Computação Gráfica 2D

Isabel Harb Manssour

Porto Alegre, agosto de 2003

Roteiro

- **Estruturas de Dados para Objetos e Cenas**
- *Pipeline* de Visualização 2D
- Transformações Geométricas 2D
- *Pipeline* de Visualização 2D
- Visualização de Objetos 2D
- *Pipeline* de Visualização 2D
- Preenchimento de polígonos

Estruturas de Dados para Objetos e Cenas

- Estudo:
  - Modelagem
  - Estrutura de dados

Estruturas de Dados para Objetos e Cenas

- **Modelagem**
  - Modelo
    - objeto destinado a reproduzir por imitação
    - representação em pequena escala daquilo que se pretende executar em grande escala
    - conjunto de hipóteses sobre a estrutura ou o comportamento de um sistema físico pelo qual se procura explicar ou prever, dentro de uma teoria científica, as propriedades do sistema
  - Modelar
    - representar por meio de modelo
    - assinalar os contornos de
    - dar forma a

Estruturas de Dados para Objetos e Cenas

- Modelagem computacional
  - Modelos não são representados fisicamente
  - As unidades dos dados e parâmetros do modelo computacional são a referência para as dimensões do objeto modelado
- Modelagem (em Computação Gráfica) consiste em todo o processo de descrever um modelo, objeto ou cena, de forma que se possa desenhá-lo

Estruturas de Dados para Objetos e Cenas

- Modelos
  - Utilizados para representar entidades físicas ou abstratas e fenômenos no computador
  - Permitem a realização de simulações, testes e previsão do comportamento das entidades modeladas
- Objetivo
  - elaborar e visualizar imagens
  - representar sua estrutura e/ou comportamento

Estruturas de Dados para Objetos e Cenas

- Importante: projeto e implementação dos modelos
  - Refletir adequadamente as propriedades das entidades
  - Determinar quais informações geométricas e não-geométricas devem ser incluídas no modelo e como estas informações serão incluídas

Estruturas de Dados para Objetos e Cenas

- **Estrutura de dados**
  - Existem várias técnicas para representação e armazenamento dos objetos
  - Formas de representação dependem da natureza dos objetos e das operações/consultas que serão realizadas

Estruturas de Dados para Objetos e Cenas

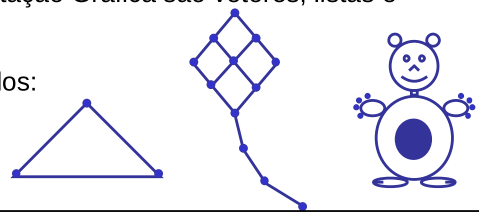
- Formas de representação (armazenadas de diferentes maneiras)
  - Entidades matemáticas com representação analítica conhecida
    - círculos, retas, elipses, curvas, etc.
    - representadas por coeficientes ou parâmetros que permitam sua reconstrução através de um procedimento
  - Pontos amostrados de algum fenômeno ou objeto fisicamente existente
    - armazenamento dos pontos e a identificação do método utilizado para a aproximação da curva ou superfície amostrada

Estruturas de Dados para Objetos e Cenas

- Formas de representação (armazenadas de diferentes maneiras)
  - Enumeração de pontos no plano
    - matriz de pontos (exemplo: imagem de satélite)
  - Decomposição planar
    - áreas de uma superfície podem ser representadas como uma subdivisão sucessiva e hierárquica do plano onde estão definidas
  - Vértices e arestas
    - forma de representação mais comum
    - contornos dos objetos são aproximados para um conjunto de segmentos de reta

Estruturas de Dados para Objetos e Cenas

- Após um objeto, ou um conjunto de objetos, ser modelado, deve-se armazenar o modelo em uma estrutura de dados adequada
- Estruturas de dados mais utilizadas na Computação Gráfica são vetores, listas e tabelas
- Exemplos:

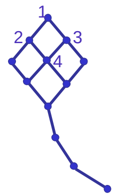


Estruturas de Dados para Objetos e Cenas

- Vetor de pontos

x1,y1	x2,y2	x3,y3

- Duas tabelas
  - Lista de vértices
  - Lista de arestas



vértices (geometria)	arestas (topologia)
x1,y1	1,2
x2,y2	1,3
x3,y3	2,4
x4,y4	3,4
...	...

Estruturas de Dados para Objetos e Cenas

- Quando o armazenamento em uma única estrutura de dados é muito custosa, ou quando é difícil representar um objeto através de primitivas gráficas, utiliza-se a modelagem baseada em procedimento (exemplos: sistemas de partículas e fractais)

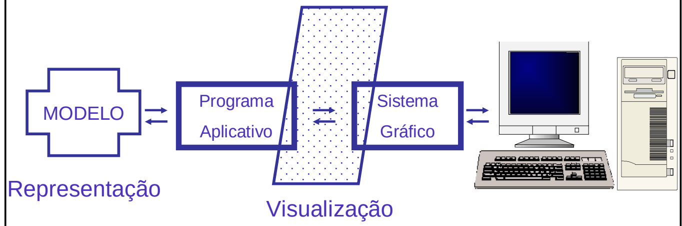
- Outra alternativa é definir um "descritor de objeto" que tenha ponteiros para as diferentes estruturas que compõem o objeto

Roteiro

- Estruturas de Dados para Objetos e Cenas
- ***Pipeline* de Visualização 2D**
- Transformações Geométricas 2D
- *Pipeline* de Visualização 2D
- Visualização de Objetos 2D
- *Pipeline* de Visualização 2D
- Preenchimento de polígonos

Pipeline de Visualização 2D

- Sistemas gráficos interativos



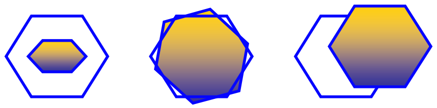
Pipeline de Visualização 2D

Roteiro

- Estruturas de Dados para Objetos e Cenas
- *Pipeline* de Visualização 2D
- **Transformações Geométricas 2D**
- *Pipeline* de Visualização 2D
- Visualização de Objetos 2D
- *Pipeline* de Visualização 2D
- Preenchimento de polígonos

Transformações Geométricas 2D

- Na maioria das aplicações os objetos podem ter suas características alteradas ou podem ser posicionados em uma cena interativamente
- Manipulação dos objetos é realizada através de transformações geométricas
  - Escala
  - Rotação
  - Translação



Transformações Geométricas 2D

- Transformações geométricas
  - Operações matemáticas que permitem alterar uniformemente o **aspecto** de objeto(s), mas não a sua **topologia**
- IMPORTANTE:
  - Alterações **afetam o aspecto** que o objeto irá assumir, mas **não a estrutura**

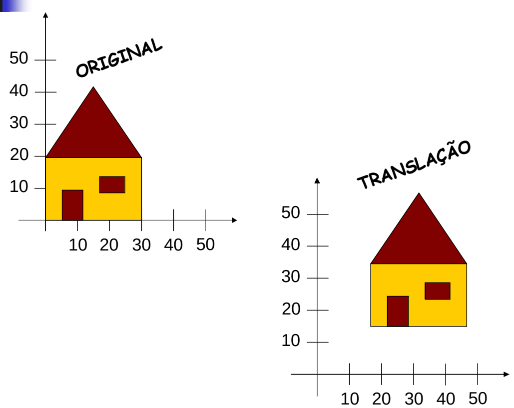
Transformações Geométricas 2D

- Primitivas gráficas
  - São os elementos básicos que compõem um desenho qualquer
  - Exemplos: ponto, reta, círculo, “polilinha”, ...
  - Possuem como parâmetros coordenadas cartesianas de pontos no espaço
  - Podem ser referenciadas por coordenadas absolutas ou relativas

Transformações Geométricas 2D

- Objetos são formados por primitivas gráficas
- As transformações geométricas são aplicadas nas coordenadas cartesianas destas primitivas gráficas
- O modelo do objeto não é alterado, pois as transformações geométricas são aplicadas no momento que este é desenhado

Transformações Geométricas 2D



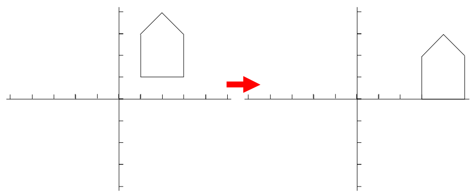
Transformações Geométricas 2D

- **Translação**
  - Objetivo: Trocar um objeto de lugar
  - Funcionamento:
    - Adição de constantes de deslocamento às coordenadas de cada ponto
    - Para cada ponto P(X,Y) é calculado o novo ponto P'(X',Y')

$$\begin{cases} X' = X + \mathbf{T_X} \\ Y' = Y + \mathbf{T_Y} \end{cases} \qquad [X' \;\; Y'] = [X \;\; Y] + [\mathbf{T_X} \;\; \mathbf{T_Y}]$$

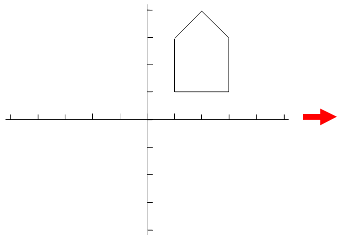
Transformações Geométricas 2D

- Translação
  - Exemplo: $T_X = 2$ e $T_Y = -1$

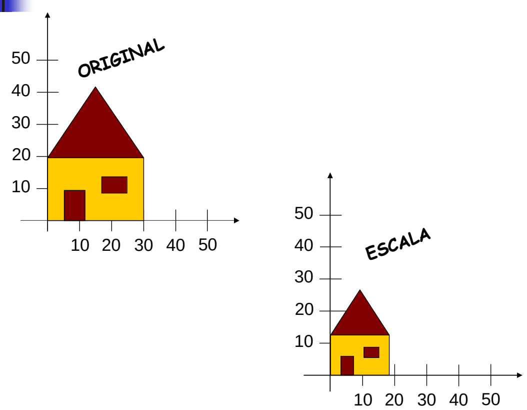


Transformações Geométricas 2D

- Translação
  - Exercício: $T_X = -5$ e $T_Y = -5$



Transformações Geométricas 2D



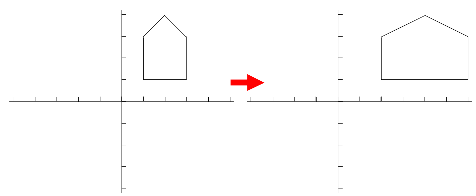
Transformações Geométricas 2D

- **Escala**
  - Objetivo: alterar o tamanho e/ou proporções do objeto aproximando ou afastando os pontos da origem
  - Funcionamento:
    - Multiplicação das coordenadas de cada ponto por fatores de escala
    - Para cada ponto P(X,Y) é calculado o novo ponto P'(X',Y')

$$\begin{cases} X' = X * \mathbf{E_X} \\ Y' = Y * \mathbf{E_Y} \end{cases} \qquad [X' \;\; Y'] = [X \;\; Y] * \begin{bmatrix} \mathbf{E_X} & 0 \\ 0 & \mathbf{E_Y} \end{bmatrix}$$

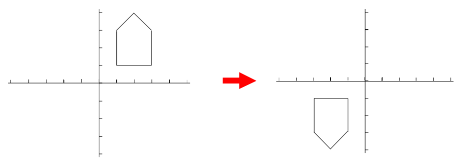
Transformações Geométricas 2D

- Escala
  - Exemplo: $E_X = 2$ e $E_Y = 1$



Transformações Geométricas 2D

- Escala
  - Exemplo: $E_X = -1$ e $E_Y = -1$



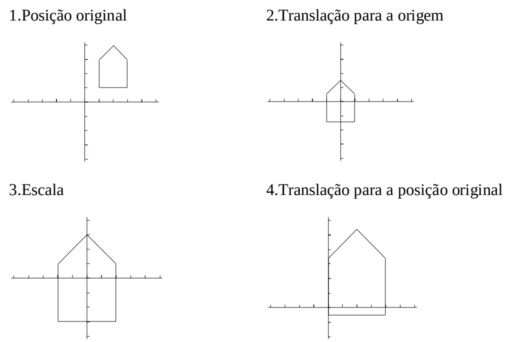
Transformações Geométricas 2D

- Escala
  - Para obter a escala em relação a um ponto qualquer

    Translação para origem→Escala→Translação para posição
  - Exemplo: $E_X = 2$ e $E_Y = 2$

Transformações Geométricas 2D

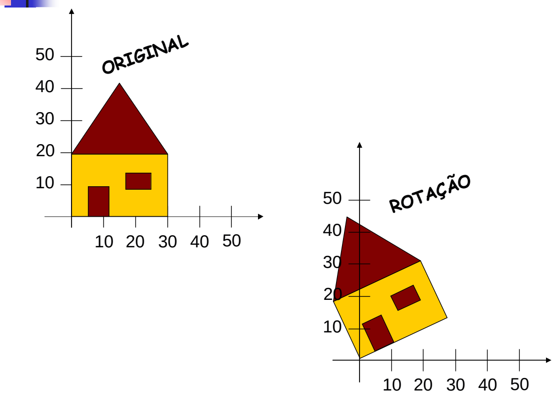
- Escala



Transformações Geométricas 2D

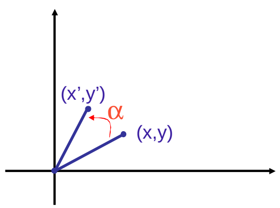
- Escala
  - É possível unir os passos descritos para obter a escala em em torno de um ponto arbitrário em uma única fórmula:
    - $X' = (X-X_r) * E_x + X_r$
    - $Y' = (Y-Y_r) * E_y + Y_r$
    - (X_r, Y_r) é o ponto de referência, em torno do qual será feita a escala

Transformações Geométricas 2D



Transformações Geométricas 2D

- **Rotação**
  - Objetivo: rotação de um objeto através de um ângulo α sobre a origem



Transformações Geométricas 2D

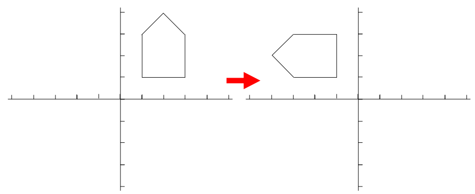
- Rotação
  - Funcionamento
    - Multiplicação das coordenadas de cada ponto pelo seno e coseno do ângulo
    - Para cada ponto P(X,Y) é calculado o novo ponto P'(X',Y')

$$\begin{cases} X' = X * \cos\alpha - Y * \operatorname{sen}\alpha \\ Y' = X * \operatorname{sen}\alpha + Y * \cos\alpha \end{cases}$$

$$[X' \ \ Y'] = [X \ \ Y] * \begin{bmatrix} \cos\alpha & \operatorname{sen}\alpha \\ -\operatorname{sen}\alpha & \cos\alpha \end{bmatrix}$$

Transformações Geométricas 2D

- Rotação
  - Exemplo: $\alpha = 90^{o}$



Transformações Geométricas 2D

- Rotação
  - Para obter a rotação em em torno de um ponto arbitrário

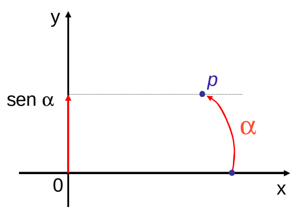
    Translação para origem→Rotação→Translação para posição

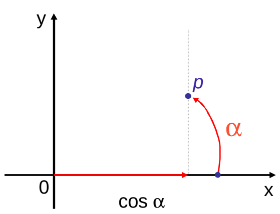
  - Unindo este três passos em uma única fórmula tem-se:
    - $X' = (X - X_r) * \cos\alpha - (Y - Y_r) * \text{sen}\,\alpha$
    - $Y' = (Y - Y_r) * \cos\alpha + (X - X_r) * \text{sen}\,\alpha$
    - (X_r, Y_r) é o ponto de referência, em torno do qual será feita a rotação

Transformações Geométricas 2D

- Rotação
  - Revisão
    - sen X cos
    - graus X radianos

grau	sen	cos
0	0	1
90	1	0
180	0	-1
...	...	...





Transformações Geométricas 2D

- **Coordenadas Homogêneas**
  - Representações matriciais das transformações:
    - Translação ≠ Rotação, Escala
  - As transformações devem ser tratadas da mesma maneira para que possam ser combinadas: **Coordenadas Homogêneas**
    - P(x,y) → P(wx, wy, w), para w≠0
    - Em CG: w = 1
    - Pontos: $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$
    - Matrizes: $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Transformações Geométricas 2D

- Coordenadas Homogêneas
  - Representação das transformações geométricas em coordenadas homogêneas:
    - Translação: $\begin{bmatrix} X' & Y' & 1 \end{bmatrix} = \begin{bmatrix} X & Y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$
    - Escala: $\begin{bmatrix} X' & Y' & 1 \end{bmatrix} = \begin{bmatrix} X & Y & 1 \end{bmatrix} \begin{bmatrix} E_x & 0 & 0 \\ 0 & E_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$
    - Rotação: $\begin{bmatrix} X' & Y' & 1 \end{bmatrix} = \begin{bmatrix} X & Y & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & sen\alpha & 0 \\ -sen\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Transformações Geométricas 2D

- **Composição das Transformações**
  - Usando coordenadas homogêneas e a representação matricial pode-se combinar as matrizes em uma só (multiplicando-as)
  - Depois, multiplica-se cada um dos pontos do(s) objeto(s) pela matriz resultante

Transformações Geométricas 2D

- Composição das Transformações
  - Exemplo:
    - Pontos: $P_1[1, 1, 1]$, $P_2[2, 2, 1]$ e $P_3[3, 1, 1]$
    - Transformações: $E_x=2$ e $E_y=2$, $T_x=2$ e $T_y=1$

$$P_1': \begin{bmatrix}1 & 1 & 1\end{bmatrix}\begin{bmatrix}2 & 0 & 0\\0 & 2 & 0\\0 & 0 & 1\end{bmatrix}=\begin{bmatrix}2 & 2 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\2 & 1 & 1\end{bmatrix}=\begin{bmatrix}4 & 3 & 1\end{bmatrix}$$

$$P_2': \begin{bmatrix}2 & 2 & 1\end{bmatrix}\begin{bmatrix}2 & 0 & 0\\0 & 2 & 0\\0 & 0 & 1\end{bmatrix}=\begin{bmatrix}4 & 4 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\2 & 1 & 1\end{bmatrix}=\begin{bmatrix}6 & 5 & 1\end{bmatrix}$$

$$P_3': \begin{bmatrix}3 & 1 & 1\end{bmatrix}\begin{bmatrix}2 & 0 & 0\\0 & 2 & 0\\0 & 0 & 1\end{bmatrix}=\begin{bmatrix}6 & 2 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\2 & 1 & 1\end{bmatrix}=\begin{bmatrix}8 & 3 & 1\end{bmatrix}$$

Transformações Geométricas 2D

- Composição das Transformações
  - Exemplo combinando as matrizes:
    - Pontos: $P_1[1, 1, 1]$, $P_2[2, 2, 1]$ e $P_3[3, 1, 1]$
    - Transformações: $E_x=2$ e $E_y=2$, $T_x=2$ e $T_y=1$

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

$$P_1': \begin{bmatrix}1 & 1 & 1\end{bmatrix}\begin{bmatrix}2 & 0 & 0\\0 & 2 & 0\\2 & 1 & 1\end{bmatrix}=\begin{bmatrix}4 & 3 & 1\end{bmatrix} \qquad P_2': \begin{bmatrix}2 & 2 & 1\end{bmatrix}\begin{bmatrix}2 & 0 & 0\\0 & 2 & 0\\2 & 1 & 1\end{bmatrix}=\begin{bmatrix}6 & 5 & 1\end{bmatrix}$$

$$P_3': \begin{bmatrix}3 & 1 & 1\end{bmatrix}\begin{bmatrix}2 & 0 & 0\\0 & 2 & 0\\2 & 1 & 1\end{bmatrix}=\begin{bmatrix}8 & 3 & 1\end{bmatrix}$$

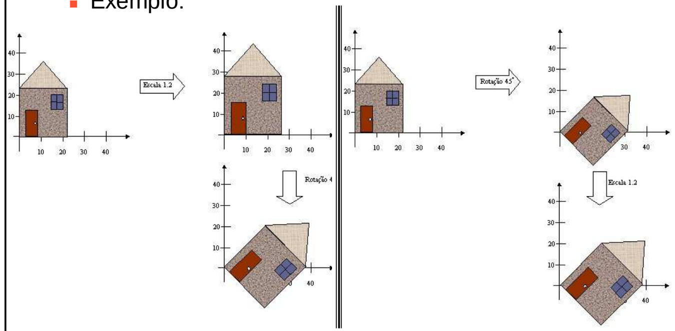
Transformações Geométricas 2D

- **Comutatividade das Transformações**
  - Problema que deve ser considerado na composição das matrizes
  - Ordem da multiplicação das matrizes, assim como da aplicação das transformações geométricas, altera a matriz resultante

rotação . escala = escala . rotação
translação . escala ≠ escala . translação
translação . rotação ≠ rotação . translação

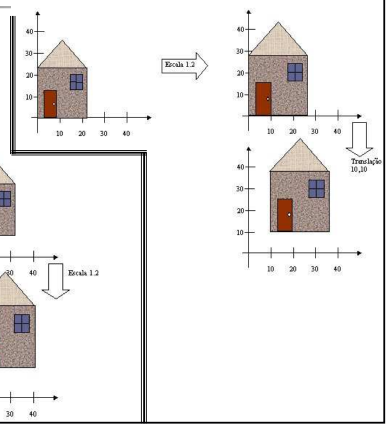
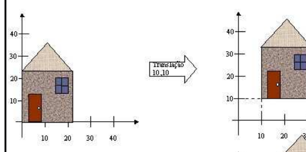
Transformações Geométricas 2D

- Comutatividade das Transformações
  - Exemplo:



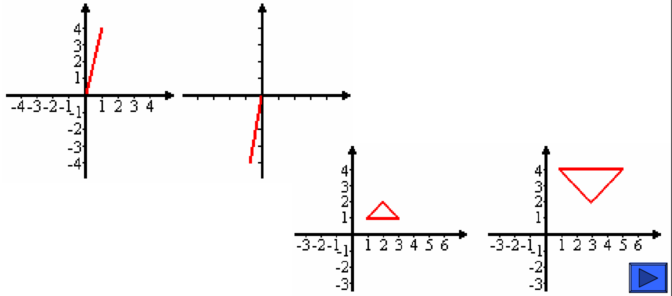
Transformações Geométricas 2D

- Comutatividade das Transformações
  - Exemplo:

Transformações Geométricas 2D

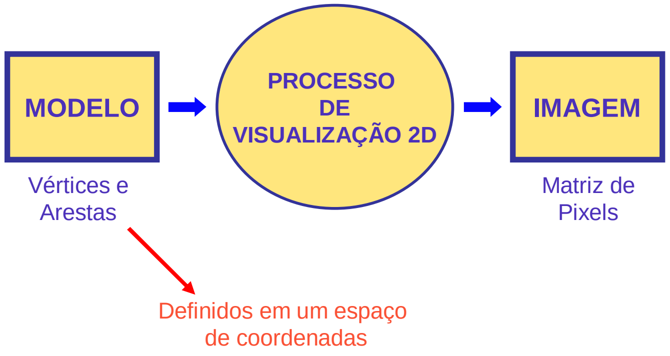
- **Exercício 1**
  - Indique quais transformações geométricas podem ter sido aplicadas nos objetos abaixo:



Roteiro

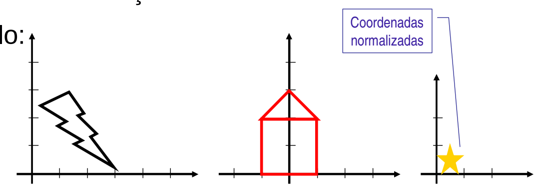
- Estruturas de Dados para Objetos e Cenas
- *Pipeline* de Visualização 2D
- Transformações Geométricas 2D
- ***Pipeline* de Visualização 2D**
- Visualização de Objetos 2D
- *Pipeline* de Visualização 2D
- Preenchimento de polígonos

Pipeline de Visualização 2D



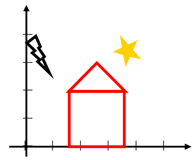
Pipeline de Visualização 2D

- Sistema de coordenadas onde se define o modelo
  - **SRO - Sistema de Referência do Objeto**
- O SRO consiste em um espaço de coordenadas adequado à criação dos modelos
- Portanto, a escolha do tamanho do SRO é feita para facilitar a criação do modelo
- Exemplo:

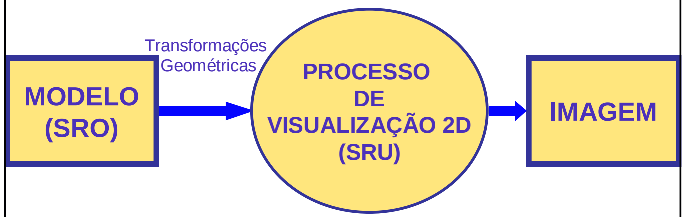


Pipeline de Visualização 2D

- Depois de modelados, os objetos são posicionados no Universo
- Universo
  - Região do plano/espaço utilizada em uma aplicação
  - **SRU - Sistema de Referência do Universo**
- Exemplo:



Pipeline de Visualização 2D

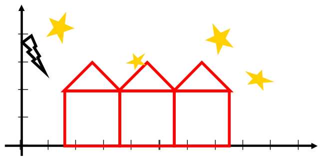


Roteiro

- Estruturas de Dados para Objetos e Cenas
- *Pipeline* de Visualização 2D
- Transformações Geométricas 2D
- *Pipeline* de Visualização 2D
- **Visualização de Objetos 2D**
- *Pipeline* de Visualização 2D
- Preenchimento de polígonos

Visualização de Objetos 2D

- Muitas vezes os objetos de uma cena são derivados de um mesmo modelo
- Exemplo:



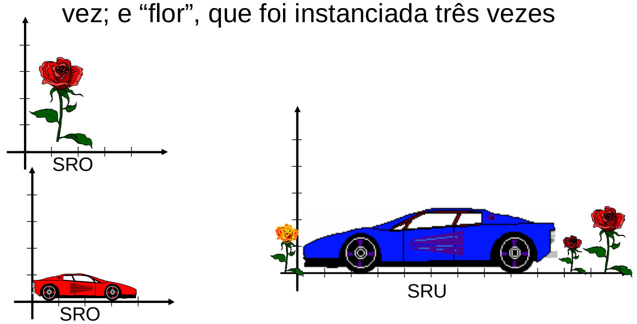
- Neste caso, os objetos são instâncias de um mesmo modelo

Visualização de Objetos 2D

- Sendo assim, uma das etapas do processo de visualização 2D é o **instanciamento**, que permite que se crie cópias do modelo
- Instâncias
  - Possuem a mesma estrutura básica, diferindo apenas no tamanho, na cor e no posicionamento
  - São alterações do modelo baseadas em parâmetros pré-definidos chamados parâmetros de instanciamento
  - Exemplos de parâmetros de instanciamento:
    - Tamanho, posição, orientação, cor, material,

Visualização de Objetos 2D

- Na figura abaixo foram criadas instâncias de dois modelos: “carro”, que foi instanciado uma vez; e “flor”, que foi instanciada três vezes



Visualização de Objetos 2D

- Para armazenar as instâncias de uma aplicação pode-se usar tabelas ou listas. Por exemplo:

Modelo	Cor	Tx	Ty	Ex	Ey	α
Carro						
Flor						
Flor						
Flor						

- Vantagens na utilização de instancias
  - Tornam o universo da aplicação menor, pois para representar objetos diferentes basta armazenar o nome do modelo e os parâmetros de instanciamento
  - Permitem representar um grande número de objetos diferentes a partir de um mesmo modelo

Visualização de Objetos 2D

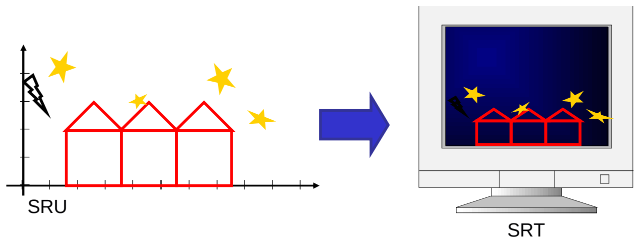
- Resumindo:
  - Os modelos são criados em um SRO para posterior instanciamento no universo (SRU)
  - As informações do modelo, bem como das instâncias, referem-se à aplicação e não ao dispositivo
  - Em outras palavras, os modelos são criados independentes do dispositivo, com as coordenadas definidas em relação ao sistema de referência adotado

Visualização de Objetos 2D

- Portanto, para visualizar os modelos é necessário realizar uma conversão das coordenadas do modelo para coordenadas proporcionais às dimensões da tela

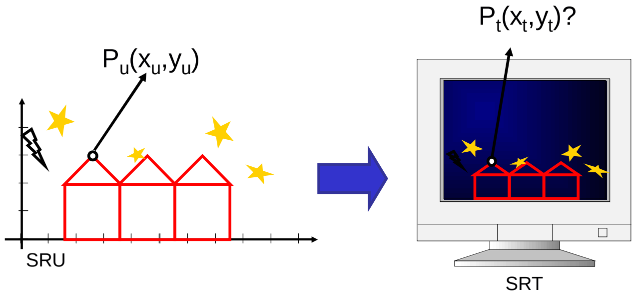
Visualização de Objetos 2D

- Dois sistemas de coordenadas envolvidos:
  - SRU: Sistema de Referência do Universo
  - SRT: Sistema de Referência da Tela



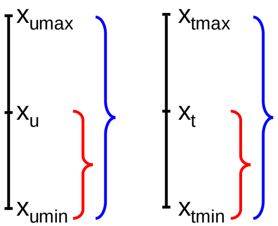
Visualização de Objetos 2D

- Como fazer o mapeamento?



Visualização de Objetos 2D

- Para encontrar a coordenada x_t

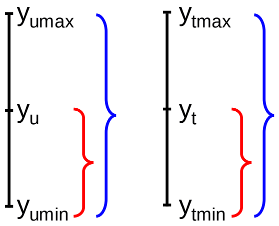


$$\frac{x_t - x_{tmin}}{x_u - x_{umin}} = \frac{x_{tmax} - x_{tmin}}{x_{umax} - x_{umin}}$$

$$x_t = \frac{(x_{tmax} - x_{tmin}) * (x_u - x_{umin})}{x_{umax} - x_{umin}} + x_{tmin}$$

Visualização de Objetos 2D

- Para encontrar a coordenada y_t é o mesmo procedimento...



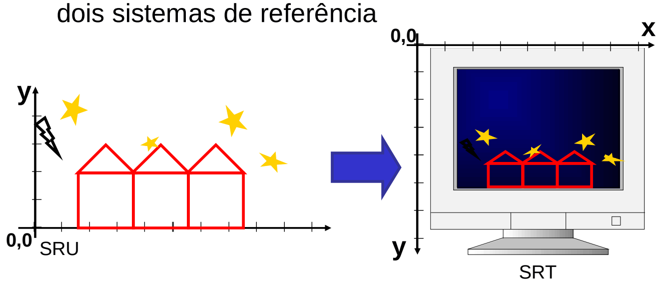
$$\frac{y_t - y_{tmin}}{y_u - y_{umin}} = \frac{y_{tmax} - y_{tmin}}{y_{umax} - y_{umin}}$$

$$y_t = \frac{(y_{tmax} - y_{tmin}) * (y_u - y_{umin})}{y_{umax} - y_{umin}} + y_{tmin}$$

...correto? NÃO

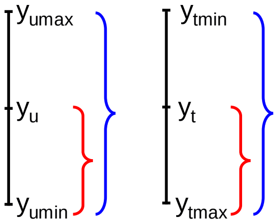
Visualização de Objetos 2D

- Deve-se inverter a relação entre o eixo y dos dois sistemas de referência



Visualização de Objetos 2D

- Para encontrar a coordenada y_t



$$\frac{y_t - y_{tmax}}{y_u - y_{umin}} = \frac{y_{tmin} - y_{tmax}}{y_{umax} - y_{umin}}$$

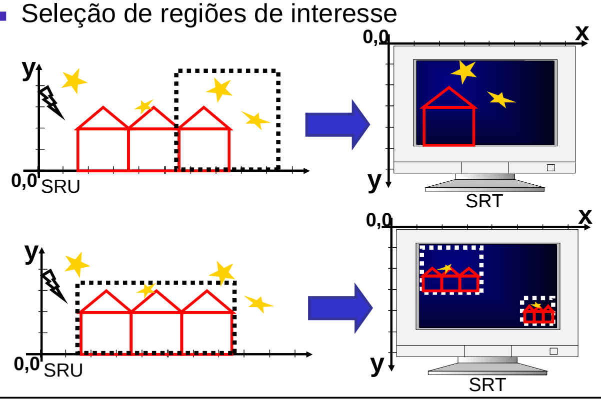
$$y_t = \frac{(y_{tmin} - y_{tmax}) * (y_u - y_{umin})}{y_{umax} - y_{umin}} + y_{tmax}$$

Visualização de Objetos 2D

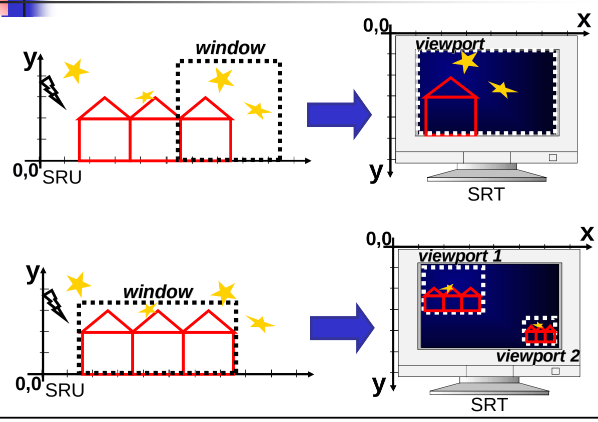
- Janela de seleção ou ***window***
  - Área do universo de interesse
  - Coordenadas do SRU
- Janela de exibição ou ***viewport***
  - Área da tela para onde o conteúdo da *window* será mapeado
  - Coordenadas do SRT
- *Zoom*?
  - Diminuir o tamanho da *window*

Visualização de Objetos 2D

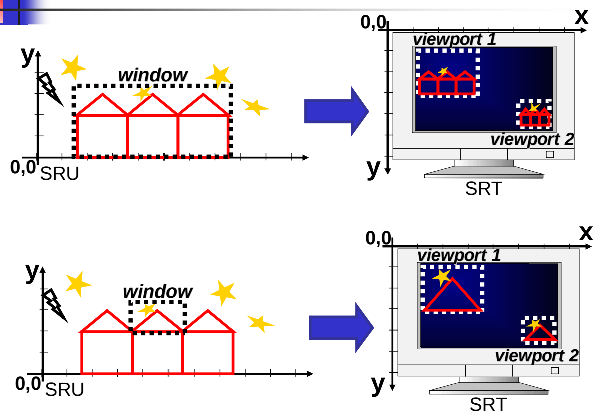
- Seleção de regiões de interesse



Visualização de Objetos 2D



Visualização de Objetos 2D

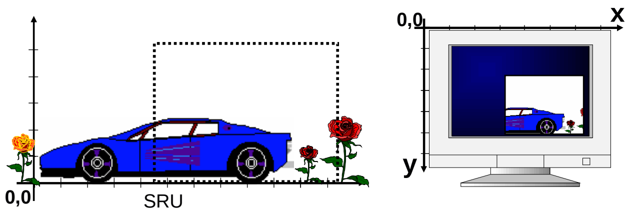


Visualização de Objetos 2D

- Exercício 2
  - Considerando um universo onde x_{umin}=0, x_{umax}=10, y_{umin}=0, y_{umax}=8, e que o objeto definido abaixo foi mapeado para um dispositivo de 1280x1024, apresente as coordenadas do objeto no SRT.
    - P_1(3,2)
    - P_2(4,7)
    - P_3(5,2)
    - P_4(2,6)
    - P_5(6,6)

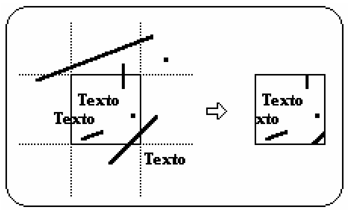
Visualização de Objetos 2D

- Uma etapa de recorte torna-se necessária para eliminar os objetos que ficam, total ou parcialmente, fora da *window*



Visualização de Objetos 2D

- **Recorte**
  - Processo de retirada dos objetos que não estão dentro da *window*
  - Elementos
    - **Pontos**
    - **Retas**
    - **Caracteres**



Visualização de Objetos 2D

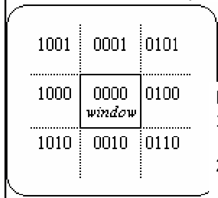
- **Recorte**
  - **Pontos**
    - Simples
    - Considerando os limites da *window*, Xmin, Xmax, Ymin e Ymax, P(X,Y) é visível se
      - Xmin ≤ X ≤ Xmax
      - Ymin ≤ Y ≤ Ymax
    - Caso estas condições não sejam satisfeitas, o ponto não está visível e deve ser recortado

Visualização de Objetos 2D

- **Recorte**
  - **Retas**
    - Mais importante
    - Basicamente, consiste em verificar quais são os pontos de intersecção da reta com os “lados” da window
    - Algoritmo de Cohen-Sutherland

Visualização de Objetos 2D

- **Recorte**
  - Algoritmo de Cohen-Sutherland para Recorte de Retas
    - Primeiro passo: codificar os extremos da linha a ser recortada



Número de quatro *bits*:
1° *bit*: em 1 se o ponto está à esquerda da *window*,
em 0 se o ponto não está à esquerda da *window*;
2° *bit*: em 1 se o ponto está à direita da *window*,
em 0 se o ponto não está à direita da *window*;
3° *bit*: em 1 se o ponto está abaixo da *window*,
em 0 se o ponto não está abaixo da *window*;
4° *bit*: em 1 se o ponto está acima da *window*,
em 0 se o ponto não está acima da *window*.

Visualização de Objetos 2D

- **Recorte**
  - Algoritmo de Cohen-Sutherland para Recorte de Retas
    - Segundo passo:
      - Decidir se a linha está **TODA DENTRO da *window*** e pode ser exibida sem recorte (verificar se os códigos possuem valor 0000, ou então fazer um *OR* dos dois códigos - se der zero a linha está toda dentro)
      - Decidir se a linha está **TODA FORA da *window*** e não pode ser exibida (realizar um *AND* dos códigos, se o resultado for diferente de 0 a linha está toda fora)

Visualização de Objetos 2D

- **Recorte**
  - Algoritmo de Cohen-Sutherland para Recorte de Retas
    - Terceiro passo: prossegue com o algoritmo...

1) Cálculo dos códigos de P1 e de P2
   Se P1 estiver fora da *window* seguir para o passo 2;
   Senão trocar P1 com P2;
2) Verificar, pelo código, se o P1 encontra-se à esquerda da *window*
   Se não estiver seguir para o passo 3;
   Se estiver
      Calcular ponto de intersecção Pi da reta P1-P2 com lado esquerdo da *windo*
      Colocar Pi em P1 e recalcular o código de P1;
      Seguir para o passo 6;
3) Verificar, pelo código, se o P1 encontra-se à direita da *window*
   Se não estiver seguir para o passo 4;
   Se estiver
      Calcular ponto de intersecção Pi da reta P1-P2 com lado direito da *window*;
      Colocar Pi em P1 e recalcular o código de P1;
      Seguir para o passo 6;

Visualização de Objetos 2D

4) Verificar, pelo código, se o P1 encontra-se acima da *window*
   Se não estiver seguir para o passo 5;
   Se estiver
      Calcular ponto de intersecção Pi da reta P1-P2 com lado de cima da window
      Colocar Pi em P1 e recalcular o código de P1;
      Seguir para o passo 6;
5) Verificar, pelo código, se o P1 encontra-se abaixo da *window*
   Se não estiver seguir para o passo 6;
   Se estiver
      Calcular ponto de intersecção Pi da reta P1-P2 com lado de baixo da windo
      Colocar Pi em P1 e recalcular o código de P1;
      Seguir para o passo 6;
6) Verificar se a nova linha P1-P2 está toda dentro ou toda fora da *window*
   Se (P1 *OR* P2) = 0
      A linha recortada está toda dentro da window;
      Encerra o algoritmo;
   Senão Se (P1 *AND* P2) ≠ 0
      A linha está toda fora da window;
      Encerra o algoritmo;
   Senão Volta ao passo 1

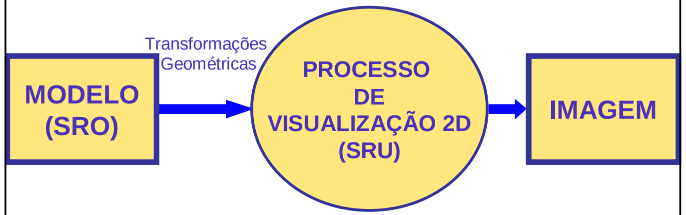
Visualização de Objetos 2D

- **Recorte**
  - **Caracteres**
    - Caractere composto por uma coleção de segmentos de reta
      - Cada linha deve ser recortada individualmente
      - Lento, e, além disso, geralmente utiliza-se uma matriz de pontos para representar um caractere
    - Tratar o caractere como uma entidade "indivisível"
      - *String* pode ser recortado caractere a caractere, considerando o seu envelope
    - Tratar todo texto ou *string* como "indivisível"
    - Calcular a a intersecção da borda da *window* com a matriz de pontos que representa o caractere

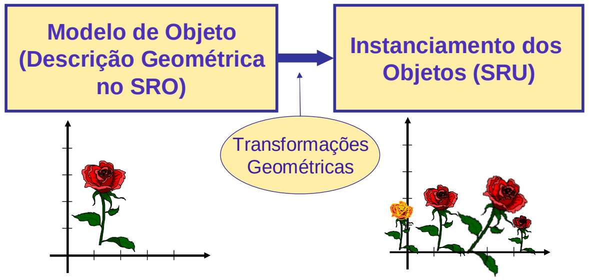
Roteiro

- Estruturas de Dados para Objetos e Cenas
- *Pipeline* de Visualização 2D
- Transformações Geométricas 2D
- Pipeline de Visualização 2D
- Visualização de Objetos 2D
- ***Pipeline* de Visualização 2D**
- Preenchimento de polígonos

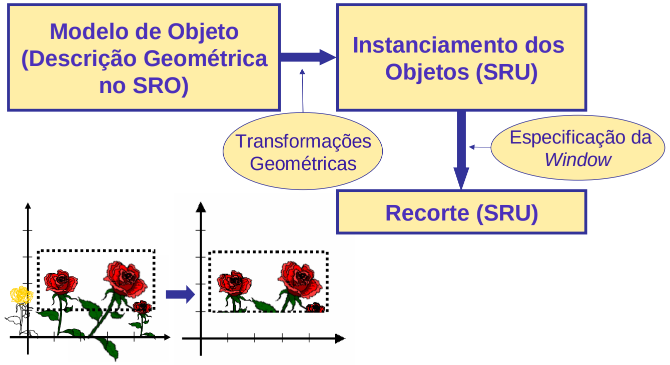
Pipeline de Visualização 2D



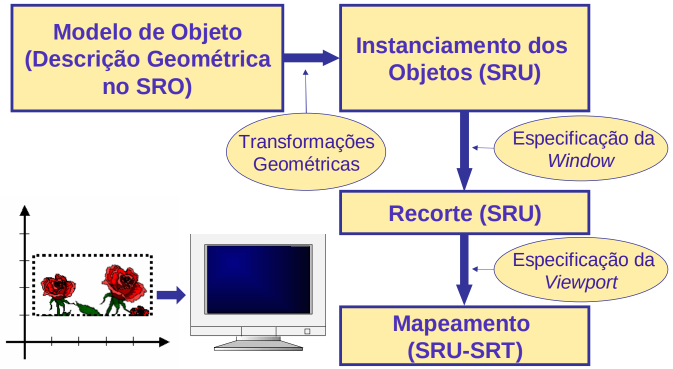
Pipeline de Visualização 2D



Pipeline de Visualização 2D

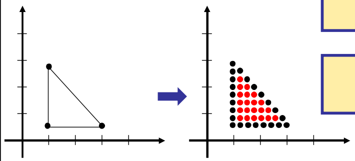


Pipeline de Visualização 2D



Pipeline de Visualização 2D

Modelo de Objeto (Descrição Geométrica no SRO) → Instanciamento dos Objetos (SRU) → Recorte (SRU) → Mapeamento (SRU-SRT) → Conversão Vetorial-Matricial

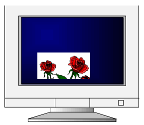


Pipeline de Visualização 2D

Modelo de Objeto (Descrição Geométrica no SRO) → Instanciamento dos Objetos (SRU) → Recorte (SRU) → Mapeamento (SRU-SRT) → Conversão Vetorial-Matricial → **Imagem(Matriz de pontos)**

Pipeline de Visualização 2D

Modelo de Objeto (Descrição Geométrica no SRO) → Instanciamento dos Objetos (SRU) → Recorte (SRU) → Mapeamento (SRU-SRT) → Conversão Vetorial-Matricial → Imagem(Matriz de pontos)



Pipeline de Visualização 2D

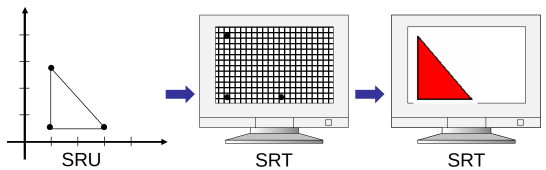
Modelo de Objeto (Descrição Geométrica no SRO) → Instanciamento dos Objetos (SRU) → Recorte (SRU) → Mapeamento (SRU-SRT) → Conversão Vetorial-Matricial → **Imagem(Matriz de pontos)**

Roteiro

- Estruturas de Dados para Objetos e Cenas
- *Pipeline* de Visualização 2D
- Transformações Geométricas 2D
- Pipeline de Visualização 2D
- Visualização de Objetos 2D
- *Pipeline* de Visualização 2D
- **Preenchimento de polígonos**

Preenchimento de Polígonos

- Uma das etapas do *pipeline* de visualização 2D é a conversão vetorial-matricial
- Esta etapa inclui:
  - Desenhos de linhas
  - Preenchimento de polígonos



Preenchimento de Polígonos

- Algoritmo básico para desenho de linhas
  - Calcula extremidades da linha na tela (P1,P2)
  - Calcula coeficientes da equação da reta:

$$\boxed{y = mx + b} \qquad m = \frac{y2 - y1}{x2 - x1}$$

$$b = y1 - m \,.\, x1$$

  - Exemplo
    - Desenhar linha de P1=(1,1) a P2=(5,7)
    - ***for* x *in* [1,5]**
      - **y = m . x + b**
      - **liga ponto (x,y)**

Fonte: C. Freitas, J. Scharcanski & S. Olabarriaga, UFRGS (2001)

Preenchimento de Polígonos

- Algoritmo básico para desenho de linhas
  - Problemas
    - inclinação das linhas?
    - muito lento (operações com números reais)
- Algoritmos mais eficientes para desenho de linhas
  - Buscam
    - eliminar ou reduzir operações com números reais
    - aproveitar coerência espacial (similaridade de valores referentes a pixels vizinhos)
  - Exemplos
    - DDA: *Data Differential Analyser*
    - *Bresenham*

Fonte: C. Freitas, J. Scharcanski & S. Olabarriaga, UFRGS (2001)

Preenchimento de Polígonos

- Algoritmo DDA

```
x = x1, y = y1
dx = x2 - x1
dy = y2 - y1
Se abs(dx) > abs(dy)
   steps = abs(dx)
Senão
   steps = abs(dy)
increX = dx/steps
increY = dy/steps
plot( round(x), round(y) )
for k in [0, steps]
      x = x+increX, y = y+increY
      plot( round(x), round(y) )
```

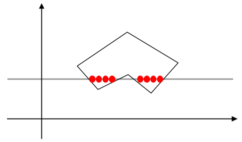
  - Exemplo
    - Desenhar linha de P1=(1,1) a P2=(5,7)

Preenchimento de Polígonos

- Abordagens para preenchimento de polígonos em sistemas matriciais
  - Determinar os intervalos nos quais cada linha varrida atravessa a área do polígono
  - Começar a "pintar" a partir de um ponto de dentro do polígono até encontrar as suas bordas

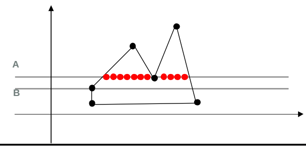
Preenchimento de Polígonos

- Algoritmo de preenchimento de polígono: ***Scan-Line***
  - Para cada linha varrida que atravessa o polígono são localizados os pontos de interseção com as arestas do polígono
  - Pontos de interseção são ordenados da esquerda para a direita
  - *Pixels* entre cada par de interseção são "setados" com a cor especificada



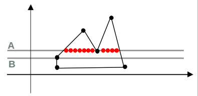
Preenchimento de Polígonos

- Algoritmo de preenchimento de polígono: ***Scan-Line***
  - Tratamento especial: interseção com os vértices do polígono
    - *Scan-line* que passa pelo vértice atravessa duas arestas do polígono na mesma posição, o que faz com que dois pontos sejam adicionados à lista de interseções da *scan-line*
    - Exemplo próxima figura:
      - *scan-line* A:correto (interseção ocorre com um nº par de arestas)
      - *scan-line* B: três pontos de interseção (identificação incorreta da área do polígono)



Preenchimento de Polígonos

- Algoritmo de preenchimento de polígono: ***Scan-Line***
  - Tratamento especial para interseção com os vértices do polígono
    - Processamento adicional para determinar corretamente os pontos do interior do polígono
    - Diferença entre A e B:
      - *scan-line* A: as duas arestas estão acima da *scan-line*
      - *scan-line* B: as duas arestas que compartilham um vértice estão em lados opostos da *scan-line*
    - Os vértices que conectam arestas que ficam em lados opostos no *scan-line* são adicionados apenas uma vez na lista de pontos de interseção



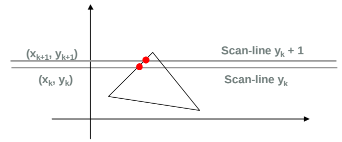
Preenchimento de Polígonos

- Algoritmo de preenchimento de polígono: ***Scan-Line***
  - Para fazer o processamento se considera a relação entre as propriedades das partes que compõem a cena
  - Cálculos incrementais
    - Ao longo de uma *scan-line*
    - Entre *scan-lines* (inclinação de uma aresta é constante)

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

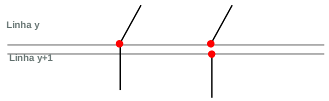
$$y_{k+1} - y_k = 1$$

$$x_{k+1} = x_k + \frac{1}{m}$$



Preenchimento de Polígonos

- Algoritmo de preenchimento de polígono: ***Scan-Line***
  - Tratamento especial para interseção com os vértices do polígono
    - As arestas do polígono podem ser processadas para determinar quando a intersecção com os vértices será adicionada uma ou duas vezes na lista de intersecções
    - Neste caso, uma maneira de garantir quando se deve contar o vértice como um ou dois pontos de interseção consiste em diminuir uma das arestas e dividir o vértice em dois, como mostra a figura abaixo



Preenchimento de Polígonos

- Algoritmo de preenchimento de polígono: ***Boundary-Fill***
  - Processamento começa a partir de um ponto do interior do polígono, que é então preenchido (ou pintado) até que sua borda seja encontrada
  - Se a borda possui uma única cor, o preenchimento é realizado *pixel* a *pixel* até que a cor da borda seja encontrada

Preenchimento de Polígonos

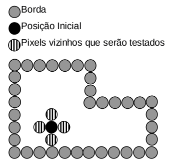
- Algoritmo de preenchimento de polígono: ***Boundary-Fill***
  - Parâmetros de entrada:
    - Um ponto (x,y) do interior do polígono
    - Cor da sua borda
  - Posições "vizinhas" do ponto (x,y) são testadas:
    - Se não for a cor da borda o ponto é pintado com a cor de preenchimento (até que todos os *pixels* do polígono tenham sido testados)
  - Dois exemplos de métodos recursivos
    - *BoundaryFill4* e *FloodFill4*

Preenchimento de Polígonos

```
void BoundaryFill4 (int x, int y, int fill, int boundary)
{
    int current;
    current = getPixel(x,y);
    if (  (current != boundary)  &&  (current != fill)  )
    {
          setColor(fill);
          setPixel(x,y);
          BoundaryFill4 (x+1, y, fill, boundary);
          BoundaryFill4 (x-1, y, fill, boundary);
          BoundaryFill4 (x, y+1, fill, boundary);
          BoundaryFill4 (x, y-1, fill, boundary);
    }
}
```

Preenchimento de Polígonos

```
void FloodFill4 (int x, int y, int fillcolor, int oldcolor)
{
    if (  getPixel(x,y) == oldcolor  )
    {
          setColor(fillcolor);
          setPixel(x,y);
          FloodFill4 (x+1, y, fillcolor, oldcolor);
          FloodFill4 (x-1, y, fillcolor, oldcolor);
          FloodFill4 (x, y+1, fillcolor, oldcolor);
          FloodFill4 (x, y-1, fillcolor, oldcolor);
    }
}
```



Referências

- PINHO, Márcio. S. **Manipulação de Imagens**. Disponível em http://www.inf.pucrs.br/~pinho/CG/Aulas/Img/IMG.htm. Esta página também está disponível em http://www.inf.pucrs.br/~flash/cg/Aulas/Img/IMG.htm.
- FOLEY, James D., et al. **Computer Graphics: Principles and Practice**. 2ª Ed., New York, Addison Wesley, 1990.
- HEARN, Donald; BAKER, M. Pauline. **Computer Graphics - C Version**. 2nd Ed. Upper Saddle River, New Jersey: Prentice Hall, 1997, 652 p.