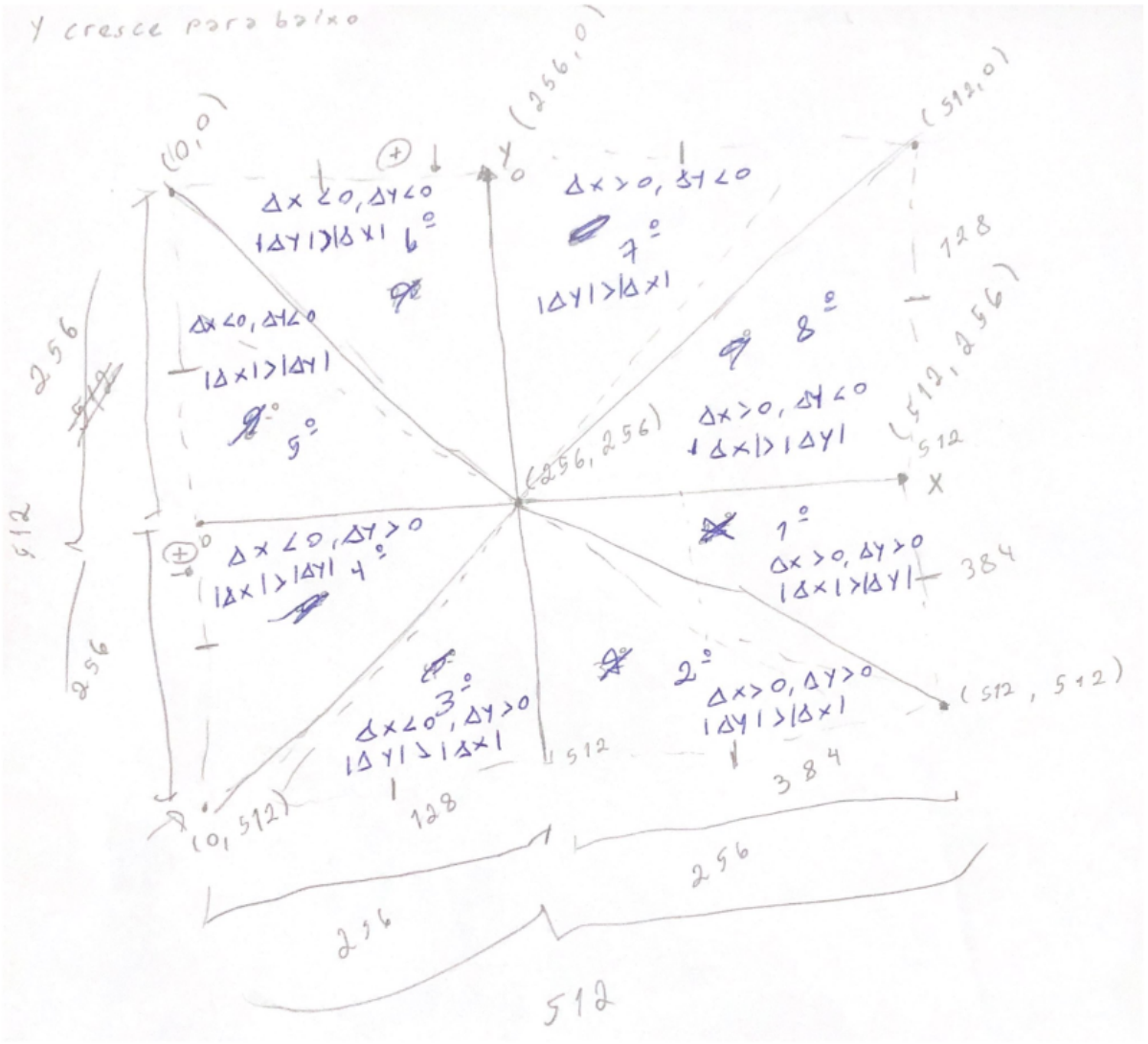
- **DrawTriangle**

  Implementação do método **myDrawTriangle** que realiza a rasterização das linhas entre os três vértices de um triângulo. Esse método foi simples de implementar, pois com o método de rasterizar uma linha entre dois pontos, é apenas necessário para mostrar o triângulo na tela, chamar o método **myDrawLine** três vezes passando o vértice a com vértice b, o vértice a com o vértice c e por fim o vértice b com o vértice c do triângulo, então o método **myDrawLine** vai rasterizar as arestas desse triângulo na tela.

  Segue o teste de desenho do triângulo com vértices em pa(256, 0), pb(0, 512) e pc(512, 512) da tela.



- Rascunho utilizado para o entendimento do funcionamento da tela e os octantes.



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