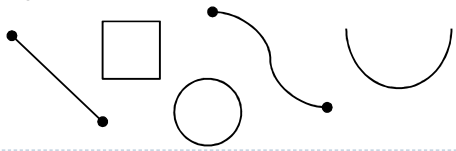
Primitivas Gráficas 2D

GBC204 – Computação Gráfica
Prof. Dr. rer. nat. Daniel Duarte Abdala

O que é uma Primitiva Gráfica?

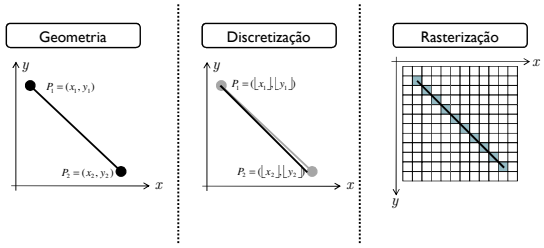
- Representação discreta em grade de um elemento geométrico fundamental, e.g. Ponto, Linha, Retângulo, Círculo, etc.
- Outra Definição (Formal): *"Aproximar primitivas matemáticas (ideais) em termos de vértices (pixels) a serem plotados em uma malha discreta e finita de elementos de figura finitos."*



Primitiva Gráfica - Ponto

- Postulado (aceito a priori) fundamental da geometria;
- Ponto – Localização única no espaço;
  - Não possui dimenção;
  - Elemento básico a partir do qual todos os outros objetos da geometria são construídos;
  - Representado gráficamente utilizando usualmente um pequeno círculo ou "marca";
  - **Postulado da Existência**: *"Numa reta, bem como fora dela há infinitos pontos. Num plano há infinitos pontos."*
- Como o computador é uma máquina discreta, aproximações devem ser feitas:
  - Ponto é aproximado para um pixel ou pel (picture element);
  - Menor unidade representável em um display gráfico;
  - Elemento fundamental a partir do qual todas as demais primitivas gráficas são construídas.
  - **Postulado da Existência (CG)**: *"Numa reta, bem como fora dela há um número finito de pels. Num plano há um número finito de pels."*

Processo para Representação Geométrica em Computador



Representação de Pontos

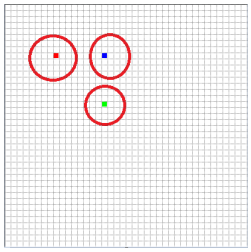
- Pontos são realizados sob a forma de pixels, ou mais especificamente pels;
- Basicamente, altera-se o valor do respectivo elemento no framebuffer;
- Programacionalmente:

```
setPixel(x,y);
```

```
screen.set_at(x,y,cor)
```

Exemplo (python)

```
import sys,pygame
from pygame import gfxdraw

pygame.init()
screen = pygame.display.set_mode((50,50))
screen.fill((255,255,255))

screen.set_at((10,10),(255,0,0))
screen.set_at((20,20),(0,255,0))
screen.set_at((20,10),(0,0,255))
pygame.display.flip()

while 1:
  for event in pygame.event.get():
      if event.type == pygame.QUIT: sys.exit()
```



Primitiva Gráfica - Linhas

- **Postulado da Determinação:** *"Dois pontos distintos delimitam uma única reta que passa por eles."*;
- Computacionalmente lidamos com segmentos de retas, ou seja, apenas a parte da reta delimitada pelo ponto inicial e final;
- **PROBLEMA:** converter a representação matemática de um segmento de reta definida por infinitos pontos entre os pontos inicial e final para uma representação discreta alinhada a grade de representação de finitos pels.

Matematicamente ...

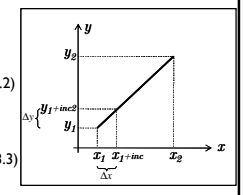
Equação da Reta: $y = mx + b$ (eq. 3.1)

Isolando o termo "m":

$$y_1 = mx_1 + b$$
$$y_2 = mx_2 + b$$
$$y_2 - mx_2 = y_1 - mx_1$$
$$mx_1 - mx_2 = y_2 - y_1$$
$$m(x_2 - x_1) = (y_2 - y_1)$$

$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{(eq. 3.2)}$$

$$b = y_1 - mx_1 \quad \text{(eq. 3.3)}$$



A partir de qualquer valor de x, o incremento discreto de y pode ser calculado:

$$\Delta y = m\Delta x \quad \text{(eq. 3.4)}$$

similarmente:

$$\Delta x = \frac{\Delta y}{m} \quad \text{(eq. 3.5)}$$

Algoritmo DDA – Digital Differential Analyzer

- Algoritmo de conversão de linhas que se baseia no cálculo de Δx ou Δy;
- A linnha é amostrada em intervalos unitários incrementais em uma coordenada (x ou y) e então se determina o valor inteiro correspondente para a outra coordenada;
- Considere um linha com inclinação positiva (m>0);
- Se m ≥1 → amostragem em x (Δx=1) e os valores de y serão computados utilizando a expressão a seguir:

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

onde k indica a próxima iteração discreta a partir do Ponto inicial.

DDA – C (pseudocódigo)

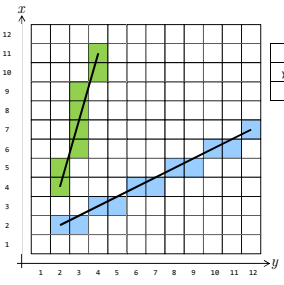
```
#define ROUND(a) ((int)(a+0.5))

void linhaDDA(int xa, int ya, int xb, int yb)
{
  int dx = xb-xa, dy = yb-ya, steps, k;
  float xIncrement, yIncrement, x = xa, y = ya;

  if (abs(dx)>abs(dy)) steps = abs(dx);
  else steps = abs(dy);
  xIncrement = dx/(float)steps;
  yIncrement = dy/(float)steps;

  setPixel(ROUND(x), ROUND(y));
  for(k=0; k<steps;k++){
    x += xIncrement;
    y += yIncrement;
    setPixel(ROUND(x), ROUND(y));
  }
}
```

DDA - Python

```
import sys,pygame
from pygame import gfxdraw

pygame.init()
screen = pygame.display.set_mode((400,400))
screen.fill((0,0,0))
pygame.display.flip()
White = (255,255,255)
def ROUND(n):
  return int(n+0.5)
```

```
def dda(x1,y1,x2,y2):
  x,y = x1,y1
  length = (x2-x1)
  if length <= (y2-y1)
    length = y2 - y1
  dx = (x2-x1)/float(length)
  dy = (y2-y1)/float(length)
  screen.set_at((ROUND(x),ROUND(y)),white)

  for i in range(length):
    x += dx
    y += dy
    screen.set_at((ROUND(x),ROUND(y)),white)
  pygame.display.flip()

dda(10,10,50,50)

while 1:
  for event in pygame.event.get():
    if event.type == pygame.QUIT: sys.exit()
```

Exemplo



$dda(2,4,4,11)$

$dx = 4-2 = 2, dy = 11-4 = 7, steps = 7$

x	4	5	6	7	8	9	10	11
yinc	-	2.285	2.571	2.857	3.143	3.428	3.714	3.999
y	2	2	3	3	3	3	4	4

$dda(2,2,12,7)$

$dx = 12-2 = 10, dy = 7-2 = 5, steps = 5$

x	2	3	4	5	6	7	8	9	10	11	12
yinc	-	2.5	3	3.5	4	4.5	5	5.5	6	6.5	7
y	2	2	3	3	4	4	5	5	6	6	7

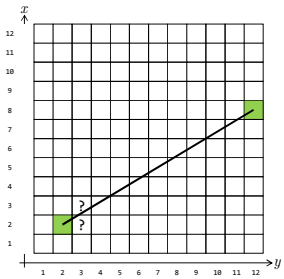
Problemas com o algoritmo DDA

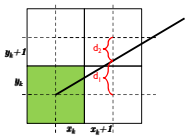
- (↑) O algoritmo dda elimina a necessidade da multiplicação na eq. 3.1;
- (↓) A acumulação de erros de arredondamento nas adições sucessivas em ponto flutuante podem levar ao cálculo de posições de pels que se distanciam do caminho original da linha em caso de longos segmentos de retas;
- (↓) Operação de arredondamento e aritmética de ponto flutuante são custosas.

Algoritrmo de Bresenham

- Bresenham [BRE65] propôs um algoritmo consideravelmente mais preciso e eficiente para rasterização de linhas;
- O algoritmo ainda apresenta a vantagem de ser adaptado para a rasterização de círculos;
- Inicialmente considera-se o caso de linhas com inclinação positiva menor que 1 ($0 \le m \le 1$);
- Neste caso, inicia-se no ponto terminador mais a esquerda $p_1(x_1,y_1)$ e incrementa-se a coordenada x atualizando no frame buffer o pel que possua o valor de y mais próximo da linha real;
- Assumindo que o pel(x_k,y_k) esteja no frame buffer, então deve-se decidir qual o pel a atualizar no frame buffer na coluna x_k+1;
- Há neste caso, duas possibilidades: (x_k+1,y_k) e (x_k+1,y_k+1);

Bresenham Motivação





Matematicamente...

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

então:

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

e:

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

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

Matematicamente...

Um parâmetro decisório pode ser alcançado pela manipulação da equação acima de modo que apenas cálculos inteiros estejam envolvidos.

$m = \Delta x / \Delta y$ Onde Δx e Δy são as separações vertical e horizontal respectivamente dos pontos terminadores $p_1(x_1,y_1)$ e $p_2(x_2,y_2)$ do segmento de reta.

$$\Delta x = x_2 - x_1 \qquad \Delta y = y_2 - y_1$$

$$c = 2\Delta y + \Delta x(2b-1)$$

Note que c é independente da posição do próximo pel e consequentemente, pode ser eliminado.

$$p_k = \Delta x(d_1 - d_2)$$
$$p_k = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

Matematicamente...

No passo k+1 o parâmetro de decisão avaliado pela equação anterior será:

$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$$

Subtraindo as duas equações, obtêm-se:

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

Como $x_{k+1}=x_k+1$, têm-se:

$$p_{k+1} - p_k = 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

Note que o termo $(y_{k+1}-y_k)$ será sempre 0 ou 1;

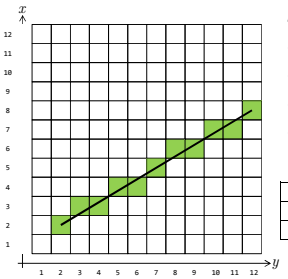
O primeiro parâmetro decisório p_0 avaliado na posição (x_1,y_1) e m avaliado como $\Delta y/\Delta x$:

$$p_0 = 2\Delta y - \Delta x$$

Bresenham Python (1º quadrante)

```
import sys,pygame
from pygame import gfxdraw

pygame.init()
screen = pygame.display.set_mode((400,400))
screen.fill((0,0,0))
pygame.display.flip()

white=(255,255,255)

#Esta funcao funciona apenas
#para o primeiro quadrante
def bresenham(x1,y1,x2,y2):

    #carrega no fb o pixel (x1,y1)
    screen.set_at((x1,y1),white)
    #computa os deltas necessarios
    dx    = x2 - x1
    dy    = y2 - y1
    dy2   = 2*dy
    dydx2 = dy2 - 2*dx
    pant  = dy2 - dx
    x = x1
    y = y1
    for i in range(dx):
        if pant < 0:
            screen.set_at((x+1,y),white)
            pant = pant + dy2
        else:

            screen.set_at((x+1,y+1),white)
            pant = pant + dydx2
            y += 1;
        x += 1;
    pygame.display.flip()

bresenham(10,10,50,50)

while 1:
    for event in pygame.event.get():
            if event.type == pygame.QUIT: sys.exit()
```

Exemplo Bresenham



$$bresenham(2,2,12,8)$$
$$dx = 12-2 = 10$$
$$dy = 8-2 = 6$$
$$dy2 = 2\times 6 = 12$$
$$dydx2 = 12-20 = -8$$
$$p_0 = 12-10 = 2$$

x	2	3	4	5	6	7	8	9	10	11	12
P_{x+1}	2	-6	6	-2	10	2	-6	6	-2	10	–
y	2	3	3	4	4	5	6	6	7	7	8

Importância do Desempenho (Linhas)

- Em resumo, linhas são fáceis de se calcular e plotar:
  - Para decidir exatamente onde um ponto recai na latice de plotagem basta que se resolva a equação da reta;
  - Resultados reais podem ser discretizados por operação de arredondamento;
- No entanto:
  - Cálculos em ponto flutuante são caros;
  - Arredondamentos são caros;
  - Multiplicações (não por potências de 2) e divisões são caras;
- Em última instância:
  - A saída é bidimencional (monitores e displays);
  - Linhas e pontos são as únicas primitivas realmente plotadas;
  - Centenas de milhares de linhas podem ser renderizadas por frame;
- Requisito Fundamental: **Minimizar o custo computacional para processamento de linhas!**

Em OpenGL (Pontos)...

- A primitiva gráfica mais fundamental prevista em OpenGL é o ponto;
- Um ponto é representado por um vértice (vertex);
- Por definição, pontos são objetos de área zero. Consequentemente, para se visualizar pontos individuais, deve-se especificar seu tamanho;

```
glBegin(GL_POINTS)
   glVertex2f(1.0, 1.0);
   glVertex2f(2.0, 1.0);
   glVertex2f(2.0, 2.0);
glEnd();

glPointSize(n);
glEnable(GL_POINT_SMOOTH);
glHint(GL_POINT_SMOOTH_HINT,GL_NICEST);
glEnable(GL_BLEND);
glBrendFunc(GL_SRC_ALPHA,
   GL_ONE_MINUS_SRC_ALPHA);
```

Em OpenGL (Linhas)...

- Linhas são especificadas utilizando a variável do ambiente GL_LINES;
- Dois pontos são necessários para se desenhar uma linha;
- Elas devem ser especificadas em coordenadas 3D;
- Para restringi-las ao plano 2D basta que a coordenada z seja zerada;

```
glBegin(GL_Lines)
   glVertex3f(1.0, 5.0,  0.0);
   glVertex3f(1.0, 10.0, 0.0);
   glVertex3f(1.0, 5.0 , 0.0);
   glVertex3f(1.0, 10.0, 5.0);
glEnd();
```

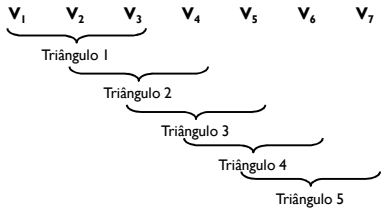
Em OpenGL (Poligonos)...

- Qualquer polígono pode ser desenhado em OpenGL;
- Eles são "quebrados" em triângulos para plotagem;
- Por definição, vértices em um polígono são desenhados no ***sentido contrário*** ao relógio;
- OpenGL possui primitivas para desenho de triângulos, quadriláteros e polígonos convexos. Normalmente estes são desenhados como ***sólidos preenchidos***;
- Frequentemente deseja-se combinar múltiplos triângulos para se formar uma superfície complexa e "contínua";
- Triângulos são preferidos em computação gráfica pois são ***garantidamente planares***!

```
glBegin(GL_TRIANGLES)
   glVertex3f(1.0, 5.0,  0.0);
   glVertex3f(1.0, 10.0, 0.0);
   glVertex3f(1.0, 5.0 , 0.0);
glEnd();

glLineWidth(n);
glEnable(GL_LINE_SMOOTH);
glHint(GL_LINE_SMOOTH_HINT,GL_NICEST);
glEnable(GL_BLEND);
glBrendFunc(GL_SRC_ALPHA,
   GL_ONE_MINUS_SRC_ALPHA);
```

Faixa (Strip) de Triângulos

- Repetir vértices comuns é laborioso e desnecessário;
- Uma faixa de triângulos usa os dois últimos pontos para especificar o triângulo;



Faixa (Strip) de Triângulos

- Desta forma, a relação entre número de vértices e triângulos é dada pela relação:

$$nt = nv - 2, \forall nv > 3$$

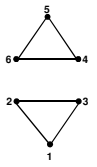
Onde:
nt : número de triângulos
nv : número de vértices

- No caso de GL_TRIANGLES, a relação é dada por:

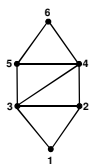
$$nt = nv/3, \forall i \, / \, nv = 3i \wedge i \in \mathbb{N}$$
$$nv = 3nt, \forall nt > 1 \wedge nt \in N$$

- Uma forma alternativa para se especificar superfícies seria utilizar GL_TRIANGLE_FAN

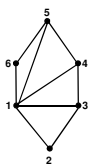
Formas de Renderização de Triângulos



GL_TRIANGLES



GL_TRIANGLE_STRIP

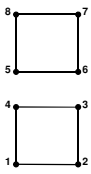


GL_TRIANGLE_FAN

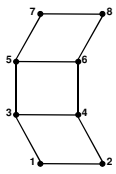
Em OpenGL (Quadirláteros)...

- OpenGL permite que quadriláteros sejam desenhados;
- Na realidade ele os quebra em dois triângulos;
- Processo similar ao adotado para triângulos simples;

```
glBegin(GL_QUADS)
  glVertex3f(1.0, 1.0,  0.0);
  glVertex3f(1.0, 5.0,  0.0);
  glVertex3f(5.0, 5.0 , 0.0);
  glVertex3f(5.0, 1.0 , 0.0);
glEnd();
```



GL_QUAD



GL_QUAD_STRIP

Primitivas Gráficas – Círculos

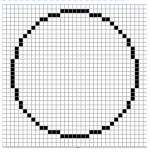
- Um círculo é definido como o conjunto de pontos, todos localizados a uma distância ***r*** de um ponto central ***c***;
- Esta relação de distância pode ser expressa pelo teorema de pitágoras, como segue:

$$(x - c_x)^2 + (y - c_y)^2 = r^2$$

- Para identificar todos os pontos do círculo, usando o eixo x como referência, tem-se:

$$y = c_y \pm \sqrt{r^2 - (c_x - x)^2}$$

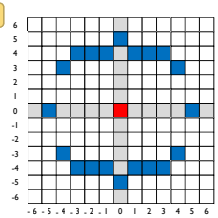
- No entanto, claramente este não é um método eficiente para geração dos pontyos do círculo;



Primitivas Gráficas – Círculos

- Problemas:
  - O método aritmético tradicional requer considerável poder computacional, em cada passo;
  - O Espaço entre pixels a serem plotados não é regular;

Círculo(0,0,5)

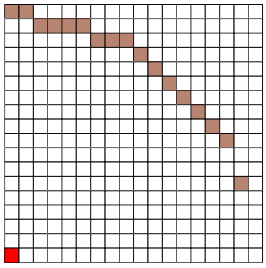


$$y = \pm\left\lfloor\sqrt{r^2 - x^2}\right\rfloor$$

x	1	2	3	4	5
$\lfloor\sqrt{25 - x^2}\rfloor$	4.89	4.58	4	3	E
y	4	4	4	3	

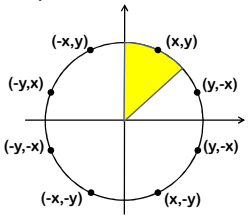
Outro Exemplo

Círculo(0,0,17)



Simetria de Octetos

- Devido a simetria do círculo, não é necessário que as coordenadas de todos os pixels do círculo sejam calculadas;
- Basta que apenas 1/8 do círculo seja calculado e os demais 7/8 dos pixels podem ser calculados de maneira não custosa unicamente por espelhamentos.

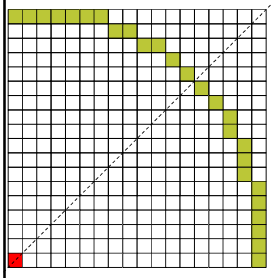


Coordenadas Polares

- Uma forma de se abordar o problema do espaçamento irregular seria calcular as coordenadas dos pontos da borda circular usando para tal coordenadas polares:

$$\begin{cases} x = c_x + r\cos\theta \\ y = c_y + r\sin\theta \end{cases} \qquad \theta = 1/r$$

- Este método resolve parcialmente o problema de espaçamento;
- No entanto também é caro computacionalmente:
  - O custo pode ser atenuado, ajustando o tamanho do passo, que em última instância, refere-se ao ângulo θ de amostragem.



x	16.99	16.88	16.73	16.5	16.27	15.95	15.57	15.15	14.67
y	0.99	1.99	2.98	3.96	4.92	5.88	6.8	7.7	8.58
θ	1/r	2/r	3/r	4/r	5/r	6/r	7/r	8/r	9/r

x	14.14	13.56	12.93	12.26	11.55	10.8	10.01	9.18	8.33
y	9.43	10.24	11.02	11.76	12.47	13.13	13.74	14.3	14.82
θ	10/r	11/r	12/r	13/r	14/r	15/r	16/r	17/r	18/r

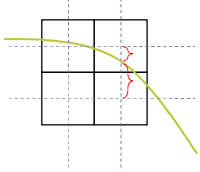
x	7.44	6.53	5.59	4.64	3.67	2.69	1.7	0.7
y	15.28	15.69	16.05	16.35	16.59	16.78	16.91	16.98
θ	19/r	20/r	21/r	22/r	23/r	24/r	25/r	26/r

Problemas com o Método de Coord. Polares

- Mesmo computando apenas 1/8 de todos os pontos que comporão a curva, o algoritmo ainda assim é muito caro;
  - *, /, sin e cos
  - Passos grandes gerarão buracos na rasterização;
  - Passos pequenos farão com q pontos sejam computados mais de uma vez;
  - A regra de passo definido como 1/r é uma aproximação;
  - Em alguns casos, dependendo do tamanho do passo, o mesmo ponto pode ser computado mais de uma vez;

Algoritmo do Círculo Baseado na Modelagem do Ponto Médio

- Midpoint Circle Algorithm
  - Baseado no algoritmo de Breseham para linhas
    - Algoritmo guloso – a cada passo deve-se decidir entre um de dois possíveis pixels;
  - Utiliza a simetria de octetos
  - Extremamente rápido!

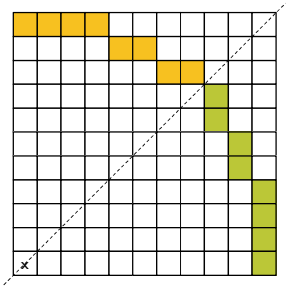


Midpoint Circle Algorithm - Bresenham

```
void midpointCircle(int r, int value)
{
  int x = 0;
  int y = r;
  int d = 1 - r;

  plotaOcteto (x, y, value);

  while( y > x )
  {
    if( d < 0 )            //selecione E
      d += ( 2 * x ) + 3;
    else {                 //selecione SE
      d += 2 * ( x - y ) + 5;
      y--;
    }
    x++;
    plotaOcteto(x, y, value);
  }
}
```

```
void plotaOcteto(int x, int y, int value)
{
  setPixel( x,  y, value);
  setPixel(-x,  y, value);
  setPixel(-y,  x, value);
  setPixel(-y, -x, value);
  setPixel(-x, -y, value);
  setPixel( x, -y, value);
  setPixel( y, -x, value);
  setPixel( y,  x, value);
}
```

Exemplo



x	y	d
0	10	-9
1	10	-6
2	10	-1
3	10	6
4	9	-8
5	9	3
6	8	-5
7	8	12

Análise de Complexidade

- Apenas aritmética inteira;
- A determinação de N não é automática;
  - Função de r, ou seja, o número de pixels a serem processados depende do raio do círculo (obviamente);
  - Seguramente N < r;
  - Algoritmo linear;
  - Multiplicações por 2 podem ser substituidas por deslocamento a esquerda (uma casa apenas);

Primitiva Gráfica - Curvas

- Matematicamente, curvas são representadas tradicionalmente utilizando equações tais como:
  - $y=x^2+2x+3$
  - $y=x^4+3x^3+2x^2+1$
  - $x^2+y^2 = r$
- Os problemas relacionados com essas representações são:
  - Difícil encontrar a equação exata para a curva desejada
  - Rasterizar tais curvas pode ser computacionalmente caro;
- A solução: Utilizar curvas paramétricas!

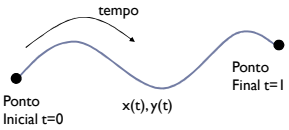
Código Exemplo (Matlab)

```
%define o num. de subdivisões
subdiv = 10
colorInc = 1/(subdiv+1);
%define um conjunto inicial de três pontos
%subplot(subdiv+1,1,1);

x = [50 250 400]
y = [50 450 300]
plot(x,y,'-o','Color',[0,0,1])
origx = x;
origy = y;

for count = 1: subdiv
   newx = [];
   newy = [];
   newx(1) = x(1);
   newy(1) = y(1);
   for allp = 1: size(x,2) - 1
      sumx = floor((x(allp) + x(allp+1) )/2);
      sumy = floor((y(allp) + y(allp+1) )/2);
      newx(allp+1) = sumx;
      newy(allp+1) = sumy;
   end;%for
   newx(size(x,2)+1) = x(size(x,2));
   newy(size(x,2)+1) = y(size(x,2));
   x = newx;
   y = newy;
   figure;
   hold on
   %subplot(subdiv+1,1,count+1);
   plot(origx,origy,'-o','Color',[0,0,1])
   plot(x,y,'-o','Color',[0, 0, colorInc*count]);
end;%for
```

Curvas Paramétricas

- Curvas 3D são representadas usando representação paramétrica, introduzindo assim uma nova variável "t";
- Q(t) = |x(t) y(t) z(t)|
- Note que x,y e z são independentes entre si, sendo dependentes apenas em t (que é a variável livre da eq.);
- Pense acerca de "t" como a progressão do tempo durante o desenho sequencial da curva;
- t = [0,1]
- http://www.inf.ed.ac.uk/teaching/courses/cg/d3/hermite.html

Metáfora do Lápis



- Critérios:
  - Controle local da forma;
  - Suavidade e Continuidade;
  - Capacidade de se avaliar derivadas;
  - Estabilidade;
  - Facilidade de renderização;

Continuidade de Curvas

- Continuidade Geométrica
  - Uma curva pode ser descrita como tendo continuidade G^n, n sendo uma medida de suavidade;
  - G^0 – As curvas tocam no ponto de junção;
  - G^1 – As curvas também compartilham uma direção tangente comum ao ponto de junção;
  - G^2 – As curvas também compartilham um centro de curvatura comum no ponto de junção.

Continuidade de Curvas

- Continuidade Paramétrica
  - C^0 – Curvas são conectadas;
  - C^1 – As primeiras derivadas são iguais;
  - C^2 – As primeiras e segundas derivadas são iguais;
  - C^n – As primeiras n-ésimas derivadas são iguais.

Curvas Paramétricas

- Utilizam-se polinômios de terceira ordem;
  - Porquê?
    - Curvas de mais baixa ordem não podem ser unidas suavemente;
    - Curvas de mais alta ordem introduzem oscilações, e são matematicamente mais complexas;
    - Ordem 3 é necessário e suficiente!

Matematicamente ...

$$Q(t) = \begin{vmatrix} x(t) & y(t) & z(t) \end{vmatrix}$$

$$\begin{cases} x(t) = a_x t^3 + b_x t^2 + c_x t + d_x \\ y(t) = a_y t^3 + b_y t^2 + c_y t + d_y \\ z(t) = a_z t^3 + b_z t^2 + c_z t + d_z \end{cases}$$

$$T = \begin{bmatrix} t^3, t^2, t, 1 \end{bmatrix} \qquad C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix} \qquad Q(t)^T = C^T \cdot T^T$$

- Note que a derivada de Q(t) é sua tangente

Matematicamente ...

$$d/dt\, Q(t) = \begin{bmatrix} d/dt\, x(t), d/dt\, y(t), d/dt\, z(t) \end{bmatrix}$$

$$\begin{cases} d/dt\, Q(t)_x = 3a_x t^2 + 2b_x t + c_x \\ d/dt\, Q(t)_y = 3a_y t^2 + 2b_y t + c_y \\ d/dt\, Q(t)_z = 3a_z t^2 + 2b_z t + c_z \end{cases}$$

$$d/dt Q(t) = [3t^2, 2t, 1, 0] \cdot C$$

Curva de Hermite

- Curvas que se juntem de maneira suave são desejadas;
- Curvas são especificadas provendo-se:
  - Os pontos terminadores;
  - As primeiras derivadas dos pontos terminadores;
- Dada a equação Q(t)=T·C:
  - Podemos fatorar a matriz C em duas outras matrizes C=G·M;
  - G – matriz da geometria;
  - M – matriz de base;
- G representa as restrições geométricas da curva (pontos terminadores e derivadas), ao passo que M será constante para todas as curvas de Hermite.

Curva de Hermite

- Olhando apenas para a coordenada x ...

$$Q(t)_x = a_x t^3 + b_x t^2 + c_x t + d_x$$

e sua derivada

$$d/dt\, Q(t)_x = 3a_x t^2 + 2b_x t + c_x$$

Sendo assim, reescrevendo de forma linear, tem-se:

$$\begin{bmatrix} Q(0)_x \\ Q(1)_x \\ Q(0)'_x \\ Q(1)'_x \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix}$$

Curva de Hermite

$$\begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} Q(0)_x \\ Q(1)_x \\ Q(0)'_x \\ Q(1)'_x \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} Q(0)_x \\ Q(1)_x \\ Q(0)'_x \\ Q(1)'_x \end{bmatrix}$$

- O mesmo desenvolvimento matemático segue para y e z.
- Então...

Curva de Hermite

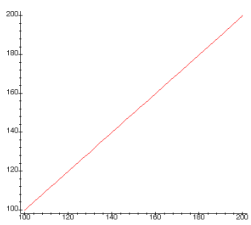
$$Q(t) = C \cdot T =$$

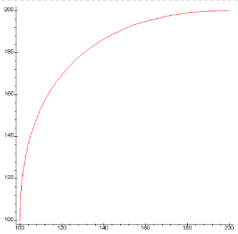
$$T \cdot M \cdot [Q(0), Q(1), Q(0)', Q(1)'] =$$

$$T \cdot M \cdot [P_1, P_4, R_1, R_4]$$

- Uma vez que M e T são conhecidos, pode-se escrever uma polinomial cúbica por inspeção simples.

Exemplos





$$G = \begin{pmatrix} 100 & 100 & 0 \\ 200 & 200 & 0 \\ 10 & 10 & 0 \\ 10 & 10 & 0 \end{pmatrix} \qquad G = \begin{pmatrix} 100 & 100 & 0 \\ 200 & 200 & 0 \\ 0 & 200 & 0 \\ 200 & 0 & 0 \end{pmatrix}$$

Curvas de Bezier

- Outra forma para se especificar curvas foi formulada por Bezier, em meados de 1972;
- Ela requer que quatro pontos (p_1, p_2, p_3, p_4) sejam especificados:
  - O primeiro e o último pontos (p_1, p_2) são os pontos terminadores da curva (na curva);
  - Os pontos intermediários (p_3, p_4) são chamados pontos de controle e residem fora da curva.
  - A curvatura é controlada pelos vetores (p_1, p_2) e (p_3, p_4).

Binômio de Newton

- Como expandir expressões do tipo $(x+y)^n$?

$$(x+y)^1 = x+y$$
$$(x+y)^2 = (x+y)(x+y) = x^2+2xy+y^2$$
$$(x+y)^3 = \big((x+y)(x+y)\big)(x+y)$$
$$= (x^2+2xy+y^2)(x+y)$$
$$= x^3+2x^2y+xy^2+x^2y+2xy^2+y^3$$
$$(x+y)^7 = (x+y)(x+y)(x+y)(x+y)(x+y)(x+y)(x+y)$$

$$(x+y)^n = \sum_{k=0}^{n}\binom{n}{k}\cdot x^{n-k}y^k$$

n – grau do binômio

$\binom{n}{k}$ – coeficiente binomial

* Número de combinações de n elementos tomados k a k.

Exemplo

$$(x+y)^4 = \sum_{k=0}^{4}\binom{4}{k}\cdot x^{4-k}y^k$$
$$= \binom{4}{0}\cdot x^{4-0}y^0 + \binom{4}{1}\cdot x^{4-1}y^1 + \binom{4}{2}\cdot x^{4-2}y^2 + \binom{4}{3}\cdot x^{4-3}y^3 + \binom{4}{4}\cdot x^{4-4}y^4$$
$$= x^4+4x^3y+6x^2y^2+4xy^3+y^4$$

Triângulo de Pascal

0: 1
1: 1 1
2: 1 2 1
3: 1 3 3 1
4: 1 4 6 4 1
5: 1 5 10 10 5 1
6: 1 6 15 20 15 6 1

l\c	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	20	15	6	1

$$t_{l,c} = t_{l-1,c} + t_{l,c-1}$$

Curva de Bezier Linear

- Caso mais simples;
- Equação paramétrica do segmento de reta;

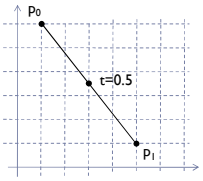
$$B(t) = (1-t)B_0 + tB_1, t \in [0,1]$$
$$B(t) = \sum_{k=0}^{1}\binom{1}{k}\cdot(1-t)^{1-k}t^kB_k$$
$$= \binom{1}{0}\cdot(1-t)^{1-0}t^0B_0 + \binom{1}{1}\cdot(1-t)^{1-1}t^1B_1 = (1-t)B_0 + tB_1$$

Pontos de controle

- Casos extremos (t=0 e t=1) são fáceis de calcular;
- Caso t=0.5:

Curva de Bezier Linear

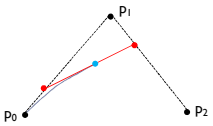
- Caso mais simples;
- Equação paramétrica do segmento de reta;

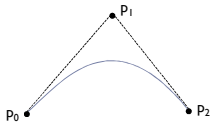


$$B(0.5) = 0.5B_0 + 0.5B_1$$
$$= 0.5\begin{pmatrix}1\\6\end{pmatrix} + 0.5\begin{pmatrix}5\\1\end{pmatrix} = \begin{pmatrix}0.5\\3\end{pmatrix} + \begin{pmatrix}2.5\\0.5\end{pmatrix} = \begin{pmatrix}3\\3.5\end{pmatrix}$$

Curva de Bezier Quadrática

$$B(t) = (1-t)^2B_0 + 2t(1-t)B_1 + t^2B_2, t \in [0,1]$$
$$B(t) = \sum_{k=0}^{2}\binom{2}{k}\cdot(1-t)^{2-k}t^kB_k$$
$$= \binom{2}{0}\cdot(1-t)^{2-0}t^0B_0 + \binom{2}{1}\cdot(1-t)^{2-1}t^1B_1 + \binom{2}{2}\cdot(1-t)^{2-2}t^2B_2 = (1-t)^2B_0 + 2t(1-t)B_1 + t^2B_2$$



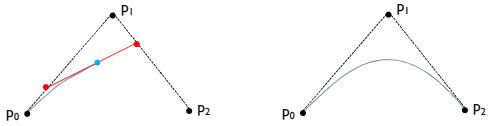


Curva de Bezier Cúbica

$$B(t) = (1-t)^3 B_0 + 3t(1-t)^2 B_1 + 3(1-t)t^2 B_3 + t^3 B_4, t \in [0,1]$$

$$B(t) = \sum_{k=0}^{3} \binom{3}{k} \cdot (1-t)^{3-k} t^k B_k$$

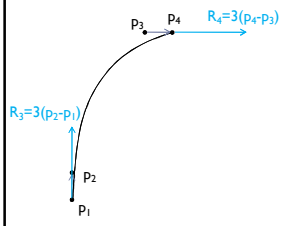
$$= \binom{3}{0} \cdot (1-t)^{3-0} t^0 B_0 + \binom{3}{1} \cdot (1-t)^{3-1} t^1 B_1 + \binom{3}{2} \cdot (1-t)^{3-2} t^2 B_2 + \binom{3}{3} \cdot (1-t)^{3-3} t^3 B_3$$



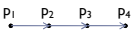
Algoritmo BezierIngênuo

```
import sys,pygame
from pygame import gfxdraw

pygame.init()
screen = pygame.display.set_mode((400,400))
screen.fill((0,0,0))
pygame.display.flip()

white=(255,255,255)

#Esta funcao funciona apenas
#para o primeiro quadrante
def bezierIngenuo(p1,p2,p3,p4):
    for t in xrange(0,1,0.01):
        omt  = 1-t
        omt2 = omt*omt
        omt3 = omt2*omt
        t2   = t*t
        t3   = t2*t
        x    = omt3 * p1[0] + ((3*omt2)*t*p1[0]) +
(3*omt*t2*p3[0])+t3*p4[0]
        y    = omt3 * p1[1] + ((3*omt2)*t*p1[1]) +
(3*omt*t2*p3[1])+t3*p4[1]
        x    = round(x,0)
        y    = round(y,0)

        screen.set_at((x,y), white)
        pygame.display.flip()

bezierIngenuo(10,10,50,50)

while 1:
    for event in pygame.event.get():
            if event.type == pygame.QUIT: sys.exit()
```

Hermite ↔ Bezier



- Tangentes nos pontos terminadores são definidas por pontos intermediários;
- Porque "3"? Porque não $R_1=(p_2-p_1)$;
- Pense em 4 pontos espaçados igualmente em uma linha!



Hermite ↔ Bezier

- As curvas são equivalentes;
- É possível formular curvas de Bezier de maneira linear;
- Primeiro, relacionamos a geometria de uma curva de Hermite com a geometria de uma curva de Bezier:

$$G_H = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

$$G_H = M_{HB} G_B$$

Hermite ↔ Bezier

$$Q(t) = T M_H G_H$$

$$G_H = [P_1, P_4, R_1, R_4]^T, T = [t^3, t^2, t, 1]$$

$$M_H = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$Q(t) = T M_H M_{HB} G_B$$

Hermite ↔ Bezier

$$M_B = M_H M_{HB} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$Q(t) = T M_B G_B$$

$$Q(t) = \begin{cases} \left(-t^3 + 3t^2 - 3t + 1\right)x_1 + \left(3t - 6t^2 + 3t^3\right)x_2 + \left(3t^2 - 3t^3\right)x_3 + t^3 x_4 \\ \left(-t^3 + 3t^2 - 3t + 1\right)y_1 + \left(3t - 6t^2 + 3t^3\right)y_2 + \left(3t^2 - 3t^3\right)y_3 + t^3 y_4 \\ \left(-t^3 + 3t^2 - 3t + 1\right)z_1 + \left(3t - 6t^2 + 3t^3\right)z_2 + \left(3t^2 - 3t^3\right)z_3 + t^3 z_4 \end{cases}$$

$$Q(t) = P_1\left(-t^3 + 3t^2 - 3t + 1\right) + P_2\left(3t^3 - 6t^2 + 3t\right) + P_3\left(-3t^3 + 3t^2\right) + P_4 t^3$$

O que mais há em relação a curvas?

- Usando o algoritmo de Casteljau, curvas de Hermite e finalmente curvas de Bezier, é possível modelar qualquer curva;
- De fato, curvas de Bezier podem ser encontradas em muitos softwares, como por exemplo o pacote MS-Office;
- Três fatos nos entanto motivam a busca de novas formas de representar curvas:
  1. Curvas são computacionalmente caras de se computar;
  2. Pontos de controle residem fora da curva;
  3. A alteração de um ponto de controle implica em possível alterações em todos os pontos da curva.

Splines

- Curvas de facto usadas em pacotes de 3D;
- **Splines Cúbicas Naturais** são curvas polinomiais cúbicas com continuidades C^0, C^1 e C^2;
  - Possuem um grau de continuidade a mais que as curvas de Bezier e Hermite!
- As curvas passam pelos pontos de controle!
- São consideradas mais suaves como elemento de interpolação.
- **O preço a ser pago:** Os coeficientes das splines cúbicas naturais são dependentes de todos os **n** (4) pontos de controle;

B-Splines (Basis Splines)

- Segmentos de curva cujo comportamento depende apenas de alguns poucos pontos de controle;
- As B-Splines são tratadas de forma um pouco diferente das curvas de Bezier ou Hermite;
- B-Splines são curvas com muitos pontos de controle, mas que são tratadas como uma sequência de segmentos de ordem cúbica;

$$BS = P_0 P_1 \cdots P_m, m \geq 3$$

- Possui m-2 segmentos. Cada segmentos é denotado por Q_3 a Q_m;
- Cada um dos m-2 segmentos de curva da B-Spline é definido por quatro dos m+1 pontos de controle.

B-Spline

- O segmento i da B-Spline é definido como:

$$Q_i(t) = T_i M_{BS} G_{BS_i}$$

- O vetor de geometria que aproxima o segmento Q_i (definido pelos pontos $P_{i-3} - P_i$)

$$G_{BS_i} = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}, 3 \leq i \leq m$$

- A matriz B-Spline base é definida por:

$$M_{BS} = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

B-Spline – Funções de Mistura

$$Blend = TM_{BS}$$

$$Blend = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

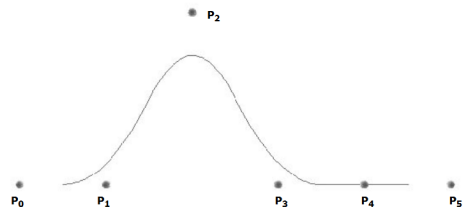
$$B_0 = \frac{1}{6}\left(-t^3 + 3t^2 - 3t + 1\right)P_i = -\frac{1}{6}(t-1)^3 P_i$$

$$B_1 = \frac{1}{6}\left(3t^3 - 6t^2 + 4\right)P_{i+1}$$

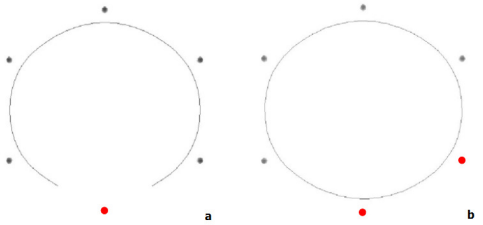
$$B_2 = \frac{1}{6}\left(3t^3 + 3t^2 + 3t + 1\right)P_{i+2}$$

$$B_3 = \frac{1}{6}t^3 P_{i+3}$$

Exemplos



Exemplos



Referências

[BRE65] Bresenham, J. E. "Algorithm for Computer Control of a Digital Plotter," IBM Systems Journal, 4(1), 1965, 25-30.