IGCE – Instituto de Geociências e Ciências Exatas
DEMAC – Departamento de Estatística, Matemática e Computação

Computação Gráfica

Daniel Pedronette
pedronette@gmail.com

Agenda

- Padronização e Pacotes Gráficos
- Primitivas Gráficas
  - Ponto
  - Reta
    - Composição de Pontos
    - Equação da Reta
    - Algoritmo DDA
    - Algoritmo de Bresenham

Padronização

- Como o interesse em CG cresceu, é importante escrever aplicações que possam rodar em diferentes plataformas.
- Um padrão para desenvolvimento de programas gráficos facilita esta tarefa eliminando a necessidade de escrever código para um *driver* gráfico distinto para cada plataforma na qual a aplicação deve rodar.

Padronização

- Para de padronizar a construção de aplicativos que se utilizam de recursos gráficos e torná-los o mais independentes possível de máquinas, e portanto facilmente portáveis, foram desenvolvidos os chamados Sistemas Gráficos.
- Resumindo: Portabilidade.

Exemplos de Padronização

- A linguagem *Postscript* que se tornou um padrão por facilitar a publicação de documentos estáticos contendo gráficos 2D e textos;
- Sistema *XWindow,* que se tornou padrão para o desenvolvimento de interfaces gráficas 2D em *workstations* UNIX;

Padrões Gráficos

- Depois de muitos esforços de diversas organizações em diversos países, surge um primeiro padrão em 1984:
  - Graphical Kernel System (GKS): foi adotado como primeiro padrão de software gráfico pela ISO (International Standard Organization) e ANSI (American National Standards Institute).
  - Desenvolvido inicialmente para 2D, mas com extensão para 3D.

Padrões Gráficos

- PHIGS (Programmer´s Hierarchical Interactive Graphics Standard): extensão do GKS
  - Incorpora funcionalidades como: renderização de superfícies, modelagem hierárquica de objetos e manipulação de figuras.
  - Extensão PHIGS+ para 3D.

Padrões Gráficos

- Enquanto GKS e PHIGS estavam sendo desenvolvidos, as *workstations* gráficas da Silicon Graphics tornaram-se muito populares.
- Essas workstations traziam uma conjunto de rotinas chamado: GL (Graphics Library)
- Esse conjunto teve uma aceitação muito rápida pela comunidade e tornou-se um padrão de fato.

Padrões Gráficos

- As rotinas do padrão GL, que haviam sido projetadas para alto desempenho e renderização em tempo real foram estendida para outros hardwares.
- Isso resultou, no início da década de 90, em uma versão desse padrão independente de hardware: o OpenGL.

Padrões Gráficos

- O OpenGL é mantido e atualizado pelo OpenGL Architecture Review Board, um consórcio de empresas relacionadas a aplicações gráficas, entre elas a Silicon Graphics.
- OpenGL foi projetado para processamento eficiente de aplicações 3D, mas pode ser utilizado para 2D, como um caso especial onde a coordenada z assume valor zero.

Padrões Gráficos

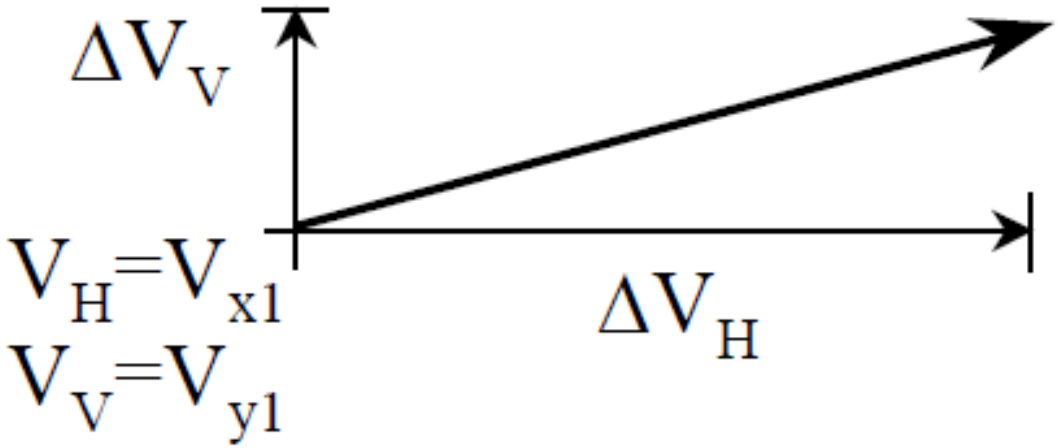
- Outros Padrões:
  - OpenInventor
  - VRML (Virtual Reality Modeling Language)
  - Java2D
  - Java3D
  - Renderman Interface

Primitivas Gráficas

- **Definição:** uma primitiva de saída é uma estrutura geométrica básica a partir da qual podem ser desenvolvidas estruturas mais complexas.
  - Exemplos: ponto, linha, círculo, curva, etc.

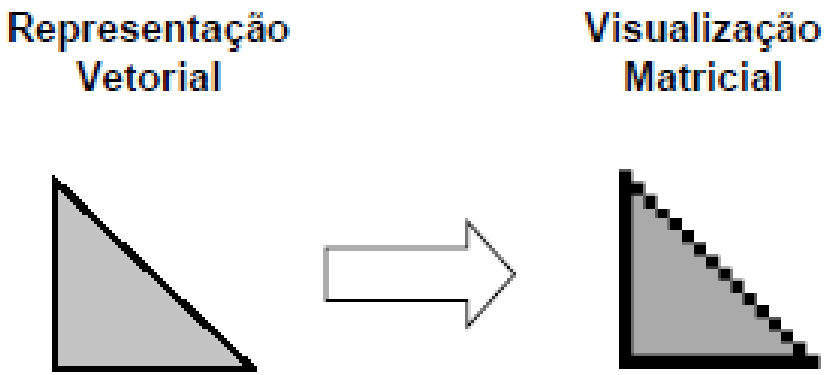
Terminais Vetoriais

- Variação linear das deflexões horizontal e vertical proporcionais às variações em X e Y.



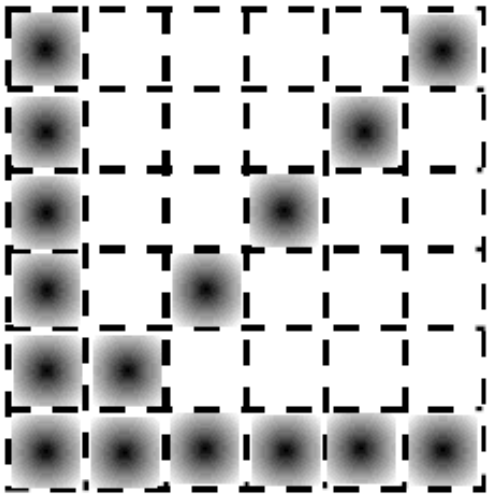
Terminais Matriciais

- Converter definições geométricas para *pixels.*
- Preenchimento do conjunto de pixels que melhor aproxima a figura desejada.



Primitivas Gráficas

- Ponto
  - Primitiva básica dos Terminais Matriciais
  - Base para todas as figuras

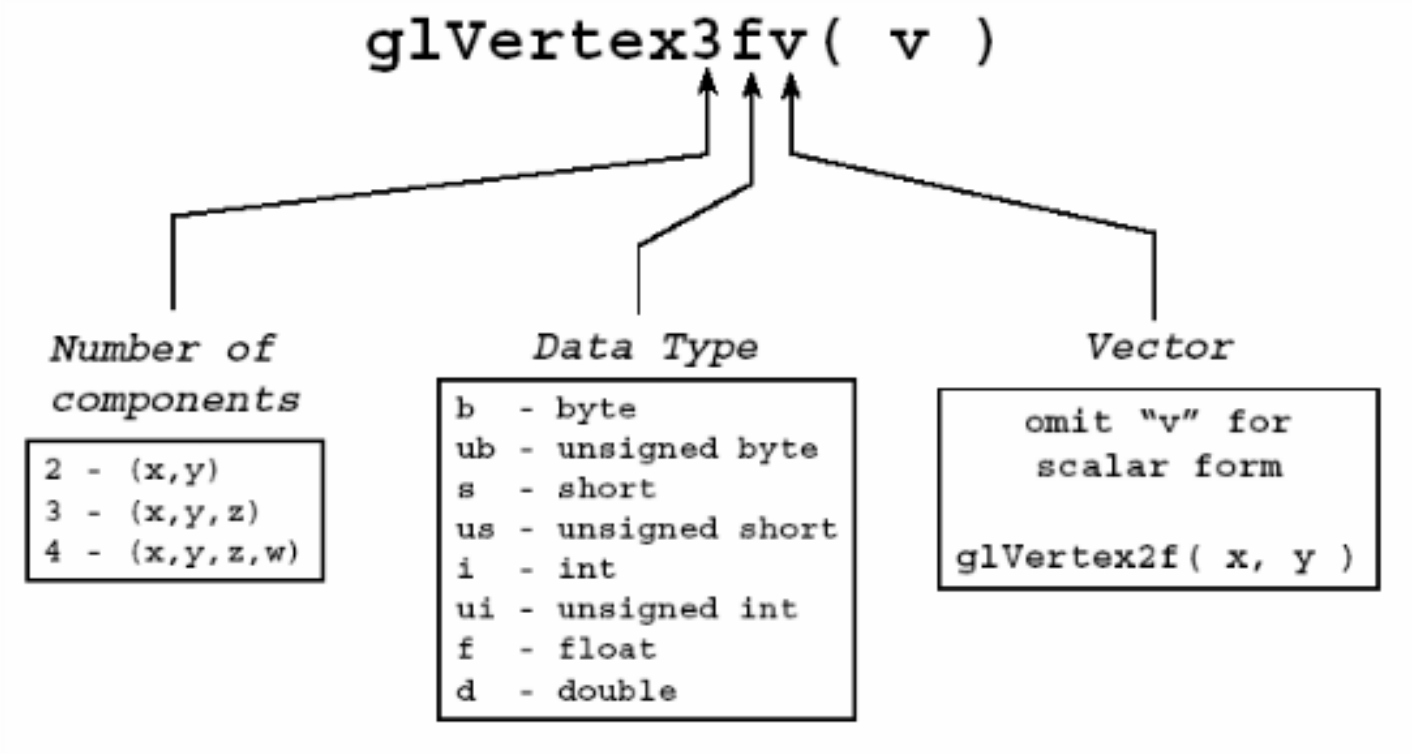


Primitivas Gráficas: Linha

- Linha: composição de pontos
- Problema: determinar os pontos que mais se aproximam da linha desejada.

- Linhas Horizontais:
  - x variável, y constante
- Linhas Verticais:
  - x constante, variação em y

OpenGL - Início



OpenGL - Início

```c
#include <GL/glut.h>
#include <stdlib.h>
#include "util.h"
#include "linhas.h"

void createEnv(void);
void mainDisplay(void);
void display(void);
void keyboard(unsigned char key, int x, int y);

/** Função Principal **/
int main(int argc, char** argv){
  glutInit(&argc, argv);                          //inicializa glut
  createEnv();                                    //cria ambiente
  glutDisplayFunc(mainDisplay);                   //funcao que sera redesenhada pelo GLUT
  glutKeyboardFunc(keyboard);                     //aguarda ESC do teclado
  glutMainLoop();                                 // mostra todas as janelas criadas
  return 0;
}

/** Criação do Ambiente **/
void createEnv () {
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);   // especifica o uso de cores e buffers
  glutInitWindowSize (WIDTH, HEIGHT);             // especifica as dimensoes da janela
  glutInitWindowPosition (0, 0);                  // especifica aonde a janela aparece na tela
  glutCreateWindow ("Computação gráfica");        // cria a janela
  glClearColor(1.0, 1.0, 1.0, 1.0);               // cor de fundo
  glOrtho (0, WIDTH, 0, HEIGHT, -1 ,1);          // modo de projecao ortogonal
}
```

(...)

OpenGL - Início

(...)

```
/** Função que será renderizada **/
void mainDisplay(void){
  glClear(GL_COLOR_BUFFER_BIT);                    // limpa a janela
  glColor3f (0.0, 0.0, 0.0);                       // cor
  glBegin(GL_POINTS);                              // primitiva utilizada
  display();                                       // => Função reponsável por gerar as primitivas
  glEnd();                                         // finaliza
  glFlush();                                       // processa todas as rotinas
}

void display () {
  int i=0;

  glVertex2i(10,10);
  glVertex2i(20,20);

  /*
  horizontalLine(20,220,50);
  */
}

/** Interação com dispositivos de Entrada **/
void keyboard(unsigned char key, int x, int y){
  switch (key) {
  case 27:                                         // tecla Esc (encerra o programa)
    exit(0);
    break;
  }
}
```

Exercício

- **Objetivo:** estabelecer primeiro contato com a biblioteca gráfica;
- **Descrição:** desenhar o esqueleto de um tabuleiro de xadrez na tela, com 8 linhas e 8 colunas.

- **Informações adicionais:** o tamanho da tela é X x Y.

Equação da Reta

- Objetivo: traçar um segmento de reta de (x_o, y_o) a (x_{end}, y_{end}).

Equação da Reta: $y = m \cdot x + b$

- Como utilizar a equação da reta para traçar o segmento desejado?

Equação da Reta

Equação da Reta: $y = m \cdot x + b$

$$m = \frac{y_{end} - y_0}{x_{end} - x_0}$$

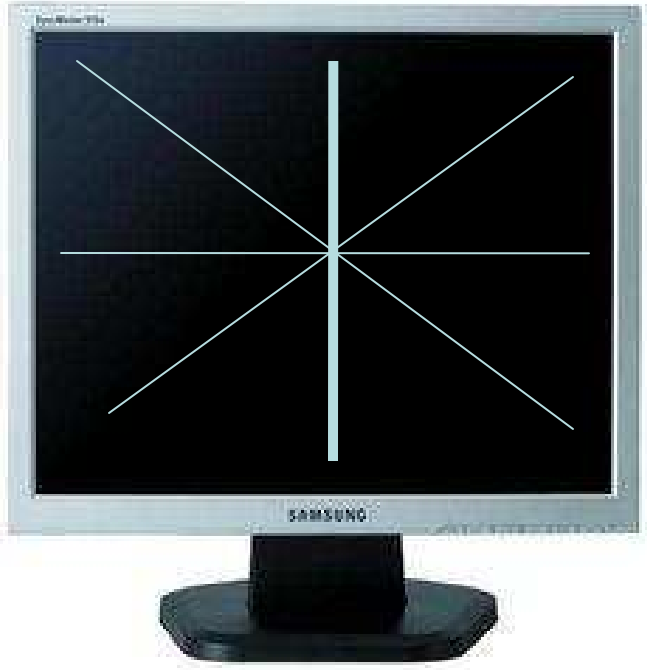
$$b = y_0 - m \cdot x_0$$

Calculados m e b, qual o algoritmo para traçar o segmento de reta?

E quando x_{end},= x_o ?

Exercício

- **Objetivo:** utilizar a equação da reta para construir uma primitiva de linha;
- **Descrição:** desenhar 8 segmentos de reta com origem no centro da tela e fim nas bordas, usando o algoritmo baseado na equação da reta.
- Atenção às retas verticais!



Exercício – Equação da Reta

```
void equacaoReta (int x0, int y0, int xend, int yend) {
  float m,b,deltax,deltay,yf;
  int i,yi,x1,x2;

  if (xend>x0) {
      x1 = x0;
      x2 = xend;
  } else {
      x1 = xend;
      x2 = x0;
  }

  deltay = (yend - y0);
  deltax = (xend - x0);

  if (deltax==0) {
     m=0;
  } else {
     m = deltay / deltax;
  }
  b = y0 - (m * x0);

  for (i=x1;i<=x2;i++){
      yf = (m * i) + b;
      yi = round(yf);
      glVertex2f(i,yi);
  }
}
```

Algoritmo DDA

- Desempenho é um requisito importante para aplicações de computação gráfica.
- Qual o problema em utilizar a equação da reta?

Algoritmo DDA

- Algoritmo incremental: baseado no ponto atual, é possível calcular o ponto seguinte.
- Vantagem: elimina as operações de multiplicação utilizadas na equação da reta.

Algoritmo DDA

- Cálculos sucessivos baseados no ponto anterior:

$$y_{k+1} = y_k + m$$

$$x_{k+1} = x_k + 1/m$$

$$m = \Delta y/\Delta x$$

Algoritmo DDA

Algoritmo 1 Implementa o algoritmo DDA

Tamanho = $abs(x_2 - x_1)$

$\Delta x = (x_2 - x_1)/Tamanho$
$\Delta y = (y_2 - y_1)/Tamanho$
$i = 1$
while $i \leq Tamanho$ **do**
desenhaPonto(Round(x), Round(y)) {**Round**: valor arredondado de um dado número real. Inteiro(-8.6) = -9; Inteiro(-8.4) = -8}
$x = x + \Delta x$
$y = y + \Delta y$
$i = i + 1$
end while

Exercício

- **Objetivo:** sedimentar conhecimentos sobre o algoritmo de linhas DDA.
- **Descrição:** analise o algoritmo DDA, e descubra qual o problema do algoritmo apresentado. Feito isso, implemente o algoritmo correto (apresentado a seguir), e refaça o exercício anterior usando esse algoritmo.

Algoritmo DDA

Algoritmo 1 Implementa o algoritmo DDA

if $abs(x_2 - x_1) \geq abs(y_2 - y_1)$ **then**
  Tamanho = $abs(x_2 - x_1)$
else
  Tamanho = $abs(y_2 - y_1)$
end if
{seleciona o maior dos valores entre Δx e Δy como unidade rasterização}
$\Delta x = (x_2 - x_1)/Tamanho$
$\Delta y = (y_2 - y_1)/Tamanho$
$i = 1$
while $i \leq Tamanho$ **do**
  desenhaPonto(Round(x), Round(y)) {**Round**: valor arredondado de um dado número real. Inteiro(-8.6) = -9; Inteiro(-8.4) = -8}
  $x = x + \Delta x$
  $y = y + \Delta y$
  $i = i + 1$
end while

Algoritmo Bresenham

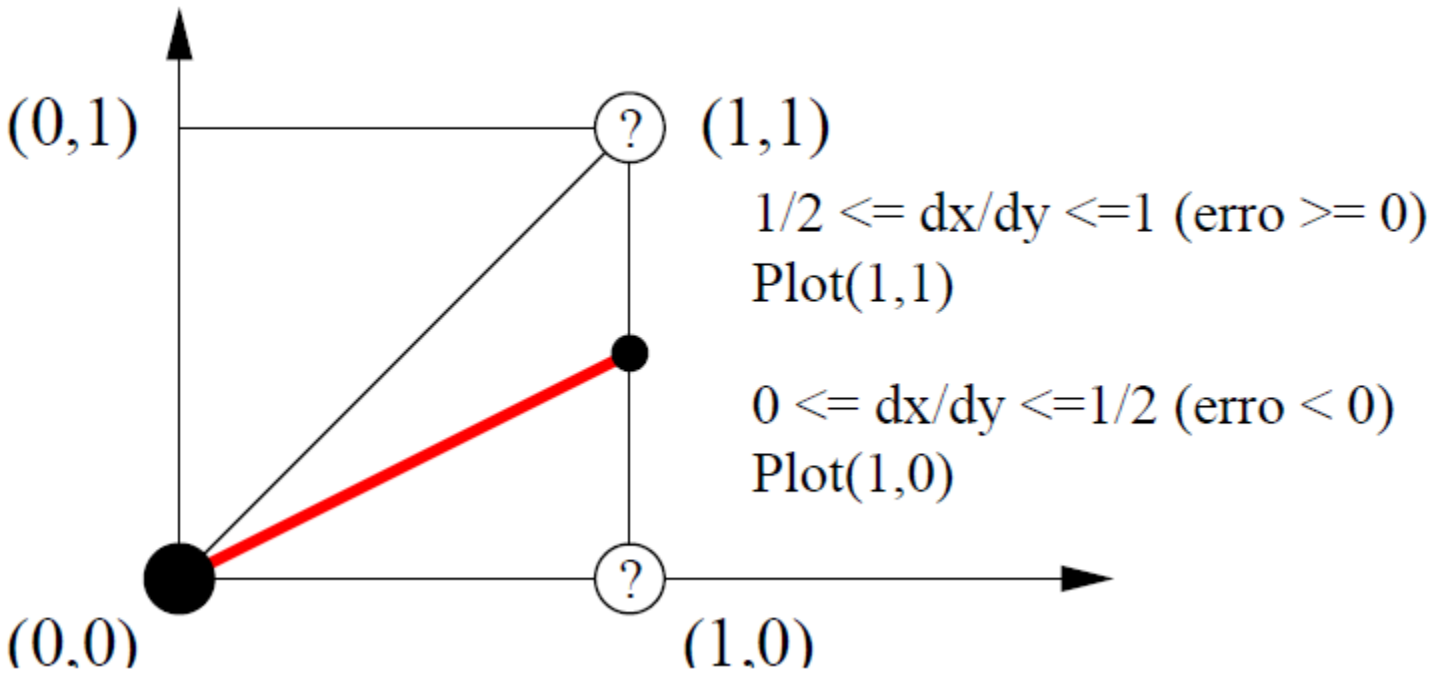
- Desempenho é um requisito importante para aplicações de computação gráfica!
- Qual o problema do algoritmo DDA?
  - Operações com ponto flutuante e arredondamentos.
  - Operações com inteiros são muito mais eficientes.
- Bresenham: implementação com ponto flutuante ou inteiros.

Algoritmo Bresenham

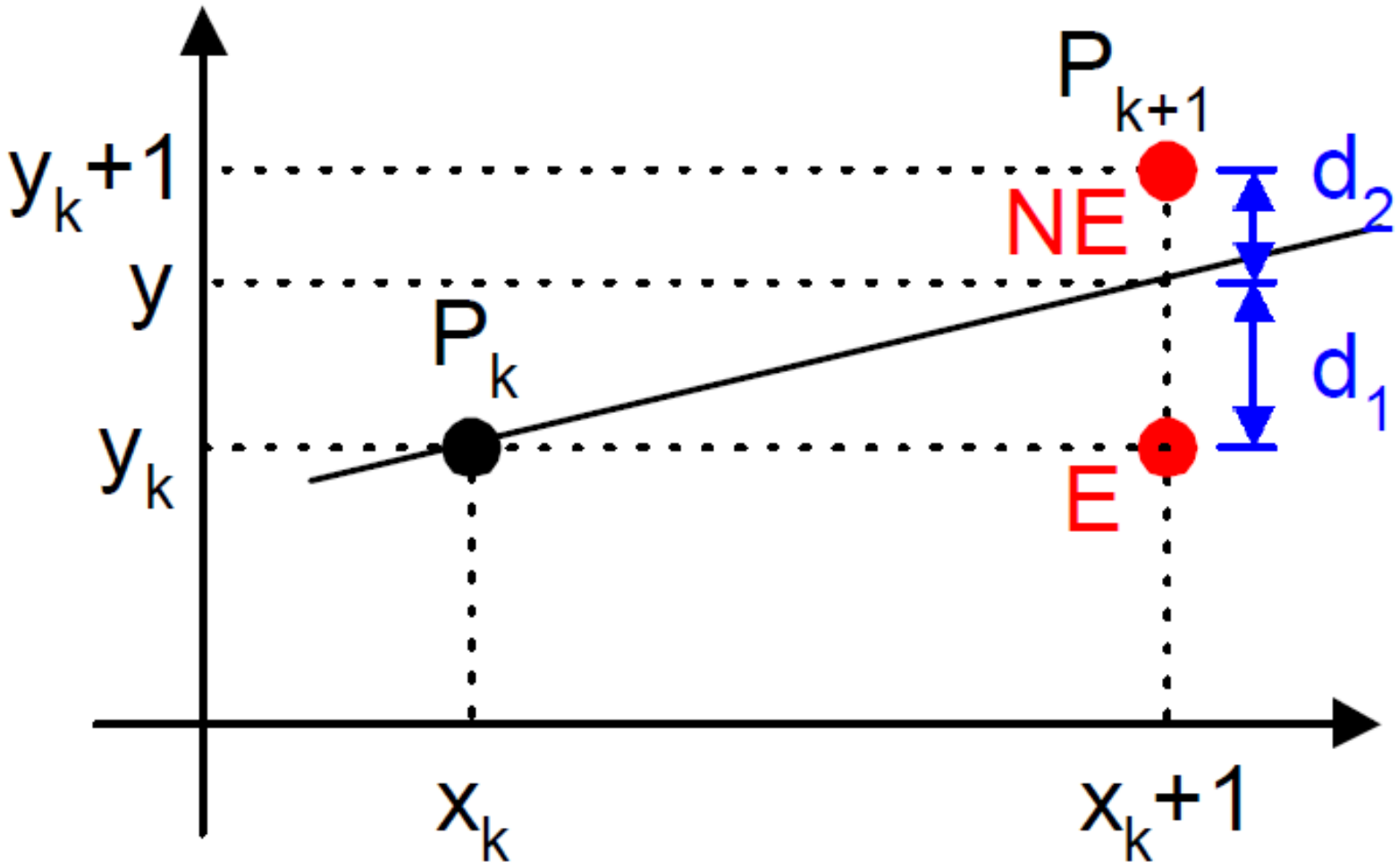
- O algoritmo procura selecionar as posições discretas ótimas para representar o segmento de reta.
- O algoritmo sempre incrementa de uma unidade a dependendo da inclinação da reta.
- O incremento na outra variável (de zero ou um) é determinado através da análise da distância entre a posição da reta real e as posições discretas mais próximas.
- Esta distância é chamada de erro.

Algoritmo Bresenham

- Ideia básica:



Algoritmo Bresenham



Algoritmo Bresenham

- Para inteiros:
- Considerando 0<=m<=1

$$y = m(x_k + 1) + b$$

- Calcula-se as distâncias d_1 e d_2:

$$d_1 = y - y_k \qquad d_1 = m(x_k + 1) + b - y_k$$

$$d_2 = (y_k + 1) - y \qquad d_2 = y_k + 1 - m(x_k + 1) - b$$

Algoritmo Bresenham

- Calcula-se a diferença entre as distâncias:

$$d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1$$

- Pode-se obter um parâmetro de decisão para o ponto na posição p_k:

$$p_k = \Delta x.(d_1 - d_2)$$

$$p_k = \Delta x.(2m(x_k + 1) - 2y_k + 2b - 1)$$

$$m = \Delta y/\Delta x$$

$$p_k = 2.\Delta y.x_k - 2.\Delta x.y_k + c$$

Algoritmo Bresenham - PF

Algoritmo 2 Implementa o algoritmo de Bresenham para retas

$x = x_1$
$y = y_1$
$\Delta x = x_2 - x_1$
$\Delta y = y_2 - y_1$
$m = \Delta y / \Delta x$
$e = m - 1/2$
for $i = 1$ to Δx **do**
    desenhaPonto(x,y)
    while $e \geq 0$ **do**
        $y = y + 1$
        $e = e - 1$
    end while
    $x = x + 1$
    $e = e + m$
end for

Algoritmo Bresenham

Algoritmo 3 Implementa o algoritmo inteiro de Bresenham para retas

```
...
ē = 2Δy − Δx
for i = 1 to Δx do
    desenhaPonto(x,y)
    while ē ≥ 0 do
        y = y + 1
        ē = ē − 2Δx
    end while
    x = x + 1
    ē = ē + 2Δy
end for
```

Algoritmo Bresenham generalizado

Algoritmo 4 Implementa o algoritmo generalizado de Bresenham para retas

$x = x_1$
$y = y_1$
$\Delta x = abs(x_2 - x_1)$
$\Delta y = abs(y_2 - y_1)$
$s1 = Sinal(x_2 - x_1)$
$s2 = Sinal(y_2 - y_1)$
if $\Delta y > \Delta x$ **then**
  Temp = Δx
  Δx = Δy
  Δy = Temp
  Troca = 1
else
  Troca = 0
end if
$\overline{e} = 2\Delta y - \Delta x$
for $i = 1$ to Δx **do**
  desenhaPonto(x,y)
  while $\overline{e} \geq 0$ **do**
    if Troca = 1 **then**
      $x = x + s1$
    else
      $y = y + s2$
    end if
    $\overline{e} = \overline{e} - 2\Delta x$
  end while
  if Troca = 1 **then**
    $y = y + s2$
  else
    $x = x + s1$
  end if
  $\overline{e} = \overline{e} + 2\Delta y$
end for

Exercício

- **Objetivo:** aplicar os conceitos aprendidos sobre o algoritmo de Bresenham.
- **Descrição:** implemente o algoritmo de Bresenham para o primeiro quadrante e teste para algumas retas que estejam nesse quadrante.

Exercício de Retas - Final

- **Objetivo:** aplicar os conceitos aprendidos em uma aplicação prática.
- **Descrição:** faça uma proteção de tela utilizando a primitiva de reta.

Exercício Final

```
void display () {
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 0.0, 0.0);
  glBegin(GL_POINTS);
```

(...)

```
  glEnd();
  glutSwapBuffers();
}
```

```
/** Função Principal **/
int main(int argc, char** argv){
  glutInit(&argc, argv);
  createEnv();
  glutDisplayFunc(display);
  glutTimerFunc(DELAY, timer, 1);
  glutMainLoop();
  return 0;
}

/** Criação do Ambiente **/
void createEnv () {
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (WIDTH, HEIGHT);
  glutInitWindowPosition (0, 0);
  glutCreateWindow ("Computação gráfica");
  glClearColor(1.0, 1.0, 1.0, 1.0);
  glOrtho (0, WIDTH, 0, HEIGHT, -1 ,1);
}

void timer(int value) {
  updateValues ();
  glutPostRedisplay();
  glutTimerFunc(33,timer, 1);
}
```

Referências

- HEARN, D. e BAKER, PAULINE - Computer Graphics with OpenGL, Prentice-Hall, 2004.
- FRANCIS S. HILL JR. - Computer Graphics, Macmillan Publishing Company, 1990.