PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

COMPUTAÇÃO GRÁFICA

RELATÓRIO DE ALGORITMOS DE RASTERIZAÇÃO

ALUNOS:
Pedro Aleph Gomes de Souza Vasconcelos – 2016.007150

Outubro de 2019
Boa Vista/Roraima

1 Introdução

Este tem como objetivo relatar a construção feita e os resultados obtidos dos algoritmos de rasterização de linha e circunferência conforme foi proposta para ser feito, a partir dos pseudocódigos apresentados em aula, exceto o de preenchimento que não pode ser concluído.

2 Desenvolvimento

Para desenvolver foi utilizado a biblioteca p5.js do JavaScript [1], ela é focada em uso do canvas, contendo várias funcionalidades para desenhar. No caso foi utilizado o próprio canvas com grid(), e rect(), para pequenos quadrados representando cada pixel. Os códigos foram primeiramente testados no editor online disponibilizado pela biblioteca [2], e foram adaptados para serem usados abrindo o arquivo index. a biblioteca funciona por padrão com duas funções setup(), onde criamos o canvas e draw(), para criar forma no canvas, a partir daí, definindo o tamanho do canvas e as variáveis a serem utilizadas, basta somente por corpo do código (baseados nos pseudocódigos) dentro da função draw().

Linhas

Para os algoritmos de linha foi utilizado um canvas 300x300, e cada rect() com tamanho 30, assim temos um plano cartesiano de 10x10, de 0 a 9. As variáveis de entrada são x1 e y1 para P1, x2 e y2 para P2.

Analítico

Dado P1 e P2, o algoritmo traça a reta a partir da equação da reta $y = m \times x + b$ percorrendo pixel por pixel em (x,y), arredondando os resultados.

```
var X, Y, x, y, m, b;
if (x1 == x2){
    X = x1 * 30;
   for (y = y1; y <= y2; y++){
     Y = y * 30;
     rect (X, Y, 30, 30);
   }
}
else{
   m = (y2 - y1) / (x2 - x1);
   b = y2 - (m * x2);
   for(x = x1; x <= x2 ; x++){
     y = (m * x) + b;
     X = x * 30;
     Y = round(y) * 30;
     rect(X, Y, 30, 30);
   }
}
```

DDA

Dado P1 e P2, o algoritmo traça a reta considerando os valores dx e dy, caso dx > dy, então os pixels são percorridos em x de x1 a x2 com incremento y com dy/dx, caso dy > dx então os pixels são percorridos em y de y1 a y2 com incremento em x com dy/dx. Também arredonda os resultados.

```
var X, Y, x, y, incremento;
if ((y2 - y1)  < (x2 - x1)){
  incremento = (y2 - y1) / (x2 - x1);
  y = y1 * 30;
  for (x = x1; x <= x2 ; x++){
      X = round(x) * 30;
      Y = round(y) * 30;
      rect(X, Y, 30, 30);
      y += incremento;
  }
}
else{
  incremento = (x2 - x1) / (y2 - y1);
  x = x1 * 30;
  for (y = y1; y <= y2 ; y++){
      X = round(x) * 30;
      Y = round(y) * 30;
      rect(X, Y, 30, 30);
      x += incremento;
  }
}
```

Brasenham

Dado P1 e P2, o algoritmo traça a reta lembrado em consideração opção de qual o próximo pixel ele seguirá, evitando o arredondamento como por exemplo dado (x, y) ele decidirá se o próximo pixel será (x+1, y) ou (x+1, y+1) , o parâmetro de decisão também utiliza dx e dy.

```
var X, Y;
if(abs(x2 - x1) > abs( y2 - y1)){
  if (x2 > x1)
    LinhaHorizontal(x1, y1, x2, y2, X, Y);
  else
    LinhaHorizontal(x2, y2, x1, y1, X, Y);
}else{
  if (y2 > y1)
    LinhaVertical(x1, y1 , x2 , y2, X , Y);
  else
    LinhaVertical(x2, y2 , x1 , y1, X , Y);
}
```

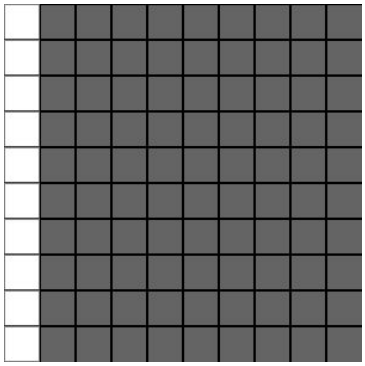
caso dx > dy

```
var dx = x2 - x1;
var dy = y2 - y1;
var yi = 1;
if (dy < 0){
  yi = -1;
  dy = -dy;
}
var y = y1;
var p = (2 * dy) - dx;
  for(var x = x1; x <= x2; x++){
    X = x * 30;
    Y = y * 30;
    rect(X, Y, 30, 30);
    if(p > 0){
      y = y + yi;
      p = p - 2 * dx;
    }
    p = p + 2 * dy;
  }
```

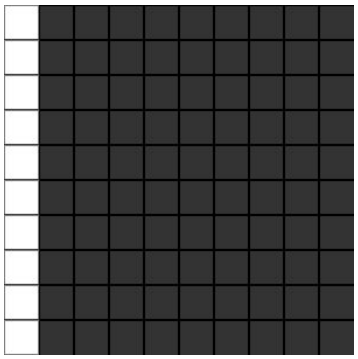
caso dy > dx

```
var dx = x2 - x1;
var dy = y2 - y1;
var xi = 1;
if (dy < 0){
  xi = -1;
  dx = -dx;
}
var x = x1;
var p = (2 * dx) - dy;
  for(var y = y1; y <= y2; y++){
    X = x * 30;
    Y = y * 30;
    rect(X, Y, 30, 30);
    if(p > 0){
      x = x + xi;
      p = p - 2 * dy;
    }
    p = p + 2 * dx;
  }
```

Quando x1=x2 (P1((0,0), P2(0,9))



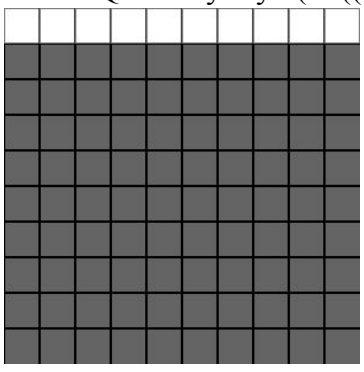
analitico



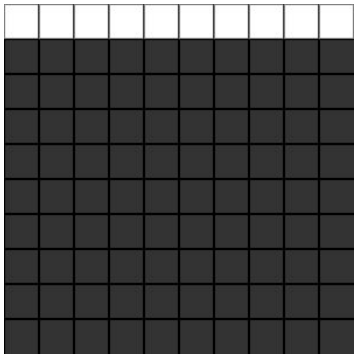
DDA

brasenham

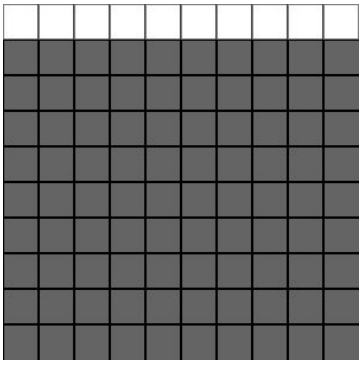
Quando y1=y2 (P1((0,4), P2(9,0))



analitico

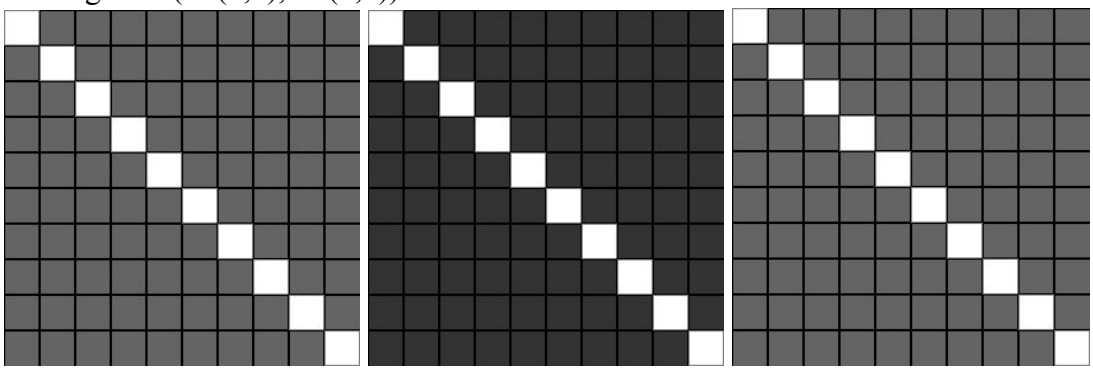


DDA



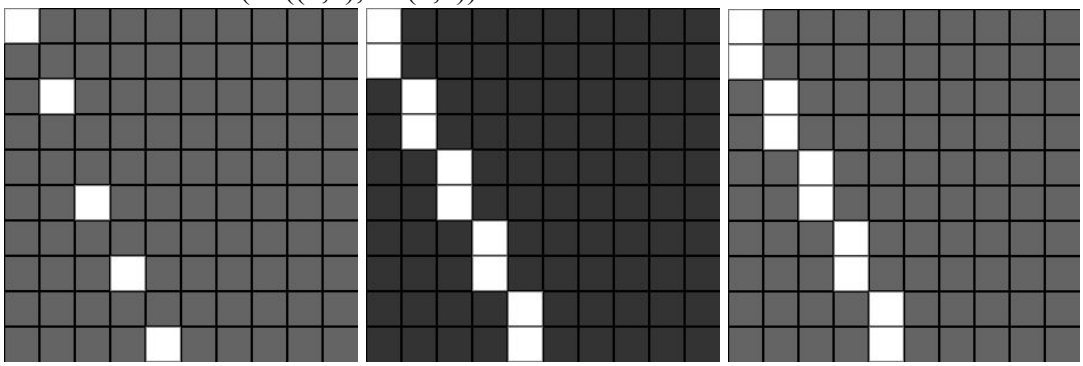
brasenham

Na diagonal (P1(0,0), P2(9,9))



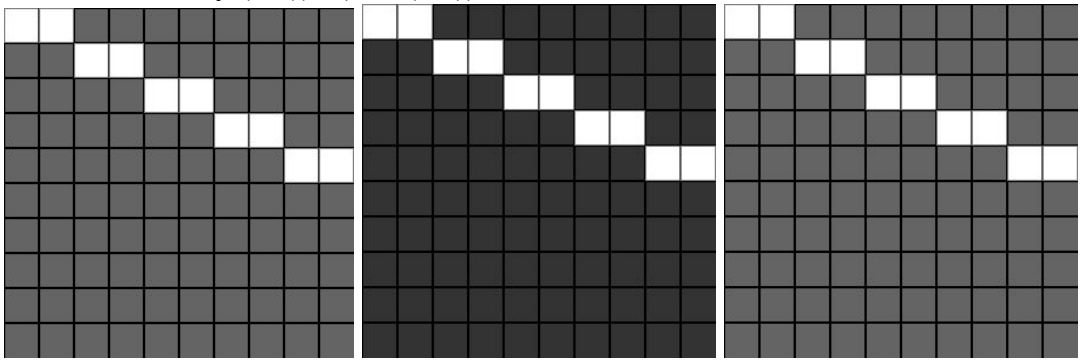
analitico DDA brasenham

Até a metade de x (P1((0,0), P2(4,9))



analitico DDA brasenham

Até a metade de y (P1((0,0), P2(9,4))



analitico DDA brasenham

Pode ser observado que DDA apresenta melhores resultados que o Analítico, já que em alguns casos preenche mais pixels, e semelhante ao Brasenham em relação aos pixels pintados, mas em retas maiores ele reproduziria mais lentamente e apresentaria erros de arredondamento.

Circunferências

Foi utilizado um canvas 600x600, com rect() de tamanho 20, com valores possíveis de 0 a 29 em x e y, as variáveis de entrada são xc, yc para o centro e r para o Raio.

Equação paramétrica

Dado xc, yc e r; o algoritmo percorre o gráfico com t de 1 a 360, inicialmente com x = xc + r e y = yc, e vai preenchendo os pixels e dando valores para x com xc + r×cos(t) e y com yc + r×sen(t). Arredonda os resultados.

```
var X, x, Y, y, t;
x = xc + r;
y = yc;
for (t = 1; t <= 360 ; t++){
    X = round(x) * 20;
    Y = round(y) * 20;
  rect(X, Y, 20, 20);

  x = xc + r * cos((PI *t)/180);
  y = yc + r * sin((PI *t)/180);
}
```

Simetria incremental

Dado os pontos de entrada, o algoritmo tem como propriedade que a circunferências é simétrica e assim ele so trabalha em um octeto e preenche o resto simetricamente. Iniciados x e y como no anterior, Enquanto x e y forem diferentes, preenche os pixels e incrementa x com a soma dos ângulos theta e alpha. sendo alpha o ângulo inicial 1/r e theta obtido pelas regras de trigonometria de cosseno e seno de alpha+theta. Arredondando os resultados (não consegui reproduzir este algoritmo).

Circunferência de Bresenham

Também trabalha com so um octante e preenche o demais pontos com simetria, evitando trabalha com operações que podem envolver pontos flutuantes., assim evitando o arredondamento, iniciando de x = 0 e y = r , escolhendo o pixel a ser preenchido a partir da equação fc(x,y), mh, md e mv, ou a partir do parâmetro de decisão calculado por sinal.

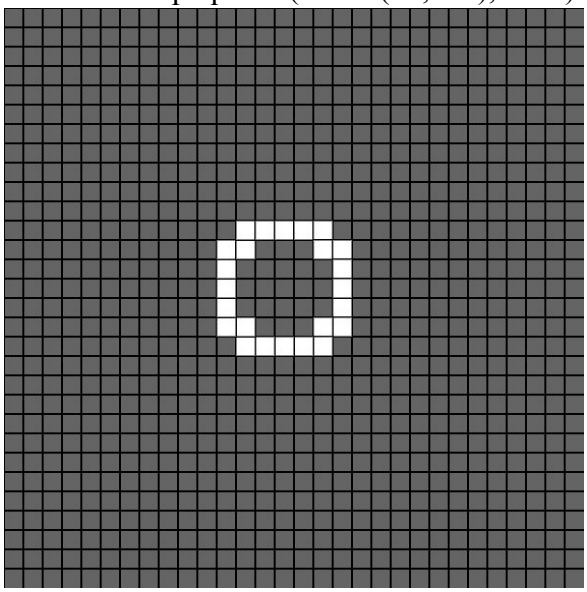
```
x = r; y = 0;
var P = 1 - r;
draw_simetric(xc, yc, x ,y);
while(x > y){
    y ++;
  if(P < 0){
    P = P + 2 * y + 1;
  }
  else{
    x --;
    P = P + 2 * (y - x) + 1;
      }
  draw_simetric(xc,yc,x,y);
  }
```

em[3],

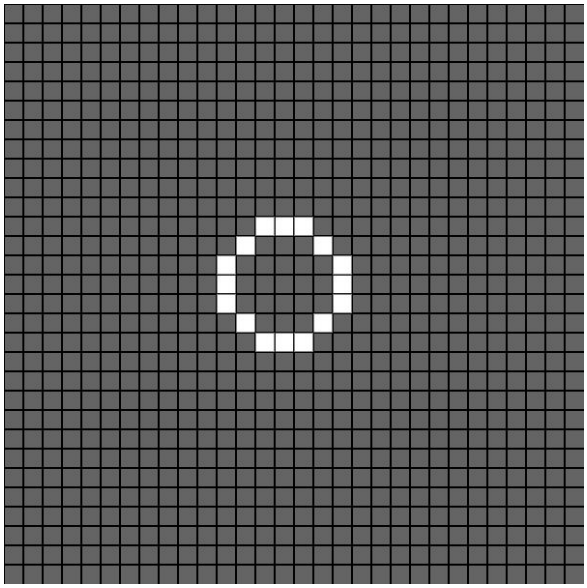
```
var d = 3 - 2 * r;
var x = 0;
var y = r;
drawCircle(xc, yc, x, y);
while ( x <= y){
  x ++;
  if ( d  < 0)
    d += 4*x + 6
  else{
    d += 4 * (x - y) + 10;
    y--;
  }
  drawCircle(xc, yc, x, y);
}
```

em [4]

Para um raio pequeno (centro(14, 14), r = 3)

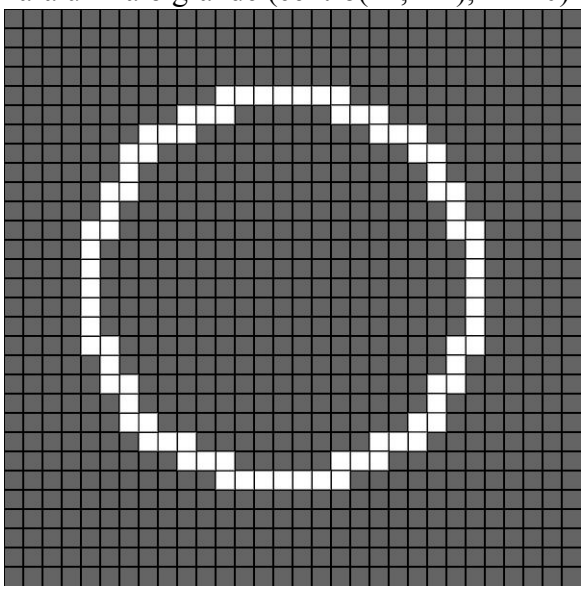


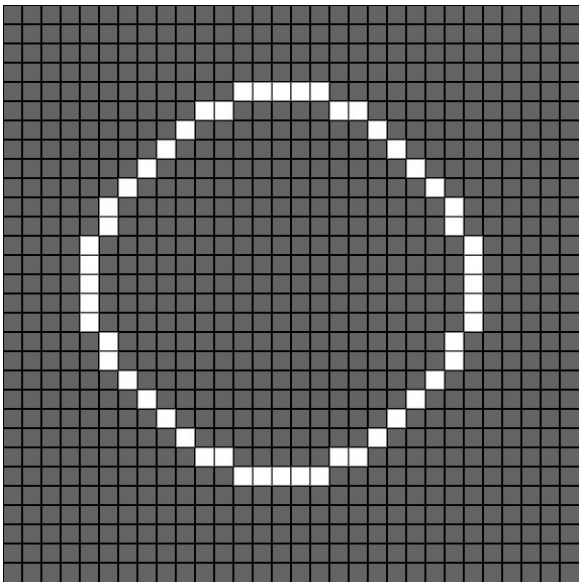
Equação Paramétrica



Brasenham

Para um raio grande (centro(14, 14), r = 10)





Observa-se que o algoritmo de equação paramétrica preenche pixels desnecessários em comparação com os demais, e para raios maiores ele irá deixar de marcar todos os pontos possíveis. Equando o de simetria incremental é semelhante ao de brasenham circular, exceto que o de brasenha em geral produz melhores resultados por trabalhar somente com inteiros, ambos se relacionam de mesma forma como brasenham e DDA nas retas.

3 Considerações finais

Este trabalho apresentou a implementação dos algoritmos rasterização de linhas e circunferências, somente foram adaptados para o programa, não foi criado nada novo para cada algoritmo. É uma boa forma de aprender o funcionamento destes na prática, e mostra como é ser trabalhoso para o computador representar formas geométricas, De maneira geral é uma ótima experiência.

4 Referências

[1] https://p5js.org/
[2] https://editor.p5js.org/
[3]https://www.geeksforgeeks.org/mid-point-circle-drawing-algorithm/
[4]https://www.geeksforgeeks.org/bresenhams-circle-drawing-algorithm/