COMPUTAÇÃO GRÁFICA

OBJETIVO

Proporcionar o aprendizado de técnicas e conceitos básicos de computação gráfica, que podem ser utilizados para o **desenvolvimento, avaliação e uso de aplicativos gráficos.**

- CAD – Arquitetura
- CAD – Engenharia Eletrônica
- CAD – Engenharia Mecânica
- Simulação Visual (Vôo, autos, cirurgias, etc.);
- Tratamento de imagens na Astronomia;
- Marketing, Turismo, Moda, etc, etc...

Aplicações da Computação Gráfica

- Atualmente a CG está presente em quase todas as áreas do conhecimento humano. Alguns exemplos:
- Engenharia que utiliza as tradicionais ferramentas CAD (*Computer-Aided Design*).
- Medicina: que trabalha com modernas técnicas de visualização para auxiliar o diagnóstico por imagens.
- Nesta área, também têm sido desenvolvidos sistemas de simulação para auxiliar no treinamento de cirurgias endoscópicas.
- Outros tipos de simuladores são usados para treinamento de pilotos e para auxiliar na tomada de decisões
- Direito (por exemplo, para reconstituir a cena de um crime).

Computação Gráfica - Aplicações

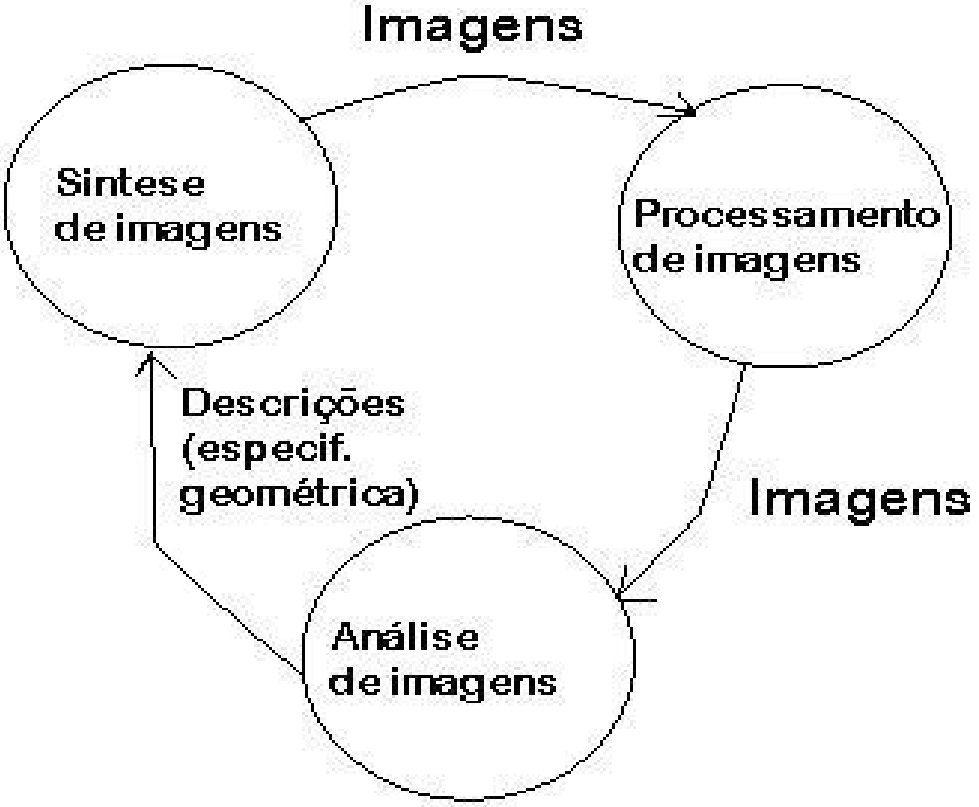
- Uma área de aplicação da CG que tem crescido muito nos últimos anos é a visão computacional.
- Inicialmente, surgiu a visualização científica, que visava o desenvolvimento de representações gráficas para grandes volumes de dados gerados, por exemplo, por satélites e equipamentos de sensoriamento remoto.
- Visualização volumétrica: trata apenas de como gerar representações para dados volumétricos, tais como dados meteorológicos, oceanográficos e médicos.
- Visualização de informações: é uma área que visa auxiliar na análise de dados financeiros, mineração de dados, estudos do mercado ou de gerenciamento de redes de computadores.

Aplicações da Computação Gráfica

- Animação é outro exemplo de aplicação que tem sido bastante desenvolvida. Através da exibição de imagens em seqüência é possível representar o comportamento de objetos reais ou simulados. Além de desenho animados por computador, a simulação de humanos virtuais e de comportamento de multidões são temas de pesquisas atuais.
- Técnicas de CG também são usadas em RV, que introduziu um novo paradigma de interface com o usuário. Através de dispositivos especiais que captam movimentos do corpo do usuário é possível interagir em diferentes ambientes projetados por computador. Simuladores para treinamento de pessoal, tratamento de fobias e jogos são alguns exemplos de aplicações de RV.

COMPUTAÇÃO GRÁFICA

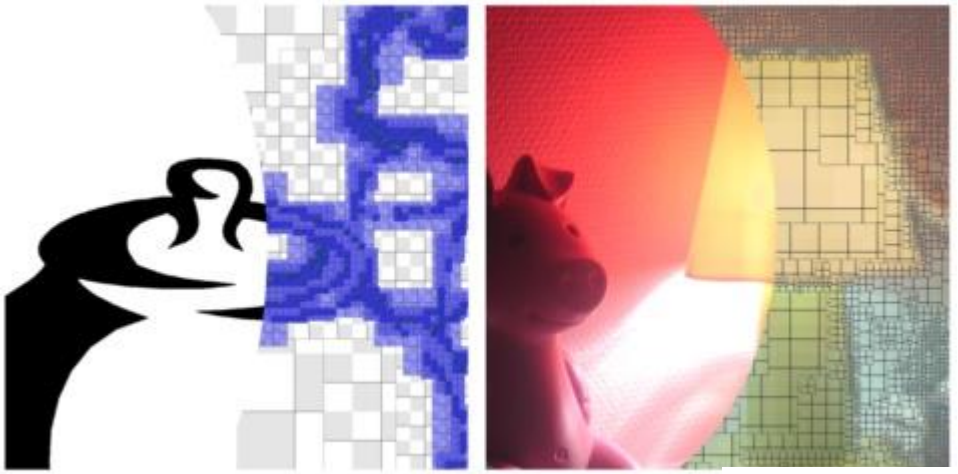
- Área da ciência da computação que estuda a geração, manipulação e interpretação de modelos e imagens de objetos utilizando computador. Tais modelos vêm de uma variedade de disciplinas, como física, matemática, engenharia, arquitetura, etc.
- Pode-se relacionar a Computação Gráfica com 3 sub-áreas [Persiano and Oliveira, 1989]:



COMPUTAÇÃO GRÁFICA

Síntese de Imagens: Área que se preocupa com a produção de representações visuais a partir das especificações geométrica e visual de seus componentes. É freqüentemente confundida com a própria Computação Gráfica. As imagens produzidas por esta sub-área são geradas a partir de dados mantidos nos chamados Display-Files.

Síntese de Imagem



Extraído de Hoppe H. **Hugues Hoppe's home page** disponível em http://research.microsoft.com/en-us/um/people/hoppe/

COMPUTAÇÃO GRÁFICA

- **Processamento de Imagens**: envolve as técnicas de transformação de Imagens, em que tanto a imagem original quanto a imagem resultado apresentam-se sob uma representação visual (**geralmente matricial**). Estas transformações visam melhorar as características visuais da imagem (aumentar contraste, foco, ou mesmo diminuir ruídos e/ou distorções).
- As imagens produzidas/ utilizadas por esta sub-área são armazenadas, ou recuperadas dos chamados Raster-Files.

Computação Gráfica

- **Análise de Imagens**: área que procura obter a especificação dos componentes de uma imagem a partir de sua representação visual. Ou seja a partir da informação pictórica da imagem (a própria imagem!) produz uma informação não pictórica da imagem (por exemplo, **as primitivas geométricas elementares que a compõem).**

Computação Gráfica

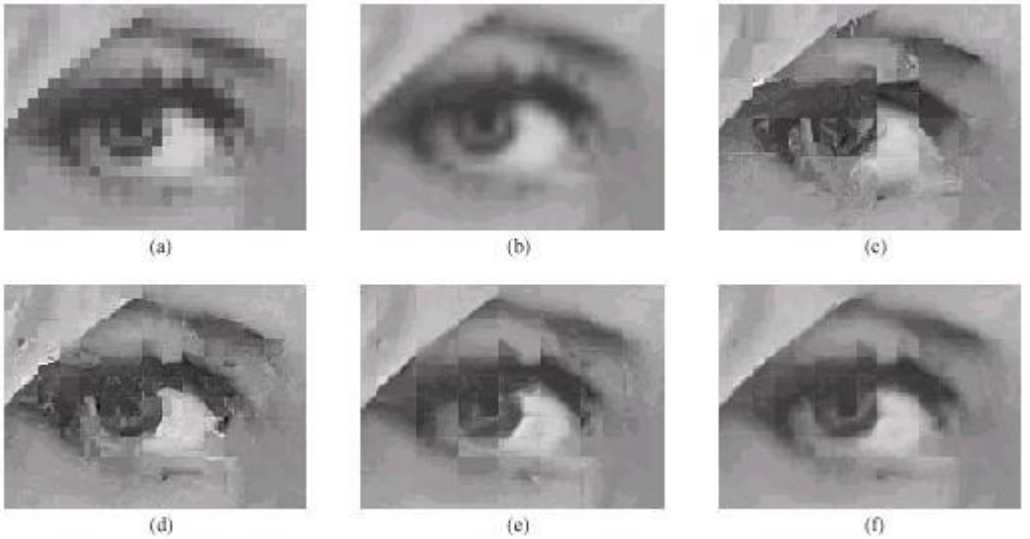
- **Visualização Computacional**, que usa técnicas de Computação Gráfica para representar informação, de forma a facilitar o entendimento de conjuntos de dados numéricos de alta complexidade. Exemplos de áreas de aplicação são: visualização de imagens médicas, meteorologia, dados financeiros, visualização de programas, dinâmica dos fluidos, e muitas outras.
- A representação gráfica (superfícies, partículas, ícones) é gerada automaticamente a partir do conjunto de dados. Ao usuário cabe definir parâmetros e atributos da imagem para melhor "navegar" seu conjunto de dados. Dessa maneira, a visualização de dados partilha de características da síntese, do processamento e da análise de dados.

Computação Gráfica x Processamento de Imagens

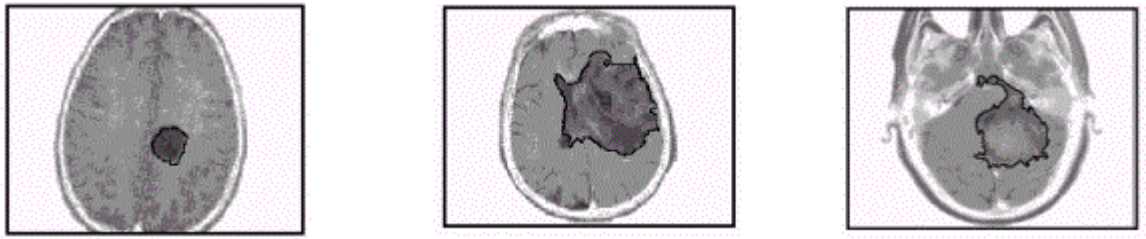
- **Computação gráfica** trata da **síntese** de imagens de objetos reais ou imaginários **a partir de modelos** computacionais.
- **Processamento de imagens** é uma área relacionada que trata do processo inverso: a análise de cenas, ou a reconstrução de modelos de objetos 2D ou 3D a partir de suas imagens.
- **Síntese de imagens** parte da descrição de objetos tais como segmentos de reta, polígonos, poliedros, esferas, etc.; e produz uma imagem que atende a certas especificações e que pode, em última instância, ser visualizada em algum dispositivo.
- As imagens em questão constituem uma representação visual de objetos bi- ou tridimensionais descritos através de especificações abstratas.

Computação Gráfica x Processamento de Imagens

- O **processamento de imagens** parte de imagens já prontas para serem visualizadas, as quais são transferidas para o computador por mecanismos diversos - digitalização de fotos, tomadas de uma câmera de vídeo, ou imagens de satélite - para serem manipuladas visando diferentes objetivos.



"*A técnica de zoom fractal está sendo muito utilizada com o objetivo conseguir uma aproximação da imagem com o mínimo de perdas na qualidade da imagem. A figura acima ilustra a comparação de várias técnicas de zoom fractal, sendo que cada uma delas apresenta valor de fator, e com isso qualidades diferentes.*"



Extraído de http://www.inf.ufsc.br/~visao/2000/fractais/index.html

Sistemas Gráficos

- A Computação Gráfica (especialmente as componentes relativas a gráficos 3D e a gráficos 3D interativos) desenvolveu-se de modo bem diverso: de simples programas gráficos para computadores pessoais a programas de modelagem e de visualização em workstations e supercomputadores.
- 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.
- Para se 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.

Traçado de Curvas em Dispositivos Gráficos Matriciais

Traçado de Primitivas

- O traçado de primitivas gráficas elementares, como segmentos de reta ou arcos de circunferência, requer a construção de algoritmos capazes de determinar na matriz de pixels da superfície de exibição quais pixels devem ser alterados de forma a simular-se a aparência do elemento gráfico desejado.
- Estes algoritmos recebem o nome de Algoritmos de Conversão Matricial, por converterem em representação matricial elementos gráficos expressos em uma representação vetorial.

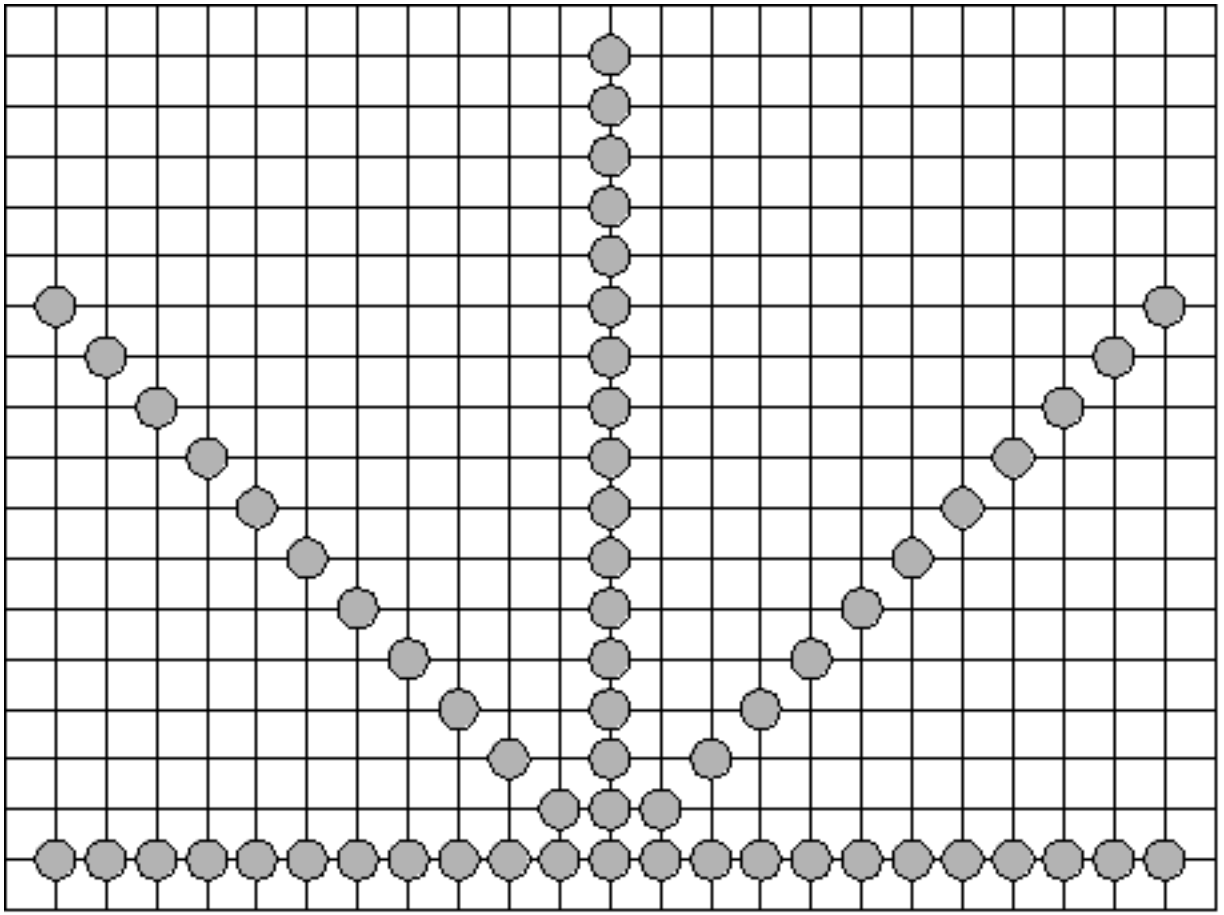
Traçado de Primitivas Gráficas

- Um elemento geométrico básico a ser traçado num terminal gráfico é o segmento de reta.
- As rotinas de traçado de linhas estão disponíveis em hardware (em chips controladores).
- O chip recebe as coordenadas dos pixels extremos, e calcula quais pixels intermediários da superfície de exibição devem ser traçados para gerar uma reta (aproximada) na tela.
- Gráficos complexos requerem o traçado de uma grande quantidade de segmentos de reta, e a velocidade é importante - daí a opção de traçado por meio de hardware especializado, que libera o processador hospedeiro para outras atividades enquanto o chip controlador calcula os pixels a serem traçados.

Traçado de Primitivas Gráficas

- Se a função de traçado precisa ser implementada em software, a rotina deve ser escrita em código de máquina otimizado para atingir o máximo de velocidade. Entretanto, é instrutivo estudar alguns algoritmos para traçado de linhas usando uma linguagem de alto nível.
- Suponhamos, que a superfície de exibição possua igual densidade de pixels na horizontal e na vertical (razão de aspecto gráfica igual a 1).

Figura 1 - Representação de segmentos de reta horizontais, verticais e diagonais.



Simetria e Reflexões

- A representação matricial de segmentos de reta horizontais, verticais, e diagonais não apresenta nenhuma dificuldade.
- Pode-se dizer que estas direções formam na realidade eixos de simetria no espaço matricial. Logo, qualquer imagem representada no espaço matricial pode ser refletida em relação a qualquer reta horizontal, vertical ou diagonal sem apresentar qualquer deformação.
- A Simetria é explorada na conversão matricial de primitivas gráficas que apresentem simetria em relação a qualquer um destes eixos.
- Na Figura 3 pode-se ver como efetuar a rotação de uma imagem utilizando de 2 reflexões

Figura 2 - Reflexão de uma imagem com relação a um eixo diagonal.

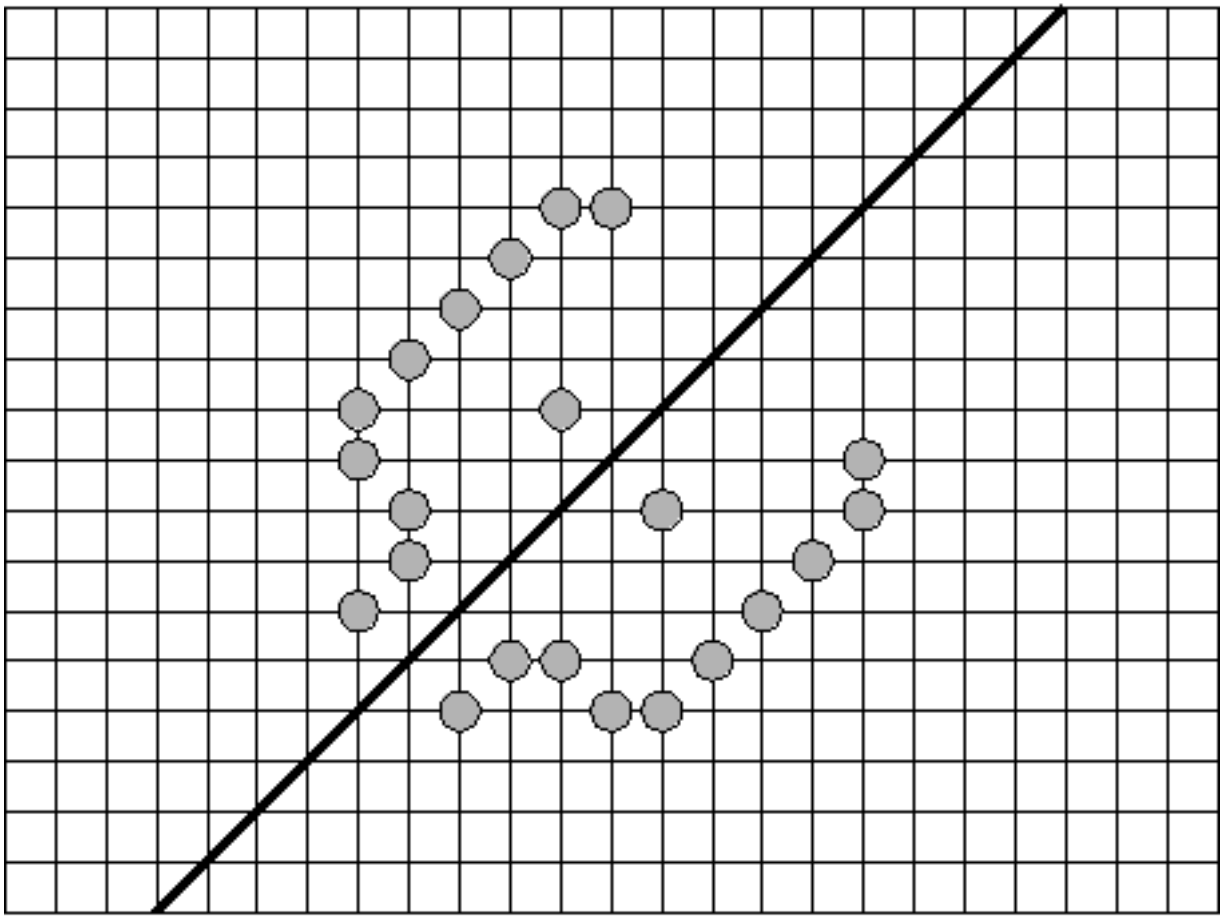


Figura 3 - Rotação de 90o obtida através de 2 reflexões, uma horizontal (a) e outra na diagonal (b).

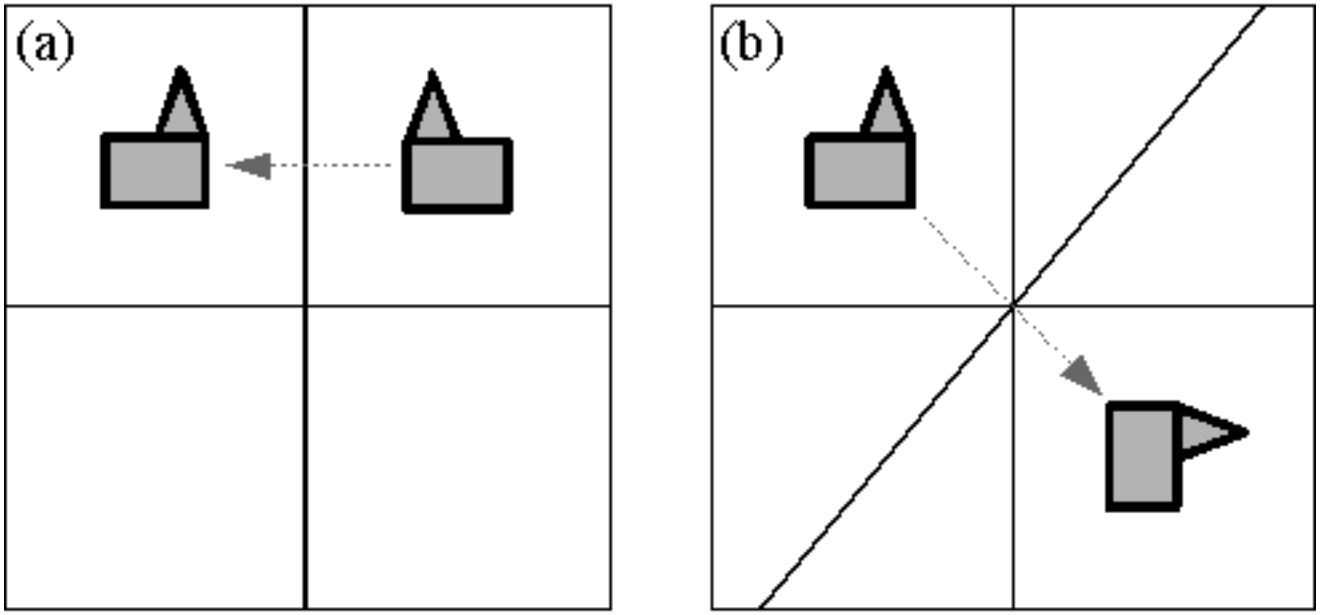
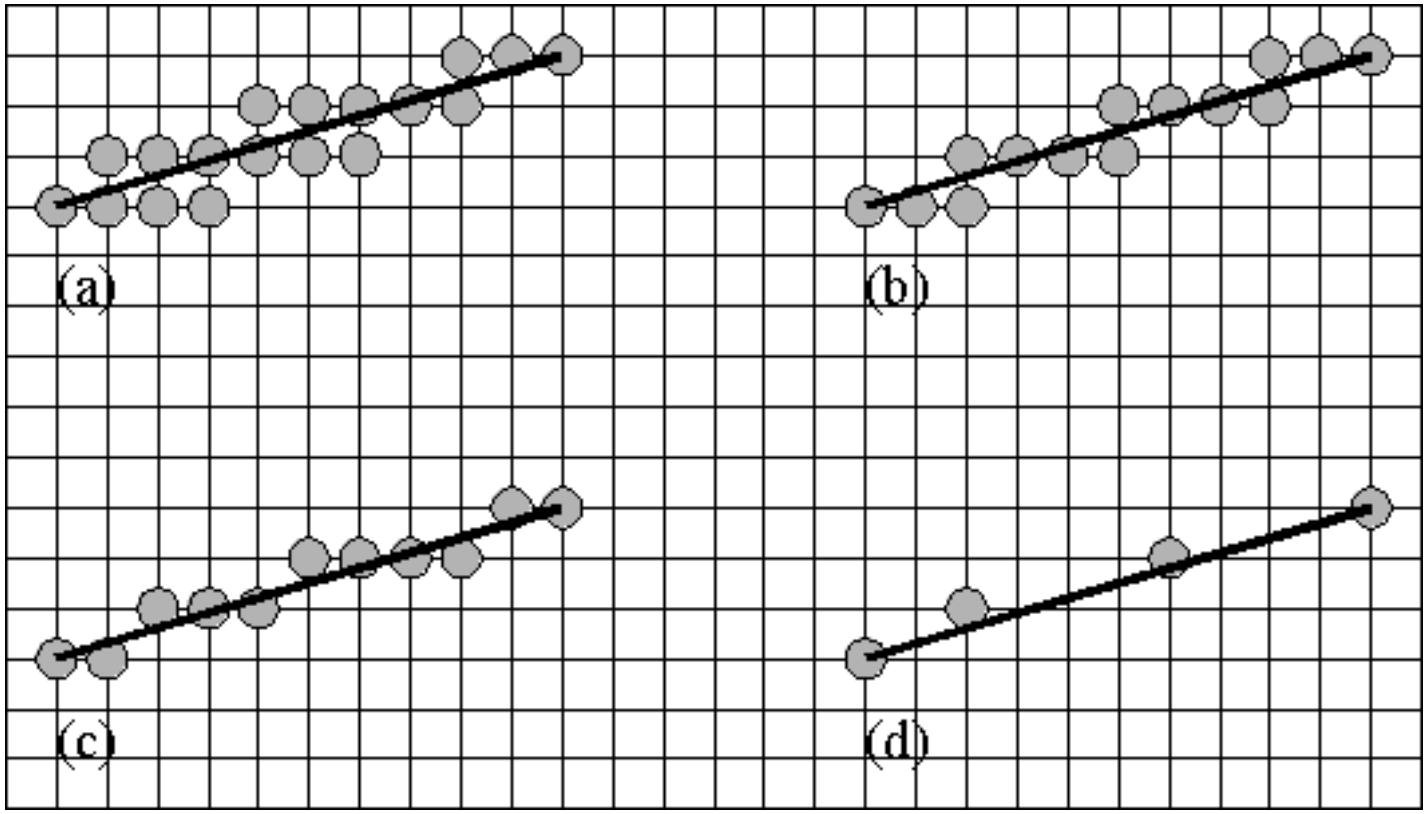


Figura 4 - Conversão matricial de segmento de reta



Conversão matricial de segmento de reta

- Alguns critérios de conversão matricial podem ser vistos através de figura 4
- Em (a) adotou-se o critério de selecionar-se os dois pixels imediatamente acima e abaixo do ponto de intersecção do segmento com cada vertical, a menos quando o segmento passa por um pixel. Restrição: dessa forma obtém-se linhas densas, como se o segmento fosse espesso.
- Em (b) o critério foi selecionar todos os pixels cujas coordenadas são obtidas arredondando-se os valores das coordenadas de algum ponto do segmento. Restrição: com segmentos a 45 o critério produz resultados semelhantes ao anterior.

Conversão matricial de segmento de reta

- A conversão o ilustrada em (c) foi obtida selecionando-se em cada vertical o pixel mais próximo do ponto de intersecção do segmento com a reta vertical. Vantagens: aparência leve e continuidade.
- Em (d) utilizou-se o mesmo critério, porém para as linhas horizontais. Restrição: descontinuidade.

Características desejáveis para os algoritmos de conversão matricial de segmentos de retas

1. Linearidade: os pixels traçados devem dar a aparência de que estão sobre uma reta. Isto é trivial no caso de segmentos paralelos aos eixos x ou y, ou com inclinação de 45 graus, mas não nos outros casos.
2. Precisão: os segmentos devem iniciar e terminar nos pontos especificados. Caso isso não ocorra, pequenos gaps podem surgir entre o final de um segmento e o início de outro.
3. Espessura (Densidade) uniforme: a densidade da linha é dada pelo número de pixels traçados dividido pelo comprimento da linha. Para manter a densidade constante, os pixels devem ser igualmente espaçados. A imagem do segmento não varia de intensidade ou espessura ao longo de sua extensão.
4. Intensidade independente da inclinação: para segmentos de diferentes inclinações.

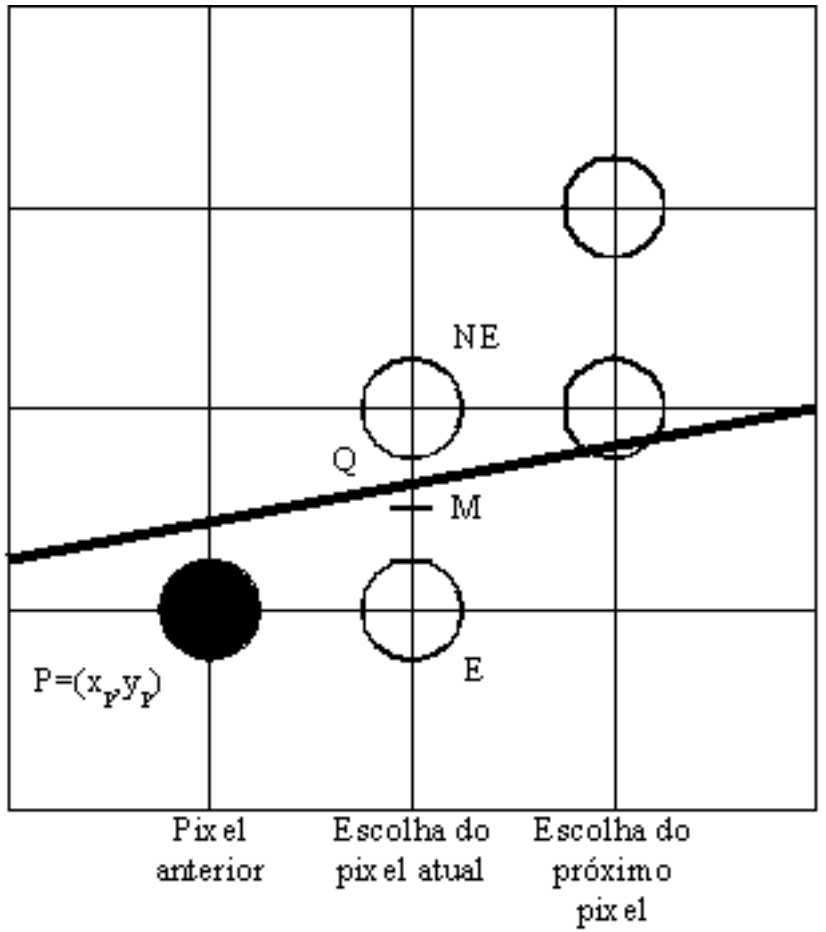
Características desejáveis para os algoritmos de conversão matricial de segmentos de retas

5. Continuidade: a imagem não apresenta interrupções indesejáveis.
6. Rapidez no Traçado dos segmentos.

Algoritmo simples para conversão matricial de retas. Algoritmo diferencial

- *Trata-se de um algoritmo pouco eficiente pois utiliza cálculos de ponto flutuante.*

```
void line(int x1, int y1, int x2, int y2, int color) {
int x, y;
float a;
int valor;
a=(y2-y1)/(x2-x1);
for (x=x1;x<=x2;x++){/* arredonda y */
y = (y1 + a * (x - x1));
write_pixel(x, y, color); }/* end for */
}/*end line */
```


NE
Q
M
E
P=(x_P,y_P)
Pixel anterior
Escolha do pixel atual
Escolha do próximo pixel

Algoritmo Incremental - Bresenham

```
void inc_line(int x1, int y1, int x2, int y2, int color){
int dx, dy, incE, incNE, d, x, y;
dx = x2 - x1;
dy = y2 - y1;
d = 2 * dy - dx; /* Valor inicial de d */
incE = 2 * dy; /* Incremento de E */
incNE = 2 * (dy - dx); /* Incremento de NE */
x = x1;
y = y1;
write_pixel(x, y, color);
```

```
while (x < x2){
    if (d <= 0) {
        /* Escolhe E */
        d = d + incE;
        x = x + 1;
        }else{
        /* Escolhe NE */
        d = d + incNE;
        x = x + 1;
        y = y + 1;
        }/* end if */
        write_pixel(x, y, color);
            }/* end while */
                }/* end inc_line */
```

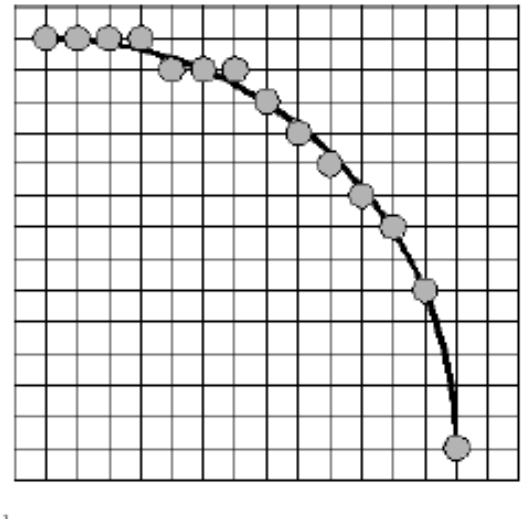
Círculos e Elipses

A equação de uma **circunferência** com centro na origem e raio R, em coordenadas cartesianas, é dada por:

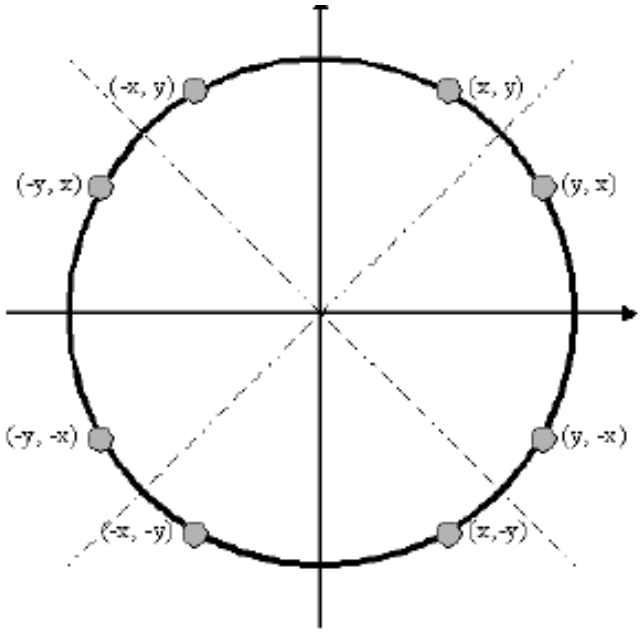
$$x^2 + y^2 = R^2$$

Portanto tem-se

$$y = \pm\sqrt{R^2 - x^2}$$



- O traçado de uma circunferência pode tirar proveito de sua simetria. Considere uma circunferência centrada na origem. Se o ponto (x, y) pertence à circunferência, pode-se calcular de sete outros pontos da circunferência.
- Consequentemente, basta computar um arco de circunferência de 45° para obter a circunferência toda.
- Para uma circunferência com centro na origem, os oito pontos simétricos podem ser traçados usando o procedimento CirclePoints.



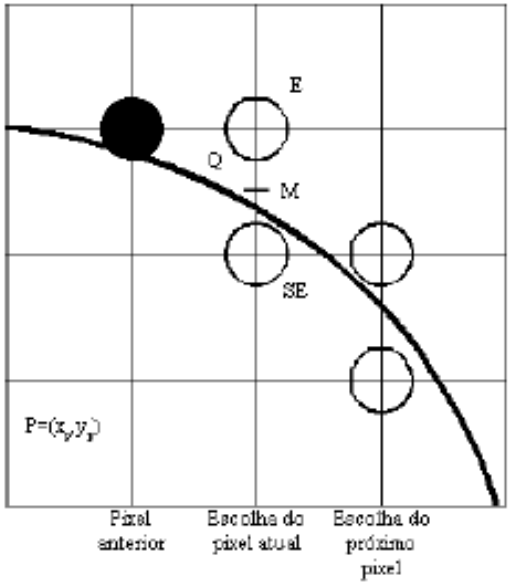

```
void CirclePoints(int x, int y, int color){

   write_pixel( x,  y, color);
   write_pixel( x, -y, color);
   write_pixel(-x,  y, color);
   write_pixel(-x, -y, color);
   write_pixel( y,  x, color);
   write_pixel( y, -x, color);
   write_pixel(-y,  x, color);
   write_pixel(-y, -x, color);
}/* end CirclePoints */
```

Os mesmos dois passos executados para o algoritmo do traçado de linhas são executados para o algoritmo de circunferência:

1. escolher o pixel com base no sinal da variável d, calculada na iteração anterior;

2. atualizar a variável d com o valor correspondente ao pixel escolhido.



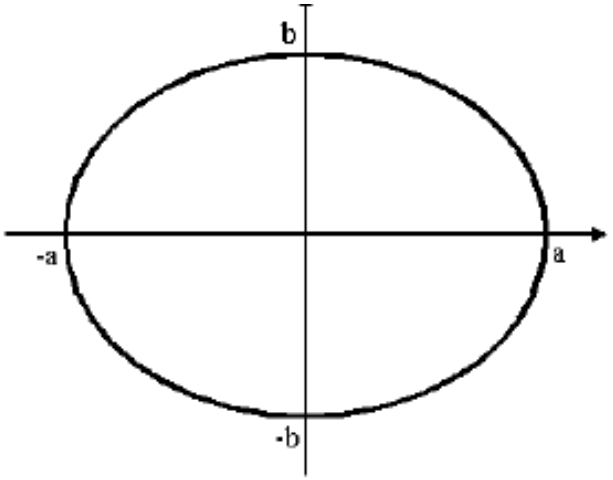
A diferença é que, na atualização de d, calcula-se uma função linear do ponto de avaliação.

Algoritmo do Ponto-Médio para circunferências utilizando aritmética inteira. (Bresenham)

```
void MidPointCircleInt(int r, int color) {
int x, int y, d;
/* Valores iniciais */
x = 0;
y = r;
d = 1 - r;
```

```
CirclePoints(x, y, color);
while (y > x) {
if (d < 0){
/* Selecione E */
d = d + 2 * x + 3;
x++;
}else
{
/* Selecione SE */
d = d + 2 * (x - y) + 5;
x++;
y--;  }/*end if*/
CirclePoints(x, y, color);
}/* end while */
}/* end MidpointCircleInt */
```

Elipses



$$Gradiente F(x,y) = \frac{\partial F}{\partial x}\vec{i} + \frac{\partial F}{\partial x}\vec{j} = 2b^2x\vec{i} + 2a^2y\vec{j} \qquad (3.2$$

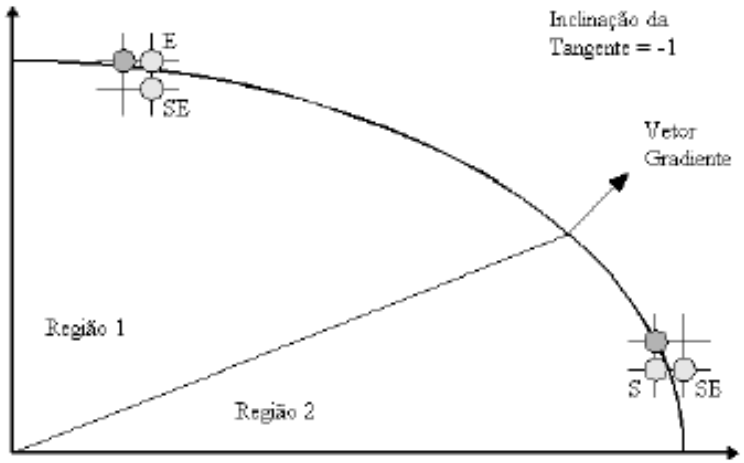


Figura 3.13: As duas regiõe adotadas, definidas pela tangente a 45°.

$F(x, y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$

Elipses

- O limite entre as duas regiões é o ponto cuja inclinação da curva é -1, e este ponto ocorre quando o vetor gradiente tem inclinação igual a 1, isto é quando os componentes nas direções i e j são de magnitudes iguais.
- A componente j do gradiente é maior na região 2 do que na região 1, e vice-versa na região 2.
- Se o próximo "ponto-médio" é:

$$a^2 (y_p - 1/2) \leq b^2(x_p + 1)$$

muda-se da região 1 para a região 2.

Elipses

- Como nos outros algoritmos de “ponto-médio” anteriores, o sinal da variável de decisão d (é o valor da função no “ponto-médio”) será usado para verificar que pixels fazem ou não parte da elipse.

```
void MidpointElipse(int a, int b, int color){
int x, int y;
float d1, d2;
/* Valores iniciais */
x = 0;
y = b;
d1 = b * b - a * a * b + a * a / 4.0;
```

```
ElipsePoints(x, y, color); /* Simetria
    de ordem 4 */

while(a * a * (y - 0.5) > b * b * (x +
    1)){

/* Região 1 */

if (d1 < 0)

d1 = d1 + b * b * (2 * x + 3);

x++

}else{

d1 = d1 + b * b * (2 * x + 3) + a * a *
    (-2 * y + 2);

x++;

y--;

} /*end if*/

ElipsePoints(x, y, color);

}/* end while */
```

```
d2 = b * b * (x + 0.5) * (x + 0.5) + a *
    a * (y - 1) * (y - 1) - a * a * b * b;

while(y > 0){

/* Região 2 */

if (d2 < 0) {

d2 = d2 + b * b * (2 * x + 2) + a * a *
    (-2 * y + 3);

x++;

y--;

}else{

d2 = d2 + a * a * (-2 * y + 3);

y--;

}/*end if*/

ElipsePoints(x, y, color);

}/* end while */

}/*end MidpointElipse*/
```

Referências

- Traina A., Oliveira, M. C. F; Apostial de Computação Gráfica , 2006. Disponível em : http://www.inf.ufes.br/~thomas/graphics/ Data de acesso: 12/3/2015. Originalmente extraída de: http://www.gbdi.icmc.usp.br/documentacao/apostilas/cg/downloads/modelagem.pdf