



UFRR

UNIVERSIDADE FEDERAL DE RORAIMA  
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO  
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CLEUSON SOUSA SANTOS

## **COMPUTAÇÃO GRÁFICA: ALGORITMOS DE RASTERIZAÇÃO DE LINHAS**

Boa Vista – Roraima,  
Novembro de 2022

CLEUSON SOUSA SANTOS

**COMPUTAÇÃO GRÁFICA: ALGORITMOS DE RASTERIZAÇÃO DE LINHAS**

Relatório apresentada ao Departamento de Ciência da Computação da Universidade Federal de Roraima, como requisito parcial para a obtenção de nota na Disciplina Computação Gráfica.

**Orientador**

ProfºDr. Luciano Ferreira Silva

Boa Vista – Roraima,  
Novembro de 2022

## RELATÓRIO

Discente: Cleuson Sousa Santos

Tema: Rasterização de retas por meios de algoritmos: Analítico, DDA e Bresenham.

---

O programa desenvolvido permite desenhar retas por meios dos algoritmos apresentados em sala de aula: Analítico, DDA e Bresenham. Implementado de maneira que se pode ver o trabalho de cada algoritmo e comparar os resultados.

### 1. Grid de Pixels

Segundo o padrão utilizado pela indústria, e o que consta na literatura, o sistema de coordenadas de pixel, define a origem de um destino de renderização no canto superior esquerdo, e avança por coordenadas positivas no eixo x para a direita e no eixo y para baixo.

Com a finalidade de tornar a rasterização mais didática e intuitiva, levou-se em consideração o sistema de coordenadas de pixels, no entanto, foi dada preferência e por transformar o sistema de coordenadas para o sistema euclidiano mais próximo do utilizado nas aulas, de tal maneira que torne melhor compreensível o resultado pretendido, tanto na renderização do resultado quanto na comparação de resultados.

A grid foi implementado como uma malha com 80 pixels de altura, com 80 pixels de largura, mais uma coluna e linha para representar o valor 0. A malha foi dividida em valores variando de -40 a 40, tanto no eixo x, quanto no eixo y.

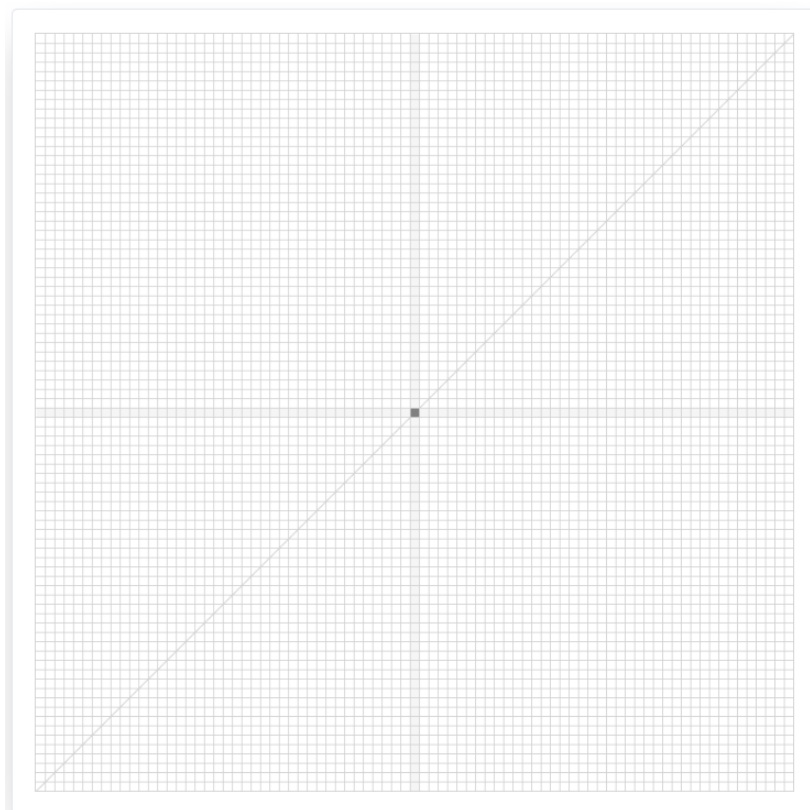


Fig 1. Malha de pixels

## 2. Controles

Foi implementado um conjunto de controles que possibilite a escolha do algoritmo que será utilizado para renderizar a linha, além de determina a coordenada inicial e a final da linha.

Método	
<input checked="" type="radio"/>	Analítico
<input type="radio"/>	Bresenham
<input type="radio"/>	DDA

Início	
x	0
y	0

Fim	
x	0
y	0

Fig 2. Controle para escolha de parâmetros da linha e comparação de algoritmo

## 3. Rasterização

Ao se escolher o algoritmo e entrar com as coordenadas os a mostrada na malha de pixels, atualizando sempre que se altera campo do formulário de controle, automaticamente.

## 4. Implementação

O programa foi implementado com JavaScript, utilizando o Framework NextJS.

a. Algoritmo Analítico.

```
/* -- Metodo Analitico -- */
export const analytic = (start, end) => {

  //Inverte pontos
  if (start.x > end.x) {
    return analytic(end, start);
  }

  let line = [];

  //Reta Vertical
  if (start.x === end.x) {

    //Inverte pontos
    if (start.y > end.y) {
      return analytic(end, start);
    }

    for (let y = start.y; y <= end.y; y++) {
      //line.push(Point(start.x, y));
    }
  }

  // Reta Inclínada
  else {
    const m = (end.y - start.y) / (end.x - start.x);
    const b = end.y - m * end.x;

    for (let x = start.x; x <= end.x; x++) {
      let y = m * x + b;
      line.push(Point(x, Math.round(y)));
    }
  }
  return line;
}
```

b. Algoritmo DDA.

```
/* -- Metodo DDA -- */
const dda = (start, end) => {

  let line = [];

  let dx = Math.abs(end.x - start.x);
  let dy = Math.abs(end.y - start.y);

  if (dx > dy) {
    // Inverte Pontos
    if (start.x > end.x) {
      const temp = start;
      start = end;
      end = temp;
    }

    let inc = (end.y - start.y) / (end.x - start.x);
    let y = start.y;

    for (let x = start.x; x <= end.x; x++) {
      line.push(Point(x, Math.round(y)));
      y += inc;
    }
  }
  else {
    // Inverte Pontos
    if (start.y > end.y) {
      const temp = start;
      start = end;
      end = temp;
    }

    let inc = (end.x - start.x) / (end.y - start.y);
    let x = start.x;

    for (let y = start.y; y <= end.y; y++) {
      line.push(Point(Math.round(x), y));
      x += inc;
    }
  }

  return line;
}
```

c. Algoritmo Bresenham.

```
function bresenham(start, end) {

    // Armazena os pontos para retorno
    const line = [];

    // Define dx e dy
    const dx = end.x - start.x;
    const dy = end.y - start.y;

    // Inverte Pontos
    if (dx < 0) {
        return bresenham(end, start);
    }

    // Define a inclinacao da reta
    let m = 0;
    if (dy < 0) {
        m = -1;
    } else {
        m = 1;
    }

    /*
    dx >= m * dy : inclina m <= 1
    dx < m * dy => |m| > 1
    */
    if (dx >= m * dy) {

        /*
        dy < 0 => y2 < y1
        dy > 0 => y2 > y1
        */
        if (dy < 0) {

            // Define y inicial como y1
            let y = start.y;

            // Define Parametro de decisao
            let d = 2 * dy + dx;

            // Percorre incrementando x em uma unidade
            for (let x = start.x; x <= end.x; x++) {
                line.push(Point(x, y));

                if (d < 0) {
                    d = d + 2 * (dy + dx);
                    y = y - 1;
                } else {
                    d = d + 2 * dy;
                }
            }
        }
        else {
            let y = start.y;
            let d = 2 * dy - dx;

            for (let x = start.x; x <= end.x; x++) {
                line.push(Point(x, y));

                if (d < 0) {
                    d = d + 2 * dy;
                } else {
                    d = d + 2 * (dy - dx);
                    y = y + 1;
                }
            }
        }
    }
    else {
```

```

    if (dy < 0) { // y2 < y1
        let x = start.x;
        let d = dy + 2 * dx;

        for (let y = start.y; y >= end.y; y--) {
            line.push(Point(x, y));
            if (d < 0) {
                d = d + 2 * dx; //varia apenas no eixo y
            } else {
                d = d + 2 * (dy + dx);
                x = x + 1;
            }
        }
    } else {
        let x = start.x;
        let d = dy - 2 * dx;

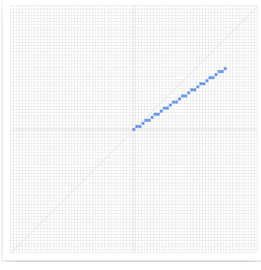
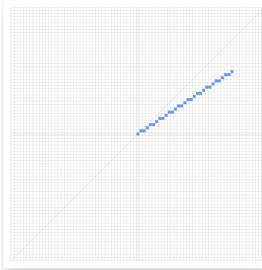
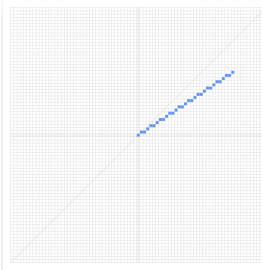
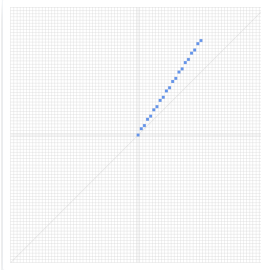


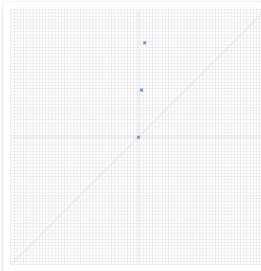
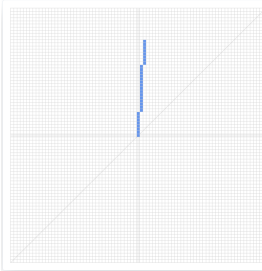

        for (let y = start.y; y <= end.y; y++) {
            line.push(Point(x, y));
            if (d < 0) {
                d = d + 2 * (dy - dx);
                x = x + 1;
            } else {
                d = d + (-2) * dx;
            }
        }
    }
}

return line;
}

```



## 5. Comparação

Coordernada	Analítico	DDA	Bresenham
inicio: (0,0) fim: (30,20)			
inicio: (0, 0) fim: (20, 30)			
inicio: (0, 0) fim: (2, 30)			

## 6. Considerações.

- Algoritmo analítico, foi o que apresentou mais facilidade, no entanto, o resultado para retas com angulos maior que  $45^\circ$  apresenta descontinuidades, e quando atinge  $90^\circ$  não apresenta nenhum ponto, visto que o algorimto de baseia no coeficiente angular, que é tangente, e portanto não há tangente para  $90^\circ$ . Nesse caso, a solução é implementar o desvio condicional para  $x_1 = x_2$ , que nessa versão está comentado.
- Algoritmo DDA, corrige os problemas de continuidade e quando  $x_1 = x_2$ , no entanto, utiliza-se de operações em pontos flutuantes, o que torna o algoritmo com custo computacional alto para grande escalas.
- Algoritmo de Bresenham, apresenta vantagens por não utilizar aritmética de ponto flutuante, no entanto, é o que apresenta a maior complexidade de implementação, neste caso, houve a necessidade de estudar os casos para os quatro quadrantes, e implementar cada caso.
- Considerou-se que a reta, é sempre um valor positivo, quando quando  $x_1 > x_2$ , inverte-se as coordenadas sem prejuízo dos algoritmos.