

Licenciatura em Engenharia Informática

Ano Lectivo de 2011/12

COMPUTAÇÃO GRÁFICA e INTERFACES

Aulas Teóricas

Prof. Manuel Próspero dos Santos

ps@di.fct.unl.pt

Departamento de Informática



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Interface

The **user interface** (or *Human Machine Interface*) is the aggregate of means by which people (*the users*) interact with a particular machine, device, computer program or other complex tool (*the system*). The user interface provides means of:

- Input, allowing the users to manipulate a system
- Output, allowing the system to produce the effects of the users' manipulation.

Ex.:

A **graphical user interface (GUI)** is a type of user interface which allows people to interact with a computer and computer-controlled devices which employ graphical icons, visual indicators or special graphical elements called "widgets", along with text, labels or text navigation to represent the information and actions available to a user. The actions are usually performed through direct manipulation of the graphical elements. Use of this acronym led to creation of the neologism *guituitive* (an interface which is intuitive).

From Wikipedia

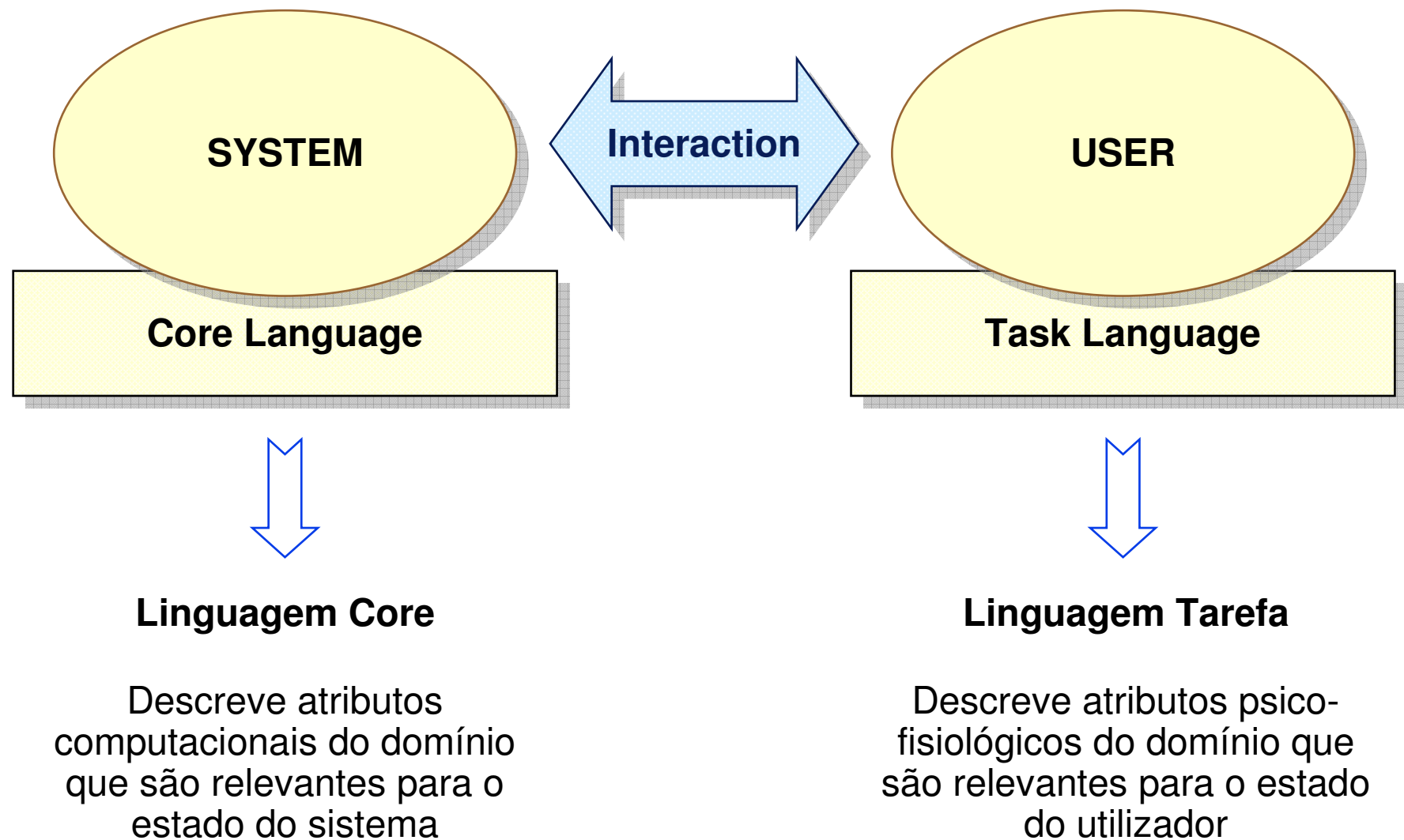
Noções gerais sobre Interação

INTERAÇÃO: a comunicação entre **utilizador** e **sistema** (sentido lato).

Propósito da Interação: ajudar o utilizador a atingir determinados objetivos num domínio de aplicação específico.

Tarefas: operações do utilizador que manipulam os conceitos do domínio da aplicação, tendo como resultado os objetivos pretendidos.

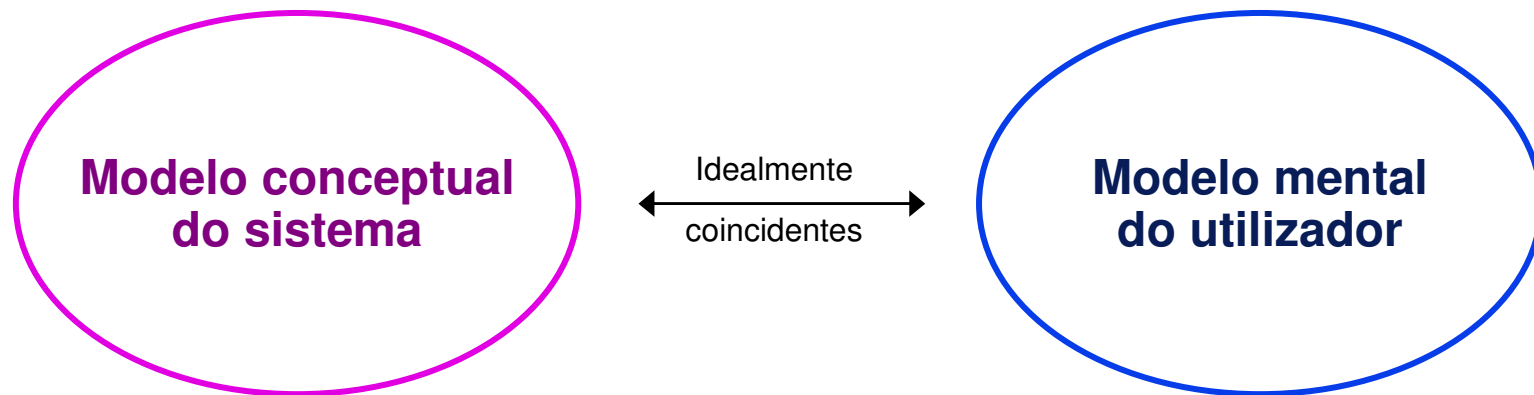
Os conceitos usados no design de um sistema e na descrição do utilizador apresentam-se diferentes, pelo que SISTEMA e UTILIZADOR são componentes distintas, cada qual com a sua linguagem própria para exprimir conceitos no domínio da aplicação.



Obs.: Para um utilizador, outros eventuais utilizadores farão parte do sistema.

Modelo conceptual de um sistema

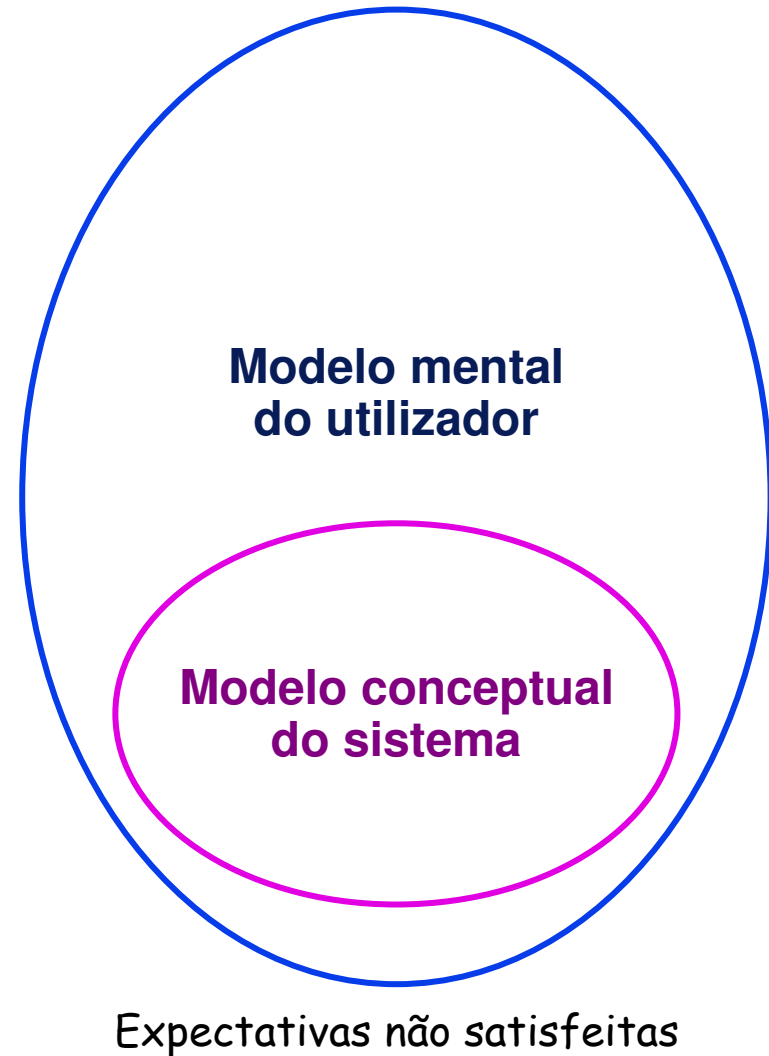
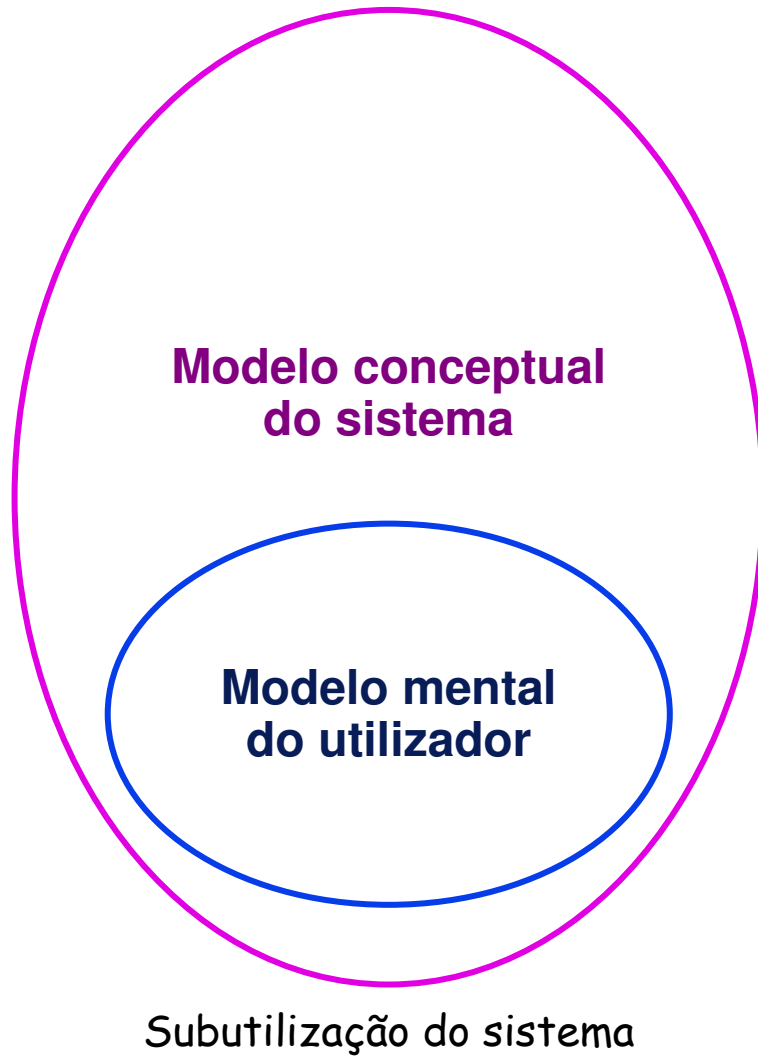
- ♦ Estrutura **conceptual** geral através da qual são apresentadas as funcionalidades do sistema.
- ♦ Idealização da responsabilidade do designer do sistema.
- ♦ Interface e documentação disponível visam facilitar o desenvolvimento de um modelo **mental** desse sistema, por parte dos utilizadores.



Modelo mental do utilizador

- ♦ O modelo mental inclui teoria e hipóteses sobre as partes e processos invisíveis de um sistema e como devem relacionar-se com as partes e os processos apreendidos através da interface e da documentação.
- ♦ O utilizador baseia-se neste seu modelo mental para
 - ♦ determinar ações e reações do sistema
 - ♦ prever acontecimentos futuros
 - ♦ procurar causas para acontecimentos observáveis
 - ♦ elaborar mnemónicas que auxiliam a relembrar relações e acontecimentos
 - ♦ compreender outros sistemas análogos.

Eventuais diferenças extremas entre os modelos



As eras de produção tecnológica de Imagens

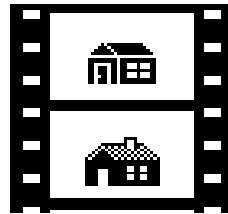
Imprensa



As eras de produção tecnológica de Imagens

Imprensa

Fotografia

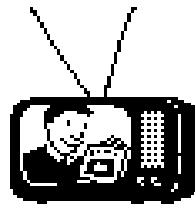


As eras de produção tecnológica de Imagens

Imprensa

Fotografia

Televisão



As eras de produção tecnológica de Imagens

Imprensa

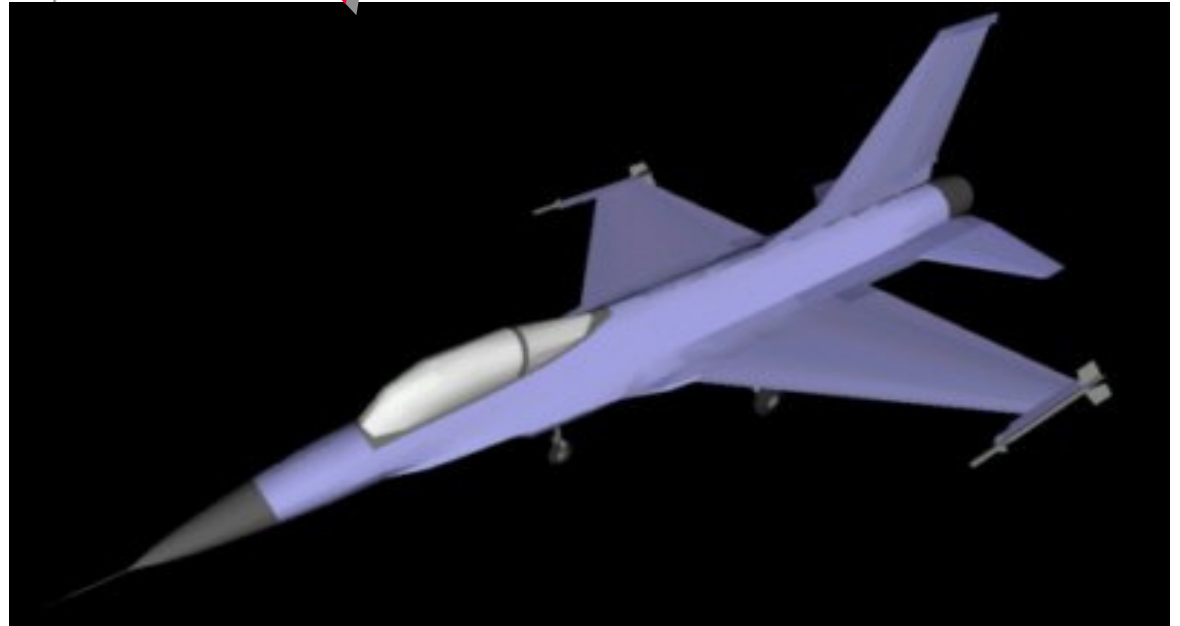
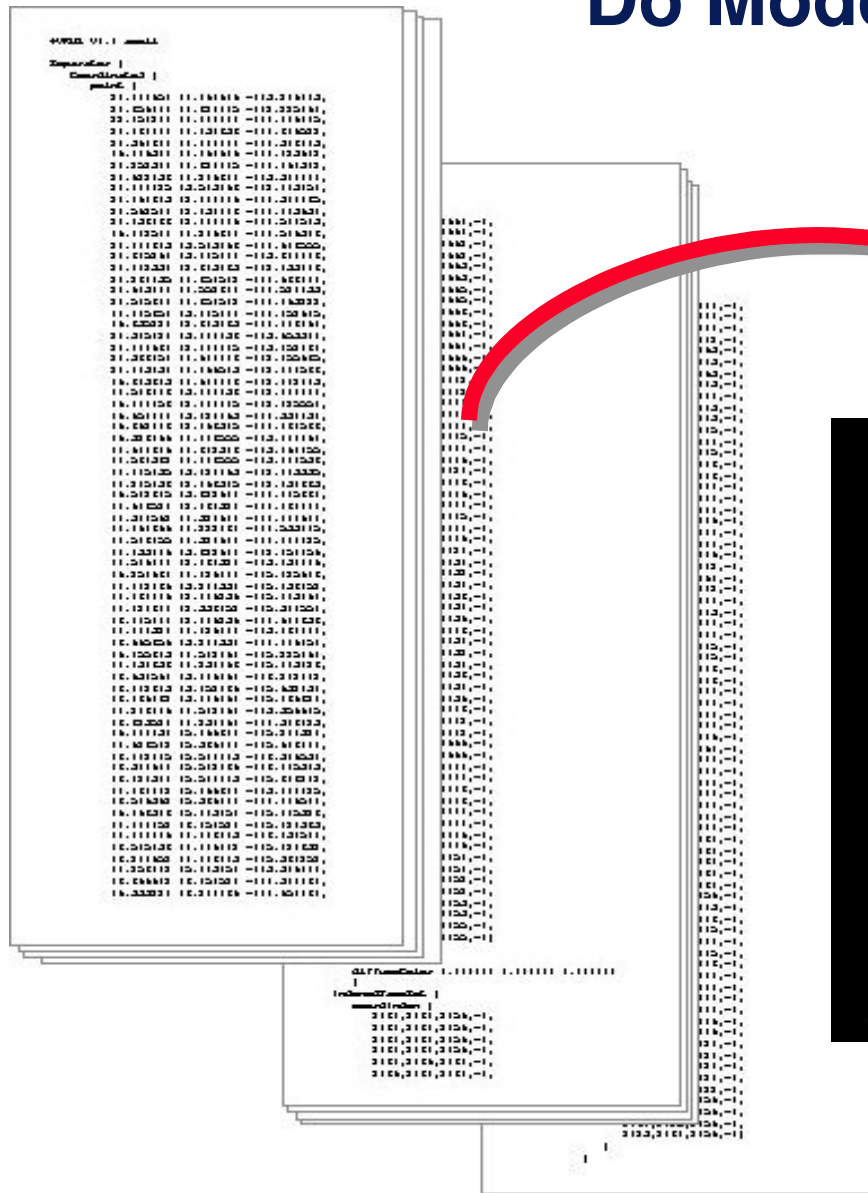
Fotografia

Televisão

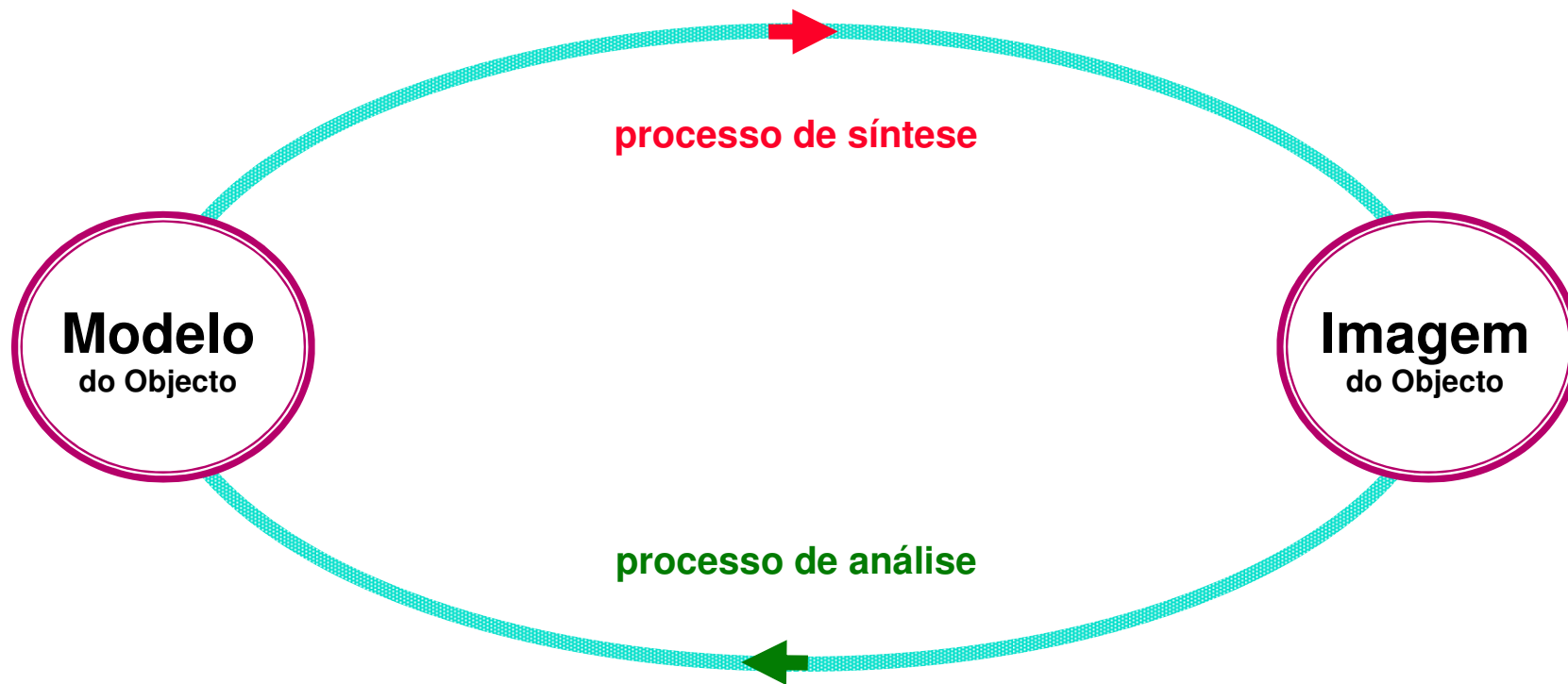
Computador



Do Modelo à Imagem



Computação Gráfica



Processamento de Imagem

A intersecção de CG com PI deveu-se à utilização de Raster Displays para a visualização

COMPUTAÇÃO GRÁFICA

Criação, armazenagem e manipulação de modelos de objetos e subsequentes imagens por meio de computador.

Sempre que exista interatividade, o utilizador controla dinamicamente as imagens quanto ao seu conteúdo, formato, tamanho, cor ou outro possível atributo, numa superfície de visualização e através de dispositivos de interação.

What is Computer Graphics?

“Computer graphics is the **technology with which pictures**, in the broadest sense of the word (synthetic graphics as well as grayscale and color images), **are captured or generated, presented, manipulated, digitally processed** in the appropriate form for the respective application and **merged with other, nongraphical application data**.

Computer graphics also includes the **computer-supported integration** and manipulation of these pictures **with other kinds of data, such as audio, speech and video** (to create multimedia systems) as well as corresponding advanced dialogue and interactive technologies.”



Prof. Dr.-Ing. José Luís Encarnação

INI GraphicsNet Foundation – Darmstadt
www.inigraphicsnet-stiftung.de

VANTAGENS DA COMPUTAÇÃO GRÁFICA

- ❑ um dos meios mais naturais para comunicação com o computador
- ❑ síntese de objetos concretos ou abstratos
- ❑ solução para problemas científicos, com grande volume de dados ou com variação dinâmica destes
- ❑ visualização e controlo do movimento para simulação e jogos, eventualmente com integração num sistema multimédia (p.ex.: inclusão de som para melhor *feedback*)
- ❑ criação de Realidade Virtual e Realidade Aumentada
- ❑ solução para maior produtividade, melhor qualidade, menor esforço de projeto e custos de conceção mais reduzidos (...)

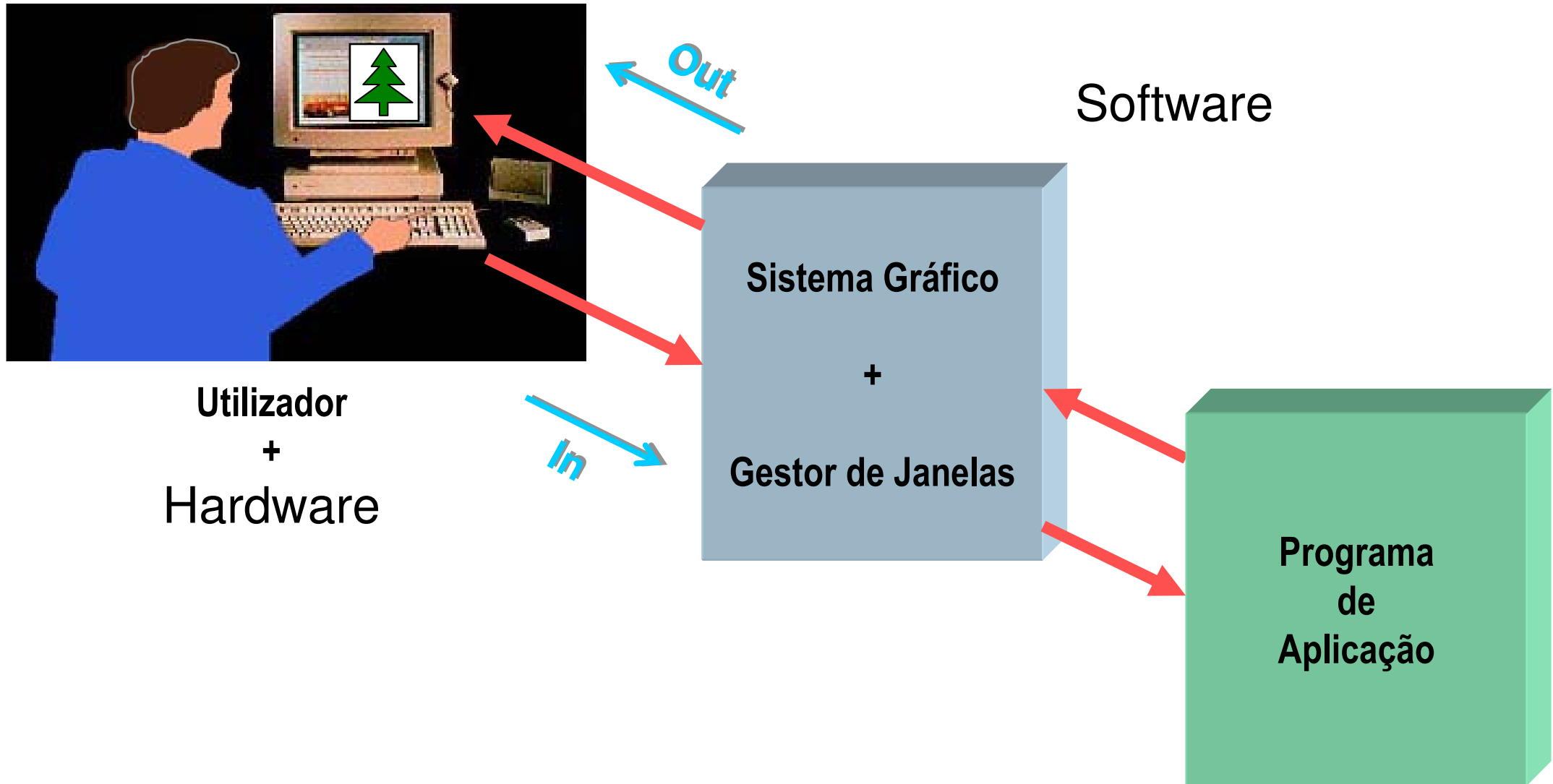
APLICAÇÕES

- Interfaces com o utilizador
- Diagramas para gestão, ciência e tecnologia
- Publicação eletrónica e escritório automático
- CAD e CADD
- Simulação e animação
- Publicidade, diversão, arte e educação
- Controlo de processos
- Cartografia (...)

O que se pede, em geral, ao Programador de aplicações gráficas?

- **Conhecimentos matemáticos sobre geometria 2D / 3D**
- **Construção de linguagens de interação e definição do diálogo**
- **Manipulação de modelos, incluindo os hierárquicos**
- **Conhecimentos de diversos algoritmos, de acordo com a aplicação**
- **Programas robustos, com facilidades de extensão, reutilização e manutenção (engenharia de software)**
- **...**

Modelo conceptual de base para o Programador



Dispositivos físicos de Entrada de Datos

IN

- ▶ Teclado
- ▶ Rato
- ▶ Joystick
- ▶ Trackball
- ▶ Mesa digitalizadora (*Tablet*)
- ▶ Telecomando

▶ (...)



Dispositivos físicos para Visualização

OUT

- ◀ Monitor
- ◀ Impressora
- ◀ Óculos para Realidade Virtual (HMD)
- ◀ Traçador (*Plotter*)



Traçador de mesa



Traçador de rolo



Paradigmas de usabilidade

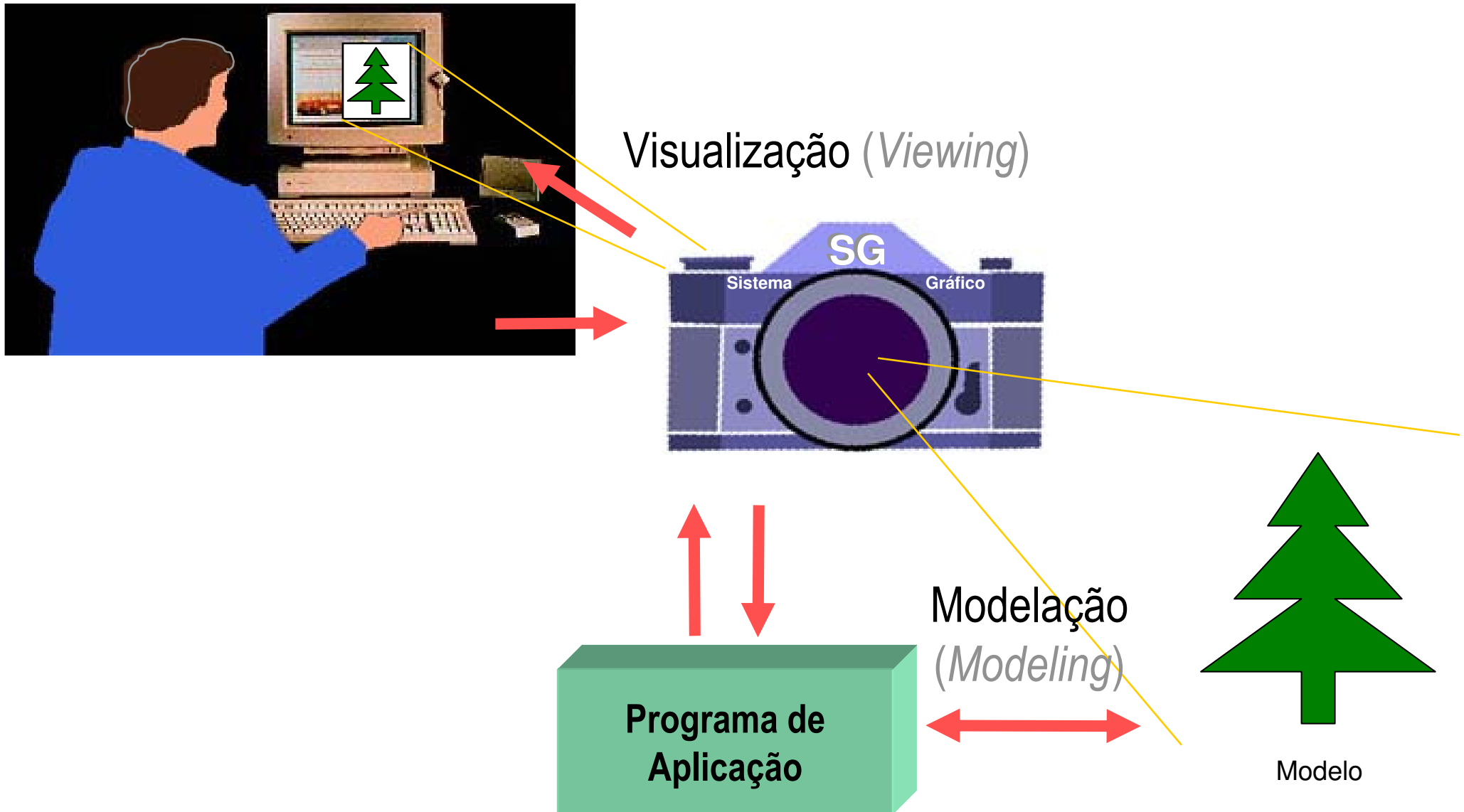
O desenvolvimento de produtos e/ou a sua utilização servem-se, frequentemente, de sistemas conhecidos (com sucesso) como paradigmas (ie, como exemplos).

A metáfora (ie, transporte da significação de um conceito para outro por relação de semelhança) é um caso paradigmático que, se for adequada, pode acelerar substancialmente e de maneira simples a familiarização dos utilizadores com os sistemas em causa (aumentando a usabilidade através da interface).

Exemplos:

- ♦ O tampo de secretária (*desktop*) no ecrã do computador.
- ♦ A tartaruga na linguagem de programação Logo, produzindo os chamados *turtle graphics*.
- ♦ A máquina fotográfica, tanto na utilização como na programação de aplicações gráficas 3D.

A metáfora da máquina fotográfica



POSSÍVEIS CRITÉRIOS PARA CLASSIFICAÇÃO DAS APLICAÇÕES

- **Tipo de objeto e representação pictórica**
 - 2D + Linha, Cor acromática, Cor cromática
 - 3D + Linha c/ ou s/ efeitos, Cor e sombras c/ ou s/ efeitos
- **Tipo de interação**
 - Grau de controlo do utilizador sobre as imagens
- **Objetivo da imagem**
 - Fim em si ou meio para outra finalidade
- **Relação objeto – imagem: temporal e/ou lógica**
 - Uma imagem relacionada com outras, ou no tempo (e.g. filme) ou por assemblagem (objeto composto)

O PROCESSO DE GERAÇÃO DA IMAGEM

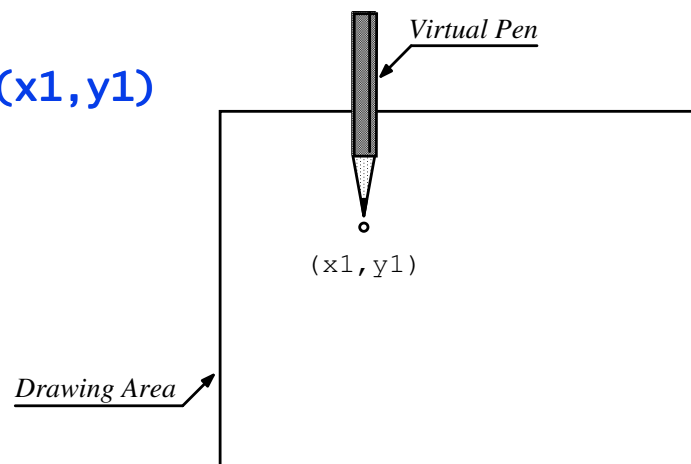
- **Construção do Modelo da Aplicação e correspondente Representação**
Coleção de dados, representando objetos físicos ou abstratos, e de relações, bem como de operações que ajudam a definir a sua estrutura e/ou comportamento. [Definição de âmbito mais alargado do que o de Computação Gráfica]
- **Descrição do Modelo ao Sistema Gráfico**
 - O quê ?
 - Como ?

Em termos de primitivas gráficas universais (pontos, linhas, polígonos, ...) em vez de estruturas de dados normalizadas dependentes da aplicação (mais restritivas e ineficientes em muitos casos).
 - Modo de interação (Pessoa-Computador) para exploração visual ou edição ?
- **Criação da Imagem**
Representação gráfica (realizada pelo SG) para ser vista pelo utilizador.

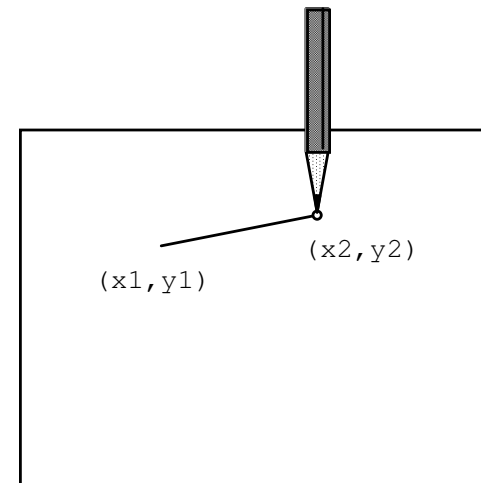
Programação com Caneta Virtual (metáfora)

Desenho de segmentos de reta só c/ Coordenadas Absolutas

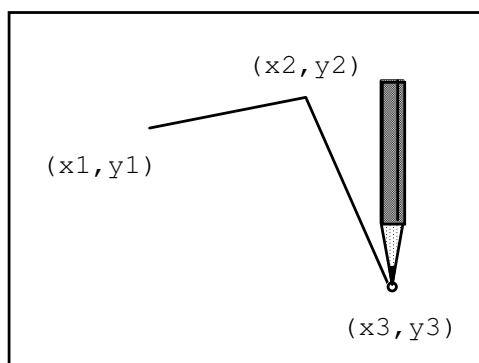
MOVE (x1,y1)



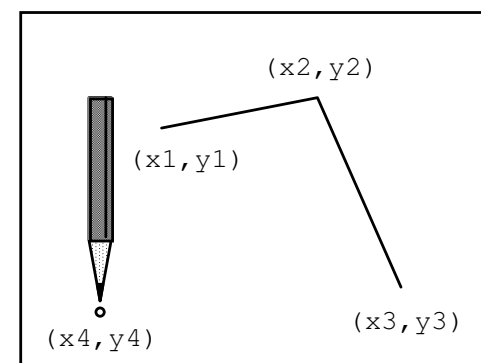
LINE (x2,y2)



LINE (x3,y3)



MOVE (x4,y4)

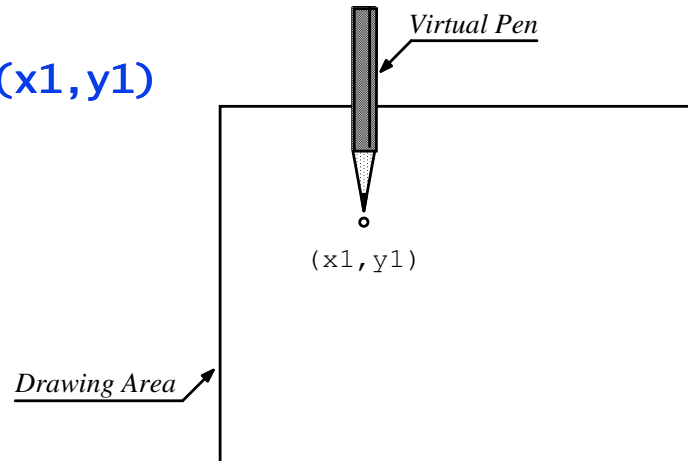


Conceito de Posição Corrente: Posição, nesse instante, da caneta virtual

Programação com Caneta Virtual (metáfora)

Desenho de segmentos de reta c/ Coordenadas Relativas

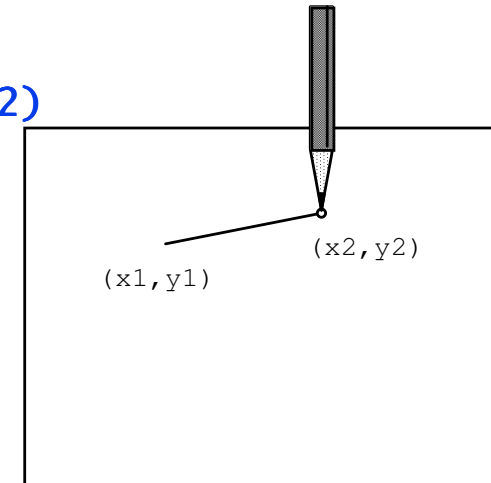
MOVE (x1,y1)



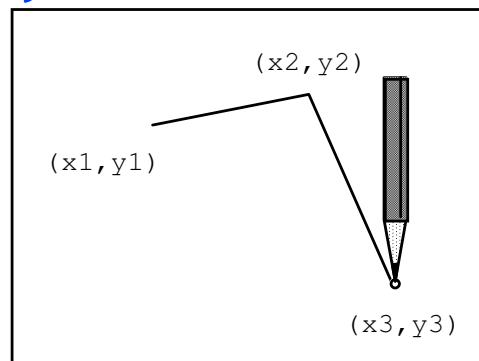
LINE_R ($\Delta x2, \Delta y2$)

$$\Delta x2 = x2 - x1$$

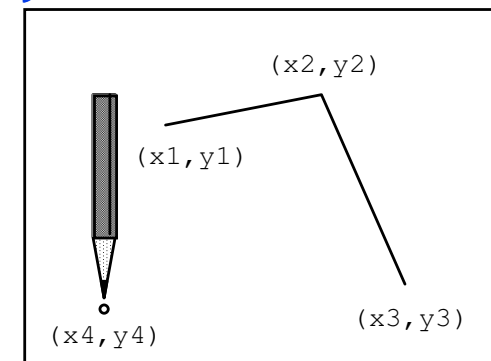
$$\Delta y2 = y2 - y1$$



LINE_R ($\Delta x3, \Delta y3$)

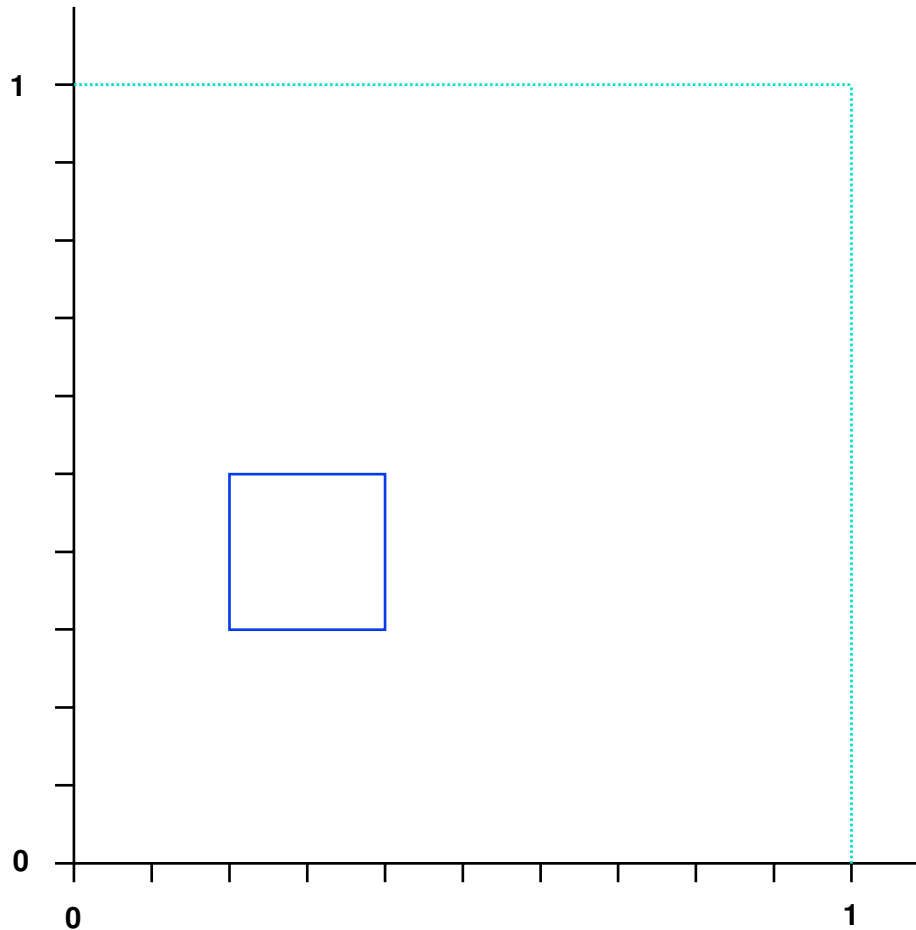


MOVE_R ($\Delta x4, \Delta y4$)



Conceito de Posição Corrente

Exemplo de aplicação no desenho de um quadrado



Dispositivo de Visualização Normalizado:

Pontos especificados em
NDC (Normalized Device Coordinates):
as coordenadas variam entre 0 e 1

C/ Posição Corrente:

```
MOVE (0.2, 0.3)
LINE (0.4, 0.3)
LINE (0.4, 0.5)
LINE (0.2, 0.5)
LINE (0.2, 0.3)
```

S/ Posição Corrente:

```
LINE (0.2, 0.3, 0.4, 0.3)
LINE (0.4, 0.3, 0.4, 0.5)
LINE (0.4, 0.5, 0.2, 0.5)
LINE (0.2, 0.5, 0.2, 0.3)
```

Características do uso da Posição Corrente:

- + Reduz o número de argumentos requeridos para cada primitiva de saída gráfica.
- Aumenta o número de primitivas de saída gráfica.
- Não tem independência da ordem de execução.
- Implica valores arbitrários em certas transições de estado do sistema gráfico (indefinição da posição corrente).

Normas gráficas ISO/ANSI (a nível API) que **não usam** o conceito de Posição Corrente:

GKS

Graphical Kernel System

PHIGS

Programmer's Hierarchical Interactive Graphics System

ISO 7942-1985(E)

Output functions

GKS functions

5.3 Output functions

POLYLINE

WSAC,SGOP L0a

Parameters:

In number of points

(2..n) I

In points

WC n × P

Effect: A sequence of connected straight lines is generated, starting from the first point and ending at the last point. The current values of the polyline attributes, as given by the GKS state list (see 6.4), are bound to the primitive. The polyline attributes are listed in 4.4.2.

If, after the workstation transformation, all points coincide, no error is generated and whether anything is drawn is workstation dependent.

References:

4.4.1

4.4.2

4.4.3

4.5.3

Errors:

5 GKS not in proper state: GKS shall be either in the state WSAC or in the state SGOP

100 Number of points is invalid

(...)

ISO/IEC 9592-1 : 1989 (E)

Output primitive functions

PHIGS Functional Specification

5.3 Output primitive functions

POLYLINE 3

(PHOP,*,STOP,*)

Parameters:

In point list

MC L(P3)

Effect: This function fully specifies the three dimensional form of the polyline primitive. Depending upon the 'edit mode', a "polyline 3" element is inserted into the open structure after the 'element pointer' or replaces the element pointed at by the 'element pointer'. The 'element pointer' is then updated to point to this "polyline 3" structure element. The value specified by the parameter is associated with the element.

When an element of this type is interpreted a connected sequence of straight lines is generated starting from the element's first point and ending at the element's last point. The current values of the polyline attributes, as defined in the PHIGS traversal state list (see 6.4), are bound to the primitive. The polyline attributes are described in 4.5.3.

A "polyline 3" element with less than two points will be placed in the open structure. When a "polyline 3" element with less than two points is interpreted it will have no visual effect.

References: 4.5.1 4.5.2 4.5.3 4.5.12 4.5.13 4.5.14 4.5.15 4.5.16 4.5.17 4.6.3

Errors:

005 Ignoring function, function requires state (PHOP,*,STOP,*)

(...)

Programa exemplo:

```
/* Access the PHIGS C binding. */
/* The next line is implementation dependent. */
#include <phigs.h>

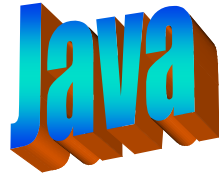
#define WS 1
#define NUM_PTS 5
#define BOX 1

Ppoint3 box_pts[] =

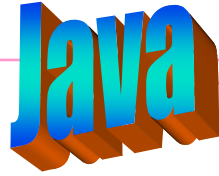
/* Define the data
   for the box. */
{{0.2,0.3,0.0}, {0.4,0.3,0.0},
 {0.4,0.5,0.0}, {0.2,0.5,0.0},
 {0.2,0.3,0.0}};

main()
{
    popenphigs(0, 0);          /* Open PHIGS. */
    popenws(WS, 0, 0);        /* Open a workstation. */
    popenstruct(BOX);         /* Define the box structure. */
    ppolyline3(NUM_PTS, box_pts); /* Insert a polyline 3 element
                                to draw the box. */
    pclosestruct();           /* Close the structure. */
    ppoststruct(WS, BOX, 1.0); /* Post the structure. */
    sleep(60);                /* Sleep for 60 seconds. */
    pclosews(WS);             /* Close the workstation. */
    pclosephigs();            /* Close PHIGS. */
}
```

Existem *bindings* (ie, regras sintáticas normalizadas) para outras linguagens



Exemplo de utilização de primitivas gráficas (nível API)



```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
/**
 * @author M. Próspero
 */
public class Casa extends java.applet.Applet {

    public void init() {
        setBackground(Color.WHITE);
    }

    public void paint(Graphics screen) {

        screen.setColor(Color.BLACK); // paredes:
        screen.drawLine(120, 220, 120, 140);
        screen.drawLine(280, 220, 280, 140);

        screen.setColor(Color.BLUE); // porta:
        int[] xPorta={140, 140, 180, 180};
        int[] yPorta={220, 160, 160, 220};
        int pontos=4;
        screen.drawPolyline(xPorta,yPorta,pontos);

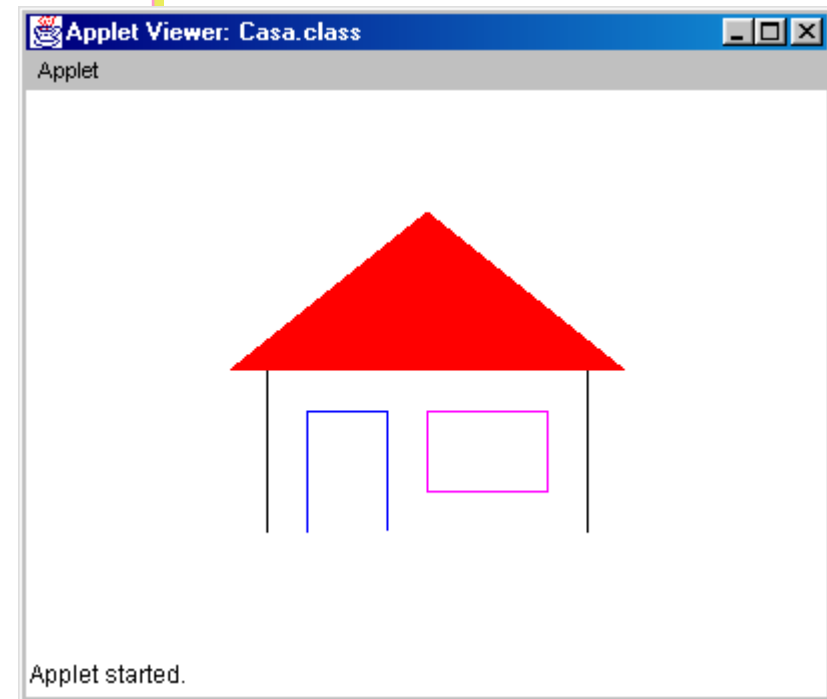
        screen.setColor(Color.MAGENTA); // janela:
        screen.drawRect(200, 160, 60, 40);

        screen.setColor(Color.RED); // telhado:
        int[] xTelhado={100, 200, 300};
        int[] yTelhado={140, 60, 140};
        Polygon telhado=new Polygon(xTelhado, yTelhado, 3);
        screen.fillPolygon(telhado);
    }
}
```

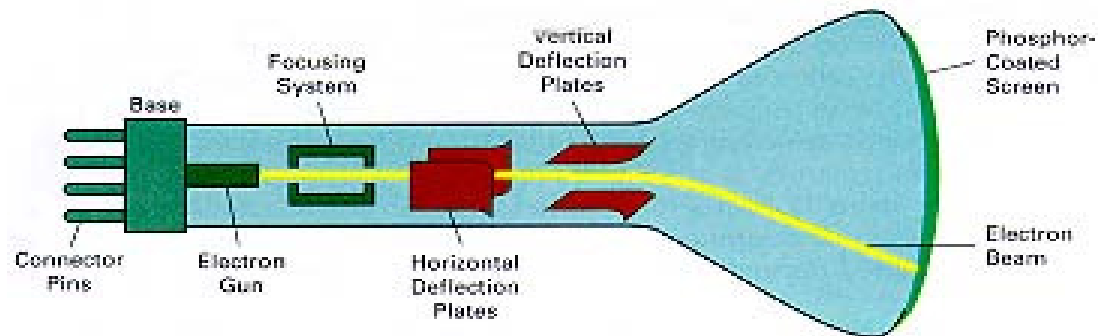
```
<HTML>
<BODY BGCOLOR="#FFCC99">

<applet code="Casa.class" width=400 height=280></applet>

</BODY></HTML>
```



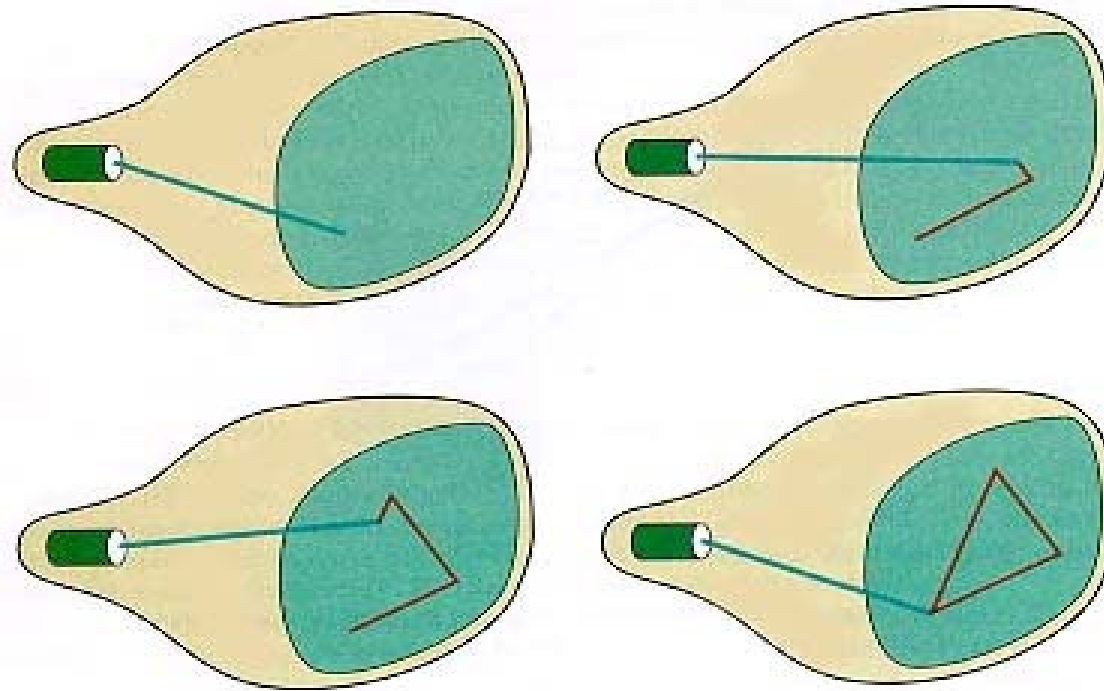
Visualização de imagens num ecrã CRT



Tubo de Raios Catódicos

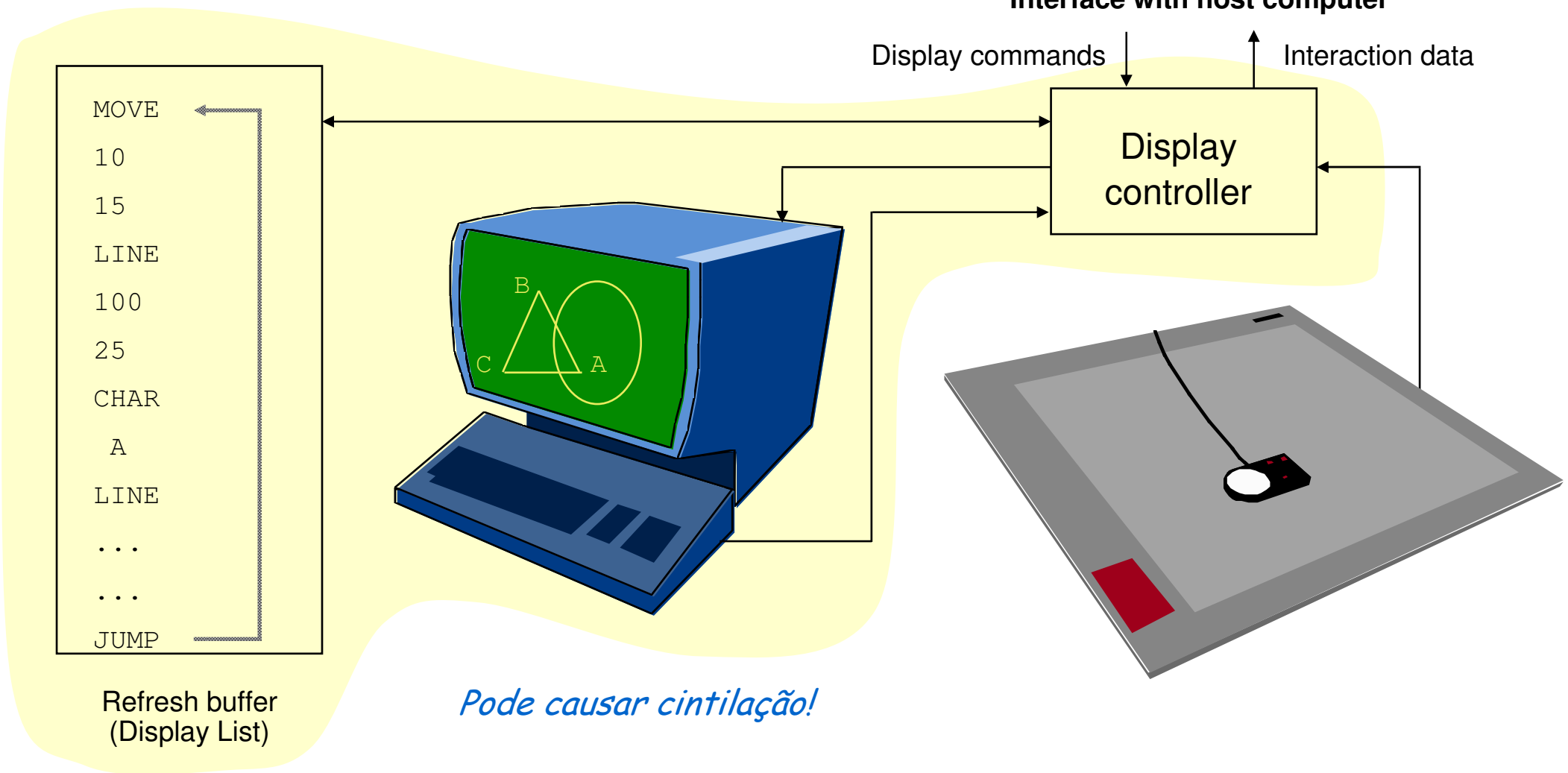
O feixe de elétrons funciona como caneta que as placas deflectoras fazem deslocar.

Dispositivo Vetorial (ou Caligráfico)



A baixa persistência do fósforo (10-60 μ s) obriga ao refrescamento do ecrã (50 Hz ou mais), sendo os gráficos apresentados continuamente redesenhados.

Anos 60



Pode causar cintilação!

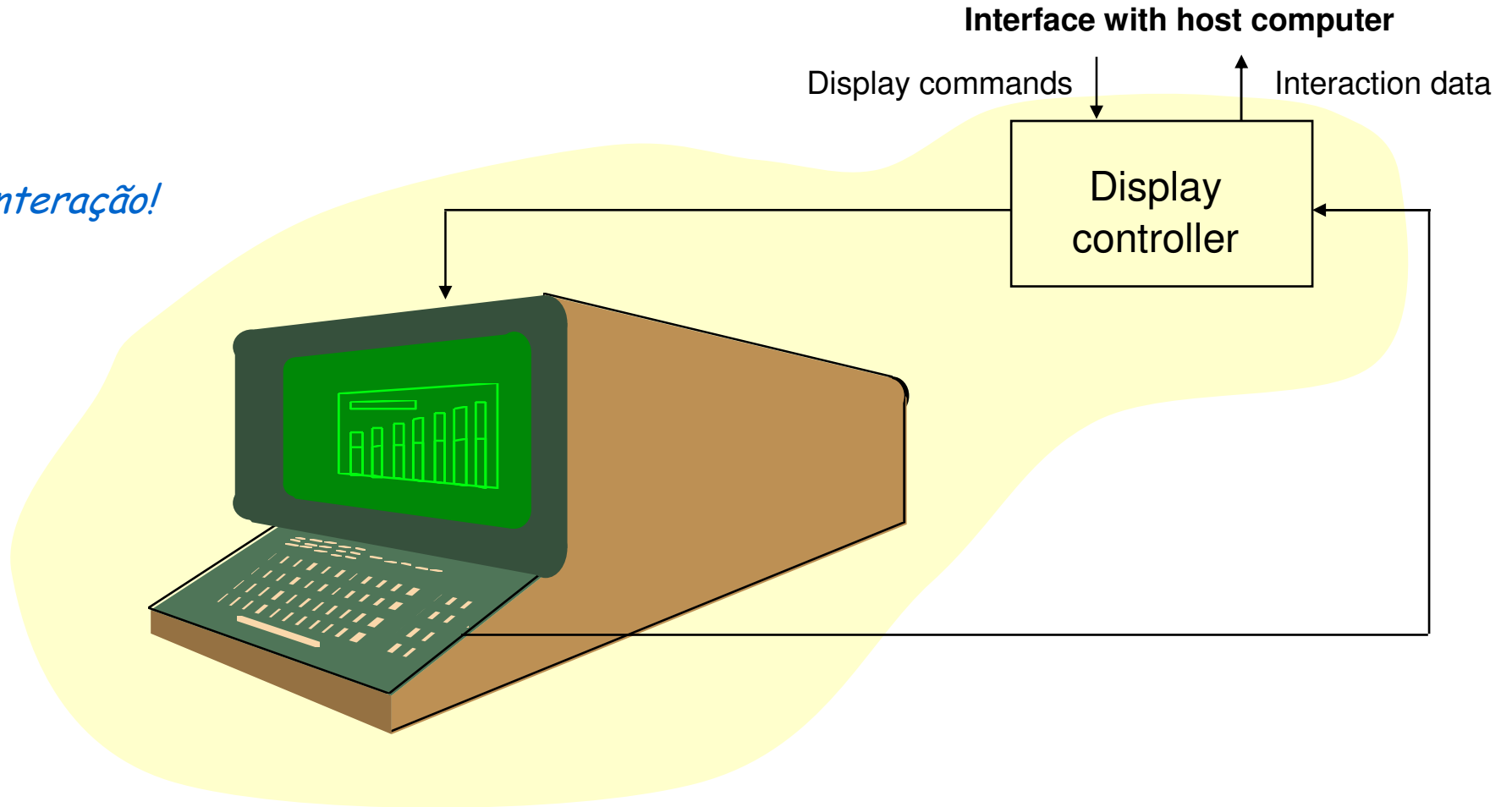
Refresh Vector Display Device

(Terminal vetorial ou caligráfico c/ refrescamento)

Refresh display $\approx$ USD 80 000
+
dedicated host computer $\approx$ USD 400 000

Anos 70

Difícil interação!

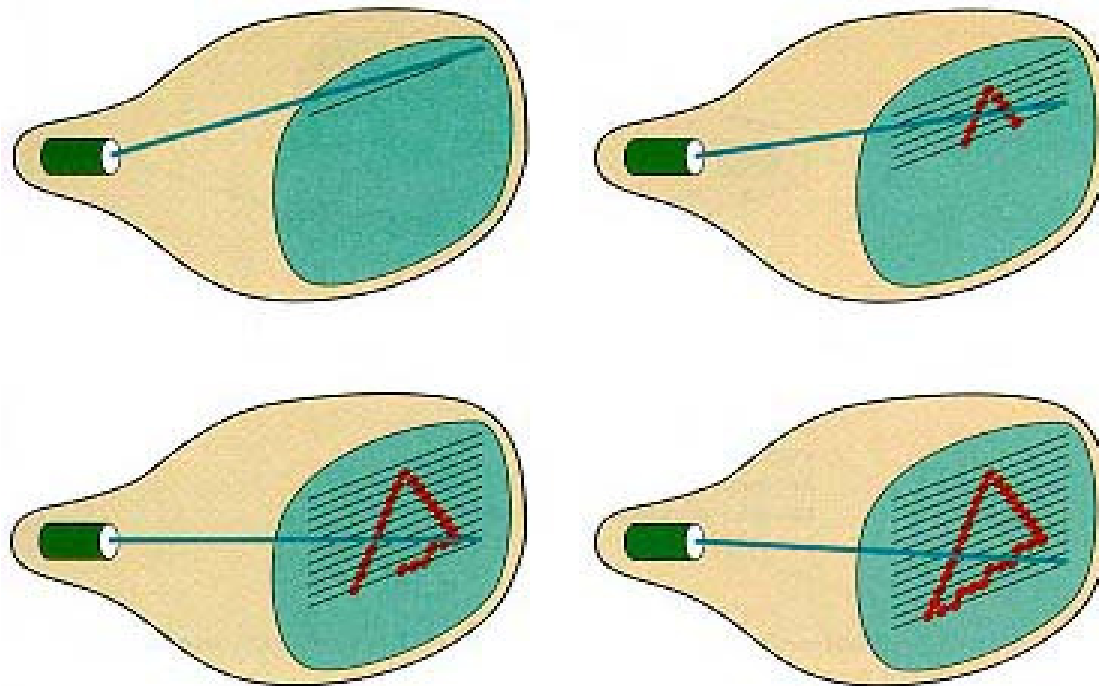


Direct-View Storage Tube Device — DVST

(Terminal vetorial ou caligráfico s/ refreshamento)

DVST ≈ USD 4 000
+
timesharing system

Dispositivo Raster

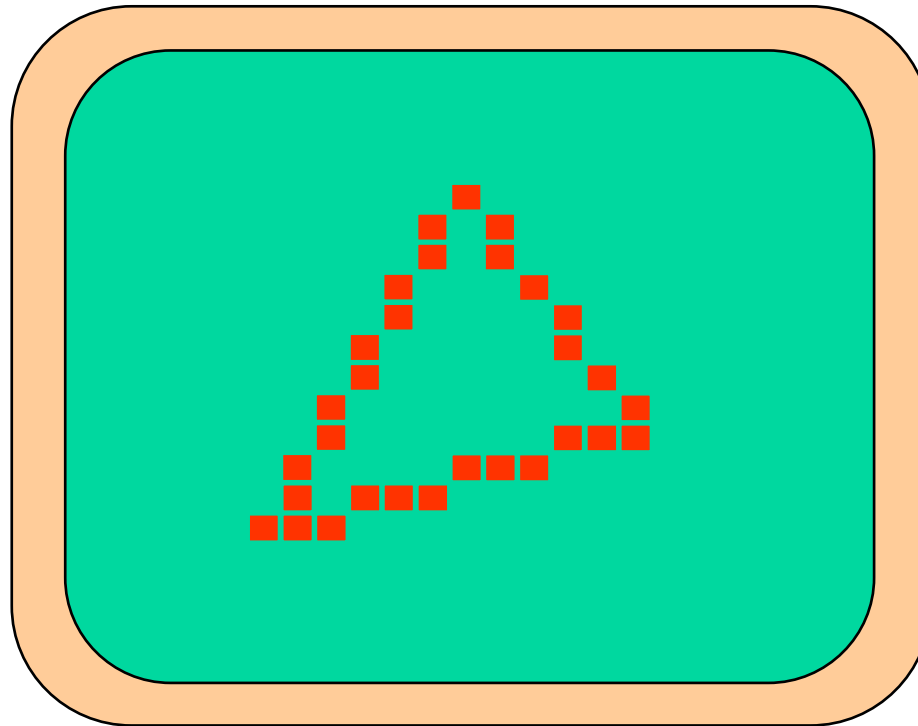


PIXEL = Picture Element

Número fixo de linhas de varrimento horizontal (da esquerda para a direita) e número fixo de pixels por linha.

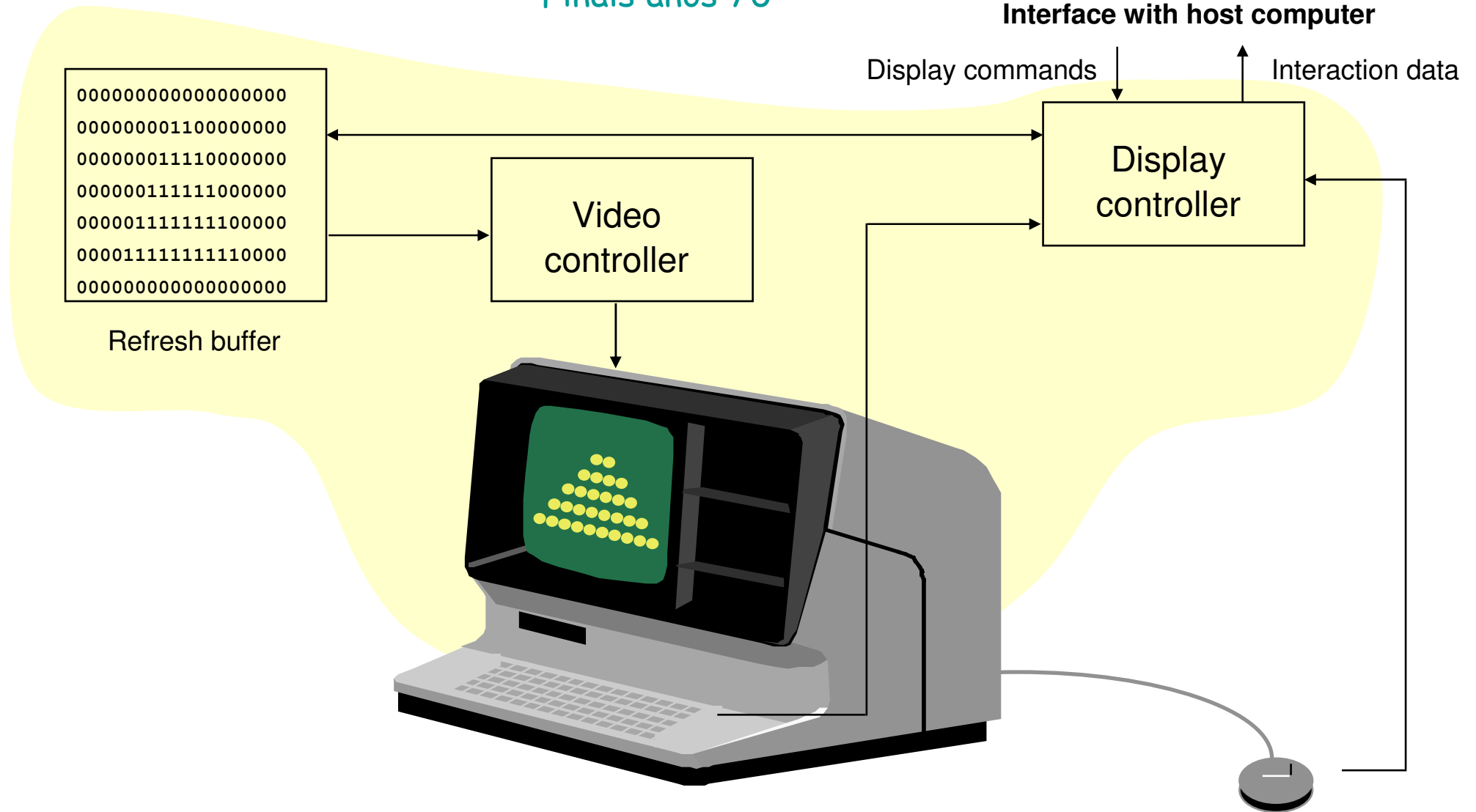
O refrescamento regular do ecrã implica um retorno vertical do(s) feixe(s) de eletrões.

Gráficos num dispositivo Raster



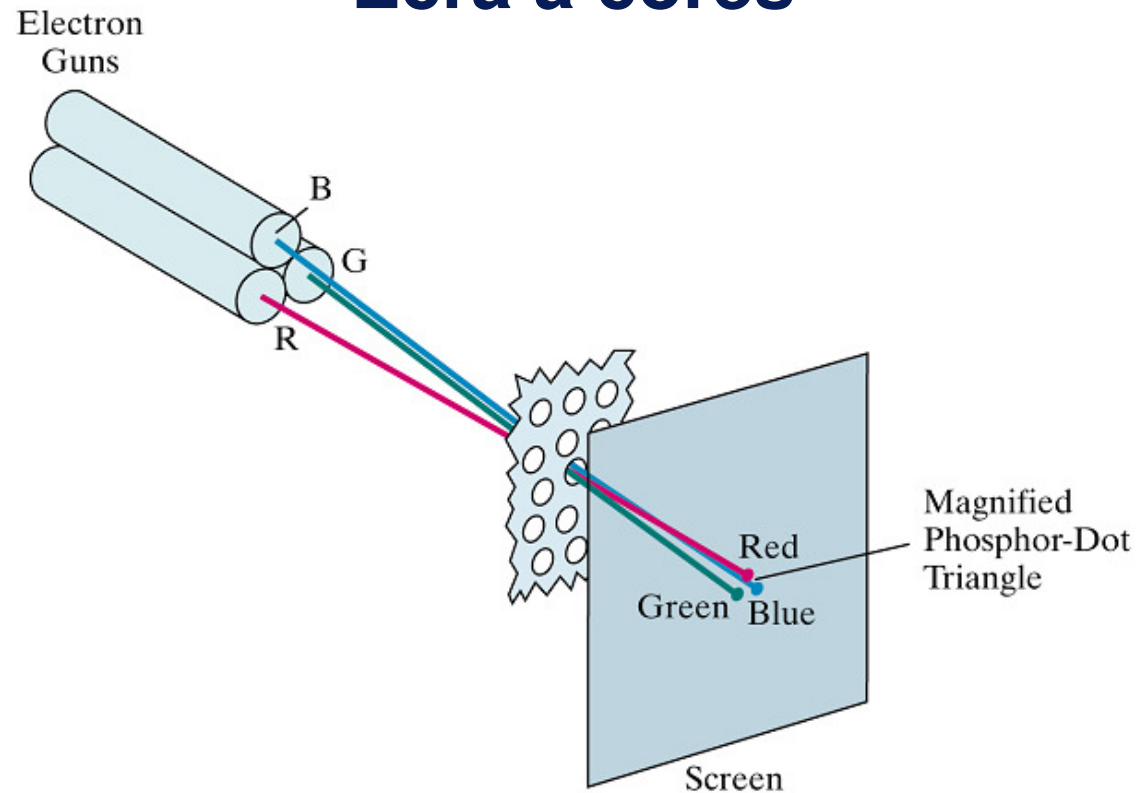
A qualidade da imagem aumentará com a resolução do ecrã.

Finais anos 70



Raster Display Device
(Terminal raster)

Ecrã a cores



from ISBN: 0130153907

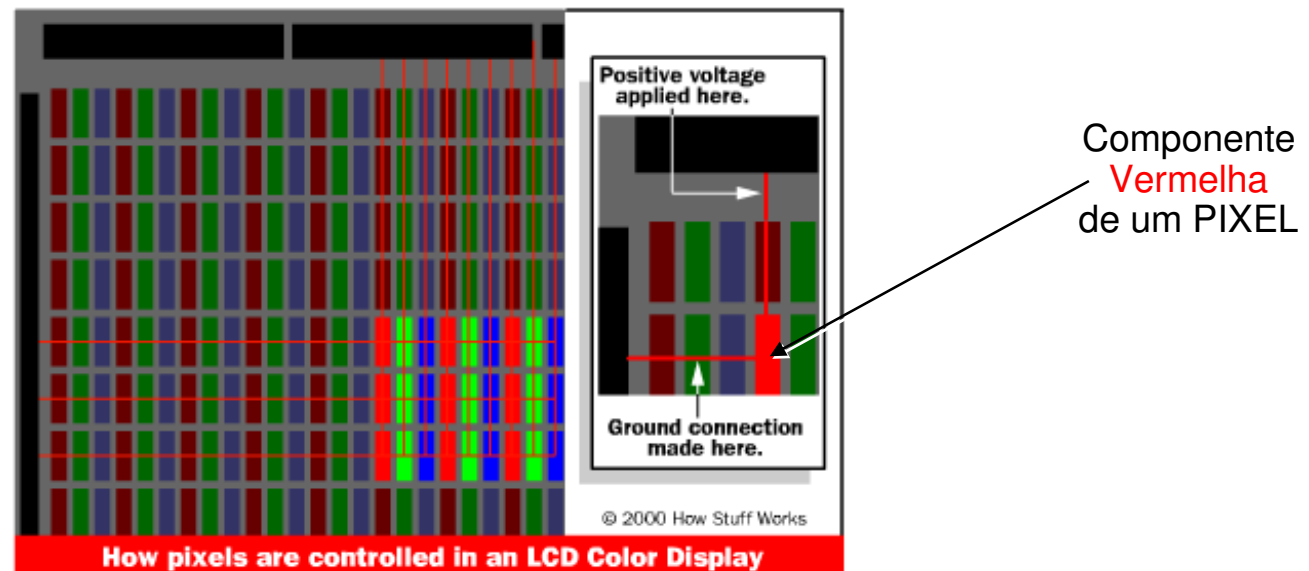
3 feixes distintos de eletrões originam cada pixel: **Red**, **Green** e **Blue**

Cada componente é controlada separadamente e conforme a cor pretendida.

Exemplo: **Red** + **Green** = **Yellow**

Visualização num ecrã LCD

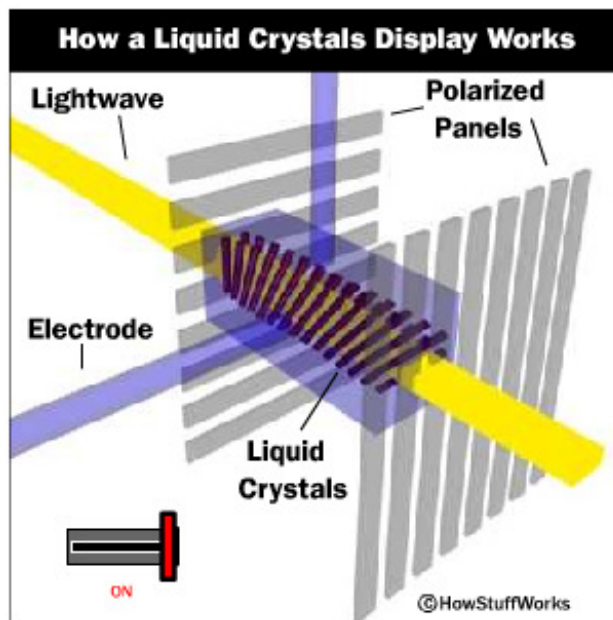
Liquid Crystals Display



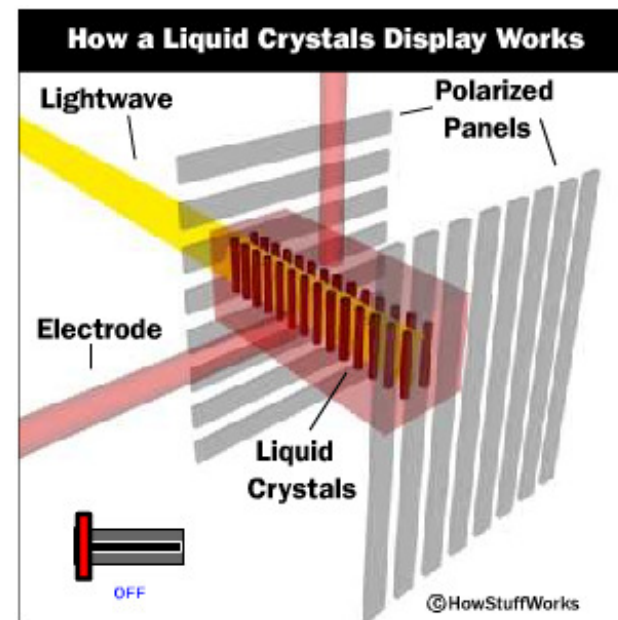
A tensão elétrica aplicada à componente de um determinado pixel faz com que esta bloqueie a luz que, num computador portátil, está a ser permanentemente emitida por detrás.

Visualização num ecrã LCD

Liquid Crystals Display



A orientação das moléculas cristalinas deixa passar a luz polarizada (torção de 90 graus do feixe)



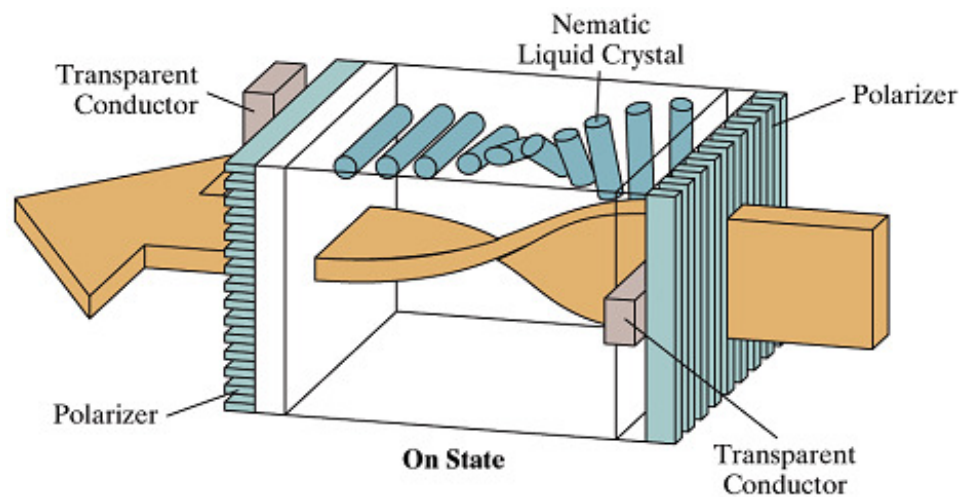
A aplicação de uma carga elétrica ao elemento cristalino age como válvula em relação ao feixe de luz

in <http://electronics.howstuffworks.com/lcd2.htm>

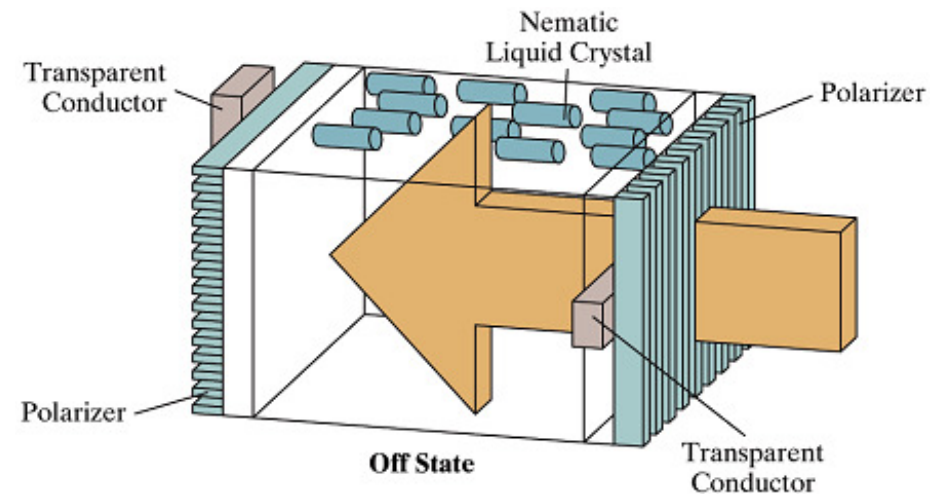
Correção: Os painéis polarizados apresentam-se trocados nas duas figuras acima.

Visualização num ecrã LCD

Liquid Crystals Display



A orientação das moléculas cristalinas deixa passar a luz polarizada (torção de 90 graus do feixe)



A aplicação de uma carga elétrica ao elemento cristalino age como válvula em relação ao feixe de luz

from ISBN: 0130153907

Limitações encontradas ao longo da história da Computação Gráfica

- ➔ **Custo do equipamento gráfico**
- ➔ **Recursos de computação**
- ➔ **Dificuldade na escrita de grandes programas interativos**
- ➔ **Não portabilidade da programação**

Alguns marcos cronológicos na História da Computação Gráfica

1950 1960 1970 1980 .

Dual 16" CRT displays
Plotters

Lightpens
High speed microfilm recorders
Colour displays
Tablets

Mouse
DVST

Raster displays

Number of display systems in the world by 1964 $\approx$ 100



Sketchpad (Ivan Sutherland) (1963)

1st research projects & commercial products

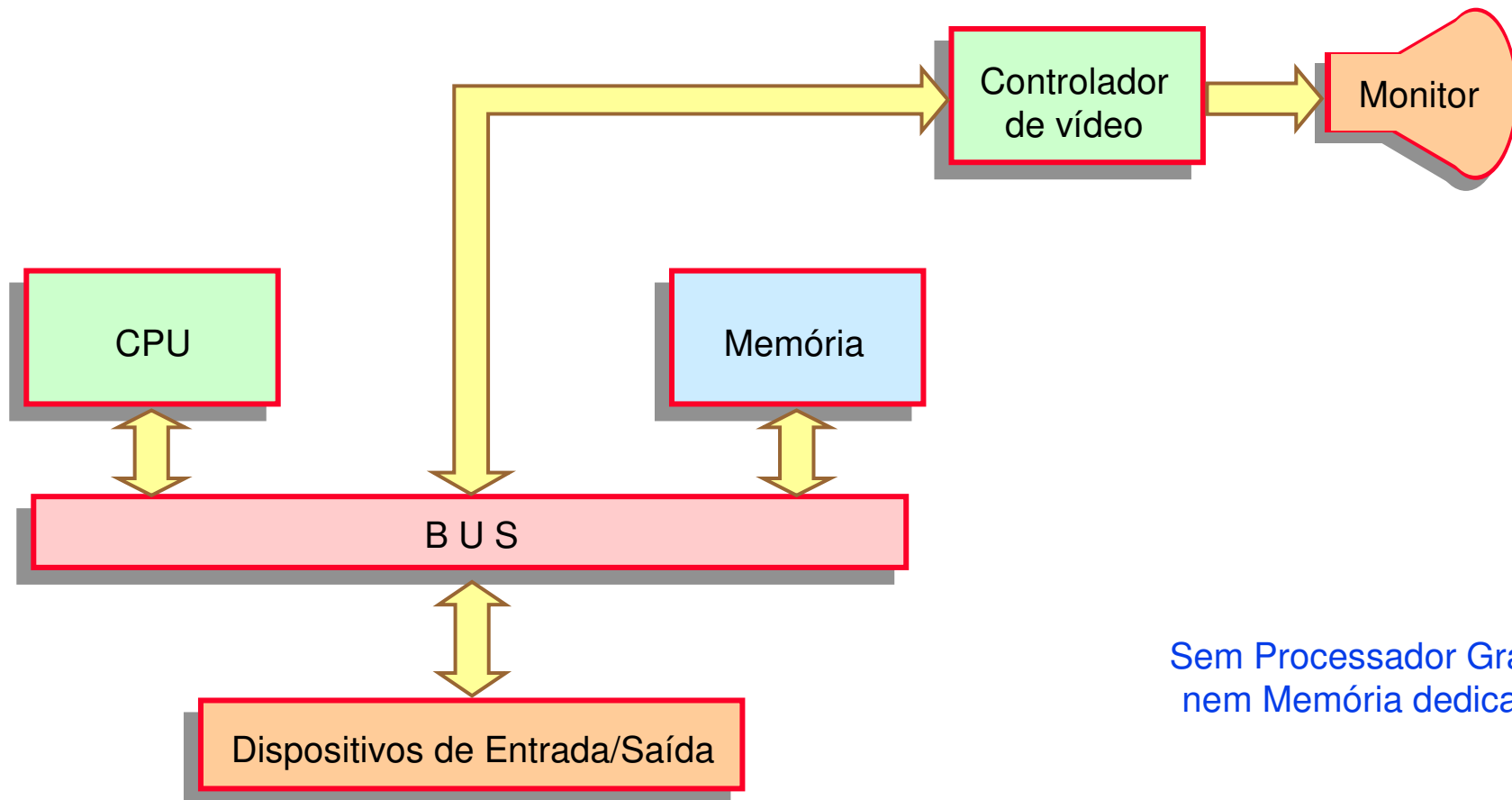
80's Workstations, PC, Mac

API ISO standards: **GKS** (J. Encarnação et al.) (1985) + **PHIGS** (1989)...

90's High powered and low cost single user systems + multimedia + www + ...

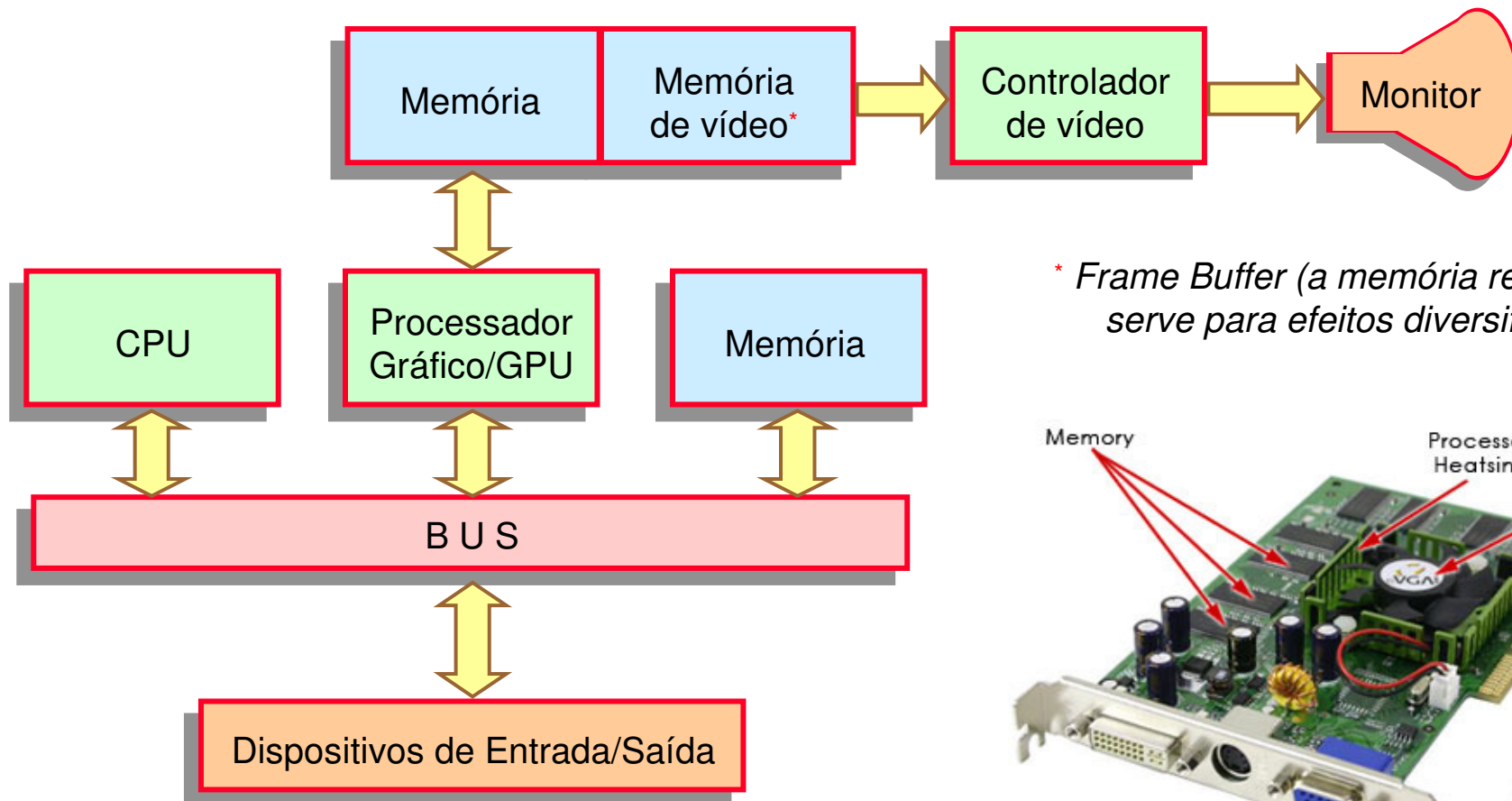
00's GPGPU, **ISO SVG**, **ISO X3D**, **API OpenCL**, ...

Arquitetura de um sistema gráfico raster

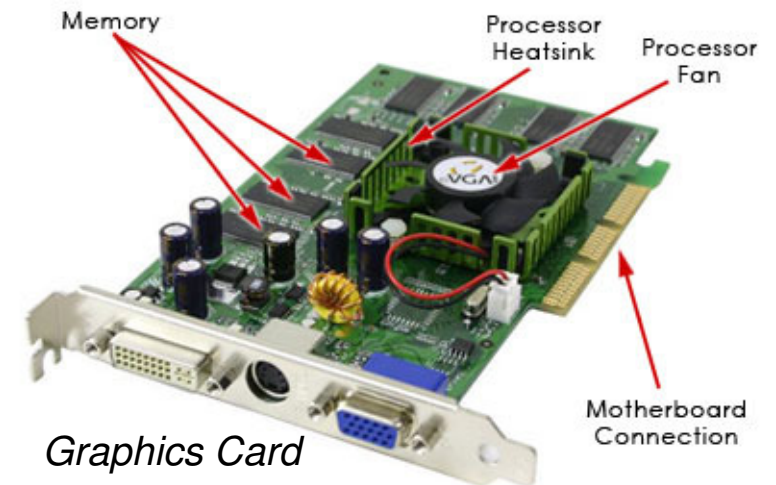


Sem Processador Gráfico
nem Memória dedicada

Arquitetura de um sistema gráfico raster

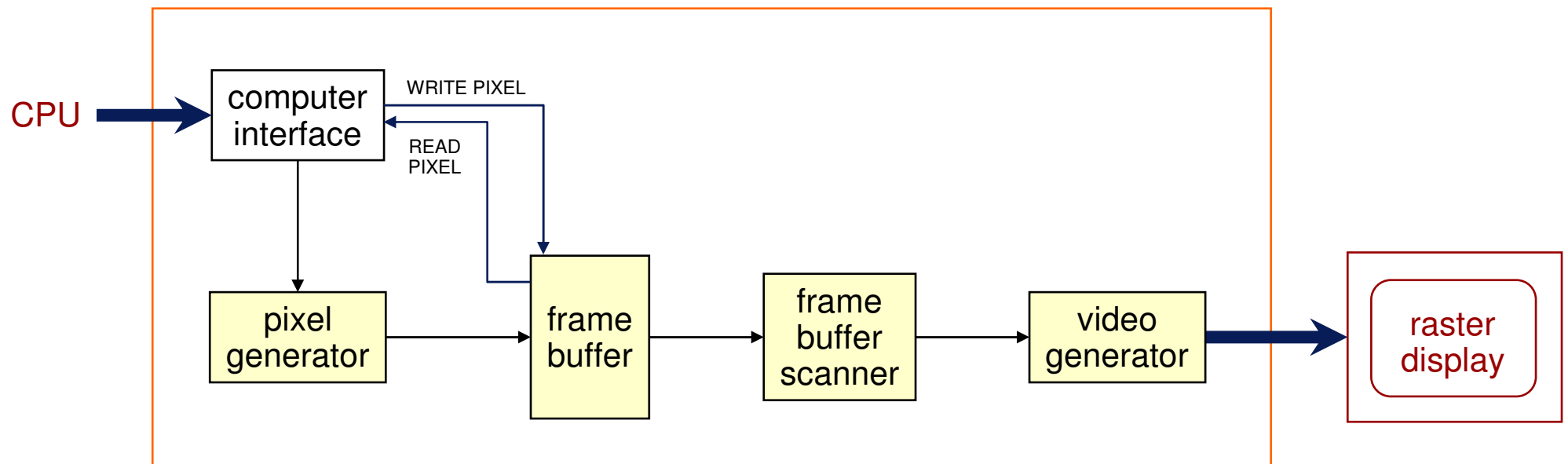


* *Frame Buffer (a memória restante serve para efeitos diversificados)*



Graphics Card

Diagrama funcional básico duma placa gráfica raster



Modificações dinâmicas (*feedback* da interação) requerem uma geração de pixels a, pelo menos, 10 Hz, ao passo que o refrescamento exige 25 Hz, no mínimo.

Funções e Componentes básicas

Principais funções duma placa gráfica

- Fornecer os sinais adequados ao sistema de visualização (*display*), interpretando os comandos vindos do CPU e construindo a imagem em pixels na memória de refrescamento.
- Executar o varrimento da memória de refrescamento e interpretar o seu conteúdo.
- Gerar os três sinais de saída independentes correspondentes às três cores primárias R, G e B.

Pixel Generator (integrado no GPU)

- Interpretador das instruções do CPU (e.g. *line*), mas escrevendo no *frame buffer* só quando o *scanner* não o estiver a utilizar (tipicamente, com um só *buffer*, no retorno vertical do feixe).

Frame Buffer

- Memória de refrescamento, cujas dimensões em x e em y tanto podem ser potências de 2 como os valores correspondentes às dimensões do sistema de visualização.

Frame Buffer Scanner

- Módulo de varrimento da memória de refrescamento (da esquerda para a direita e de cima para baixo) com modulação do sinal de saída proporcionalmente aos valores nessa memória.

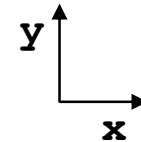
Video Generator

- Produz o sinal de vídeo para o sistema de visualização.

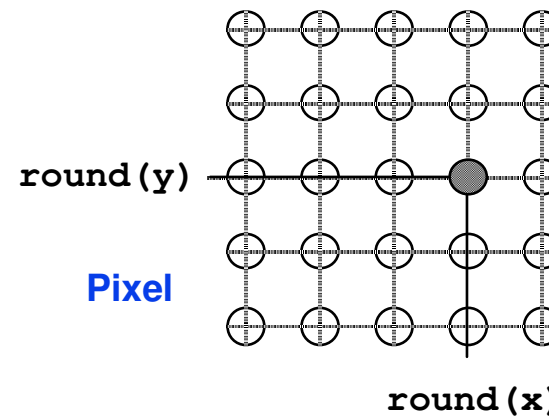
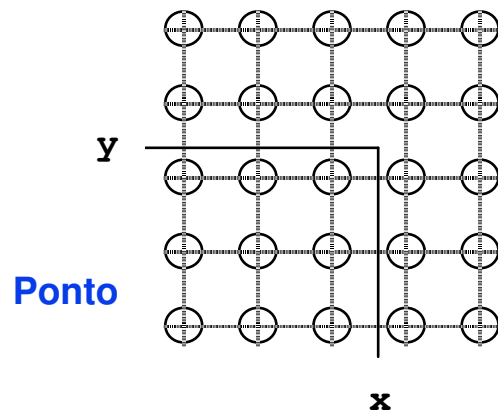
RASTERIZAÇÃO DE PRIMITIVAS GRÁFICAS

Compromisso em tecnologia Raster $\begin{cases} \text{imagens de boa qualidade} \\ \text{algoritmos de execução rápida} \end{cases}$

Nota: Salvo indicação em contrário, usaremos o seguinte sistema de coordenadas cartesianas



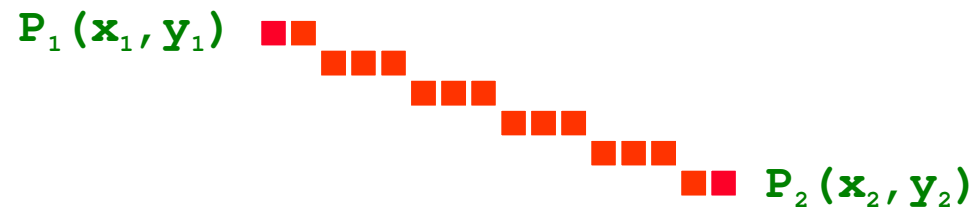
PONTOS



```
WRITE_PIXEL(round(x), round(y), value)
```

LINHAS

Segmentos de reta definem-se pelas coordenadas dos extremos (P_1 e P_2), por hipótese valores inteiros.



A cor, a grossura e o tipo de traço são atributos possíveis para as linhas.

Há implementações desta primitiva em que o pixel correspondente a P_2 não é ativado, permitindo que não seja escrito mais do que uma vez se se tratar de uma linha poligonal (e.g. um triângulo $[P_1 , P_2 , P_3]$).

Existem modos de escrita no *frame buffer* que podem originar efeitos diferentes quando se repete a escrita de um mesmo pixel (caso do modo XOR).

MODOS DE ESCRITA

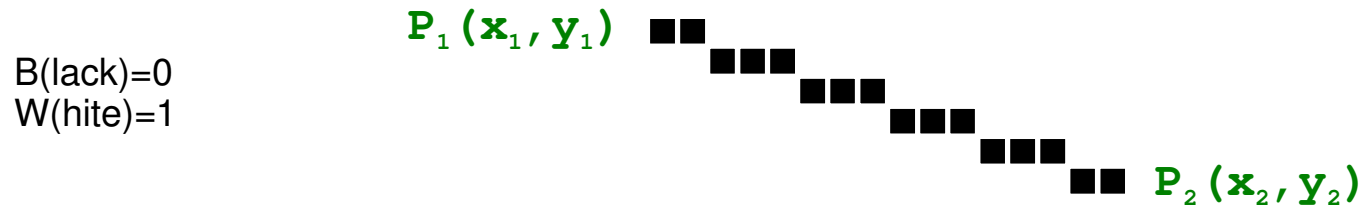
Escrita de pixels (**Source**) em modo XOR (disjunção exclusiva) no *frame buffer* (**Destination/Result**)

Source	Destination	Result
0	0	0
0	1	1
1	0	1
1	1	0

OBS.:

O modo por omissão costuma ser **Result=Source**, o que se resume a cópia.

Exemplo aplicado a linhas B&W (1 bit/pixel), usando cor W sobre fundo W:



Com outras cores, fazer procedimentos independentes para R, G e B

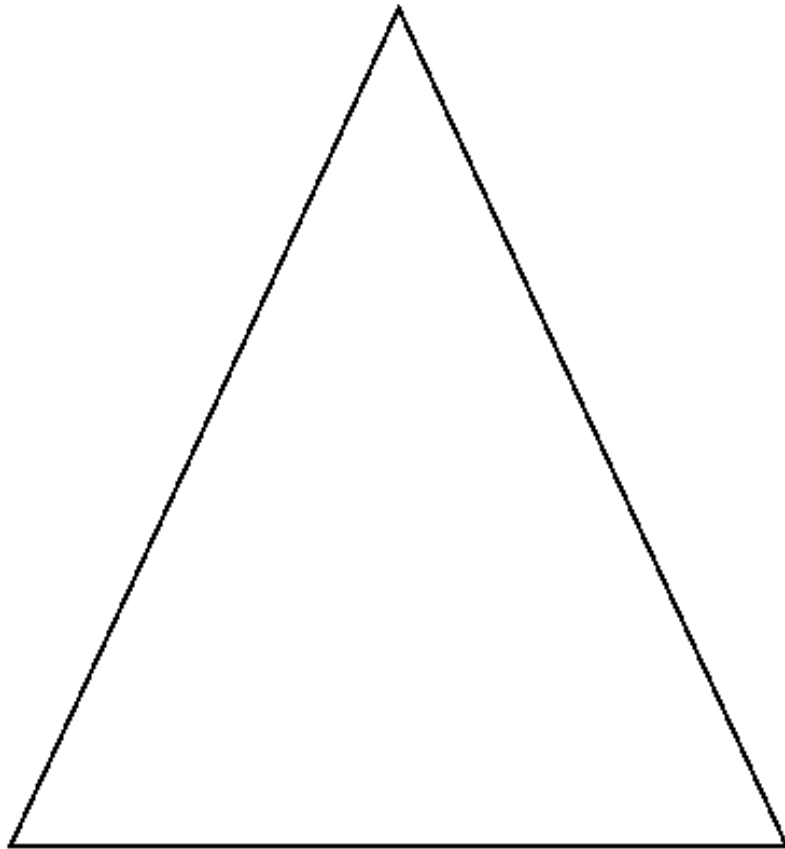
Assim, pode-se criar *rubber-banding* desenhando simplesmente 2 vezes, em cor W, o segmento a modificar (sendo a 1.^a vez sobre a posição anterior, para apagar essa versão, e a 2.^a actualizando-se apenas o ponto P_2 para a posição do cursor).

Pergunta: O que acontece quando uma linha de cor B, escrita em modo XOR, intersecta outra linha da mesma cor já existente no *frame buffer*?

Outros exemplos de utilização de primitivas gráficas (nível API)

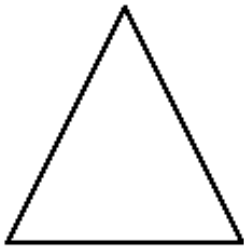
Programação em PostScript

```
%!PS
% Triangle_1
% Triângulo centrado em página A4
% ( unidades: 72 dpi )
72 144 moveto
306 648 lineto
540 144 lineto
closepath
stroke
showpage
```



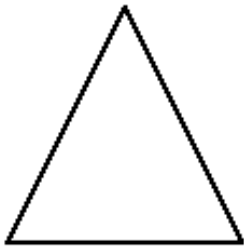
Programação em PostScript

```
%!PS
% Triangle_1
% Triângulo centrado em página A4
% ( unidades: 72 dpi )
72 144 moveto
306 648 lineto
540 144 lineto
closepath
stroke
showpage
```



Programação em PostScript

```
%!PS
% Triangle_2
% Triângulo no topo da página
/cm {28.35 mul} def
5 cm 20 cm moveto
7.5 cm 25 cm lineto
10 cm 20 cm lineto
closepath
stroke
showpage
```

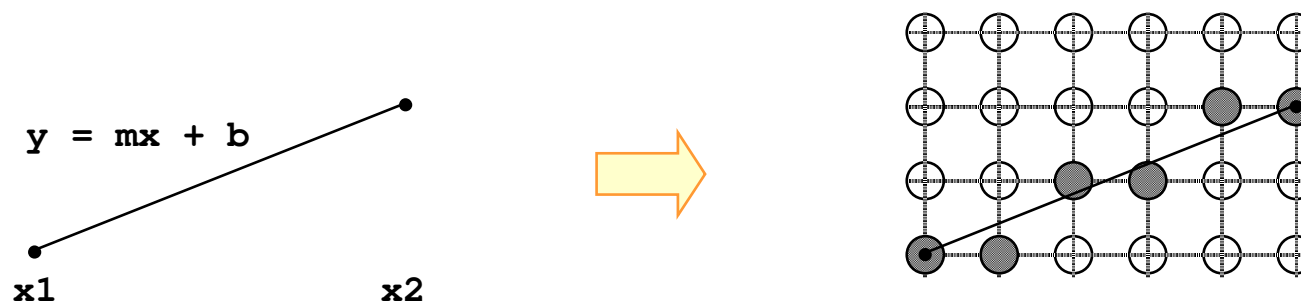


Programação em PostScript

```
%!PS
% Triangle_3
% A mesma figura que Triangle_2
/cm {28.35 mul} def
5 cm 20 cm moveto
7.5 cm 25 cm lineto
10 cm 20 cm lineto
5 cm 20 cm lineto
stroke
showpage
```

ALGORITMOS DE RASTERIZAÇÃO DE LINHAS

Tratamento de segmentos de reta



```
for x from x1 to x2 step Δx do
  y = m*x + b
  WRITE_PIXEL(x, round(y), value)
endfor
```

Pseudo-código
(a sintaxe correta
dependerá da
linguagem e do
sistema)

$\Delta x \rightarrow$ Valor inteiro (=1)

$y, m, b \rightarrow$ Não são, em geral, valores inteiros

$\Rightarrow$ Há métodos algorítmicos mais eficientes!

Como reduzir o tempo de execução ?

ALGORITMO INCREMENTAL (*DDA Algorithm* ^{*})

$$y_{i+1} = m x_{i+1} + b = m (x_i + \Delta x) + b = m x_i + m \Delta x + b = y_i + m \Delta x$$

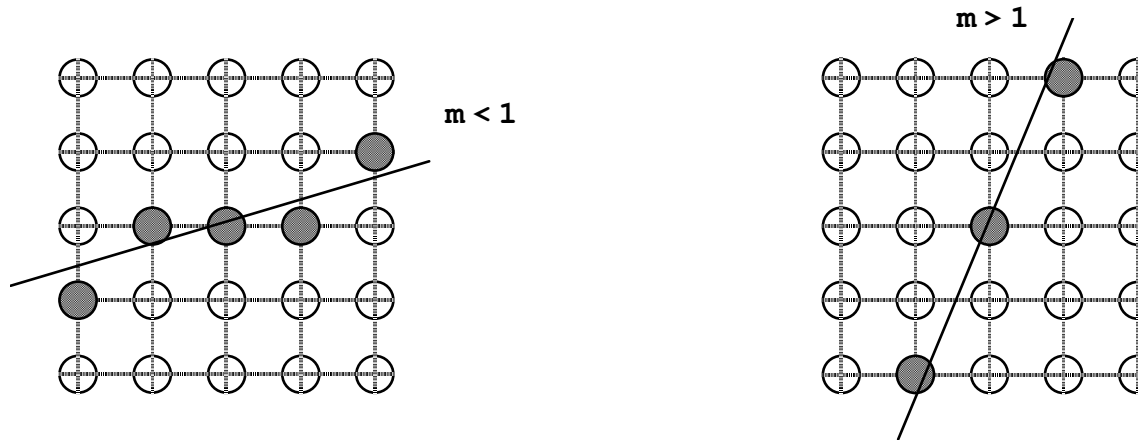
$$\Delta x = 1 \Rightarrow \Delta y = m \Delta x = m$$

```
procedure LINE (x1,y1,x2,y2,value: integer);  
                                     { when -1 ≤ m ≤ +1 only }  
var   dx,dy,x,y,m: real;  
  
begin  
    if x1<>x2 then  
        begin  
            dy := y2-y1; dx := x2-x1; m := dy/dx; y := y1;  
            for x:=x1 to x2 do  
                begin  
                    WRITE_PIXEL(x,round(y),value); y := y + m  
                end  
            end  
        else if y1=y2 then WRITE_PIXEL(x1,y1,value)  
    end;  
end;
```

^{*} *Digital Differential Analyser Algorithm*

ALGORITMO INCREMENTAL

E se for $m > 1$?



$m > 1 \Rightarrow \Delta y > 1 \Rightarrow$ Ficam pixels por intensificar !

Solução: Troca de variável independente, resultando $\Delta y = 1$ e $\Delta x = 1/m$.

Avaliação do algoritmo:

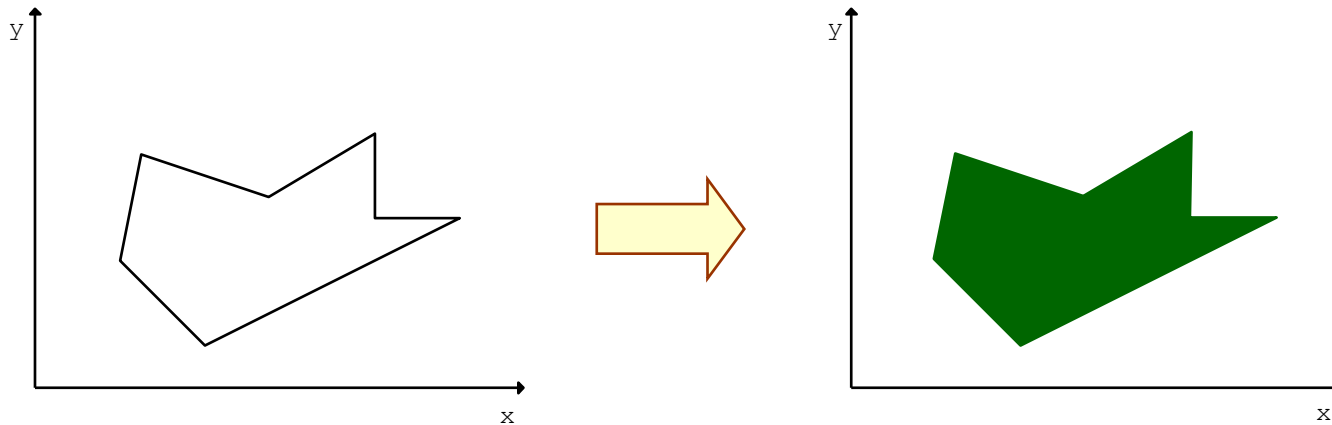
- ⊕ Eliminação do produto $m \cdot x$
- ⊖ Acumulação de erros

Há algoritmos melhores do que este!

POLÍGONOS

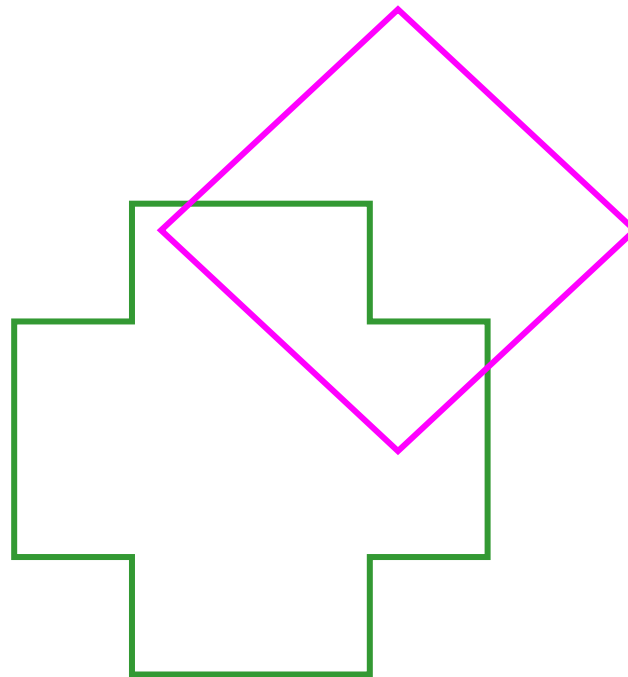
FILL AREA

Objetivo: Preenchimento de um polígono definido pela sequência dos respectivos vértices (pontos 2D), independentemente dos valores preexistentes na memória de refrescamento.



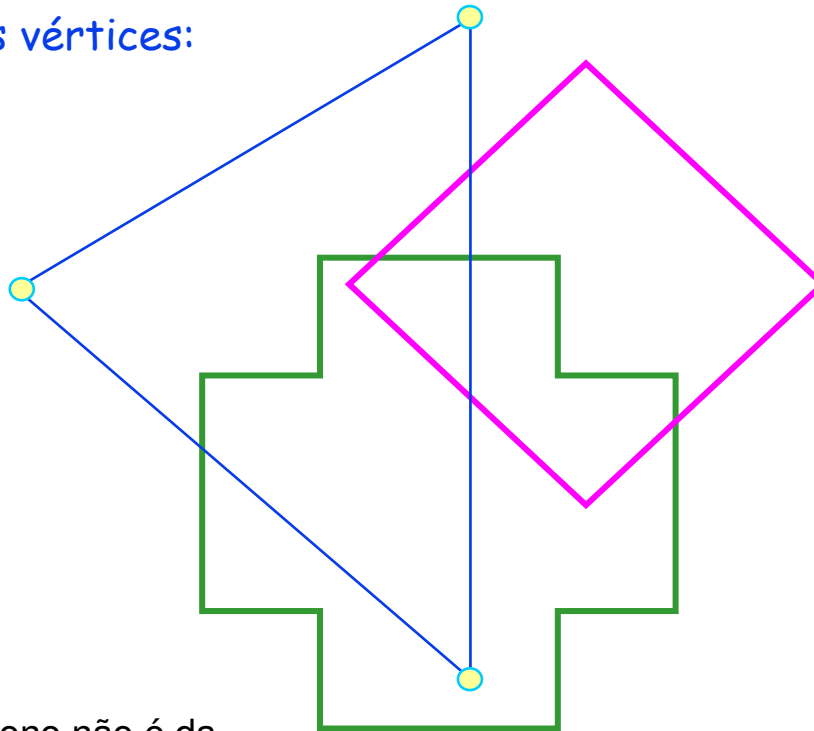
FILL AREA

Exemplo de ecrã numa situação inicial:



FILL AREA

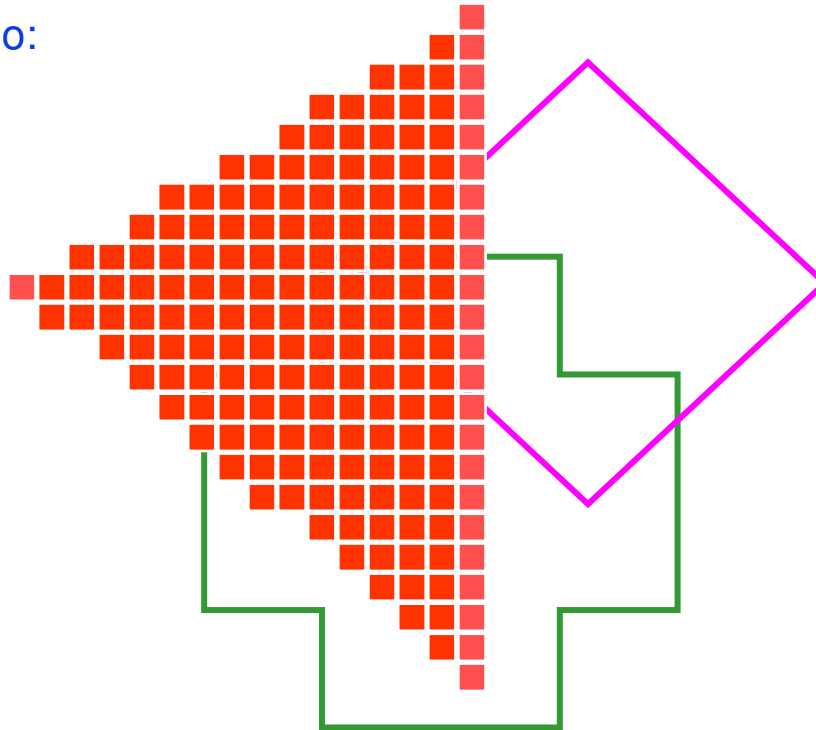
Novo polígono, definido pelos seus vértices:



NB: O desenho da fronteira do polígono não é da responsabilidade do algoritmo de FILL_AREA .

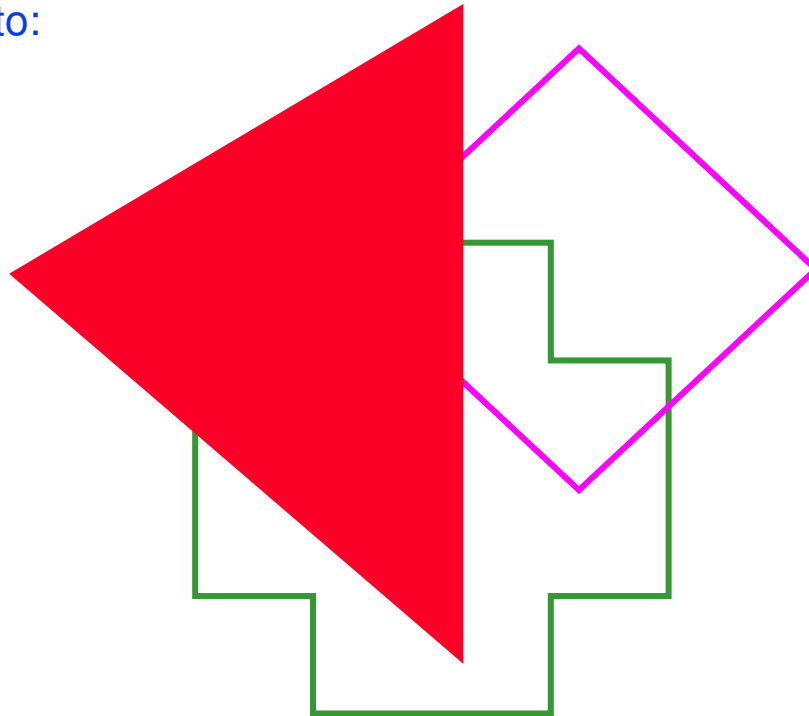
FILL AREA

Preenchimento do triângulo:
(em baixa resolução)

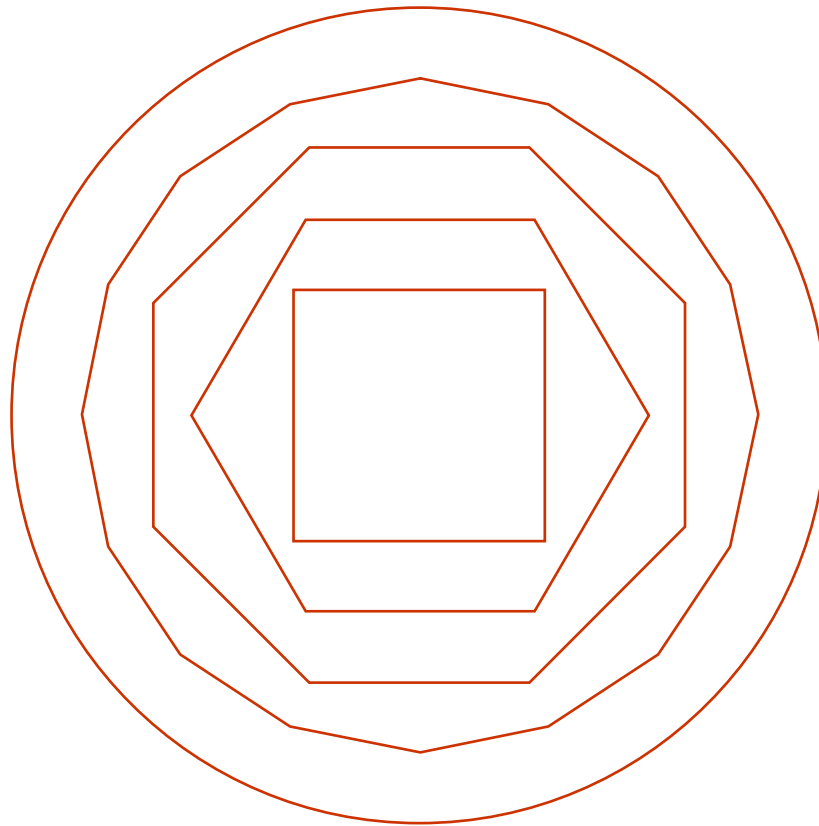


FILL AREA

Resultado do preenchimento:
(na resolução do ecrã)



Rasterização de linhas curvas genéricas



Embora haja algoritmos para casos particulares, de uma forma geral bastará encontrar uma aproximação razoável por segmentos de reta e executar o algoritmo de rasterização destes para cada um.

Representação matemática de pontos

A cada ponto $P(x,y)$ no plano far-se-á corresponder um vetor

$$P = \begin{bmatrix} x \\ y \end{bmatrix}$$

Assim, uma transformação linear, dada por uma matriz M , poderá transformar P em P' escrevendo-se:

$$P' = M.P$$

Por conseguinte, o operador M deverá ser uma matriz 2x2.

Mas, em alternativa, os pontos também se poderão especificar em Coordenadas Homogéneas^{*}.

^{*} As vantagens serão referidas em capítulos posteriores

Pontos em Coordenadas Homogêneas

2D

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

$W \neq 0$

$$\begin{bmatrix} x.W \\ y.W \\ W \end{bmatrix}$$

$$\begin{bmatrix} X/W \\ Y/W \end{bmatrix}$$

$W \neq 0$

$$\begin{bmatrix} X \\ Y \\ W \end{bmatrix}$$

NOTA: Se for por escolha, convém fazer $W=1$.

Pontos em Coordenadas Homogêneas

3D

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$W \neq 0$

$$\begin{bmatrix} x.W \\ y.W \\ z.W \\ W \end{bmatrix}$$

$$\begin{bmatrix} X/W \\ Y/W \\ Z/W \end{bmatrix}$$

$W \neq 0$

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

NOTA: Se for por escolha, convém fazer $W=1$.

Curva

Especificada por uma ou mais equações com **uma** só variável independente.

◆ Descrição não-paramétrica

- Forma explícita

$$Q = (x , y=f(x) , z=g(x))$$

- Forma implícita

$$Q = (F(x,y,z)=0 , G(x,y,z)=0)$$

◆ Descrição paramétrica

$$Q = (x=f(t) , y=g(t) , z=h(t))$$

$$a \leq t \leq b$$

Superfície

Especificada por uma ou mais equações com **duas** variáveis independentes.

◆ Descrição não-paramétrica

- Forma explícita

$$Q = (x , y , z=f(x,y))$$

- Forma implícita

$$Q = (F(x,y,z)=0)$$

◆ Descrição paramétrica

$$Q = (x=f(s,t) , y=g(s,t) , z=h(s,t))$$

$$a \leq t \leq b$$

$$c \leq s \leq d$$

Curvas e Superfícies

◆ Descrição não-paramétrica

- Forma explícita

- 😊 Facilidade de cálculo

- 😞 Não pode representar uma correspondência que não seja função

- 😞 Não se podem aplicar diretamente transformações por operadores matriciais

- ◆ Exemplo: $y = \sqrt{4 - x^2}$ e $y = -\sqrt{4 - x^2}$ com $-2 \leq x \leq 2$

não sendo direto aplicar-se uma matriz **M** de transformação

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Curvas e Superfícies

◆ Descrição não-paramétrica

- Forma implícita

- 😊 Representação de correspondências que não sejam funções

- 😞 Pode ser difícil a determinação das raízes

- 😞 Não se podem aplicar diretamente transformações por operadores matriciais

- ◆ Exemplo:

$$x^2 + y^2 - 4 = 0$$

com $-2 \leq x \leq 2$

Curvas e Superfícies

◆ Descrição paramétrica

- 😊 Representação de correspondências que não sejam funções
- 😊 Podem aplicar-se diretamente transformações por operadores matriciais

◆ Exemplo:

$$\begin{aligned}x &= 2 \cos t \\y &= 2 \sin t\end{aligned}\quad \text{com } 0 \leq t \leq 2\pi$$

ou

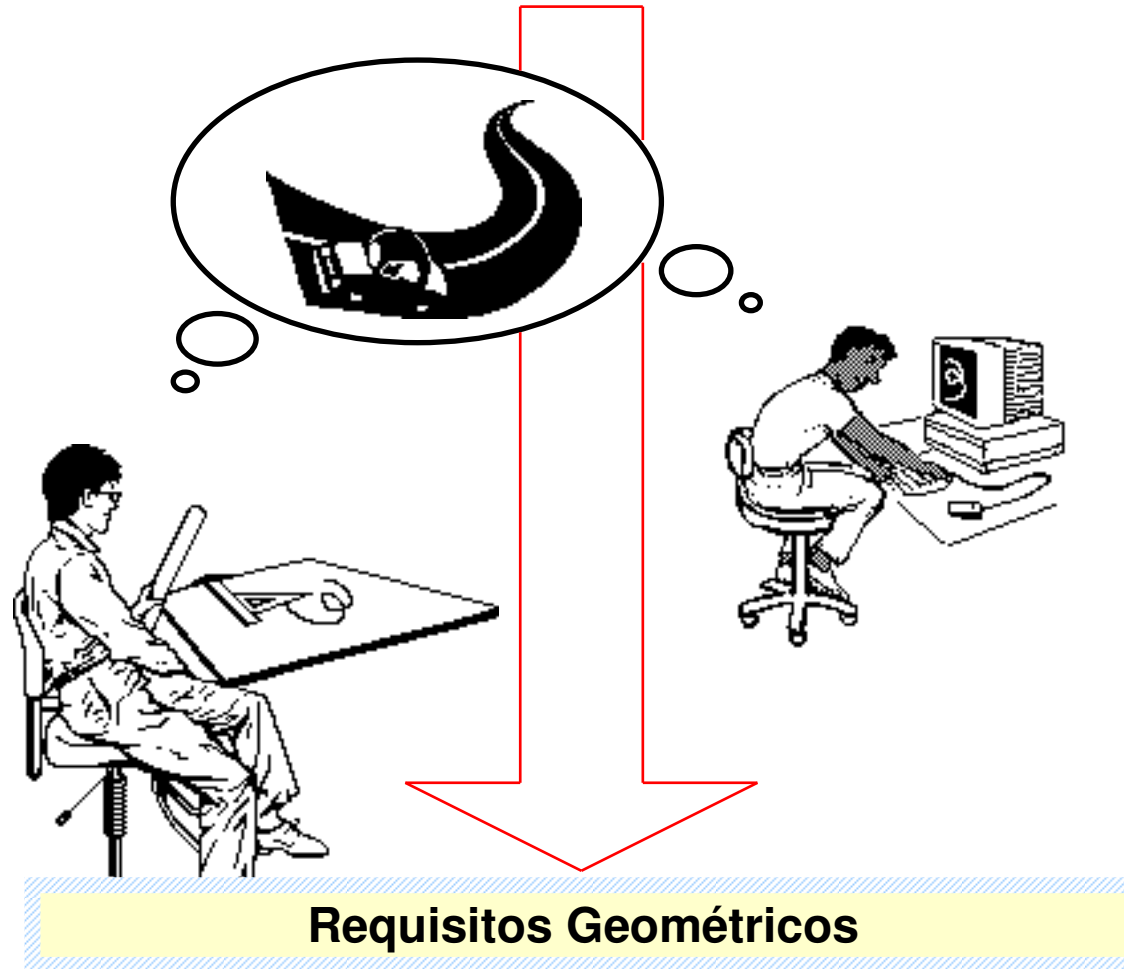
$$\begin{aligned}x &= 2 \cos (2\pi t) \\y &= 2 \sin (2\pi t)\end{aligned}\quad \text{com } 0 \leq t \leq 1$$

sendo já direto aplicar-se uma matriz **M** de transformação

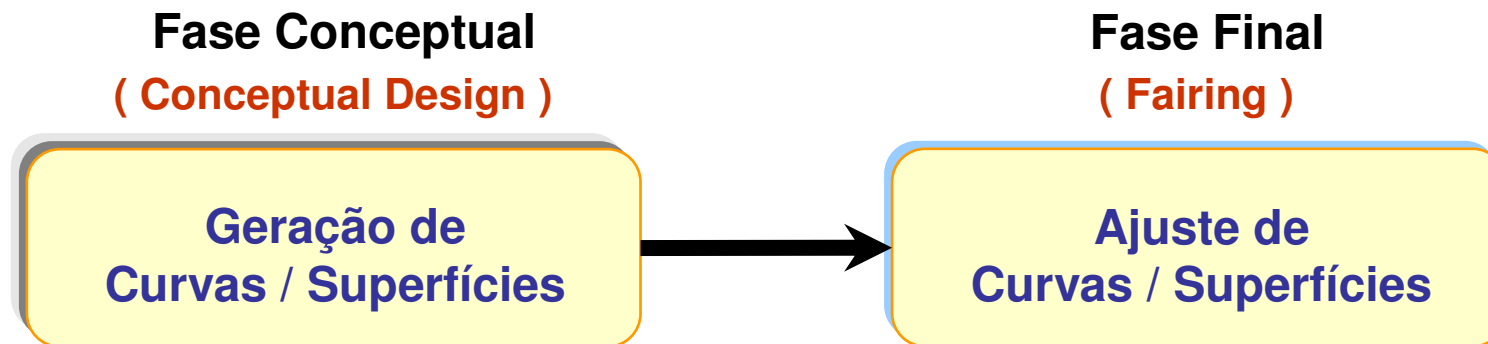
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

O processo de Design

Requisitos Técnicos, Requisitos Físicos, Requisitos Económicos, etc.



Fases no processo de Design

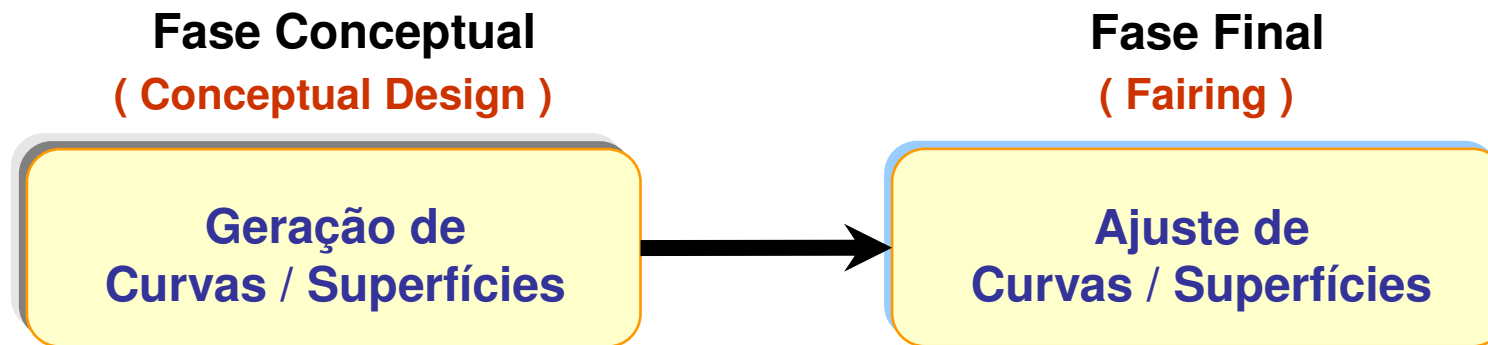


The Design Problem

Given: A set of geometric requirements [such as offsets, slopes, second derivatives, areas, volumes, centroids, ...] sufficient to characterize the intended shape.

Find: A curve (surface) meeting these requirements.

Fases no processo de Design



The Fairing Problem

Given: A curve (surface), or a corresponding set of offset points, approximating an intended shape.

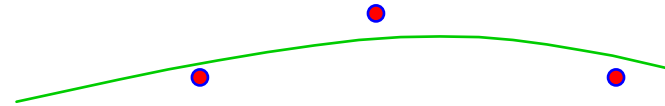
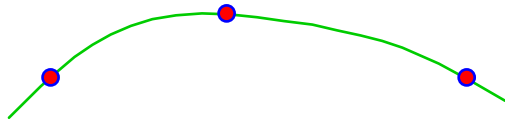
Find: A fairer curve (surface) according to some fairness criterion without deviating too far from the given shape.

What does fair mean?

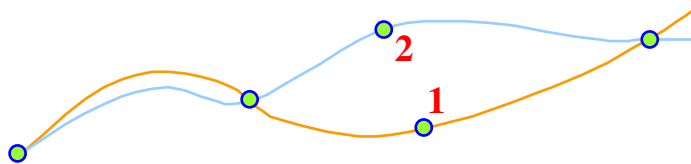
- Pleasing to the eye (aesthetic requirements)
- Related to the absence of unwanted deficiencies, particularly oscillations
- Spline fairness

Alguns requisitos no design de Curvas

- ◆ Interpolar ou aproximar um certo número de pontos conhecidos, obtendo-se a equação da curva



- ◆ Controlar através de pontos conhecidos e de forma previsível: **local** (preferível) ou **globalmente**

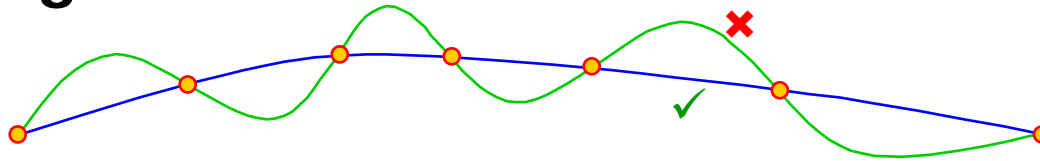


- ◆ Haver independência da forma da curva em relação ao sistema de eixos usado para especificar os pontos
- ◆ Permitir correspondências que não sejam funções

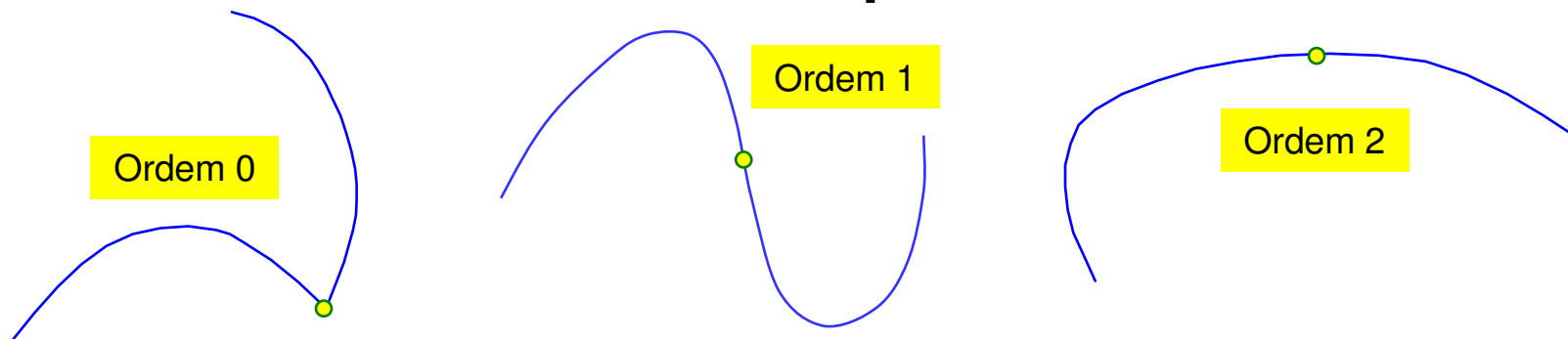


Alguns requisitos no design de Curvas

- ◆ **Existir tendência para suavizar pequenas irregularidades**



- ◆ **Disponibilizar grande versatilidade de formas**
- ◆ **Permitir a continuidade entre os troços que constituam uma curva complexa**

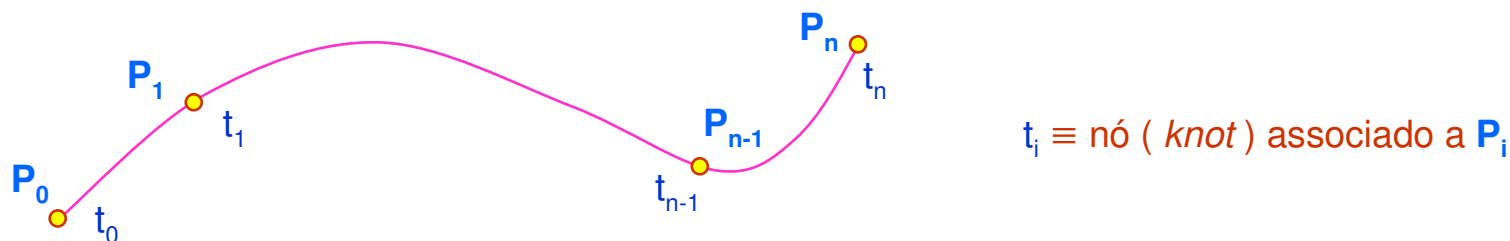


1.^a conclusão: Usar descrições paramétricas.

Curvas de Interpolação

◆ Problema:

Encontrar uma curva $Q(t)$ que passe por $n+1$ pontos P_i



Como consequência da descrição paramétrica, a escolha dos valores dos nós influencia o andamento da curva.

● Condições do problema:

- $Q(t_i) = P_i$, $i = 0..n$
- Continuidade das funções e suas derivadas

● Resolução:

- Usar polinómios interpoladores

Interpolação de Lagrange

Existe um e um só polinómio de grau n que resolve o problema:

$$Q(t) = \sum_{i=0}^n P_i L_{in}(t)$$

em que
$$L_{in}(t) = \frac{(t - t_0) \dots (t - t_{i-1}) (t - t_{i+1}) \dots (t - t_n)}{(t_i - t_0) \dots (t_i - t_{i-1}) (t_i - t_{i+1}) \dots (t_i - t_n)}$$

Mas ...

... um polinómio de grau n tem até $n-1$ extremos relativos e $n-2$ pontos de inflexão !

Em medições precisas, quando se usam muitos pontos, o resultado poderá ser uma oscilação indesejada da curva.

Este efeito negativo poderá ser evitado ?

Não, pois basta que haja um só ponto incompatível com uma curva suave.

A maioria das aplicações não requer continuidade das derivadas de elevada ordem

A solução é a interpolação da curva por troços contínuos !



Polinómios de baixo grau



Interpolação por Splines (naturais)

A teoria dos splines trata da interpolação polinomial, na generalidade e por troços.

DEFINIÇÃO DE SPLINE :

Uma função $\mathbf{S(t)}$, escalar ou vetorial, definida no intervalo $[t_0, t_n]$, é um **SPLINE** de **ordem k** (ou **grau k-1**) se :

- $\mathbf{S(t)}$ é um polinómio de grau **k-1** em cada intervalo $[t_i, t_{i+1}]$, com $t_0 < \dots < t_i < t_{i+1} < \dots < t_n$
- $\mathbf{S(t)}$ e as suas derivadas de ordem **1..k-2** são contínuas em todo o intervalo onde é definida (isto é, $\mathbf{S(t)}$ pertence à **classe $\mathbf{C^{k-2}}$**)

$k = 4$  **SPLINE CÚBICO**

DADOS: $n+1$ pontos $\mathbf{P_i}$ ($i=0..n$) e nós $t_i \in [t_0, t_n]$ associados

OBJETIVO:

Encontrar os polinómios cúbicos interpoladores por troços e que definem a função $\mathbf{Q(t)}$ em $[t_0, t_n]$ tal que pertence a $\mathbf{C^2}$ nos nós (*knots*).

Dedução dos coeficientes

Sendo polinómio cúbico, para o intervalo $[t_k, t_{k+1}]$ pode escrever-se

$$Q_k(t) = a_k + b_k (t - t_k) + c_k (t - t_k)^2 + d_k (t - t_k)^3$$

Condições fronteira:

$$Q(t_k) = P_k$$

$$Q(t_{k+1}) = P_{k+1}$$

$$\left(\frac{dQ_k}{dt}\right)_{t=t_k} = R_k$$

$$\left(\frac{dQ_k}{dt}\right)_{t=t_{k+1}} = R_{k+1}$$

Substituindo e resolvendo dá:

$$a_k = P_k$$

$$b_k = R_k$$

$$c_k = \frac{3(P_{k+1} - P_k)}{h_k^2} - \frac{2R_k}{h_k} - \frac{R_{k+1}}{h_k}$$

$$d_k = \frac{2(P_k - P_{k+1})}{h_k^3} + \frac{R_k}{h_k^2} + \frac{R_{k+1}}{h_k^2}$$

com $h_k = t_{k+1} - t_k$

Condições adicionais

As **4** equações deduzidas para os coeficientes referem-se a cada um dos troços (intervalos), pelo que haverá ao todo **4n** equações.

As derivadas **R_i** (em número de **n+1**) ainda não são conhecidas, pelo que precisamos de outras **n+1** equações.

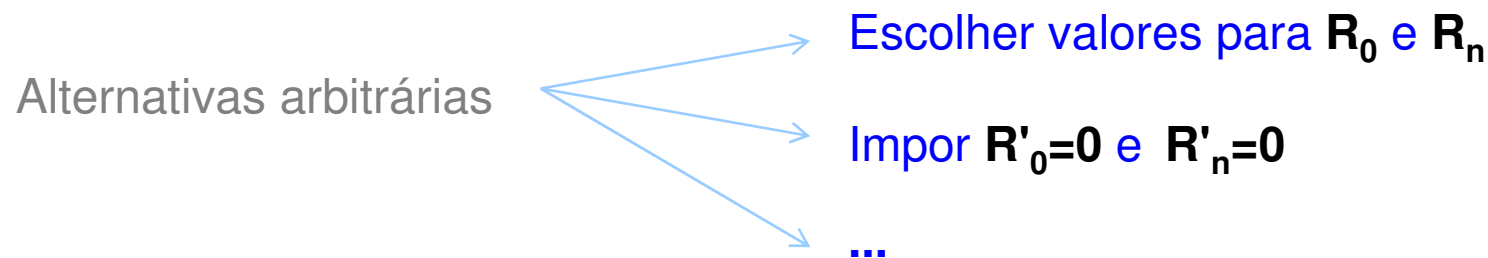
Condição para a segunda derivada:

$$\left(\frac{d^2 Q_k}{dt^2} \right)_{t=t_{k+1}} = \left(\frac{d^2 Q_{k+1}}{dt^2} \right)_{t=t_{k+1}}$$

Ao todo serão **n-1** equações, da forma:

$$2 c_k + 6 d_k (t_{k+1} - t_k) = 2 c_{k+1}$$

Para não deixarmos o sistema indeterminado, há que introduzir mais **2** equações:



Desvantagens dos Splines naturais

- ◆ Como se trata de resolver um sistema de equações global, o controlo da curva é global (e não local).
- ◆ Qualquer que seja a alternativa tomada para a resolução da indeterminação do sistema, o ajuste da curva resultante depende inteiramente da qualidade dos pontos dados, não se garantindo, portanto, a **ausência de oscilações indesejáveis** (mesmo na situação dos erros nos dados serem pequenos).

CONCLUSÃO:

Não é boa solução para design interativo!

Curvas cúbicas e notação a usar

Curva no espaço 3D (também se poderia escrever em coordenadas homogêneas):

$$Q(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} a_x t^3 + b_x t^2 + c_x t + d_x \\ a_y t^3 + b_y t^2 + c_y t + d_y \\ a_z t^3 + b_z t^2 + c_z t + d_z \end{bmatrix} = \begin{bmatrix} [t^3 \ t^2 \ t \ 1] \cdot [a_x \ b_x \ c_x \ d_x]^T \\ [t^3 \ t^2 \ t \ 1] \cdot [a_y \ b_y \ c_y \ d_y]^T \\ [t^3 \ t^2 \ t \ 1] \cdot [a_z \ b_z \ c_z \ d_z]^T \end{bmatrix}$$

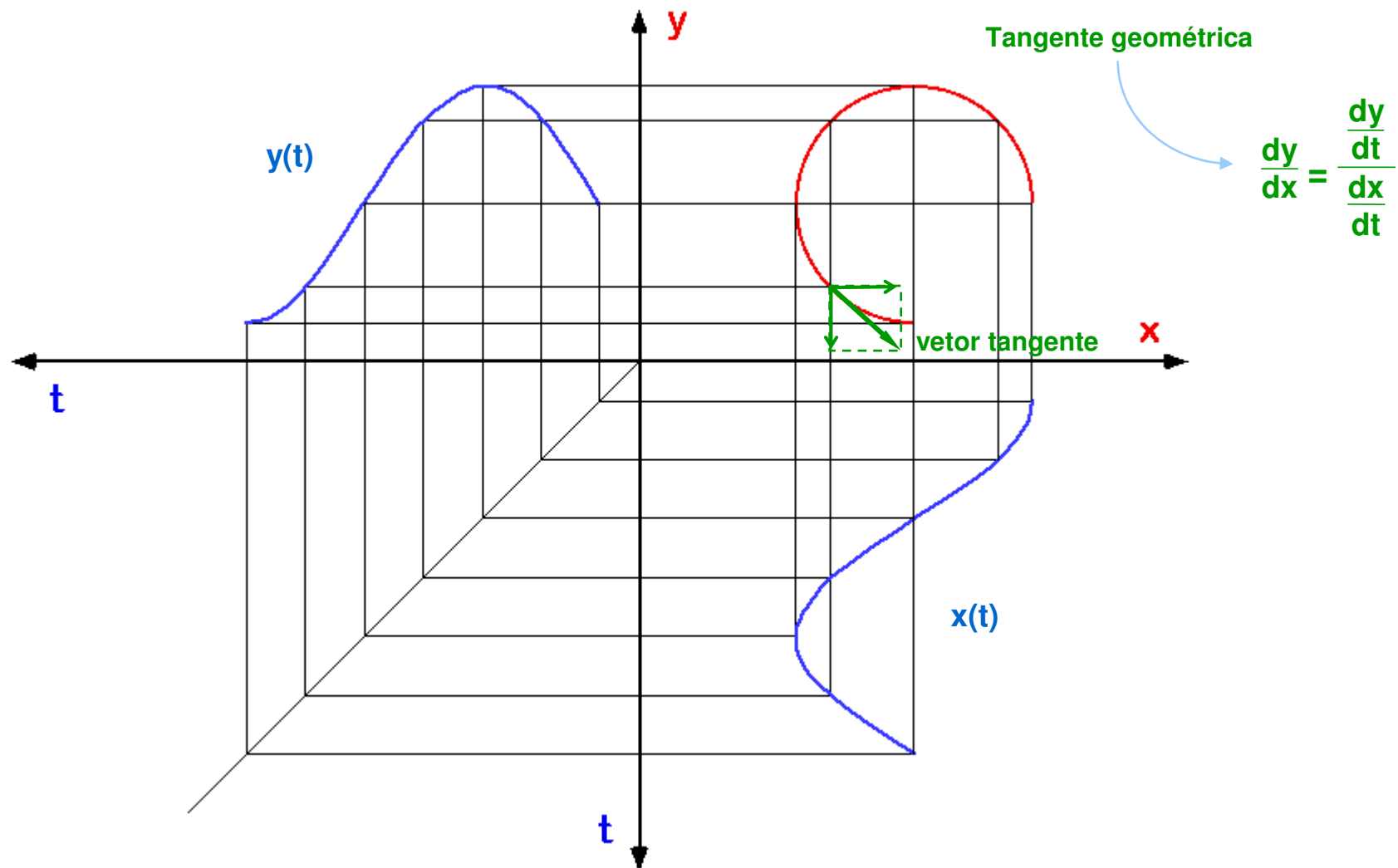
Sempre que não estiver em causa apenas uma coordenada em particular, por comodidade usar-se-á a seguinte expressão geral (vetorial):

$$Q(t) = a t^3 + b t^2 + c t + d = [t^3 \ t^2 \ t \ 1] \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = T.A$$

Conclusão:

4 coeficientes (vetoriais) arbitrários  podemos impor 4 condições

Eliminação do parâmetro em curva 2D



Exemplo com outro tipo de curva

Forma paramétrica:

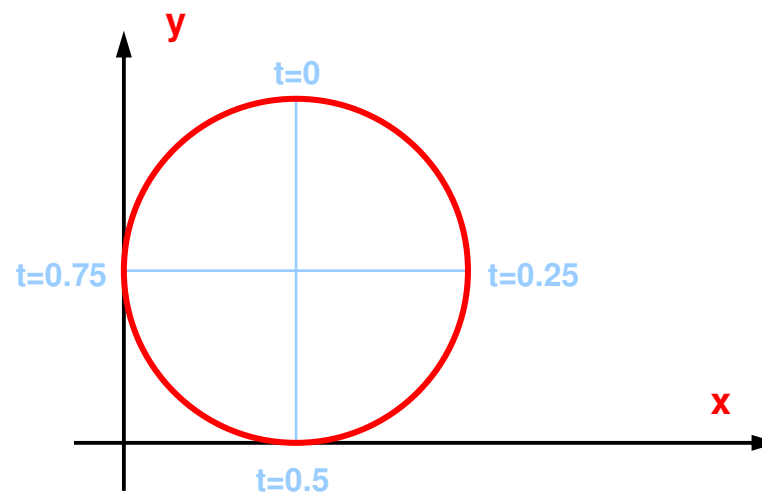
$$x = 1 + \sin(2\pi t)$$

$$y = 1 + \cos(2\pi t)$$

Componentes do vetor tangente:

$$\frac{dx}{dt} = 2\pi \cos(2\pi t)$$

$$\frac{dy}{dt} = -2\pi \sin(2\pi t)$$



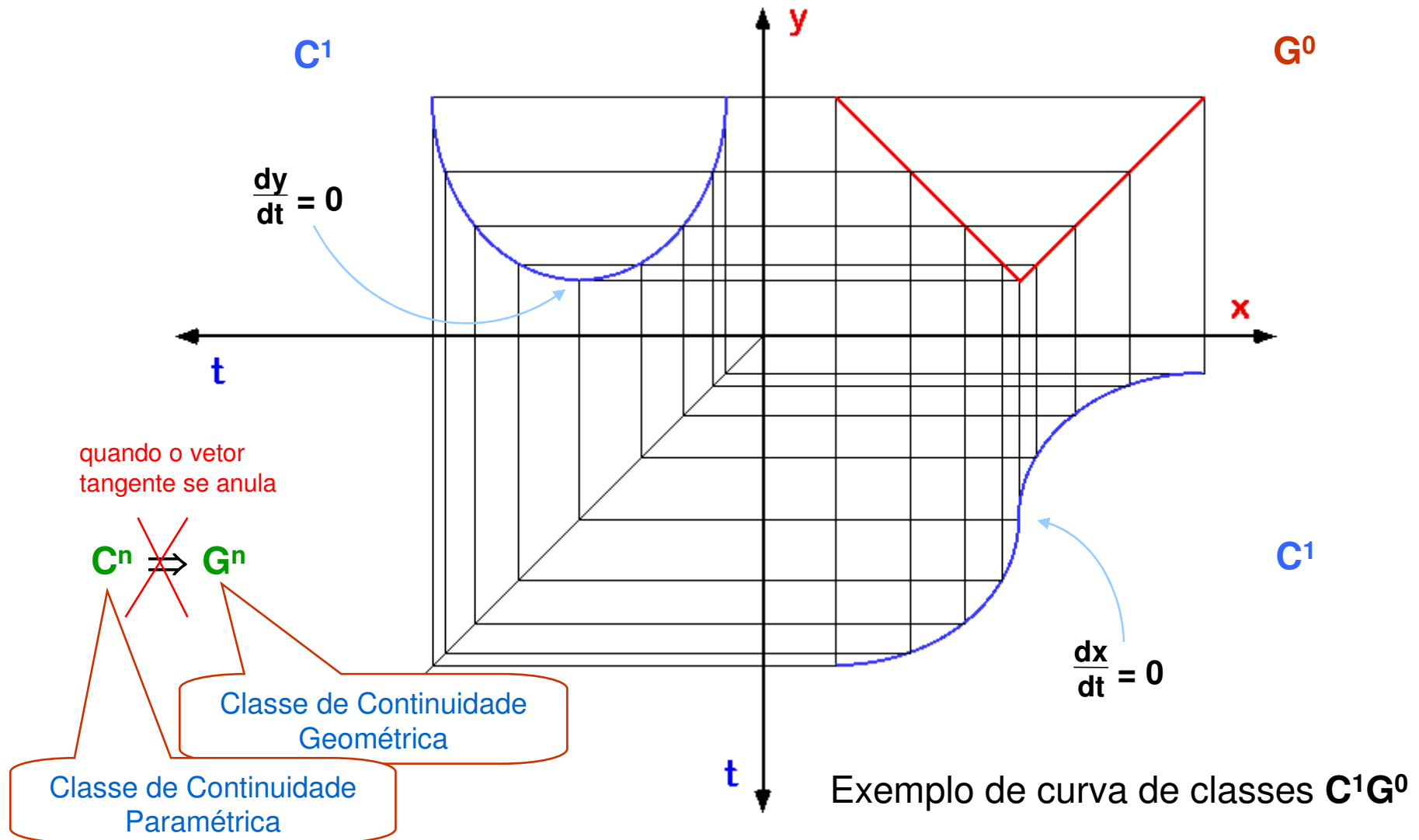
Módulo do vetor tangente:

$$|R| = \sqrt{(dx/dt)^2 + (dy/dt)^2} = 2\pi \sqrt{\cos^2(2\pi t) + \sin^2(2\pi t)} = 2\pi$$

Tangente geométrica:

$$\frac{dy}{dx} = \frac{\frac{dy}{dt}}{\frac{dx}{dt}} = -\tan(2\pi t)$$

Continuidades Paramétrica e Geométrica

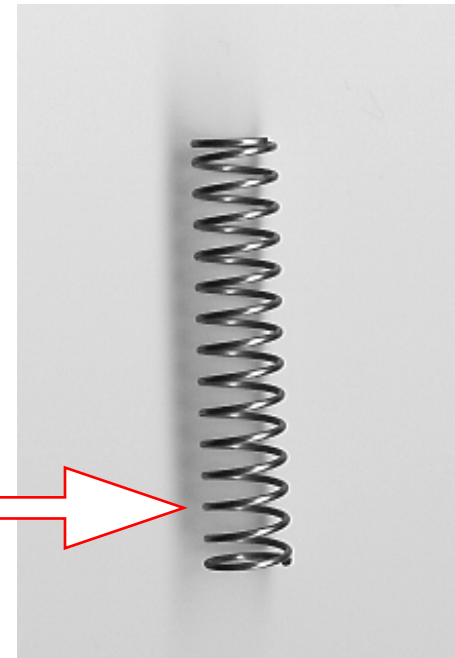


Comparação com exemplo fotográfico

Esta mola de arame não apresenta, no espaço 3D, quaisquer pontos angulosos ao longo da sua extensão...



... mas esse tipo de pontos pode aparecer numa fotografia (2D), dependendo da escolha do ponto de vista.



Curvas de Hermite

Condições: 2 pontos a interpolar e vetores tangentes nesses mesmos pontos.

$$Q(0) = P_0 \quad Q'(0) = R_0 \quad Q(1) = P_3 \quad Q'(1) = R_3$$

Na forma matricial:

$$\begin{bmatrix} Q(0) \\ Q(1) \\ Q'(0) \\ Q'(1) \end{bmatrix} = \begin{bmatrix} P_0 \\ P_3 \\ R_0 \\ R_3 \end{bmatrix}$$

Vetor da geometria de Hermite G_H

Substituindo cada elemento $Q(t)$ por $T.A$ escrever-se-á:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot A = G_H$$

Resolvendo em ordem a A dará

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot G_H = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot G_H = M_H \cdot G_H$$

Matriz de Hermite

pelo que uma curva de Hermite é da forma

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

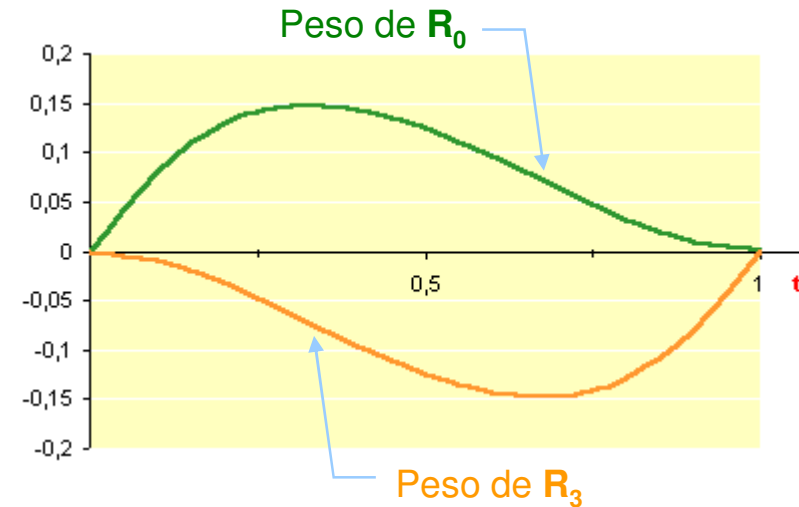
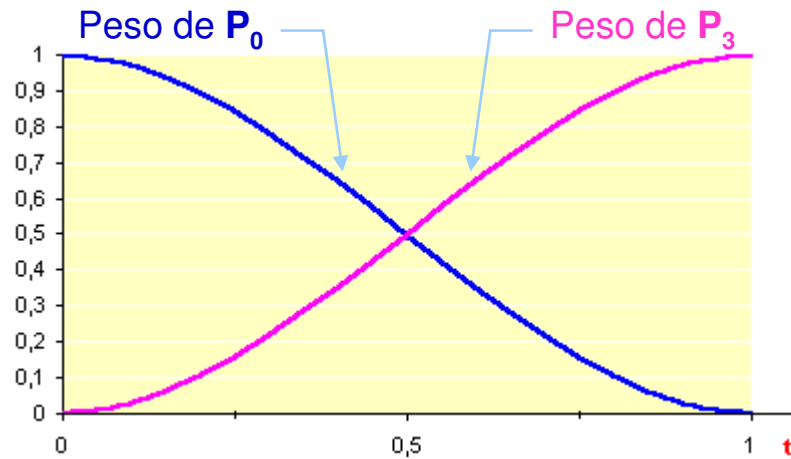
Blending functions

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

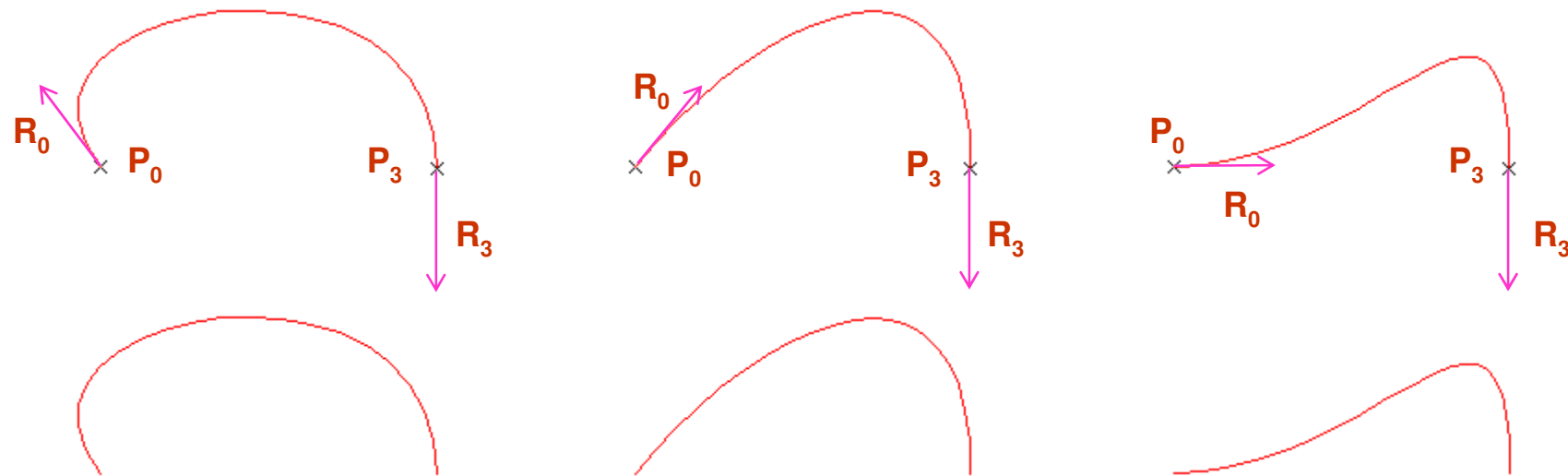
Os pesos dos elementos do vetor de geometria são funções de t :

Blending Functions

$$T \cdot M_H = [2t^3 - 3t^2 + 1 \quad -2t^3 + 3t^2 \quad t^3 - 2t^2 + t \quad t^3 - t^2]$$

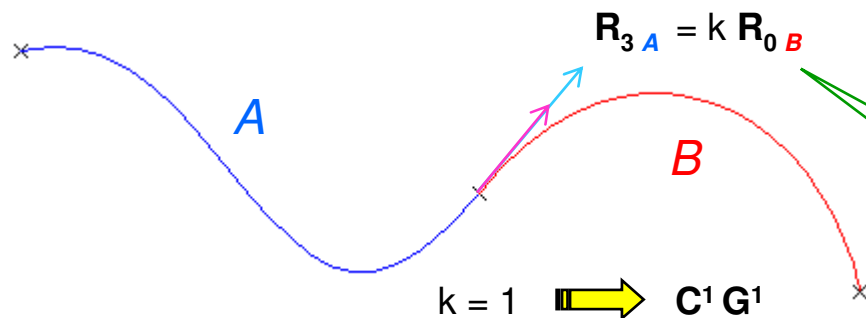
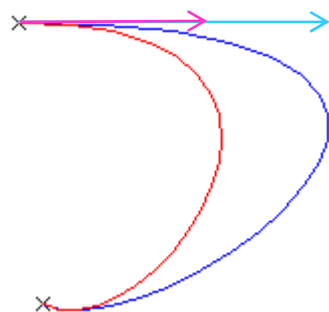


Exemplos de curvas no plano



Condição $k > 0$ para $C^0 G^1$ no ponto de junção de troços:

Influência da norma do vetor tangente:



$k = 1 \Rightarrow C^1 G^1$

o vetor tangente também será contínuo

a tangente geométrica será contínua

Curvas de Bézier

Condições: As mesmas das curvas de Hermite, introduzindo-se 2 pontos intermédios que determinam os vetores tangentes.

$$Q'(0) = R_0 = 3 (P_1 - P_0)$$

$$Q'(1) = R_3 = 3 (P_3 - P_2)$$

donde:

$$\begin{bmatrix} P_0 \\ P_3 \\ R_0 \\ R_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Vetor da geometria de Bézier

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

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

com a Matriz de Bézier:

$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Numa interface para curvas de Bézier manipulam-se apenas pontos, enquanto que para curvas de Hermite há que manipular também vetores.

Curvas de Bézier

Exemplo geral da utilização de *Blending Functions* $B_{in}(t)$ de ordem n :

$$Q(t) = P_0 B_{0n}(t) + P_1 B_{1n}(t) + P_2 B_{2n}(t) + \dots + P_m B_{mn}(t)$$

com $0 \leq t \leq 1$ e $n = m + 1$

B_{in} equivale a $B_{i,n}$

Blending Functions de ordem $n=4$ para as Curvas de Bézier:

Se $Q(t) = T \cdot M_B \cdot G_B$ então

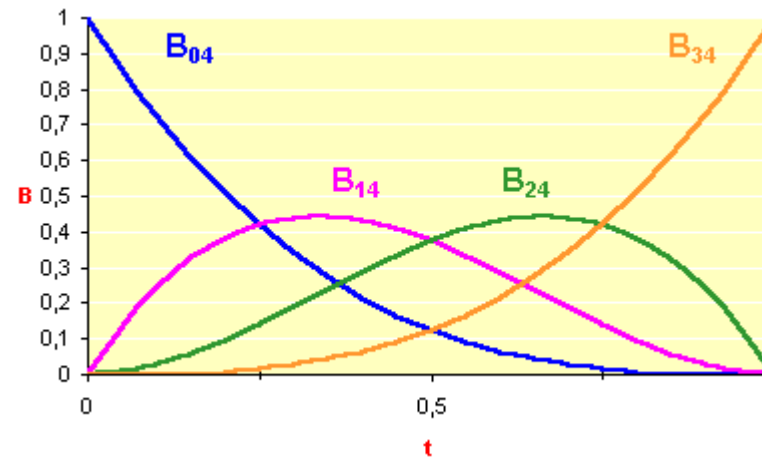
$$T \cdot M_B = [B_{04} \quad B_{14} \quad B_{24} \quad B_{34}]$$

$$B_{04}(t) = -t^3 + 3t^2 - 3t + 1$$

$$B_{14}(t) = 3t^3 - 6t^2 + 3t$$

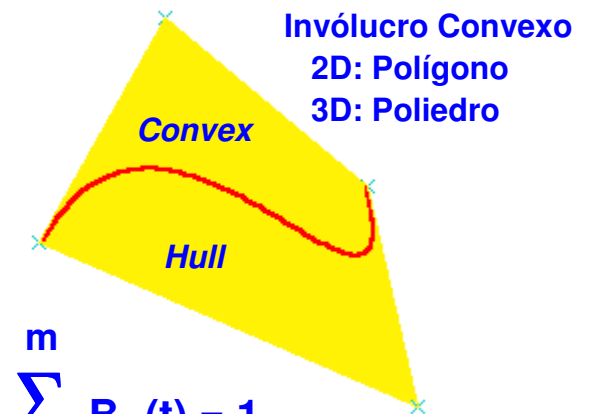
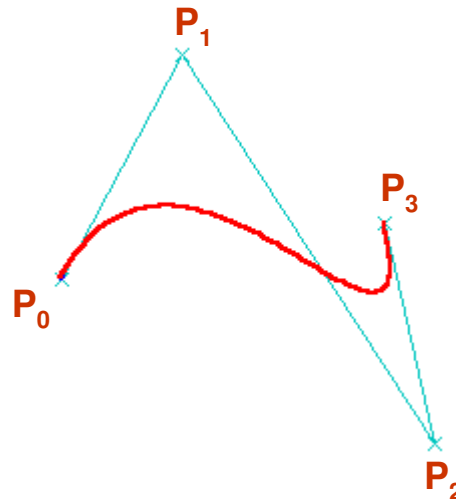
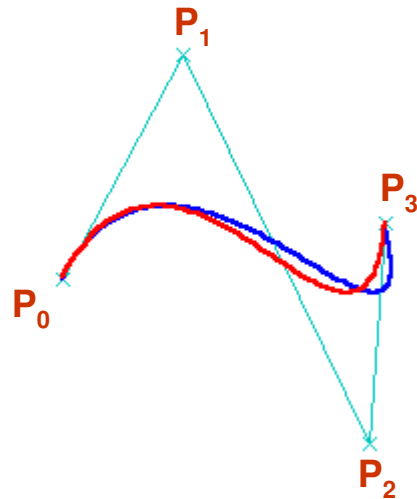
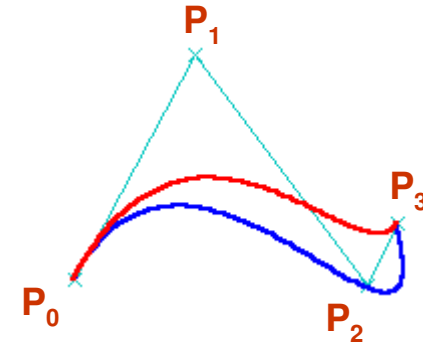
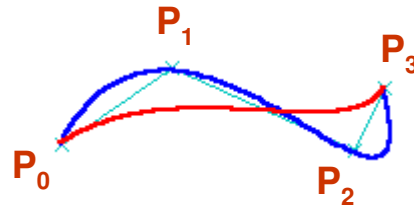
$$B_{24}(t) = -3t^3 + 3t^2$$

$$B_{34}(t) = t^3$$



Exemplo interativo

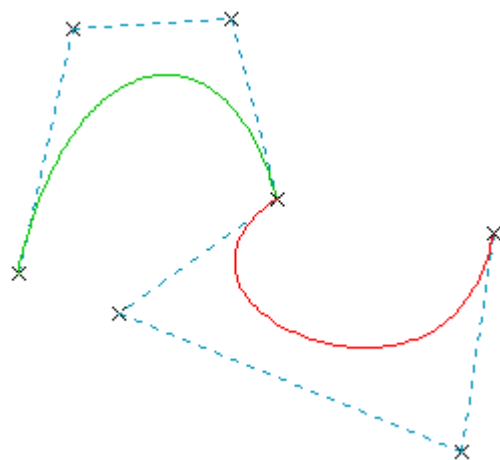
Curva pretendida:



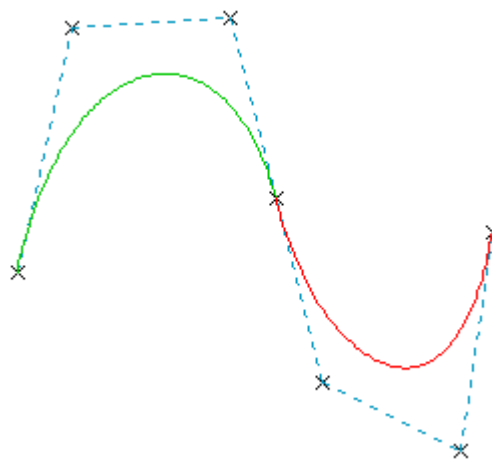
$$\sum_{i=0}^m B_{in}(t) = 1$$

$$B_{in}(t) \geq 0$$

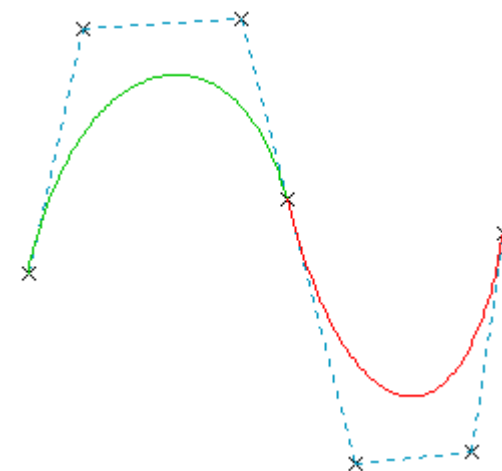
Junção de troços (segmentos) de curva



Continuidade $C^0 G^0$ na junção
(vetores tangentes com direções ou sentidos diferentes)



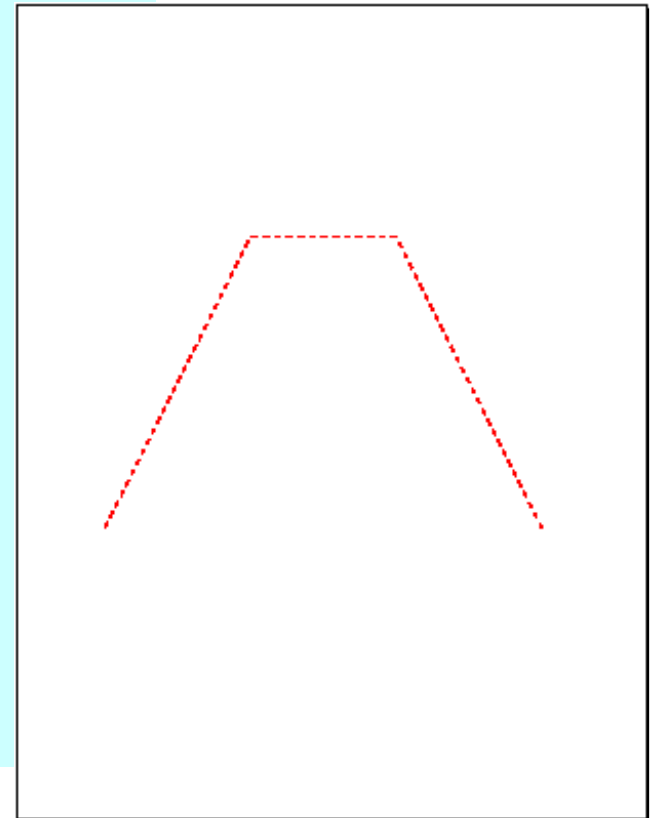
Continuidade $C^1 G^1$ na junção
(vetores tangentes iguais)



Continuidade $C^0 G^1$ na junção
(vetores tangentes apenas com a mesma direção e sentido)

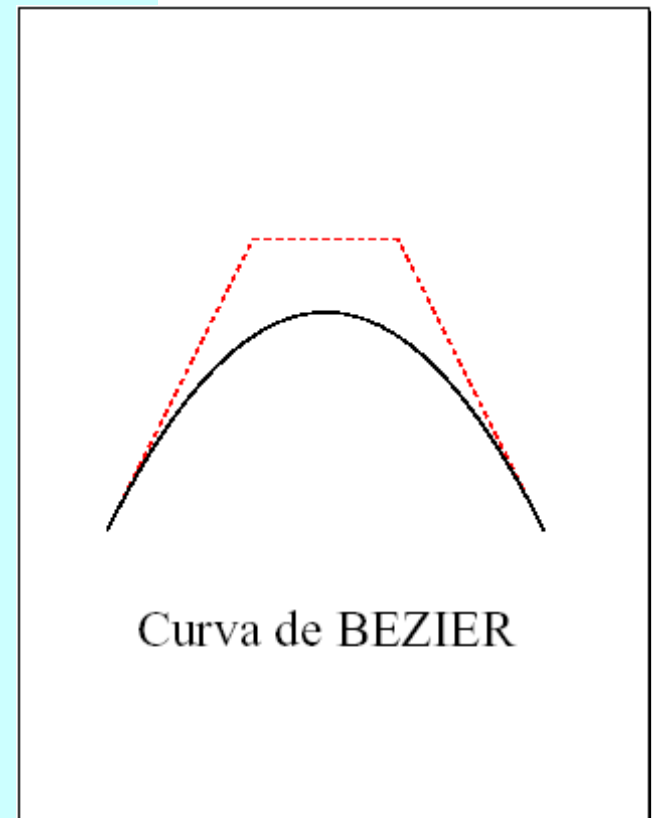
Exemplo de utilização em PostScript

```
%!PS
% Poligonal guia numa curva de Bézier (M.Próspero)
/cm {28.35 mul} def
3 cm 10 cm moveto
8 cm 20 cm lineto
13 cm 20 cm lineto
18 cm 10 cm lineto
[ 0.2 cm 0.2 cm ] 0 setdash
0.02 cm setlinewidth
1.0 0.0 0.0 setrgbcolor
stroke
showpage
```



Exemplo de utilização em PostScript

```
%!PS
% Desenho de uma curva de Bézier (M.Próspero)
/cm {28.35 mul} def
3 cm 10 cm moveto
gsave
  8 cm 20 cm lineto
  13 cm 20 cm lineto
  18 cm 10 cm lineto
  [ 0.2 cm 0.2 cm ] 0 setdash
  0.02 cm setlinewidth
  1.0 0.0 0.0 setrgbcolor
  stroke
grestore
8 cm 20 cm 13 cm 20 cm 18 cm 10 cm curveto
stroke
% Legenda:
/Times-Roman findfont
50 scalefont
setfont
4 cm 6 cm moveto
(Curva de BEZIER) show
showpage
```



Algoritmo de DE CASTELJAU

A partir dos pontos dados P_i (apresentados agora na forma P_{0i}) definem-se pontos auxiliares:

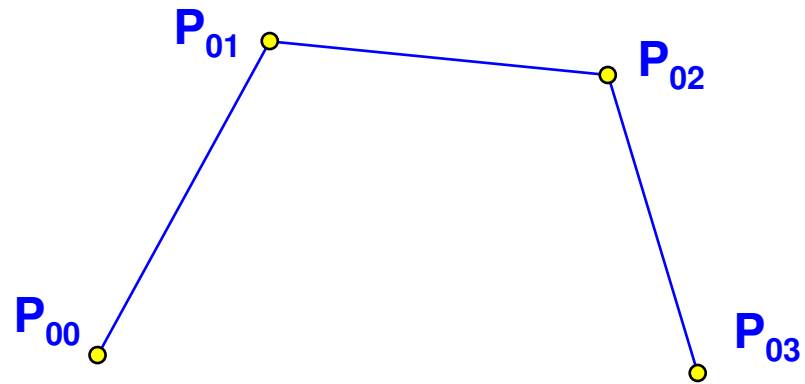
$$P_{k,n}(t) = (1 - t) P_{k-1,n-1}(t) + t P_{k-1,n}(t)$$

Exercício: Verificar que $P_{33}(t)$ corresponde à curva cúbica de Bézier.

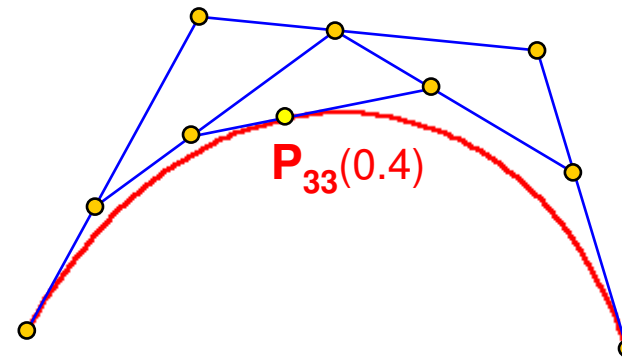
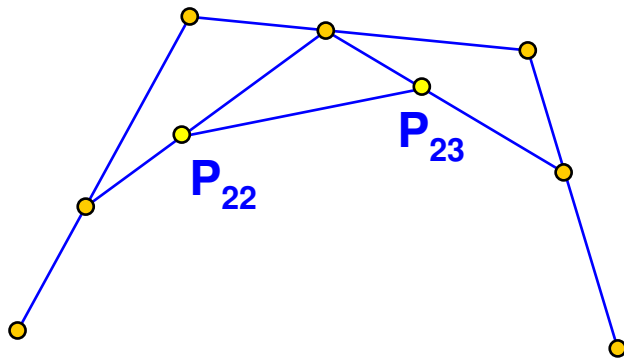
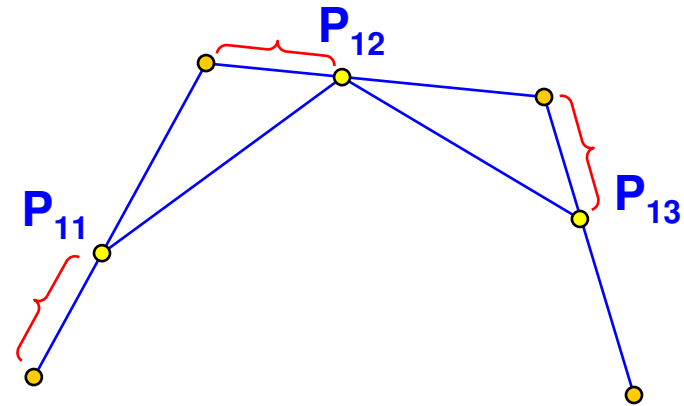
Resolução:

$$\begin{aligned} P_{33}(t) &= (1 - t) P_{22}(t) + t P_{23}(t) \\ &= (1 - t) ((1 - t) P_{11}(t) + t P_{12}(t)) + \\ &\quad t ((1 - t) P_{12}(t) + t P_{13}(t)) \\ &= (1 - t)^2 ((1 - t) P_{00} + t P_{01}) + \\ &\quad 2(t - t^2) ((1 - t) P_{01} + t P_{02}) + t^2 ((1 - t) P_{02} + t P_{03}) \\ &= (1 - t)^3 P_{00} + (3t - 6t^2 + 3t^3) P_{01} + (3t^2 - 3t^3) P_{02} + t^3 P_{03} \\ &= Q_{\text{Bézier}}(t) \end{aligned}$$

Aplicação geométrica



Para $t=0.4$:



Ver Applet em http://www.saltire.com/applets/advanced_geometry/spline/spline.htm

Aproximação Bézier-Bernstein

Generalização:

$$Q(t) = \sum_{i=0}^m P_i B_{in}(t)$$

com $0 \leq t \leq 1$ e $n = m + 1$

Curva de ordem **n**, aplicada a **m+1** pontos
e tendo como funções de peso (*Blending Functions*) os

Polinómios de Bernstein:

$$B_{k,n}(t) = \frac{(n-1)!}{k! (n-1-k)!} t^k (1-t)^{n-1-k}$$

As Curvas cúbicas de Bézier são o caso particular em que **n=4** pontos.

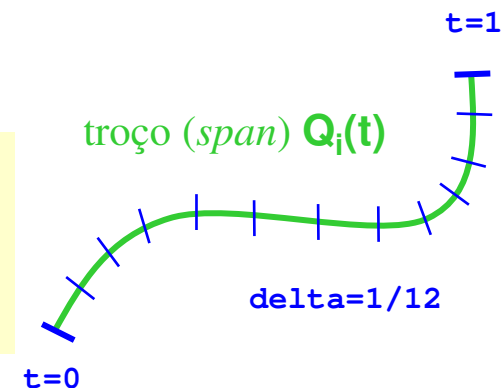
Avaliação de uma curva cúbica

Em 2D ou 3D, cada troço i tem a forma

$$Q_i(t) = T \cdot A = a * t^3 + b * t^2 + c * t + d$$

Algoritmo de visualização (aproximação a poligonal 3D):

```
for (i=first_span to last_span)
  for (j=1 to 1/delta)
    Line_3D( Q_i((j-1)*delta), Q_i(j*delta) )
```



Com este algoritmo, a curva $Q_i(t)$ é avaliada $2/\delta$ vezes (e $1+1/\delta$ recorrendo à posição corrente ou a uma variável auxiliar). Mas a eficiência aumentará se se puder reduzir o número de operações aritméticas.

Um dos métodos para tal é o de **HORNER**:

$$Q_i(t) = ((a * t + b) * t + c) * t + d$$

(Extra programa: Um outro método, ainda mais rápido, é o das **Diferenças Finitas Descendentes**)

Curvas B-Spline

$$Q(t) = T \cdot M_{Bs} \cdot G_{Bs}$$

O vetor da geometria G_{Bs} é igual ao de Bézier (G_B), sendo diferente a Matriz B-spline M_{Bs} :

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

Blending Functions:

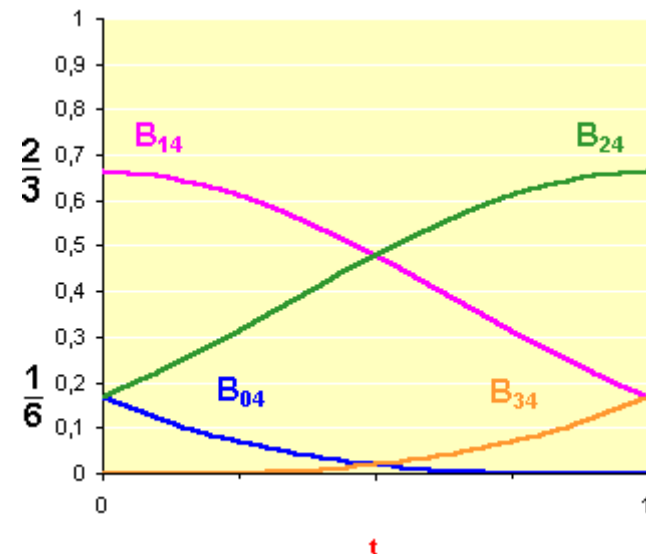
$$T \cdot M_{Bs} = [B_{04} \quad B_{14} \quad B_{24} \quad B_{34}]$$

$$B_{04}(t) = (-t^3 + 3t^2 - 3t + 1) / 6$$

$$B_{14}(t) = (3t^3 - 6t^2 + 4) / 6$$

$$B_{24}(t) = (-3t^3 + 3t^2 + 3t + 1) / 6$$

$$B_{34}(t) = t^3 / 6$$



Conclusão:

Início da curva $\Rightarrow$ vizinhança de P_1

Final da curva $\Rightarrow$ vizinhança de P_2

Propriedades: C^0

Sejam dados os seguintes vetores da geometria, geradores dos troços Q_3 e Q_4 :

$$G_{Bs_3} = \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \quad G_{Bs_4} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

Sendo

$$T \cdot M_{Bs} = \frac{1}{6} \begin{bmatrix} -t^3 + 3t^2 - 3t + 1 & 3t^3 - 6t^2 + 4 & -3t^3 + 3t^2 + 3t + 1 & t^3 \end{bmatrix}$$

e considerando o intervalo $0 \leq t \leq 1$ para cada troço, por substituição obter-se-á:

$$Q_3(1) = \frac{1}{6} \begin{bmatrix} 0 & 1 & 4 & 1 \end{bmatrix} G_{Bs_3}$$

$$Q_4(0) = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \end{bmatrix} G_{Bs_4}$$

$$Q_3(1) = Q_4(0)$$

Conclusão:  Junção dos troços num mesmo ponto, implicando continuidade C^0

Propriedades: C^1

Primeira derivada:

$$\frac{d}{dt} T \cdot M_{Bs} = \frac{1}{2} [-t^2 + 2t - 1 \quad 3t^2 - 4t \quad -3t^2 + 2t + 1 \quad t^2]$$

Considerando o intervalo $0 \leq t \leq 1$ para cada troço, por substituição obter-se-á:

$$\frac{d}{dt} Q_3(1) = \frac{1}{2} [0 \quad -1 \quad 0 \quad 1] G_{Bs_3}$$

$$\frac{d}{dt} Q_4(0) = \frac{1}{2} [-1 \quad 0 \quad 1 \quad 0] G_{Bs_4}$$

$$\frac{d}{dt} Q_3(1) = \frac{d}{dt} Q_4(0)$$

Conclusão:  Continuidade C^1 no ponto de junção dos troços

Propriedades: C^2

Segunda derivada:

$$\frac{d^2}{dt^2} \mathbf{T} \cdot \mathbf{M}_{Bs} = [-t + 1 \quad 3t - 2 \quad -3t + 1 \quad t]$$

Considerando o intervalo $0 \leq t \leq 1$ para cada troço, por substituição obter-se-á:

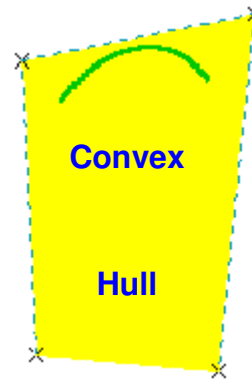
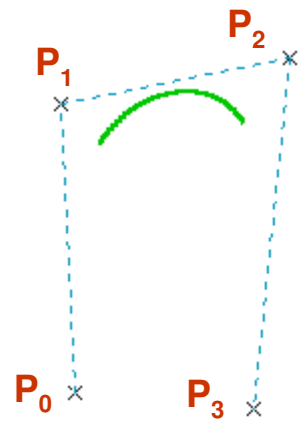
$$\frac{d^2}{dt^2} \mathbf{Q}_3(1) = [0 \quad 1 \quad -2 \quad 1] \mathbf{G}_{Bs3}$$

$$\frac{d^2}{dt^2} \mathbf{Q}_4(0) = [1 \quad -2 \quad 1 \quad 0] \mathbf{G}_{Bs4}$$

$$\frac{d^2}{dt^2} \mathbf{Q}_3(1) = \frac{d^2}{dt^2} \mathbf{Q}_4(0)$$

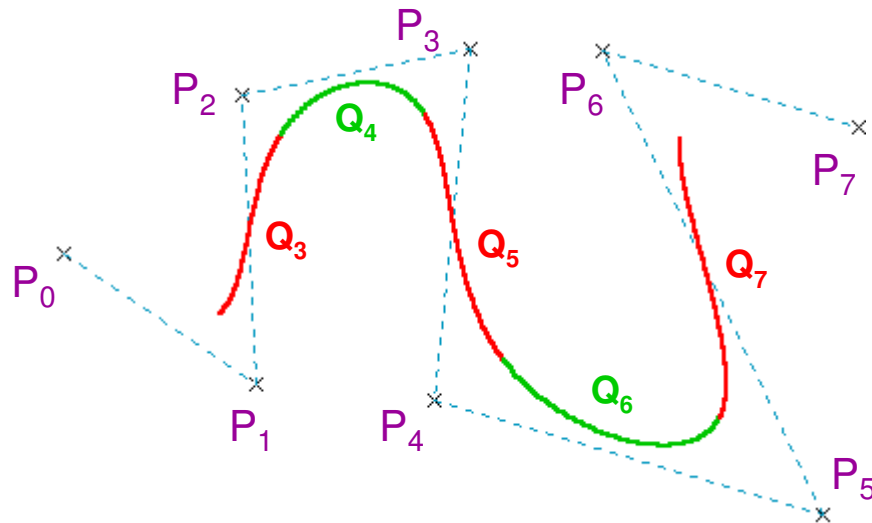
Conclusão:  Continuidade C^2 no ponto de junção dos troços

Exemplos de curvas B-spline em 2D



$$\sum_{i=0}^m B_{in}(t) = 1$$

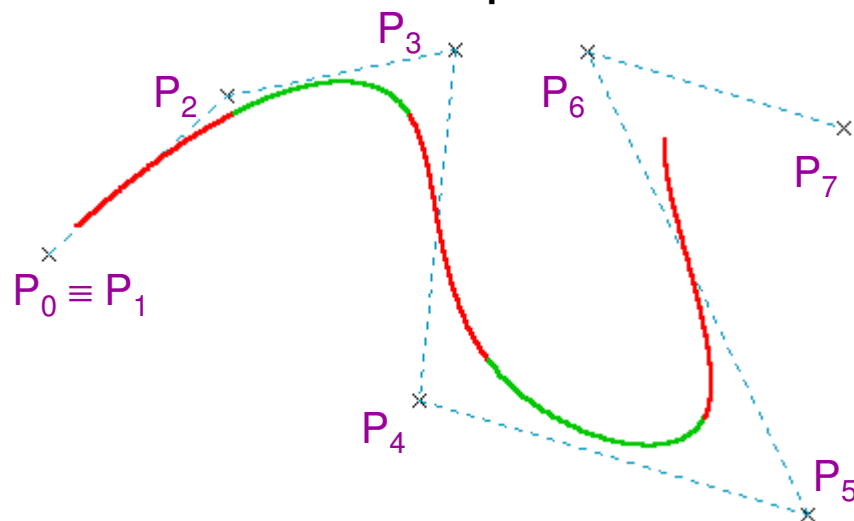
$$B_{in}(t) \geq 0$$



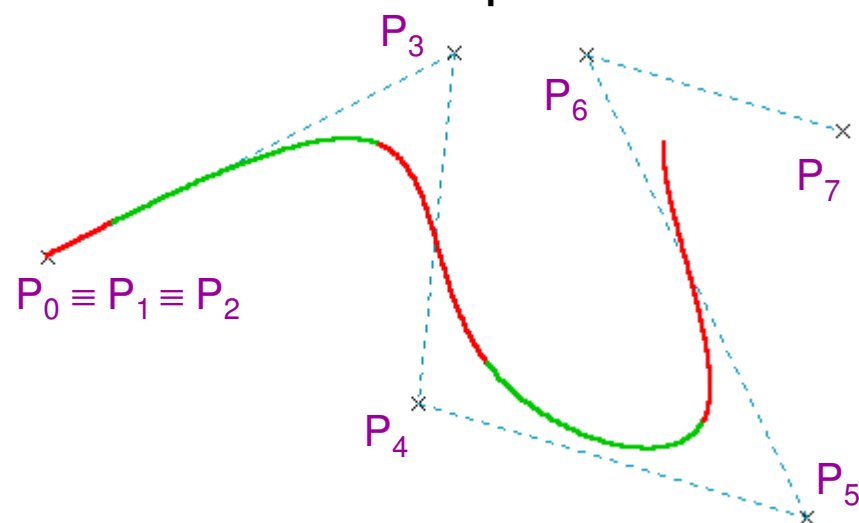
$$G_{Bsi} = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}$$

Grau de multiplicidade de pontos

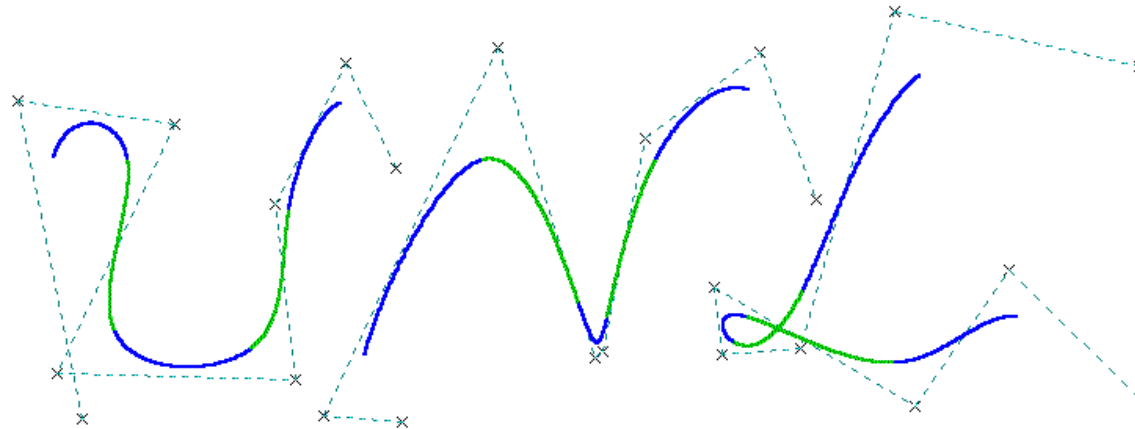
Ponto com Grau de Multiplicidade 2 :



Ponto com Grau de Multiplicidade 3 :

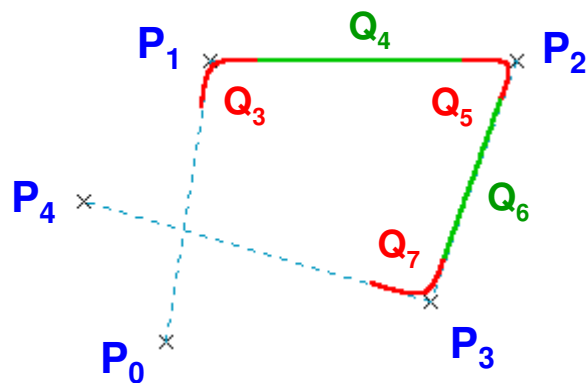


Outros exemplos (1)



Pontos com

Grau de Multiplicidade 2 :



$$G_{Bs3} = \begin{bmatrix} P_0 \\ P_1 \\ P_1 \\ P_2 \end{bmatrix}$$

$$G_{Bs4} = \begin{bmatrix} P_1 \\ P_1 \\ P_2 \\ P_2 \end{bmatrix}$$

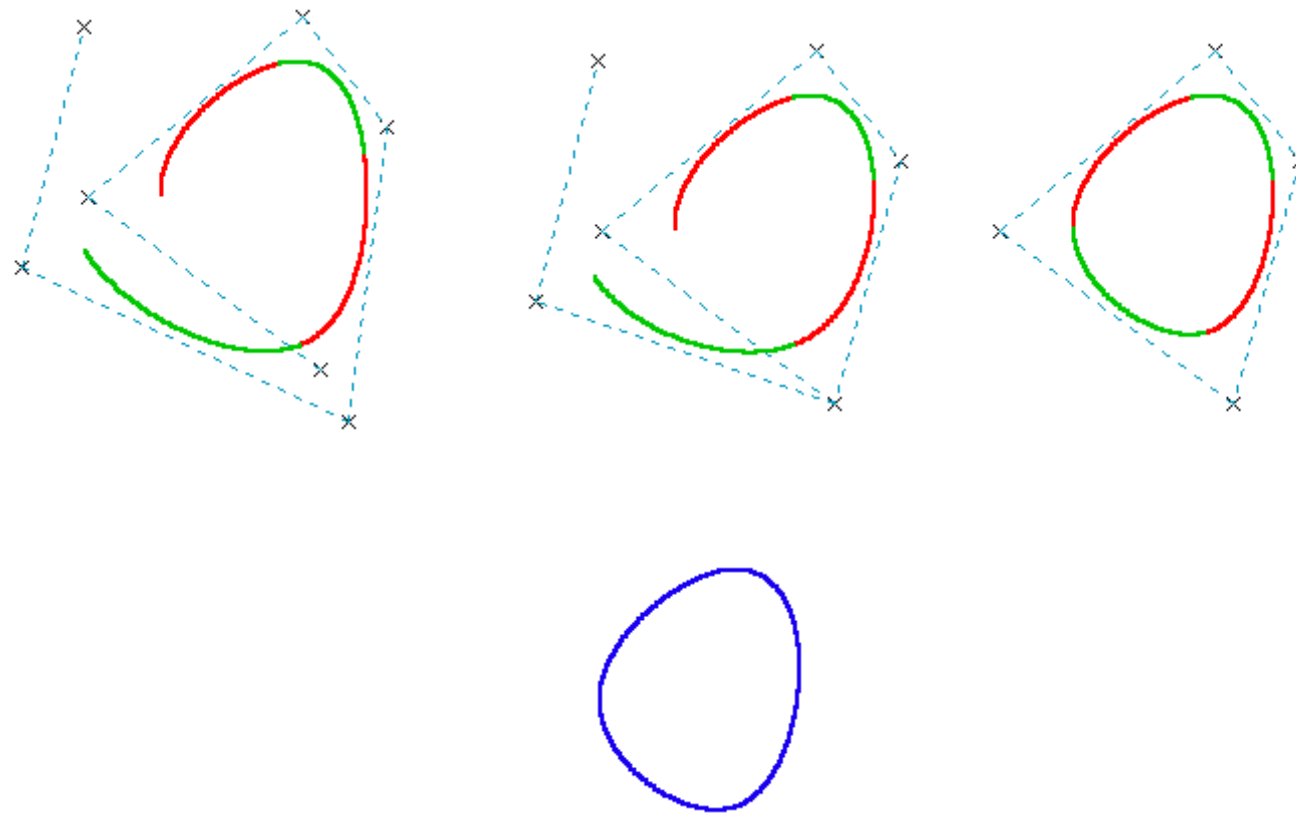
$$G_{Bs5} = \begin{bmatrix} P_1 \\ P_2 \\ P_2 \\ P_3 \end{bmatrix}$$

$$G_{Bs6} = \begin{bmatrix} P_2 \\ P_2 \\ P_3 \\ P_3 \end{bmatrix}$$

$$G_{Bs7} = \begin{bmatrix} P_2 \\ P_3 \\ P_3 \\ P_4 \end{bmatrix}$$

Outros exemplos (2)

Construção de uma curva fechada :



Continuidade e Interpolação

◆ Resultado dos graus de multiplicidade dos pontos de controlo:

- Multiplicidade 1 → continuidade $C^2 G^2$
- Multiplicidade 2 → continuidade $C^2 G^1$
- Multiplicidade 3 → continuidade $C^2 G^0$
- Multiplicidade 3 → interpola esse ponto de controlo

Interpretação geométrica:
não continuidade da
curvatura

◆ Haverá alternativa à interpolação no(s) extremo(s) de uma curva sem ser pelo aumento do grau de multiplicidade do(s) ponto(s), permitindo manter continuidade de classe $C^2 G^2$ com futuros troços adjacentes?

- Sim, por criação de novo(s) ponto(s) de controlo dito(s) “fantasma(s)”

Pontos de controlo “fantasmas” (1)

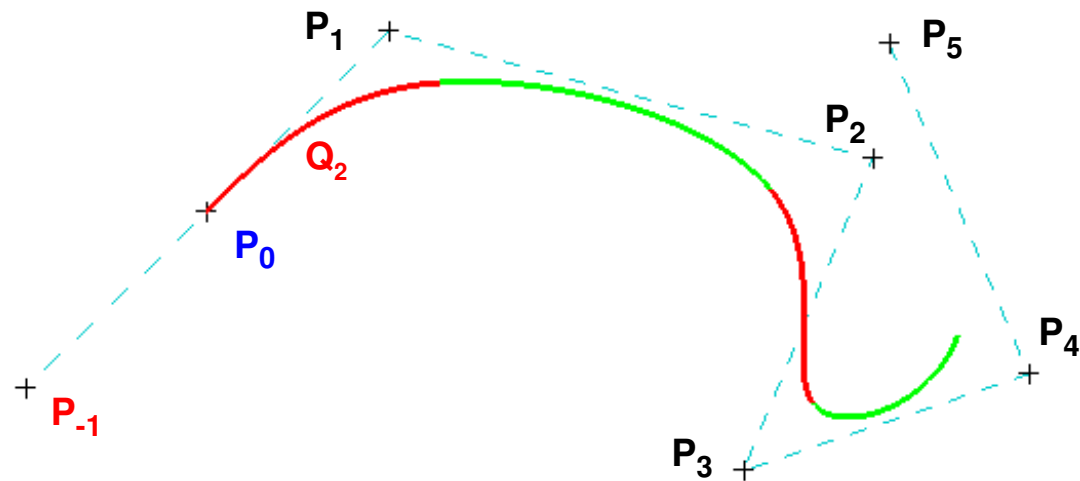
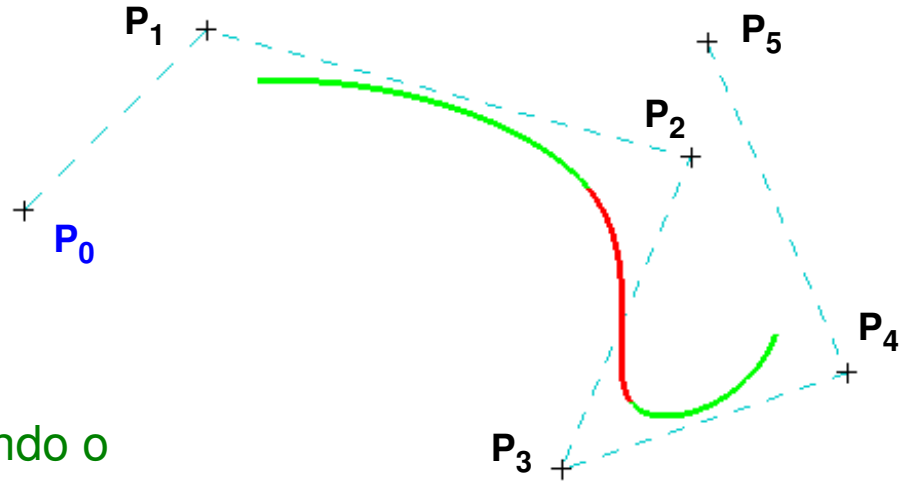
Admitindo que se quer interpolar P_0 :

Resolve-se matematicamente encontrando o ponto “fantasma” P_{-1} que verifique a condição

$$Q_2(t=0) = T \cdot M_{Bs} \cdot \begin{bmatrix} P_{-1} \\ P_0 \\ P_1 \\ P_2 \end{bmatrix} = P_0$$

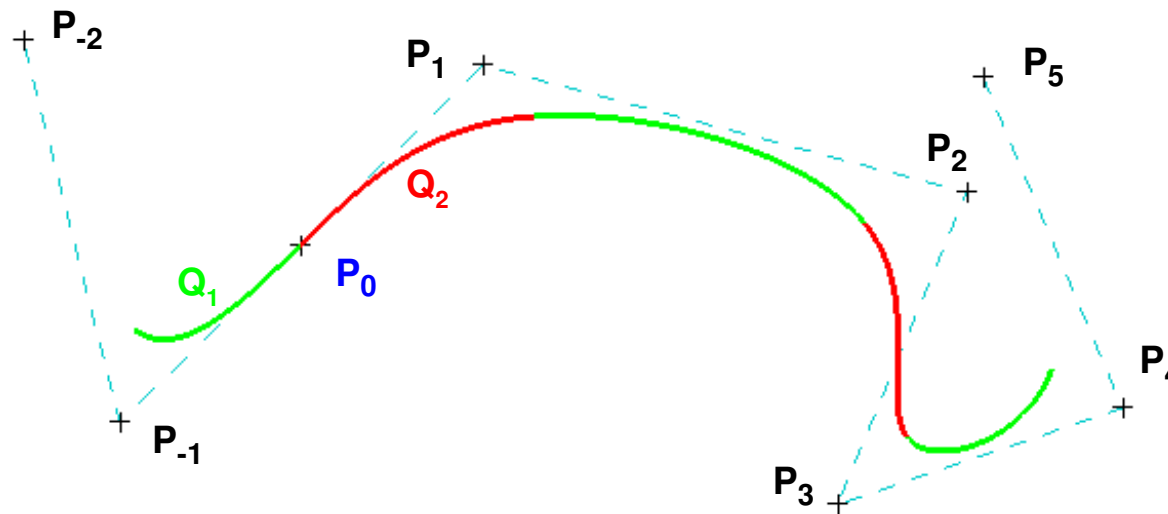
$$(P_{-1} + 4P_0 + P_1) / 6 = P_0$$

$$P_{-1} = 2P_0 - P_1$$



Pontos de controlo “fantasmas” (2)

Construção de um troço Q_1 adjacente a Q_2 , que também irá interpolar P_0 , mantendo-se continuidade $C^2 G^2$ em todas as junções de troços da curva (o que não seria possível se P_0 tivesse grau de multiplicidade superior a 1):



Curvas de Catmull-Rom

ou Splines de Overhauser

$$\mathbf{Q}(t) = \mathbf{T} \cdot \mathbf{M}_{\text{CR}} \cdot \mathbf{G}_{\text{Bs}}$$

$$\mathbf{M}_{\text{CR}} = \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

O vetor da geometria é igual ao de B-spline ($\mathbf{G}_{\text{Bs}}$)

Blending Functions:

$$\mathbf{T} \cdot \mathbf{M}_{\text{CR}} = [\mathbf{B}_{04} \quad \mathbf{B}_{14} \quad \mathbf{B}_{24} \quad \mathbf{B}_{34}]$$

$$\mathbf{B}_{04}(t) = (-t^3 + 2t^2 - t) / 2$$

$$\mathbf{B}_{14}(t) = (3t^3 - 5t^2 + 2) / 2$$

$$\mathbf{B}_{24}(t) = (-3t^3 + 4t^2 + t) / 2$$

$$\mathbf{B}_{34}(t) = (t^3 - t^2) / 2$$

As curvas não gozam da propriedade de *convex hull*

Propriedades

Considerando o intervalo $0 \leq t \leq 1$ para cada troço:

$$Q_i(0) = \frac{1}{2} \begin{bmatrix} 0 & 2 & 0 & 0 \end{bmatrix} G_{Bs_i} = P_{i-2}$$

$$Q_i(1) = \frac{1}{2} \begin{bmatrix} 0 & 0 & 2 & 0 \end{bmatrix} G_{Bs_i} = P_{i-1}$$

$$Q_{i+1}(0) = \frac{1}{2} \begin{bmatrix} 0 & 2 & 0 & 0 \end{bmatrix} G_{Bs_{i+1}} = P_{i-1}$$

$$Q_i(1) = Q_{i+1}(0)$$

C^0 e interpolação de pontos de controlo

$$\frac{d}{dt} T \cdot M_{CR} = \frac{1}{2} \begin{bmatrix} -3t^2 + 4t - 1 & 9t^2 - 10t & -9t^2 + 8t + 1 & 3t^2 - 2t \end{bmatrix}$$

$$\frac{d}{dt} Q_i(1) = \frac{1}{2} \begin{bmatrix} 0 & -1 & 0 & 1 \end{bmatrix} G_{Bs_i} = \frac{P_i - P_{i-2}}{2}$$

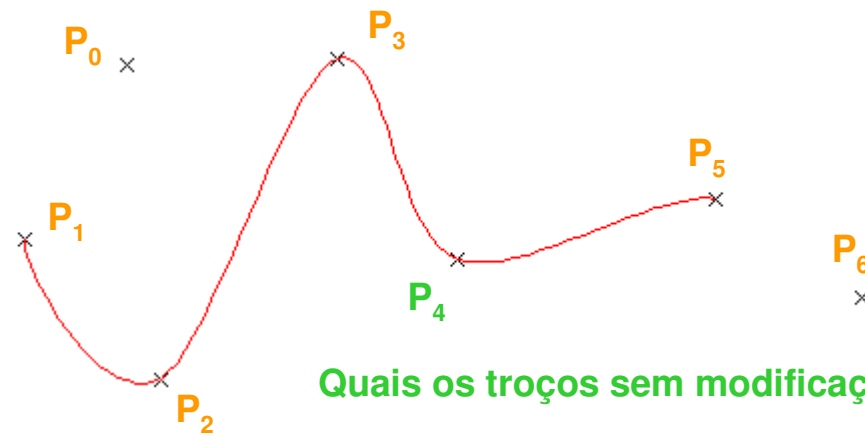
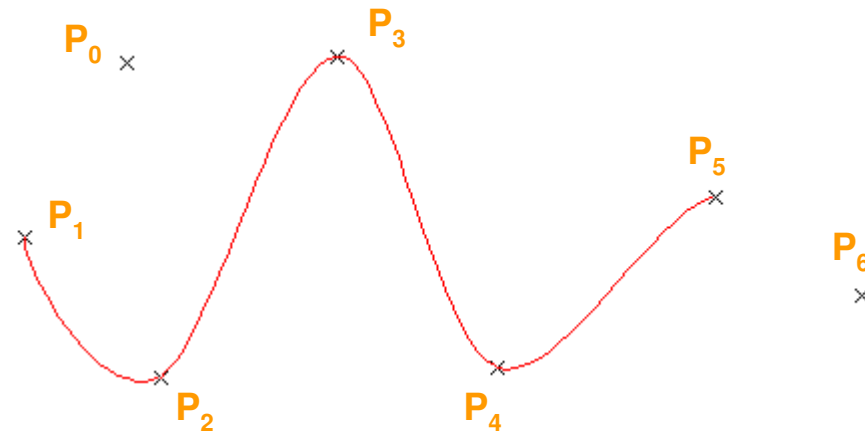
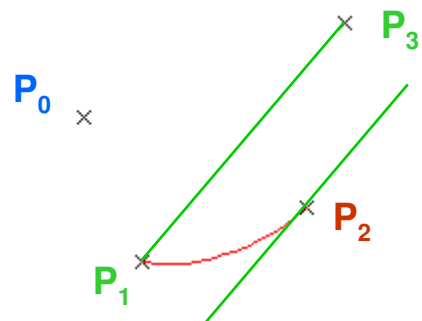
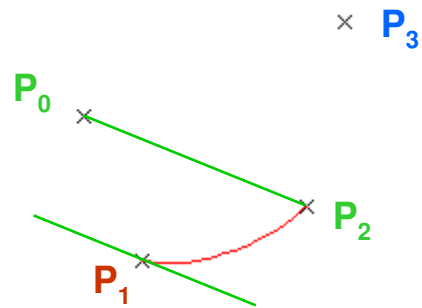
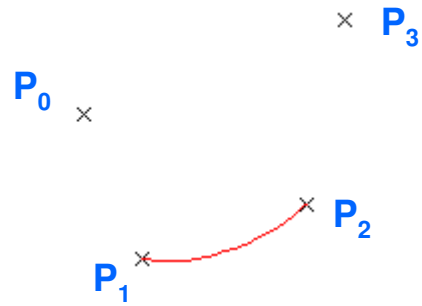
$$\frac{d}{dt} Q_{i+1}(0) = \frac{1}{2} \begin{bmatrix} -1 & 0 & 1 & 0 \end{bmatrix} G_{Bs_{i+1}} = \frac{P_i - P_{i-2}}{2}$$

$$\frac{d}{dt} Q_i(1) = \frac{d}{dt} Q_{i+1}(0)$$

Continuidade C^1

Exemplos

Direcção dos vetores tangentes



Quais os troços sem modificação?

Conversão entre curvas cúbicas

Se for construído (ou se já existir num determinado sistema) um avaliador para um certo tipo de curvas cúbicas, é possível usá-lo com qualquer outra forma de curva cúbica desde que se proceda a uma conversão entre curvas.

É o que se passa com os avaliadores (eficientes) para curvas de Bézier já existentes em PostScript e em OpenGL.

PROBLEMA:

Dada a curva $Q(t) = T \cdot M_{Bs} \cdot G_{Bs}$

encontrar G_B tal que $Q(t) = T \cdot M_B \cdot G_B$

Conversão para curva de Bézier

Solução:

$$\mathbf{M}_B \cdot \mathbf{G}_B = \mathbf{M}_{Bs} \cdot \mathbf{G}_{Bs}$$

$$\mathbf{G}_B = \mathbf{M}_B^{-1} \cdot \mathbf{M}_{Bs} \cdot \mathbf{G}_{Bs}$$

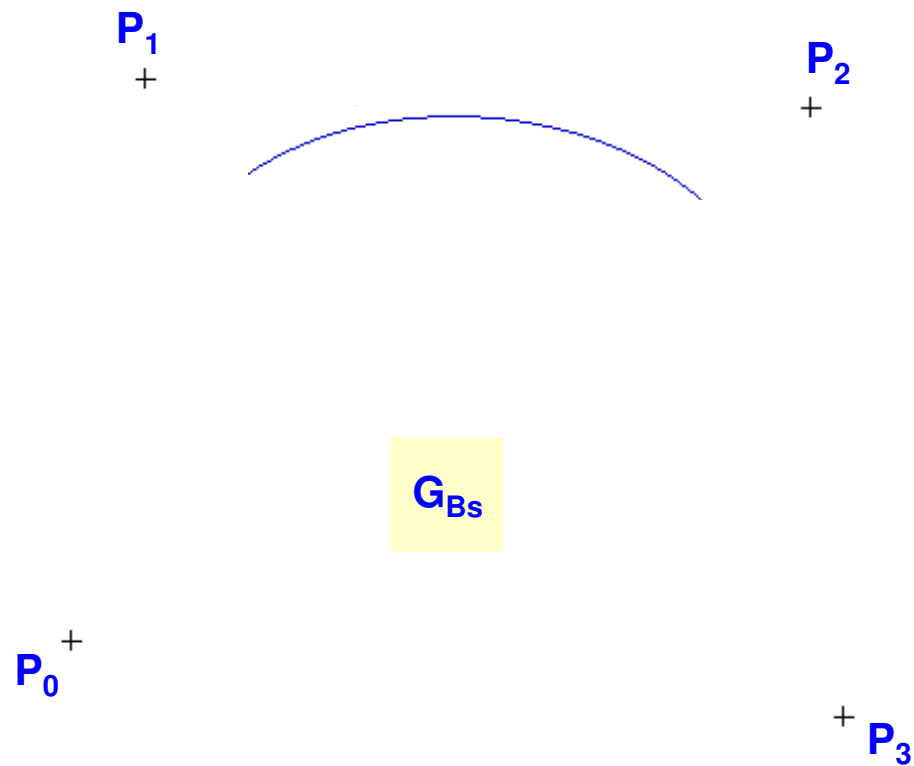
em que

$$\mathbf{M}_B^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1/3 & 1 \\ 0 & 1/3 & 2/3 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

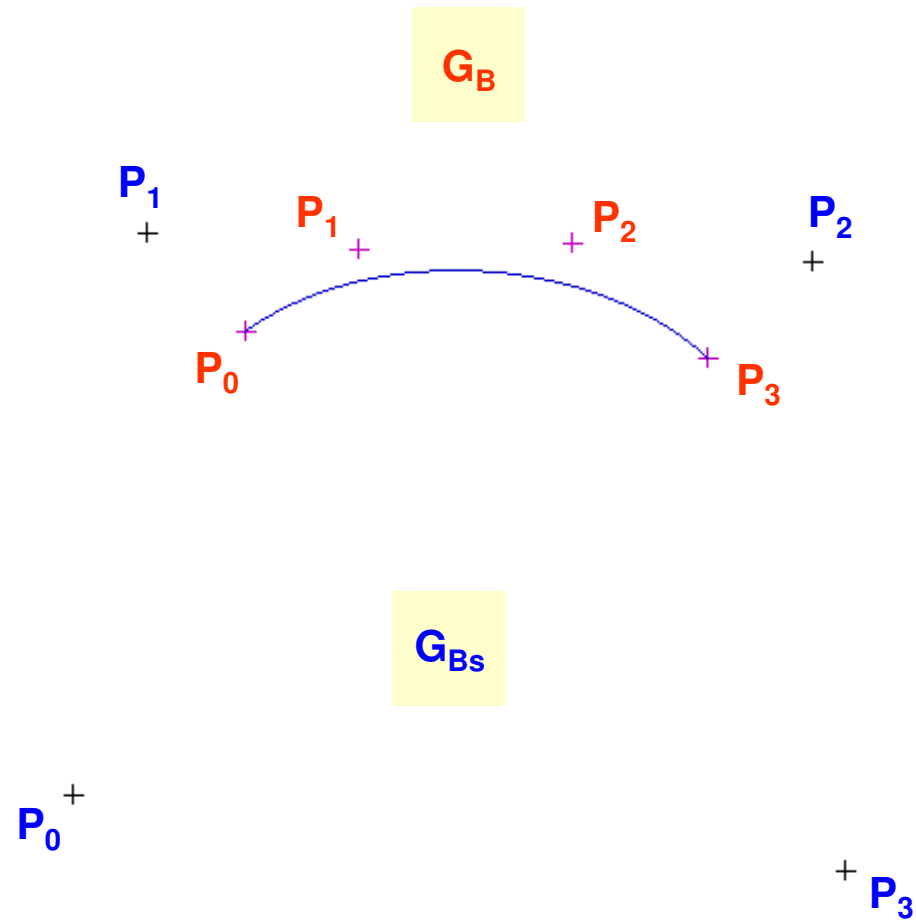
Conclusão:

Qualquer curva cúbica pode ser convertida numa curva cúbica de Bézier

Exemplo de conversão B-spline→Bézier

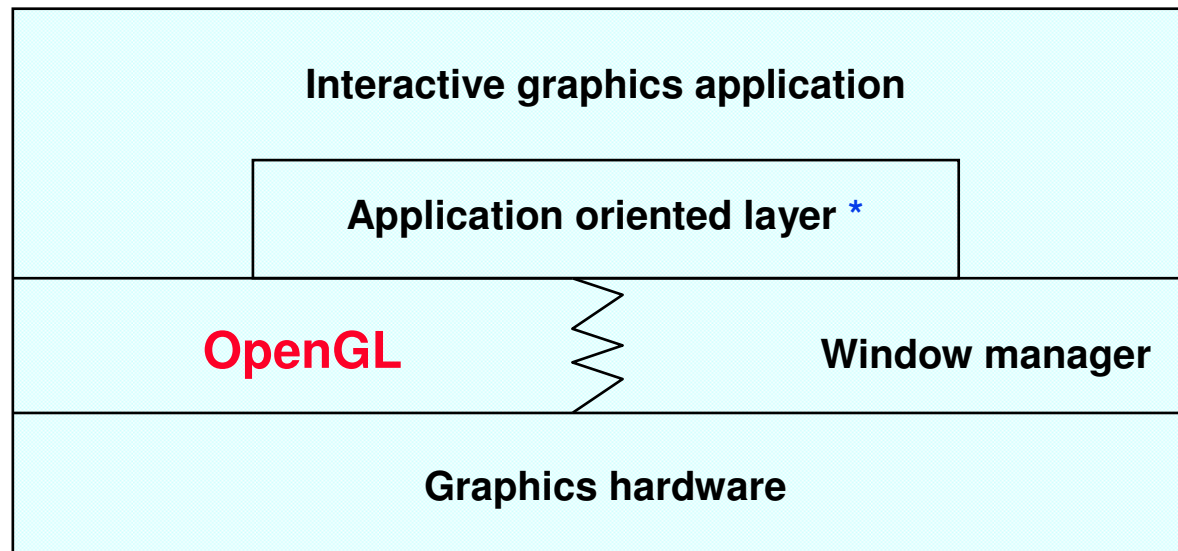


Exemplo de conversão B-spline→Bézier



The OpenGL graphics system

Layer model:



* e.g.

GLUT
JOGL
LWJGL
Open Inventor


```
// Casa.c
// Author: M.Próspero
```

```
#include <GL/glut.h>
```

```
void Desenhar (void)
```

```
{
    glMatrixMode (GL_MODELVIEW);
    glLoadIdentity ();
    glClear (GL_COLOR_BUFFER_BIT);

    glColor3f (0.0, 0.0, 0.0); // paredes:
    glBegin (GL_LINES);
    glVertex2f (120.0, -220.0);
    glVertex2f (120.0, -140.0);
    glVertex2f (280.0, -220.0);
    glVertex2f (280.0, -140.0);
    glEnd ();

    glColor3f (0.0, 0.0, 1.0); // porta:
    glBegin (GL_LINE_STRIP);
    glVertex2f (140.0, -220.0);
    glVertex2f (140.0, -160.0);
    glVertex2f (180.0, -160.0);
    glVertex2f (180.0, -220.0);
    glEnd ();

    glColor3f (1.0, 0.0, 1.0); // janela:
    glBegin (GL_LINE_LOOP);
    glVertex2i (200, -160);
    glVertex2i (260, -160);
    glVertex2i (260, -200);
    glVertex2i (200, -200);
    glEnd ();

    glColor3f (1.0, 0.0, 0.0); // telhado:
    glBegin (GL_TRIANGLES);
    glVertex2f (100.0, -140.0);
    glVertex2f (200.0, -60.0);
    glVertex2f (300.0, -140.0);
    glEnd ();

    glFlush ();
}
```

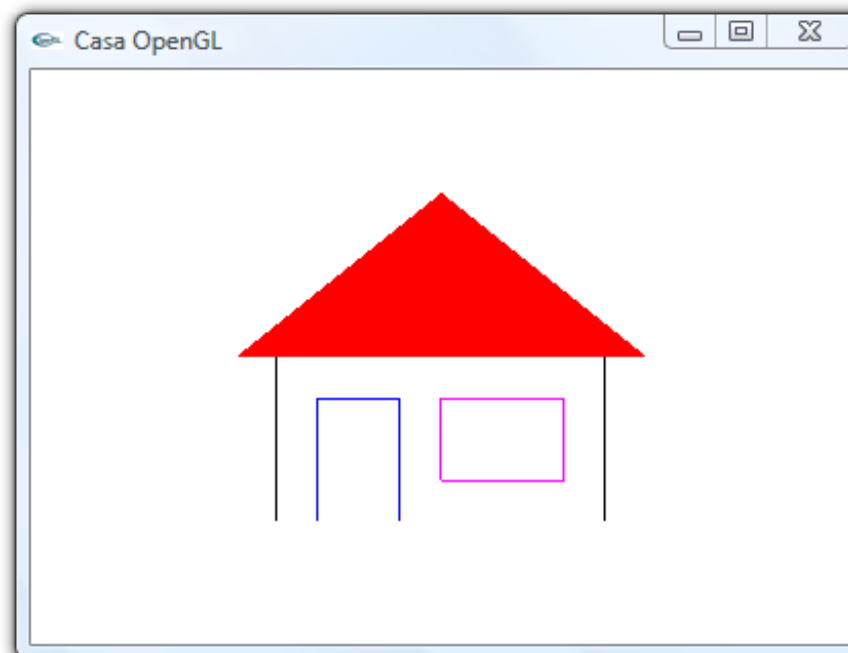
OpenGL
com C

```
void Enquadrar (GLsizei w, GLsizei h)
```

```
{
    glViewport (0, 0, w, h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluOrtho2D (0.0, 400.0, -280.0, 0.0);
}
```

```
int main (int argc, char** argv)
```

```
{
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (0, 0);
    glutInitWindowSize (400, 280);
    glutCreateWindow ("Casa OpenGL");
    glClearColor (1.0, 1.0, 1.0, 0.0);
    glutReshapeFunc (Enquadrar);
    glutDisplayFunc (Desenhar);
    glutMainLoop ();
    return (0);
}
```



```
void glBegin(GLenum mode);
```

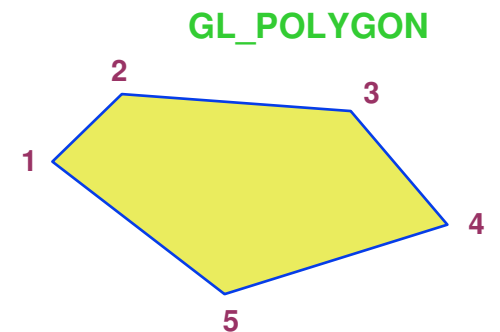
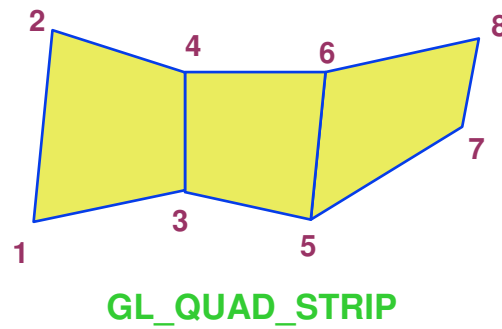
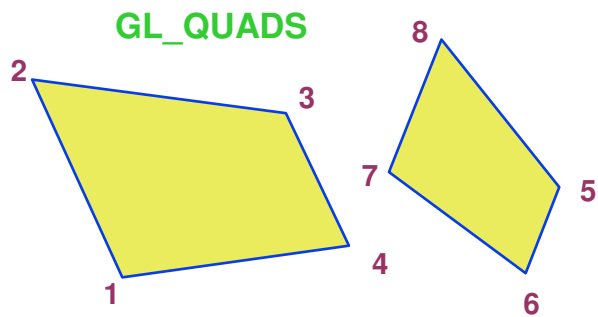
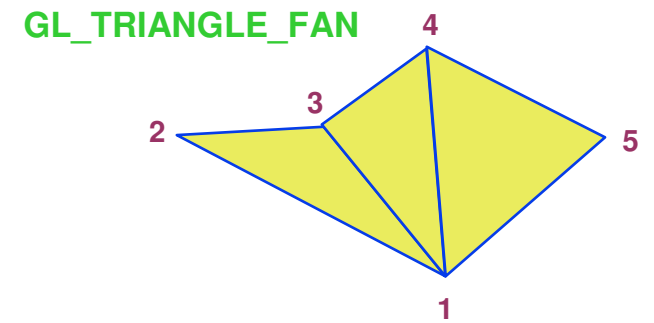
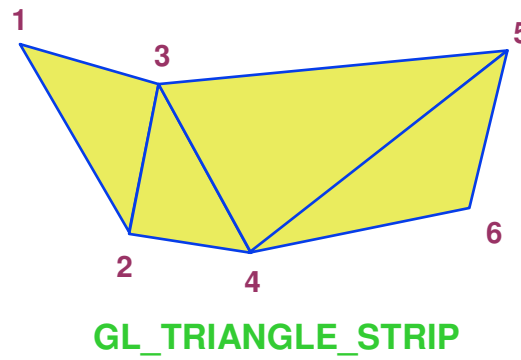
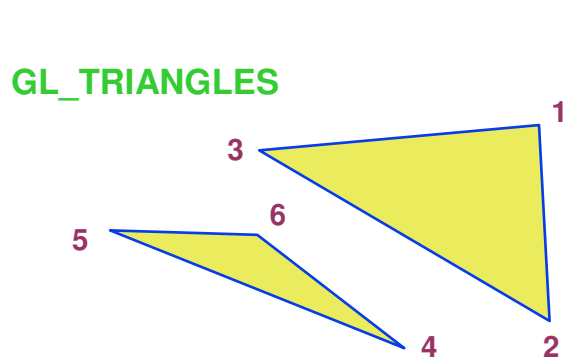
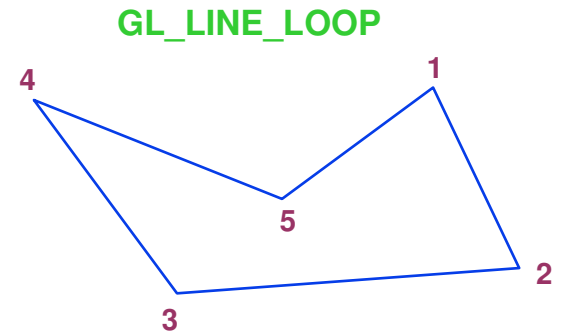
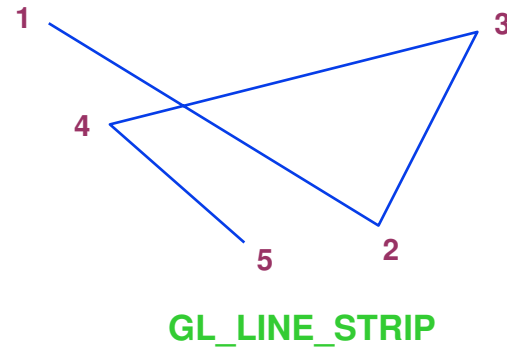
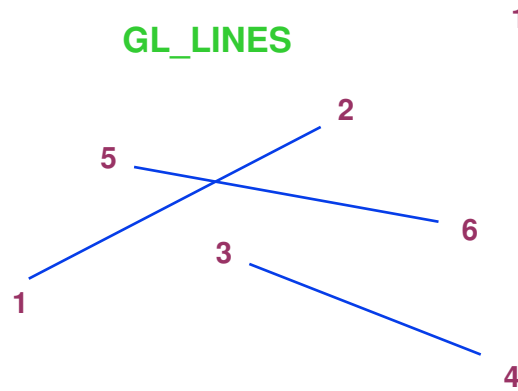
Marks the beginning of a vertex-data list that describes a geometric primitive. The type of primitive is indicated by *mode*, which can be any of the values shown in Table 2-2.

Value	Meaning
GL_POINTS	individual points
GL_LINES	pairs of vertices interpreted as individual line segments
GL_LINE_STRIP	series of connected line segments
GL_LINE_LOOP	same as above, with a segment added between last and first vertices
GL_TRIANGLES	triples of vertices interpreted as triangles
GL_TRIANGLE_STRIP	linked strip of triangles
GL_TRIANGLE_FAN	linked fan of triangles
GL_QUADS	quadruples of vertices interpreted as four-sided polygons
GL_QUAD_STRIP	linked strip of quadrilaterals
GL_POLYGON	boundary of a simple, <u>convex</u> polygon

Table 2-2 Geometric Primitive Names and Meanings

```
void glEnd(void);
```

Marks the end of a vertex-data list.



```

import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GL;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCanvas;
import javax.media.opengl.glu.GLU;
/**
 * @author M. Próspero
 */
public class Casa implements GLEventListener {

    public void init(GLAutoDrawable gLDrawable) {
        GL gl = gLDrawable.getGL();
        gl.glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
    }

    public void display(GLAutoDrawable gLDrawable) {
        GL gl = gLDrawable.getGL();
        gl.glClear(GL.GL_COLOR_BUFFER_BIT);
        gl.glMatrixMode(GL.GL_MODELVIEW);
        gl.glLoadIdentity();

        gl.glColor3d(0.0, 0.0, 0.0); // paredes:
        gl.glBegin(GL.GL_LINES);
        gl.glVertex2d(120.0, -220.0);
        gl.glVertex2d(120.0, -140.0);
        gl.glVertex2d(280.0, -220.0);
        gl.glVertex2d(280.0, -140.0);
        gl.glEnd();

        gl.glColor3d(0.0, 0.0, 1.0); // porta:
        gl.glBegin(GL.GL_LINE_STRIP);
        gl.glVertex2d(140.0, -220.0);
        gl.glVertex2d(140.0, -160.0);
        gl.glVertex2d(180.0, -160.0);
        gl.glVertex2d(180.0, -220.0);
        gl.glEnd();

        gl.glColor3d(1.0, 0.0, 1.0); // janela:
        gl.glBegin(GL.GL_LINE_LOOP);
        gl.glVertex2d(200.0, -160.0);
        gl.glVertex2d(260.0, -160.0);
        gl.glVertex2d(260.0, -200.0);
        gl.glVertex2d(200.0, -200.0);
        gl.glEnd();

        gl.glColor3d(1.0, 0.0, 0.0); // telhado:
        gl.glBegin(GL.GL_TRIANGLES);
        gl.glVertex2d(100.0, -140.0);
        gl.glVertex2d(200.0, -60.0);
        gl.glVertex2d(300.0, -140.0);
        gl.glEnd();
    }
}

```

OpenGL
com Java
= JOGL

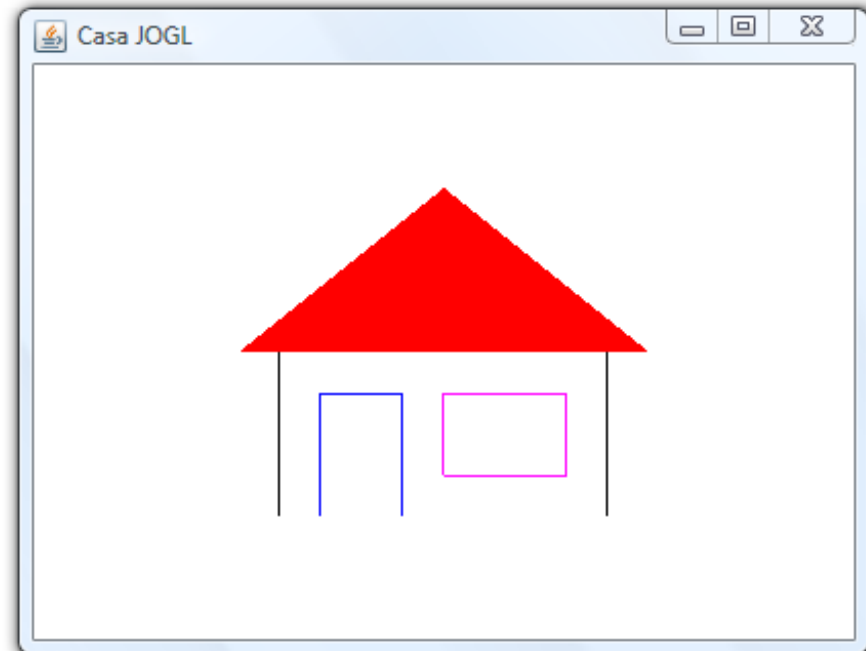
```

public void reshape(GLAutoDrawable gLDrawable, int x, int y,
                    int width, int height) {
    GL gl = gLDrawable.getGL();
    GLU glu = new GLU();
    gl.glViewport(0, 0, width, height);
    gl.glMatrixMode(GL.GL_PROJECTION);
    gl.glLoadIdentity();
    glu.gluOrtho2D(0.0, 400.0, -280.0, 0.0);
}

public void displayChanged(GLAutoDrawable gLDrawable,
                           boolean modeChanged, boolean deviceChanged) {}

public static void main(String[] args) {
    Frame frame = new Frame("Casa JOGL");
    GLCanvas canvas = new GLCanvas();
    canvas.addGLEventListener(new Casa());
    canvas.setSize(400, 280);
    frame.add(canvas);
    frame.pack();
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    frame.setVisible(true);
}
}

```



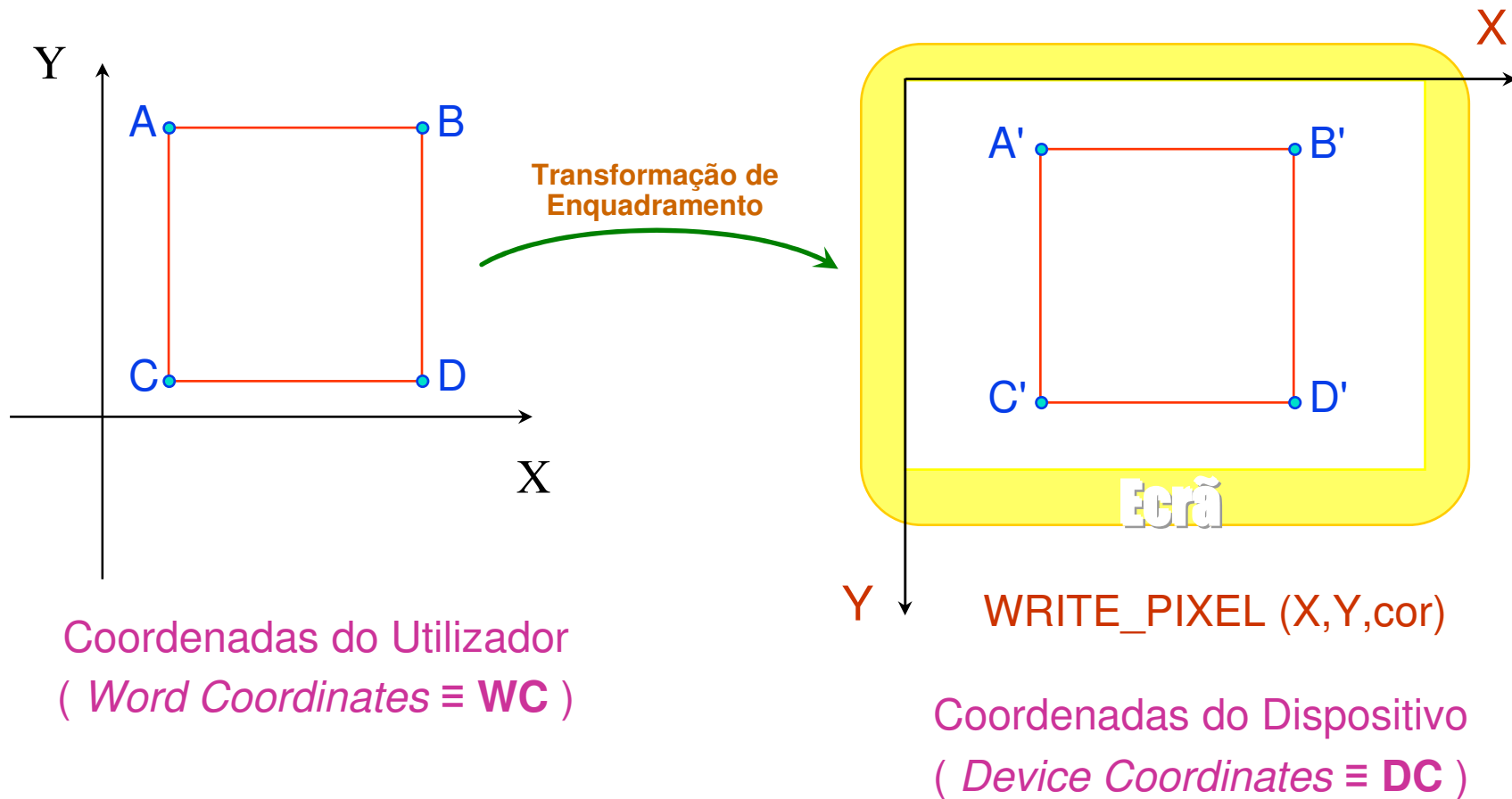
The OpenGL Syntax

<u>Prefix</u>	<u>Meaning</u>
gl	OpenGL command
GL_	OpenGL defined constant

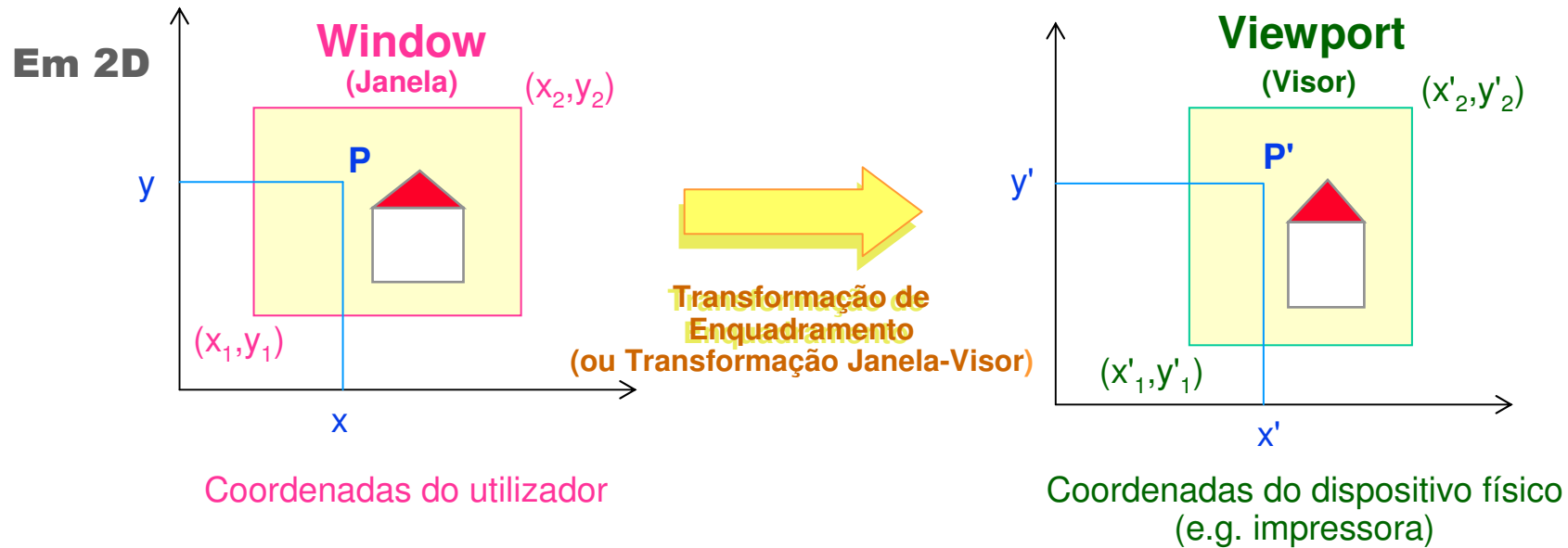
<u>Suffix</u>	<u>Meaning</u>
2...	Two arguments of type ...
3...	Three arguments of type ...
4...	Four arguments of type ...

<u>Suffix</u>	<u>Data Type</u>	<u>OpenGL Type</u>	<u>C-language Type</u>
b	8-bit integer	GLbyte	signed char
s	16-bit integer	GLshort	short
i	32-bit integer	GLint, GLsizei	long
f	32-bit floating-point	GLfloat, GLclampf	float
d	64-bit floating-point	GLdouble, GLclampd	double
ub	8-bit unsigned integer	GLubyte, GLboolean	unsigned char
us	16-bit unsigned integer	GLushort	unsigned short
ui	32-bit unsigned integer	GLuint, GLenum, GLbitfield	unsigned long
v	pointer to a vector		

Enquadramento



Enquadramento



$$\frac{x_2 - x_1}{x - x_1} = \frac{x'_2 - x'_1}{x' - x'_1}$$



$$x' = (x - x_1) (x'_2 - x'_1) / (x_2 - x_1) + x'_1$$

$$x' = (x - A) \cdot B + C$$

$$\frac{y_2 - y_1}{y - y_1} = \frac{y'_2 - y'_1}{y' - y'_1}$$



$$y' = (y - y_1) (y'_2 - y'_1) / (y_2 - y_1) + y'_1$$

$$y' = (y - D) \cdot E + F$$

Formato ou **Relação de Aspecto** (*Aspect Ratio*)
de um retângulo:

$$a = \frac{x_{\max} - x_{\min}}{y_{\max} - y_{\min}}$$

Exemplos bem conhecidos

4:3
16:9

Condição para manter as proporções da imagem no enquadramento:

$$a_{\text{janela}} = a_{\text{visor}}$$



$$\frac{x_2 - x_1}{y_2 - y_1} = \frac{x'_2 - x'_1}{y'_2 - y'_1}$$



$$E = B$$

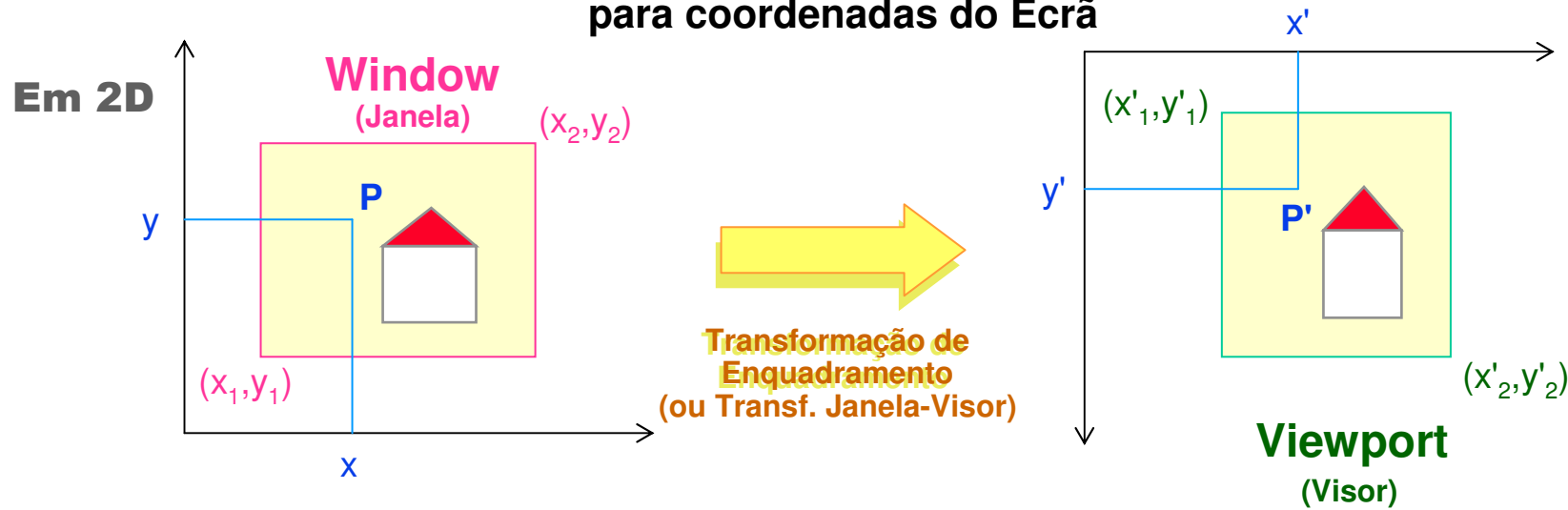
Caso contrário



distorção da imagem

Enquadramento

para coordenadas do Ecrã



$$\frac{x_2 - x_1}{x - x_1} = \frac{x'_2 - x'_1}{x' - x'_1}$$



$$x' = (x - x_1) (x'_2 - x'_1) / (x_2 - x_1) + x'_1$$

$$x' = (x - A) \cdot B + C$$

$$\frac{y_2 - y_1}{y - y_1} = \frac{y'_2 - y'_1}{y'_2 - y'}$$



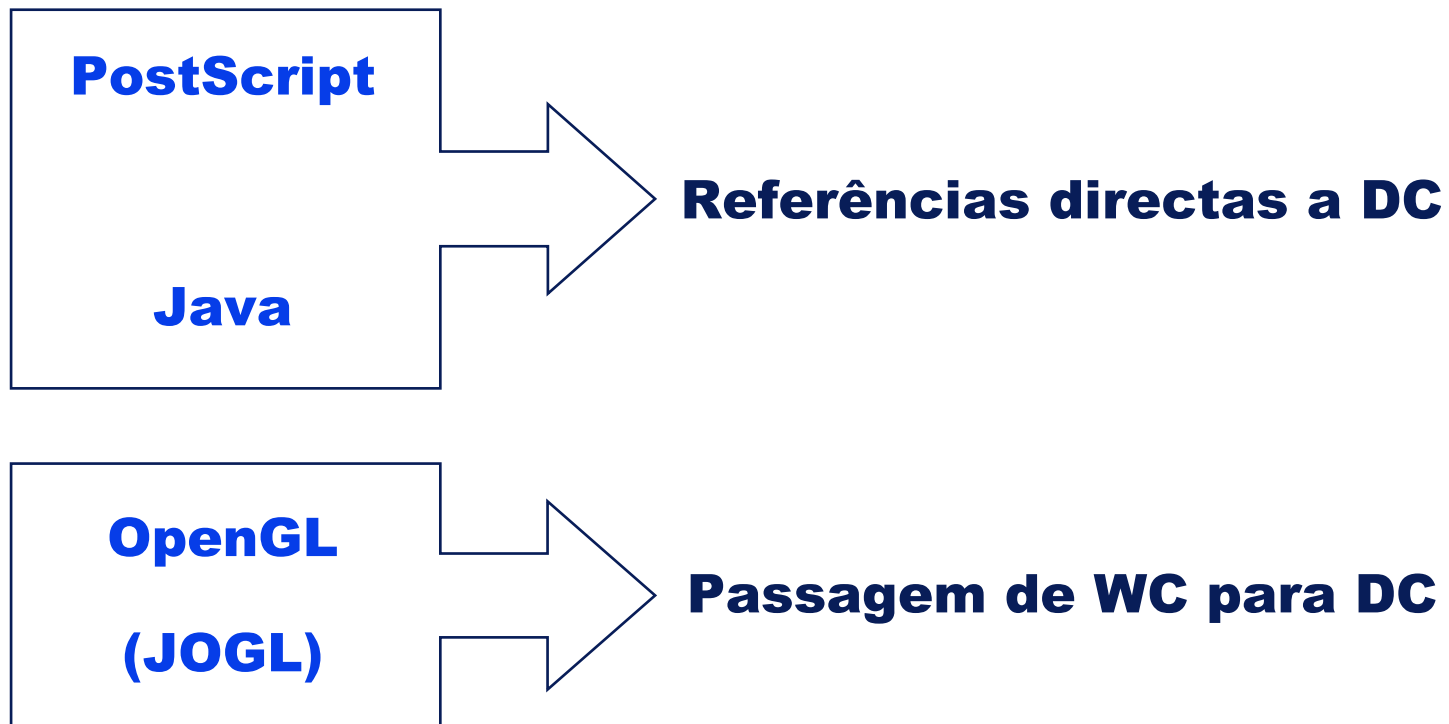
$$y' = - (y - y_1) (y'_2 - y'_1) / (y_2 - y_1) + y'_2$$

$$y' = (y - D) \cdot G + H$$

$$G = -E$$

Enquadramento

Ver os exemplos apresentados anteriormente nas aulas:



JANELA:

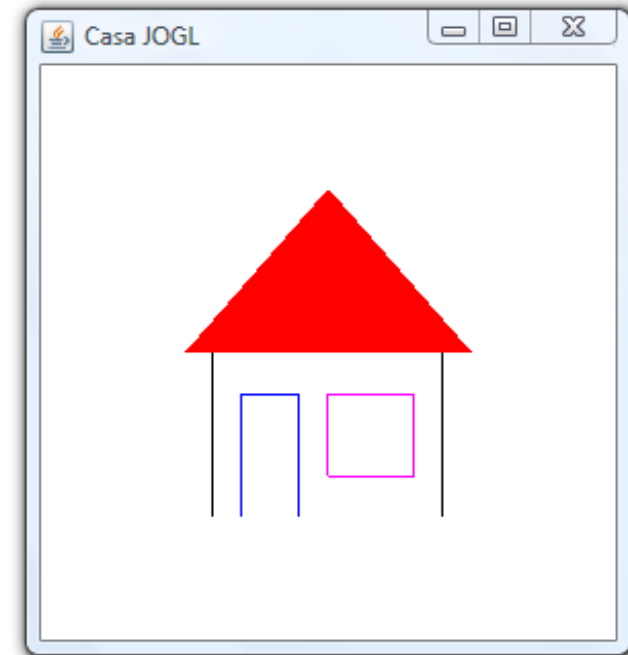
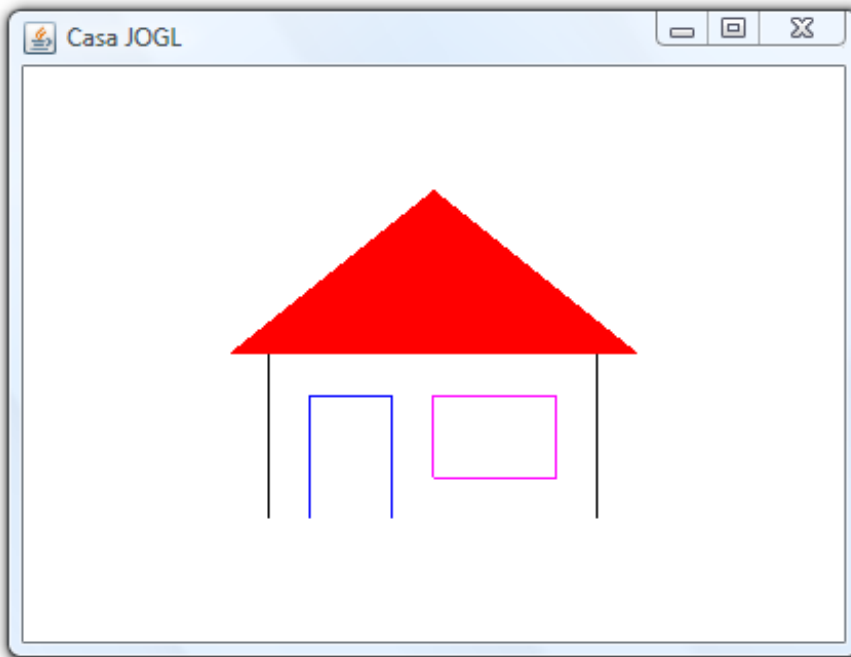
```
gluOrtho2D (0.0, 400.0, -280.0, 0.0);
```

Em WC

VISOR:

```
glViewport (0, 0, 400, 280);
```

Não exatamente
em DC



janela(0, 400, -280, 0)

visor(0, 280, 0, 280) $\rightarrow a_v = 1:1$

Em
WC

janela(0, 400, -280, 0)

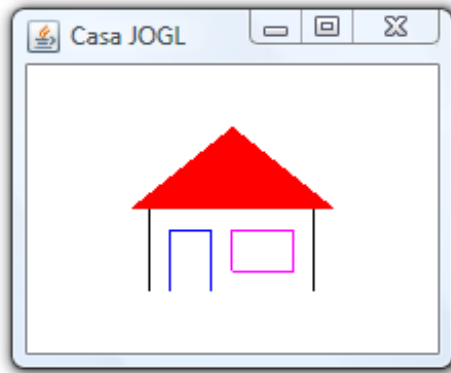
Em
DC

visor(0, 400, 0, 280)

$a = 10:7$

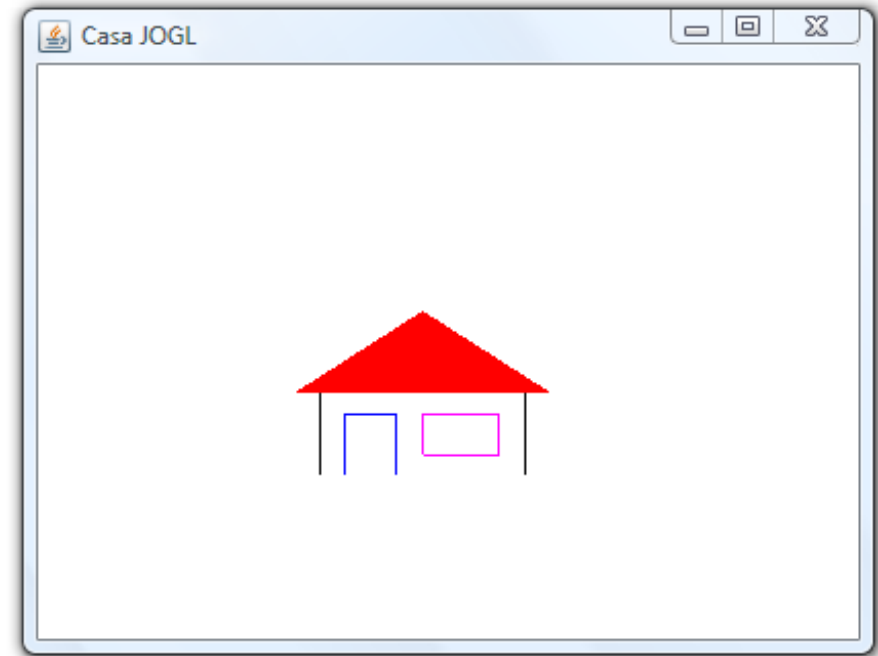
$a_j \neq a_v$

distorção da imagem



`janela(0, 400, -280, 0)`
`visor(0, 200, 0, 140)`

} $a = 10:7$



`janela(-100, 540, -380, 180)` → $a_j = 8:7$
`visor(0, 400, 0, 280)`

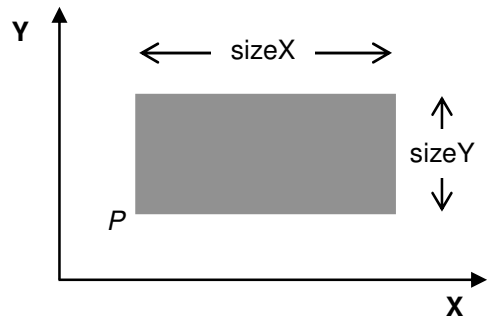


distorção da imagem

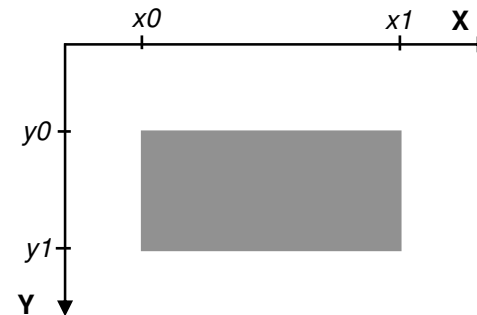
ESPECIFICAÇÃO DO VISOR NA ÁREA DE DESENHO (1)

```
void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);
```

Defines a pixel rectangle in the window into which the final image is mapped. The (x, y) parameter specifies the lower-left corner of the viewport, and $width$ and $height$ are the size of the viewport rectangle. By default, the initial viewport values are $(0, 0, winWidth, winHeight)$, where $winWidth$ and $winHeight$ are the size of the window.



`glViewport (Px, Py, sizeX, sizeY)`

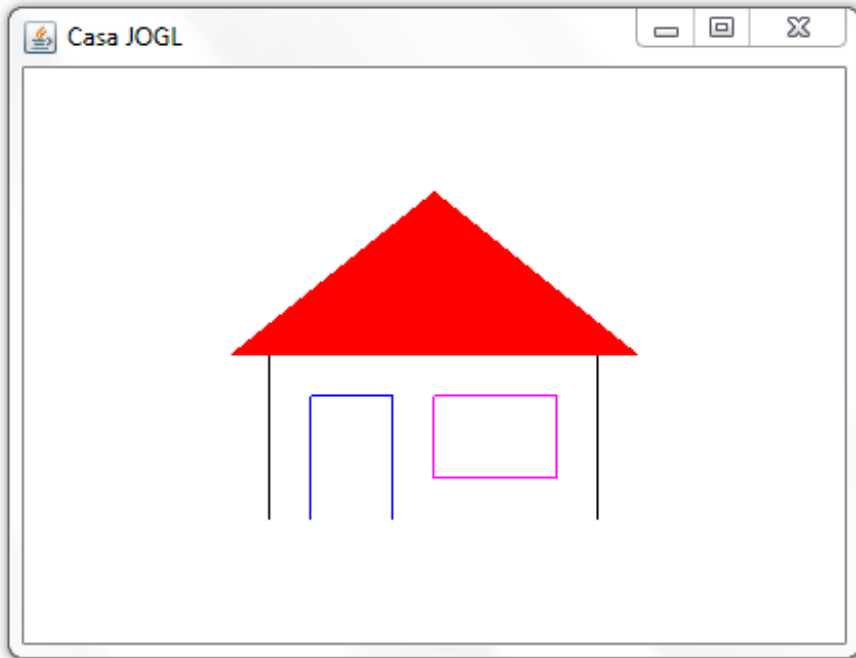


`visor (x0, x1, y0, y1)`

DC

ESPECIFICAÇÃO DO VISOR NA ÁREA DE DESENHO (2)

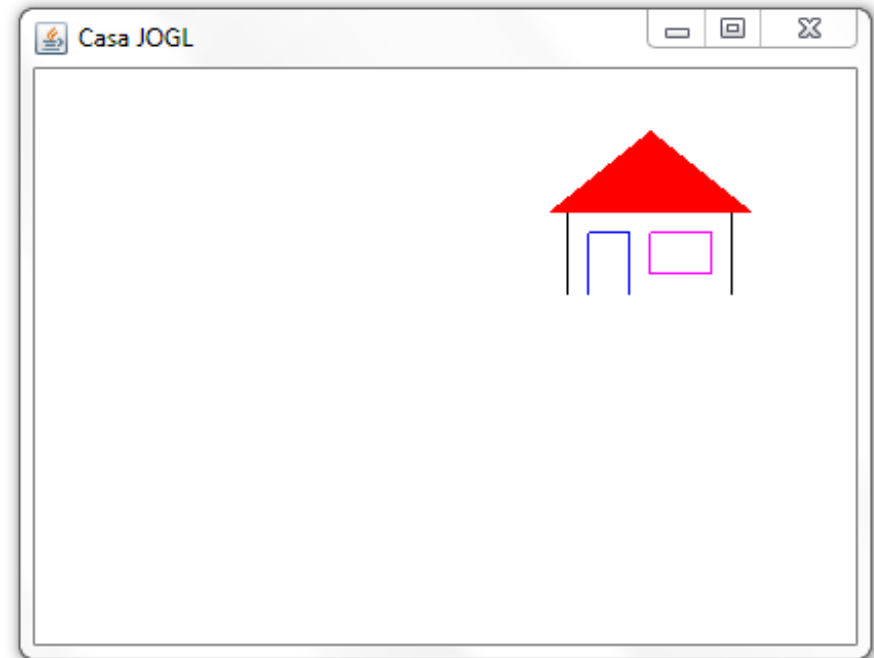
EXEMPLOS



`glViewport (0, 0, 400, 280)`

OU

`visor(0, 400, 0, 280)`



`glViewport (200, 140, 200, 140)`

OU

`visor(200, 400, 0, 140)`

Cálculo matemático por transformação linear?

Será possível que a transformação de enquadramento possa ser calculada por uma matriz M que, aplicada a um ponto P da janela (WC) o converta num ponto P' do visor (DC)?

Se se escrever P na forma do seguinte vetor

$$P = \begin{bmatrix} x \\ y \end{bmatrix}$$

então a operação de transformação teria de ser

$$P' = M.P$$

Mas não existe nenhuma matriz 2×2 que satisfaça a operação!

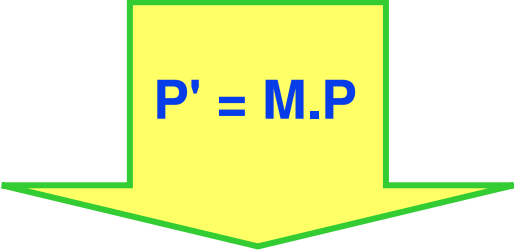
A solução está no uso de **coordenadas homogéneas**, passando M a ser do tipo 3×3 .

Enquadramento janela-visor como transformação linear

$$x' = (x - A) \cdot B + C = x \cdot B - A \cdot B + C$$

$$y' = (y - D) \cdot J + K = y \cdot J - D \cdot J + K$$

$$J = E \text{ ou } G \quad K = F \text{ ou } H$$

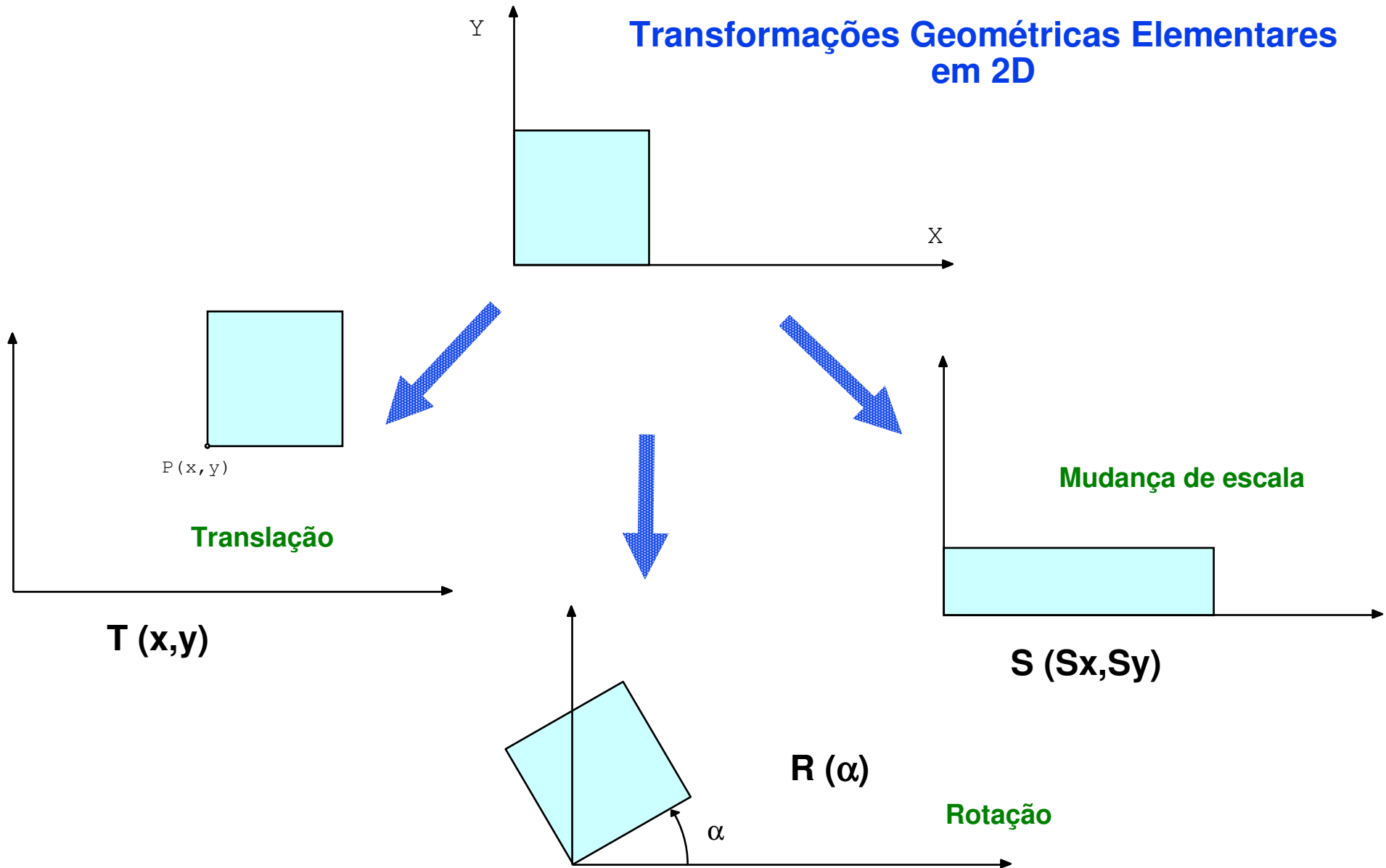

$$P' = M \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} B & 0 & C-A \cdot B \\ 0 & J & K-D \cdot J \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Pontos **P** e **P'** necessariamente em coordenadas homogêneas!

Obs.: Mais tarde se apresentará esta matriz como composição de transformações geométricas.

Transformações Geométricas Elementares em 2D



Tratamento matemático (1)

$$P(x, y) = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \leftarrow \text{em Coordenadas Homogêneas 2D}$$

$$P'(x', y') = M \cdot P(x, y)$$

Translação

$$T(T_x, T_y) = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$x' = x + T_x$$

$$y' = y + T_y$$

Mudança de escala

$$S(S_x, S_y) = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Fatores de escala

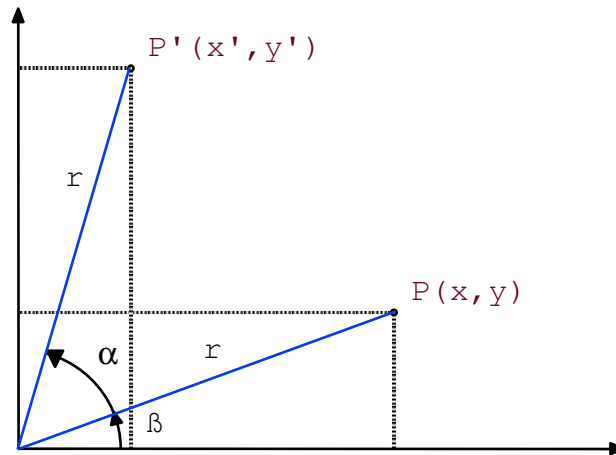
$$x' = S_x * x$$

$$y' = S_y * y$$

Tratamento matemático (2)

Rotação

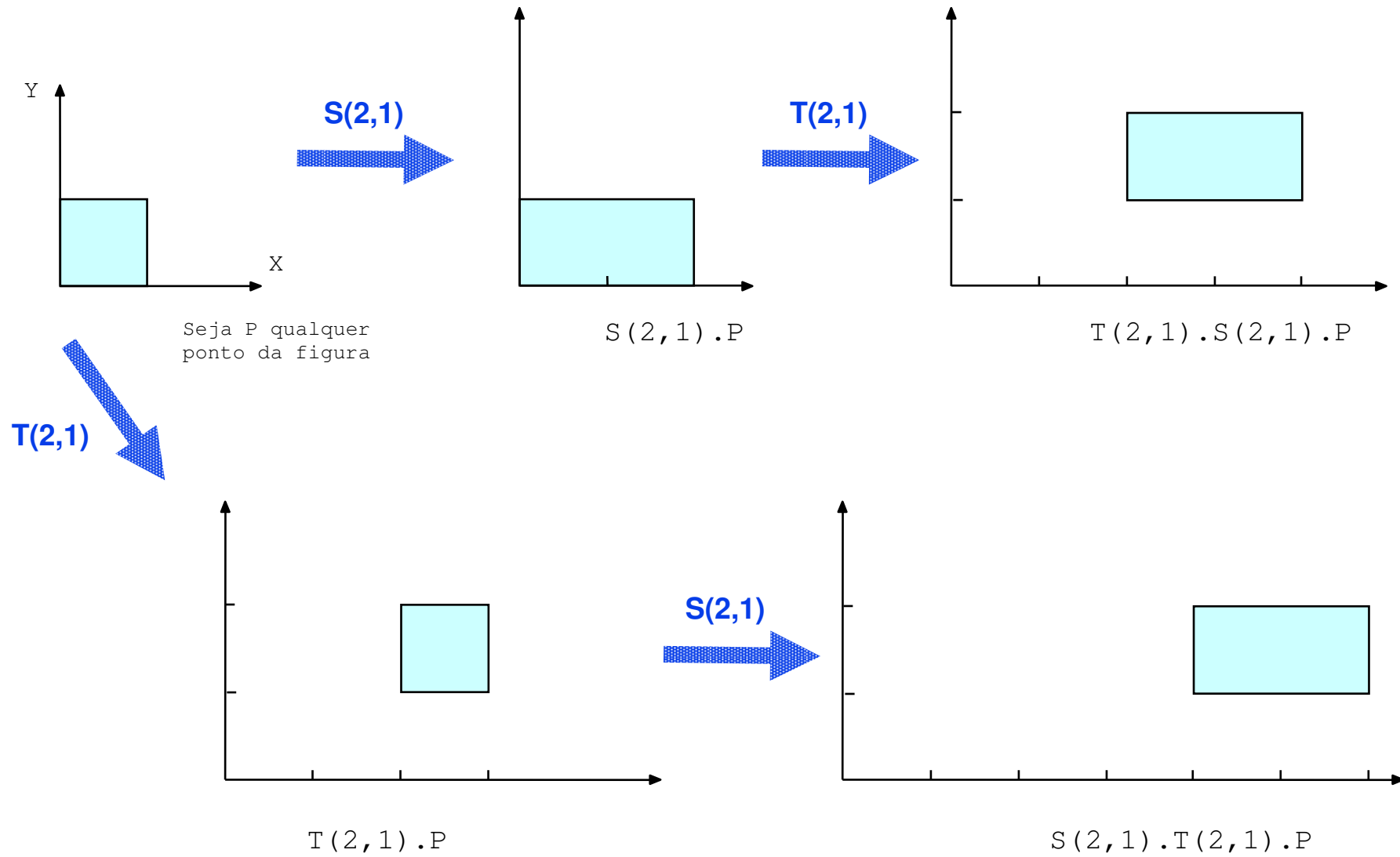
$$R(\alpha) = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



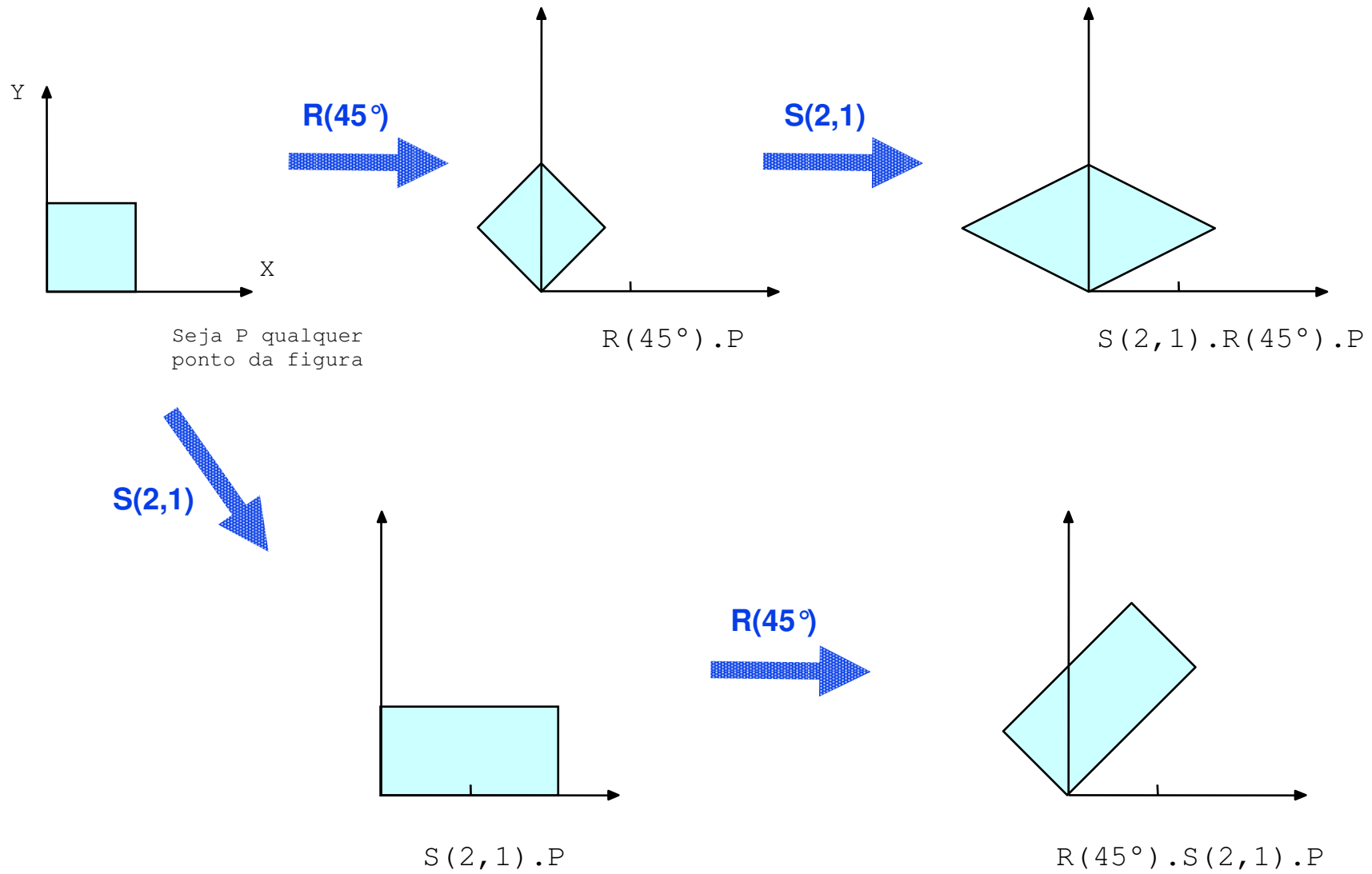
$$\begin{aligned} x' &= r \cdot \cos(\beta + \alpha) \\ &= r \cdot \cos \beta \cdot \cos \alpha - r \cdot \sin \beta \cdot \sin \alpha \\ &= x \cdot \cos \alpha - y \cdot \sin \alpha \end{aligned}$$

$$\begin{aligned} y' &= r \cdot \sin(\beta + \alpha) \\ &= r \cdot \cos \beta \cdot \sin \alpha + r \cdot \sin \beta \cdot \cos \alpha \\ &= x \cdot \sin \alpha + y \cdot \cos \alpha \end{aligned}$$

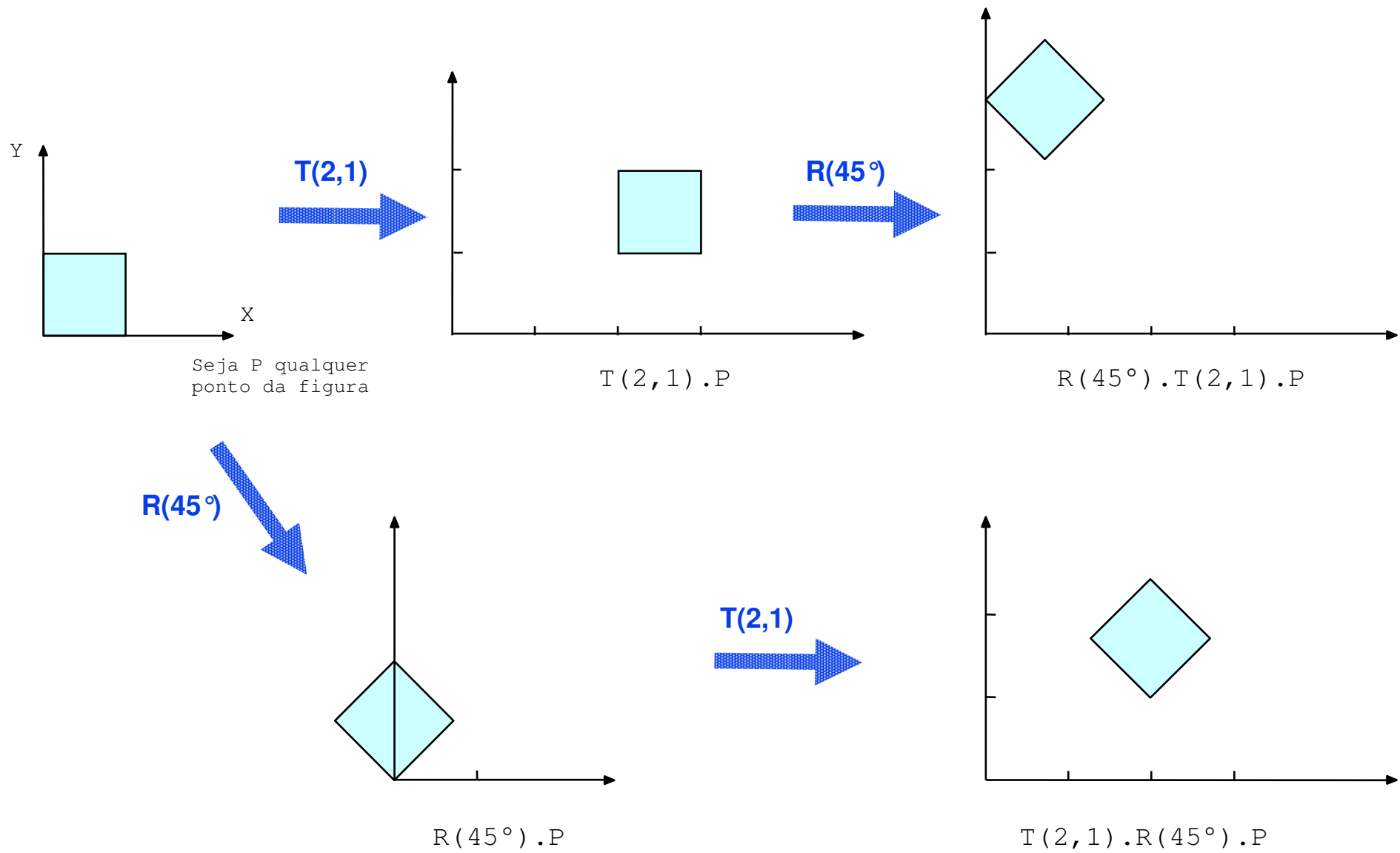
Composição de Transformações Geométricas: S e T



Composição de Transformações Geométricas: R e S



Composição de Transformações Geométricas: T e R



Quando é que se pode garantir a comutatividade?

$R(\infty).R(\beta)$

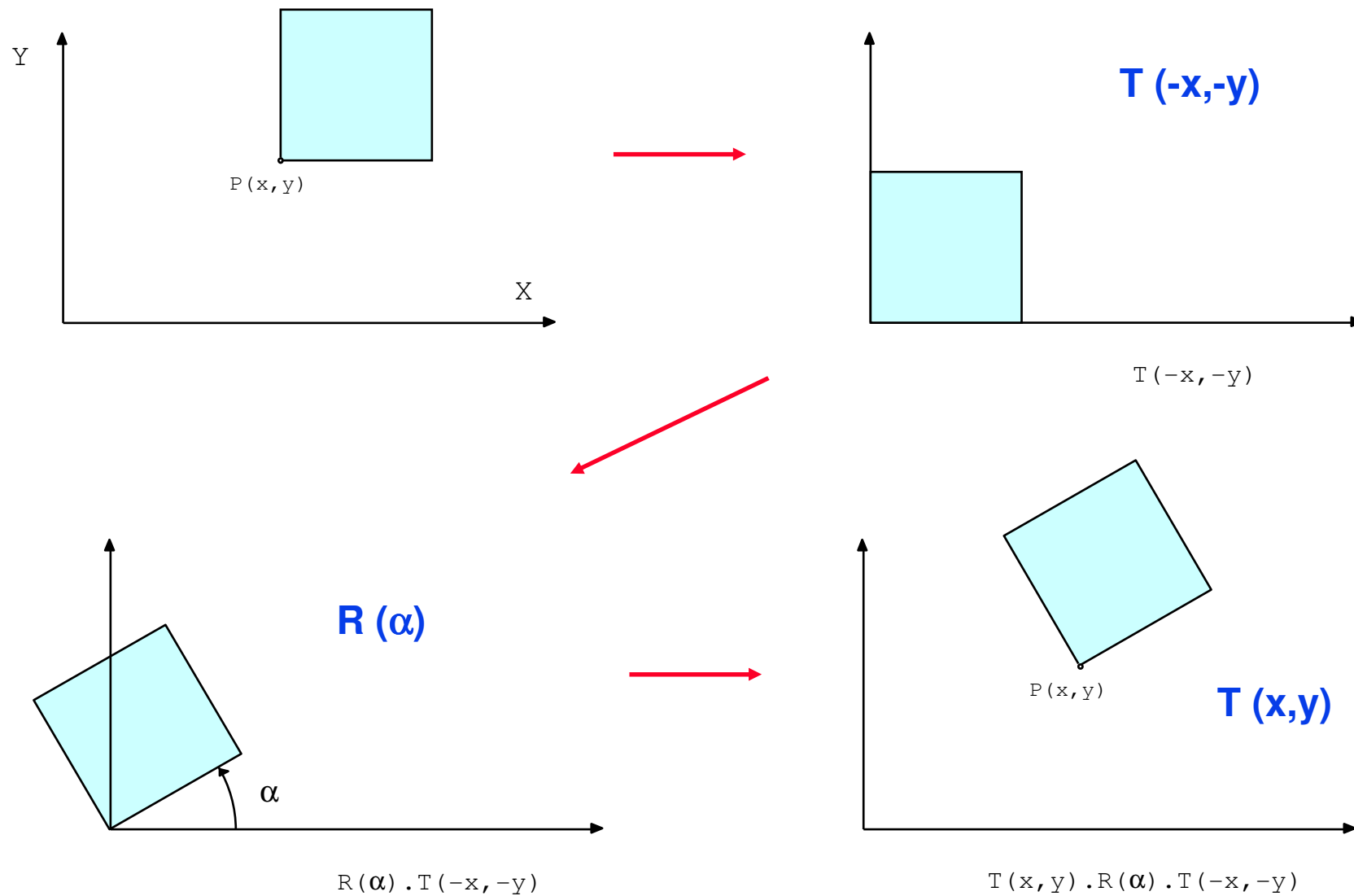
$S(K1,K2).S(K3,K4)$

$T(D1,D2).T(D3,D4)$

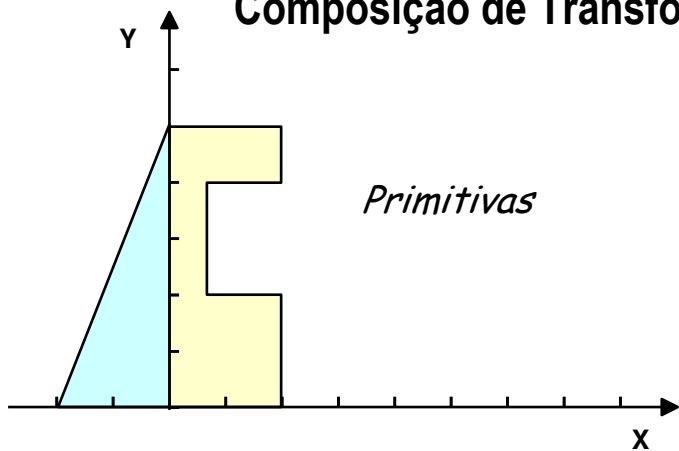
$S(K,K).R(\infty)$



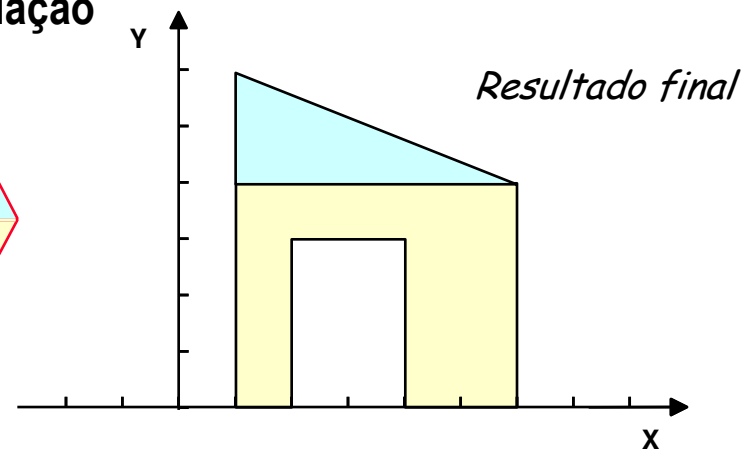
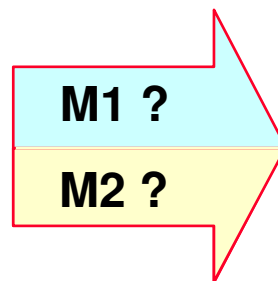
Composição de Transformações: Rotação em torno de um ponto arbitrário $P(x,y)$



Composição de Transformações em Modelação



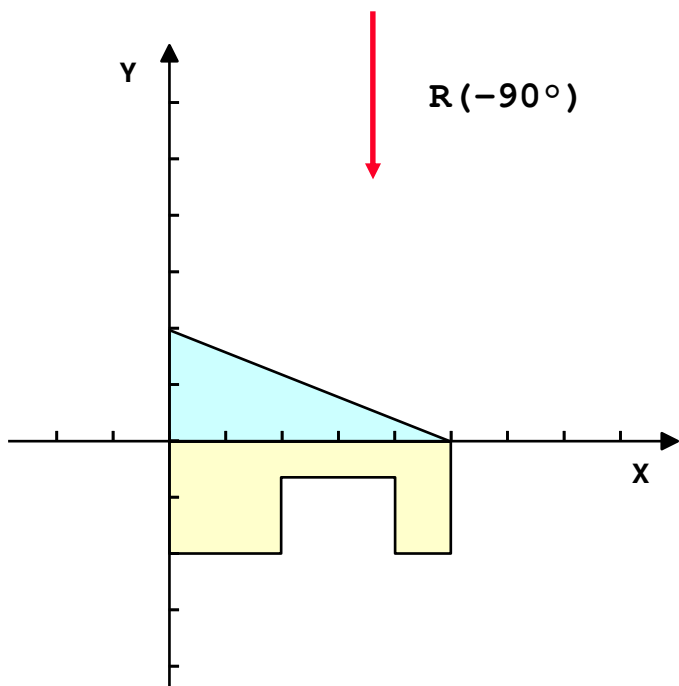
Primitivas



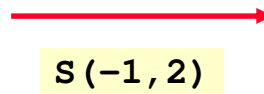
Resultado final

$$M1 = T(1,4).R(-90^\circ)$$

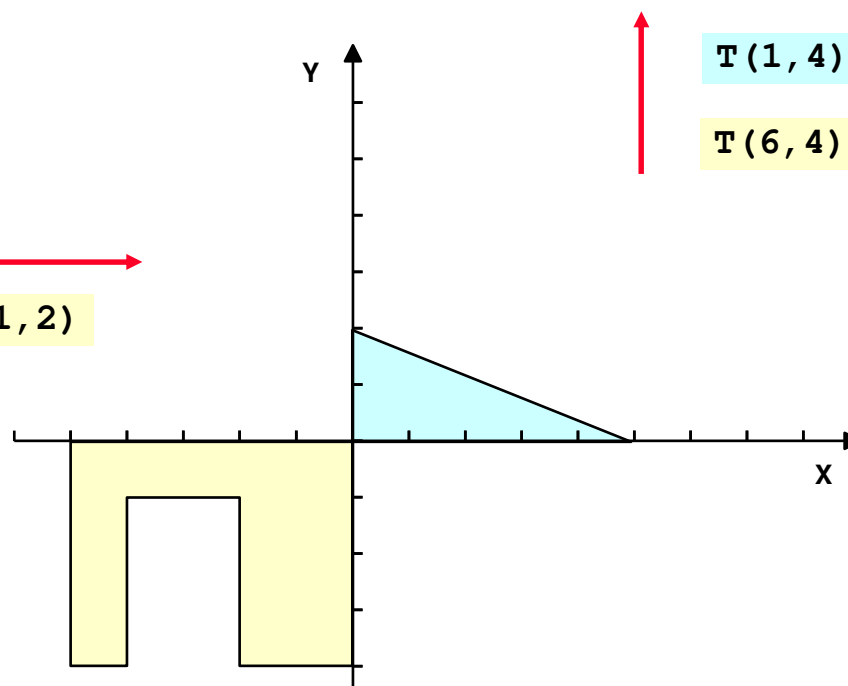
$$M2 = T(6,4).S(-1,2).R(-90^\circ)$$



$R(-90^\circ)$



$S(-1, 2)$



$T(1, 4)$

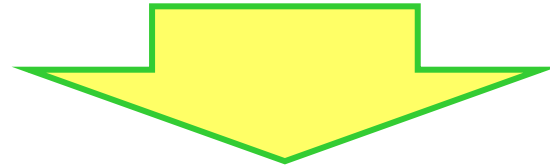
$T(6, 4)$

Aplicação da composição de transformações geométricas no enquadramento janela-visor

$$x' = (x - A) \cdot B + C$$

$$y' = (y - D) \cdot J + K$$

$$J = E \text{ ou } G, K = F \text{ ou } H$$

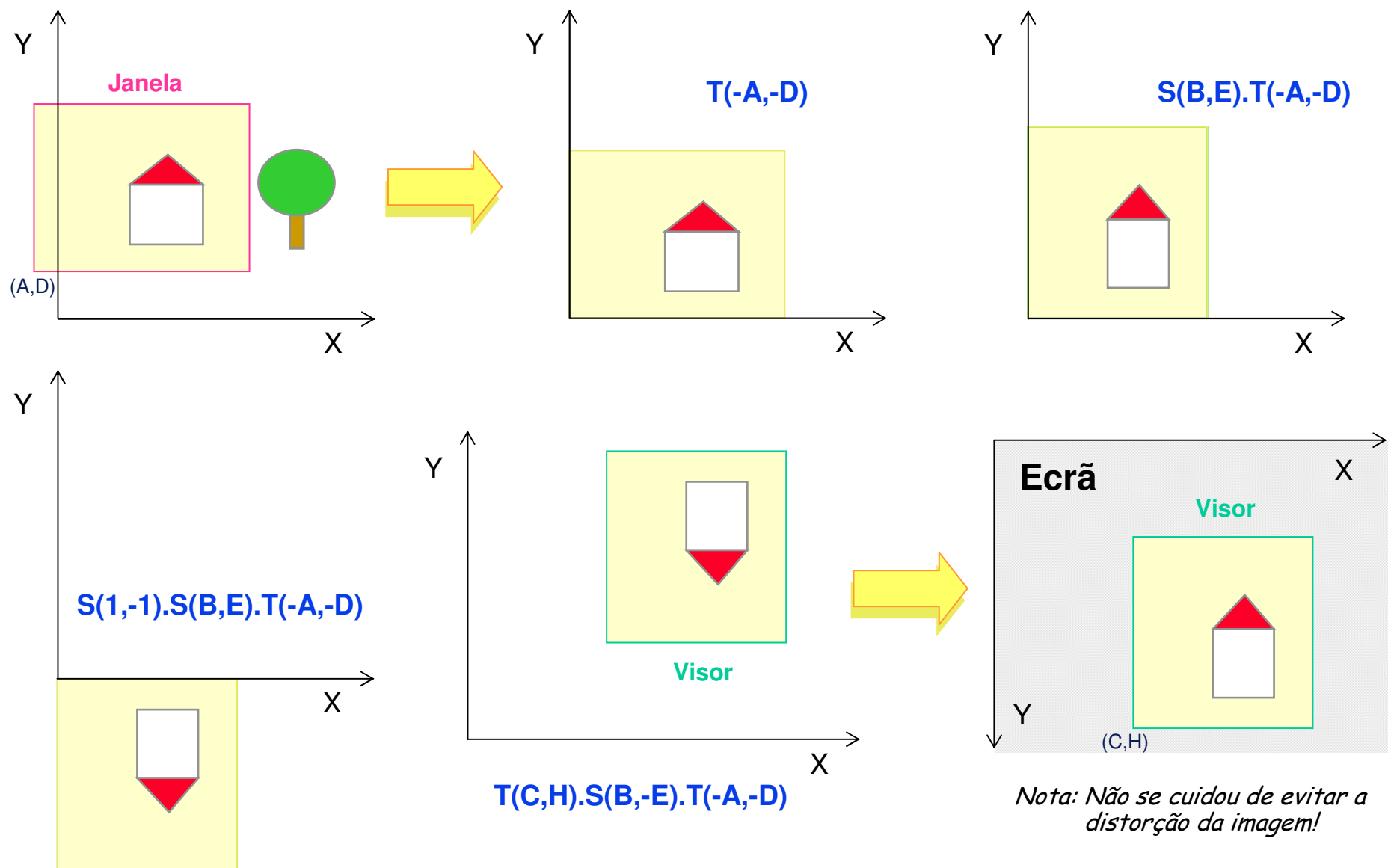


$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & C \\ 0 & 1 & K \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} B & 0 & 0 \\ 0 & J & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -A \\ 0 & 1 & -D \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

ou, numa forma mais compacta:

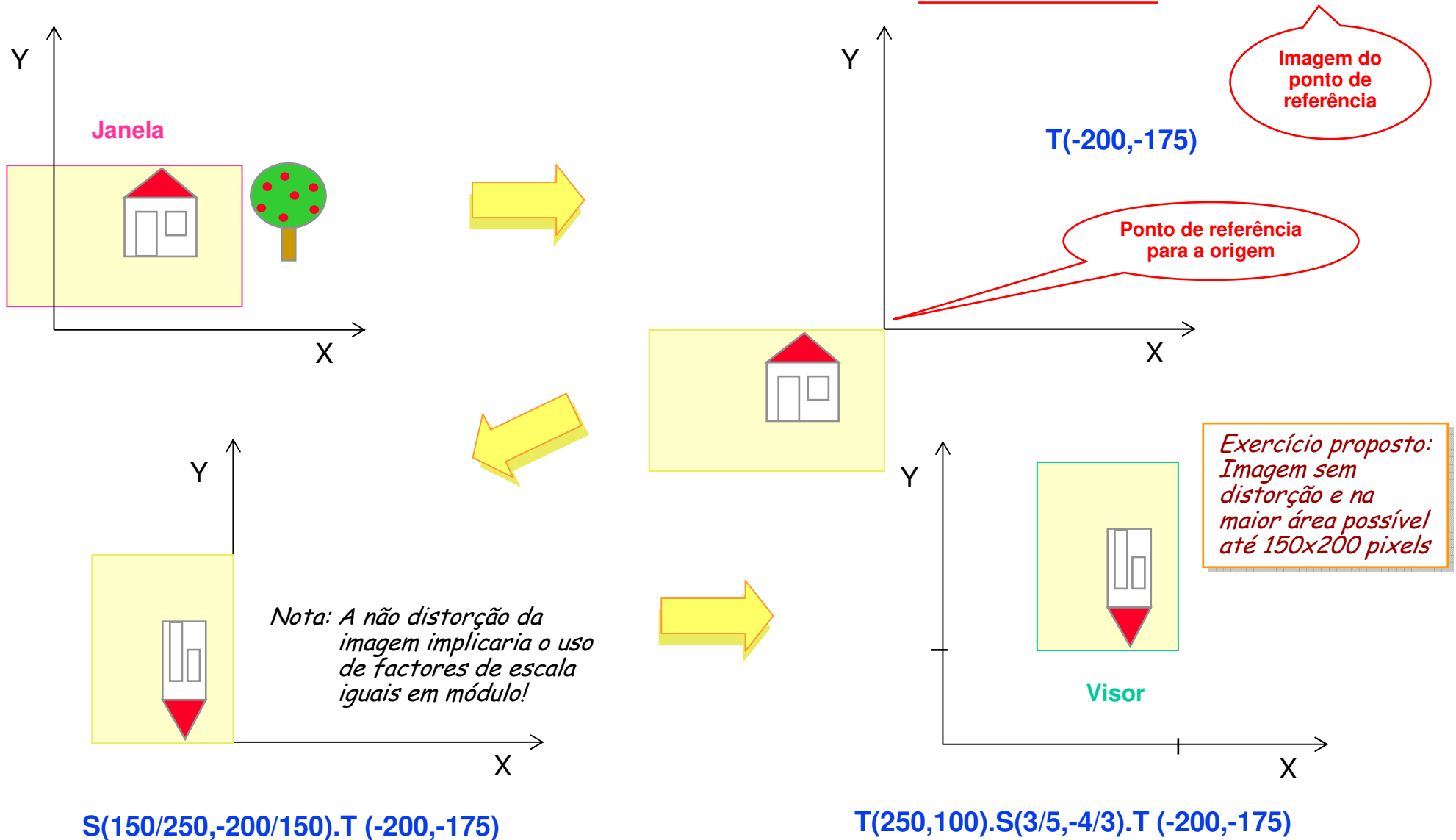
$$P' = T(C, K) \cdot S(B, J) \cdot T(-A, -D) \cdot P$$

Aplicação na dedução da transformação de enquadramento (coordenadas do ecrã)



Exemplo numérico (à escala) para ecrã:

Transformar a Janela $-50 \leq x \leq 200 \wedge 25 \leq y \leq 175$ num Visor de 150x200 pixels e canto superior direito no ponto (250,100)



Transformações Geométricas em



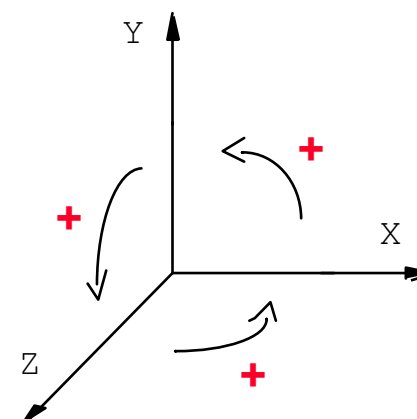
T (x,y,z)

S (Sx,Sy,Sz)

R_z(α)

R_x(α)

R_y(α)



Transformações inversas:

$$T(x,y,z) \longrightarrow T(-x,-y,-z)$$

$$S(Sx,Sy,Sz) \longrightarrow S(1/Sx,1/Sy,1/Sz)$$

$$R(\alpha) \longrightarrow R(-\alpha)$$

Transformações Geométricas em



$$T(T_x, T_y, T_z) = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S(S_x, S_y, S_z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

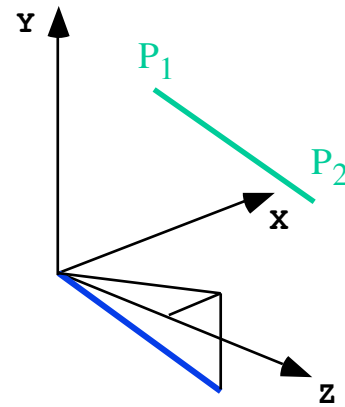
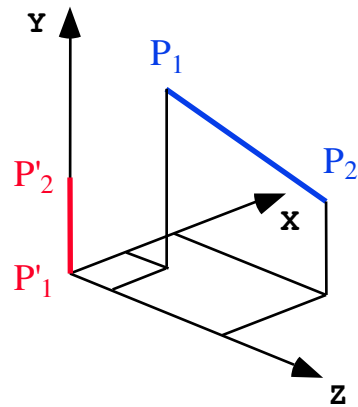
$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

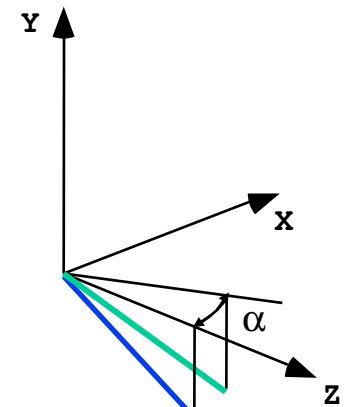
$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Exercício proposto: Com base no caso 2D, analise a comutatividade em 3D.

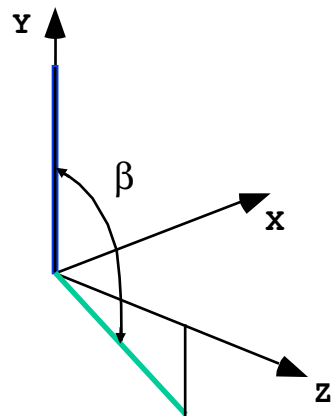
EXEMPLO de APLICAÇÃO 3D, por composição de transformações elementares: Transformar P_1P_2 em $P'_1P'_2$



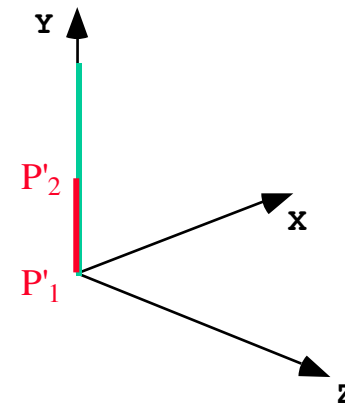
$$T(T_x, T_y, T_z)$$



$$R_y(-\alpha) \cdot T(T_x, T_y, T_z)$$



$$R_x(-\beta) \cdot R_y(-\alpha) \cdot T(T_x, T_y, T_z)$$



$$S(1, S_y, 1) \cdot R_x(-\beta) \cdot R_y(-\alpha) \cdot T(T_x, T_y, T_z)$$

OpenGL pipeline

Vertex Transformation

Exemplos em C

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glRotated(45, 0.0, 1.0, 0.0);  
glScaled(2.0, 1.0, 2.0);  
  
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
glOrtho(x1, x2, y1, y2, z1, z2);  
glMultMatrixd(matrixProj);
```

$R_Y(45^\circ).S(2,1,2)$

Multiplicação à direita

Matriz 4x4 armazenada, por colunas, num vector

A ordem de escrita das matrizes MODELVIEW e PROJECTION num programa não altera a ordem de execução no *pipeline* de OpenGL

Vertex $\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$

Modelview Matrix

Eye Coordinates

Projection Matrix

Clip Coordinates

Perspective Division

Normalized Device Coordinates

Viewport Transformation

Window Coordinates

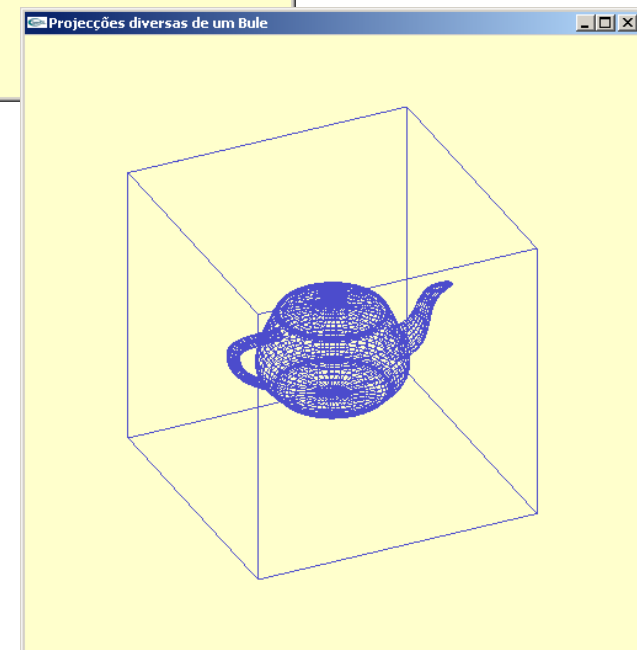
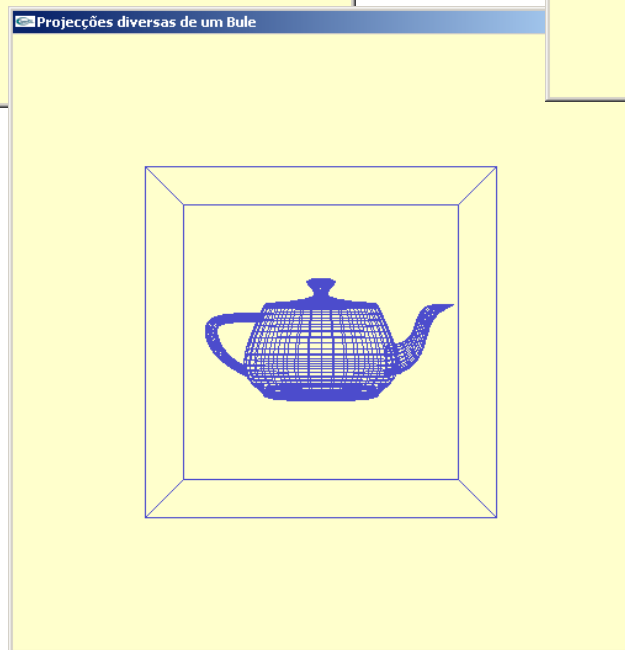
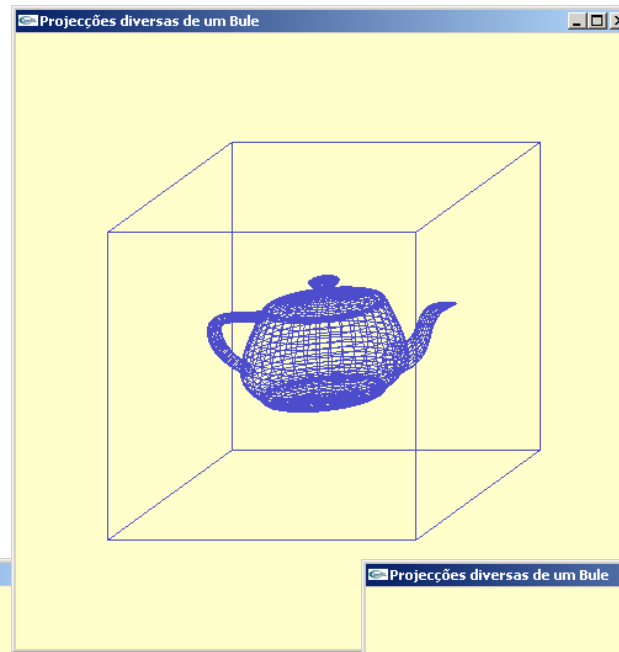
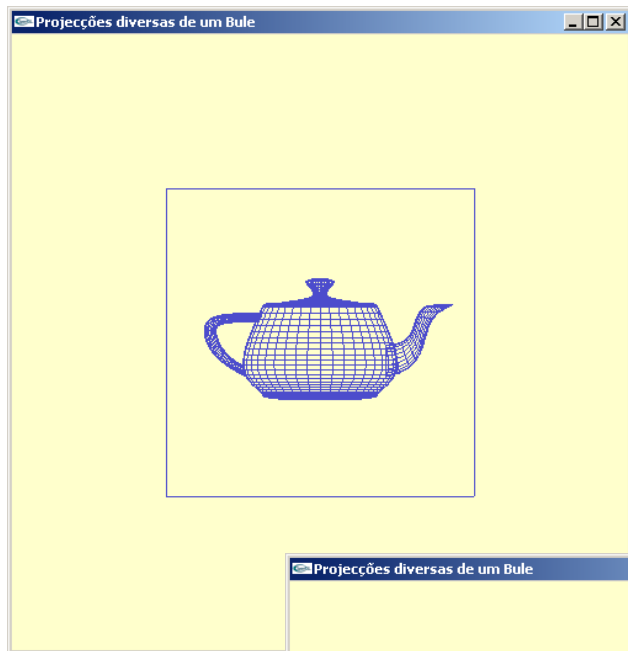
DC

Projeções Geométricas planas

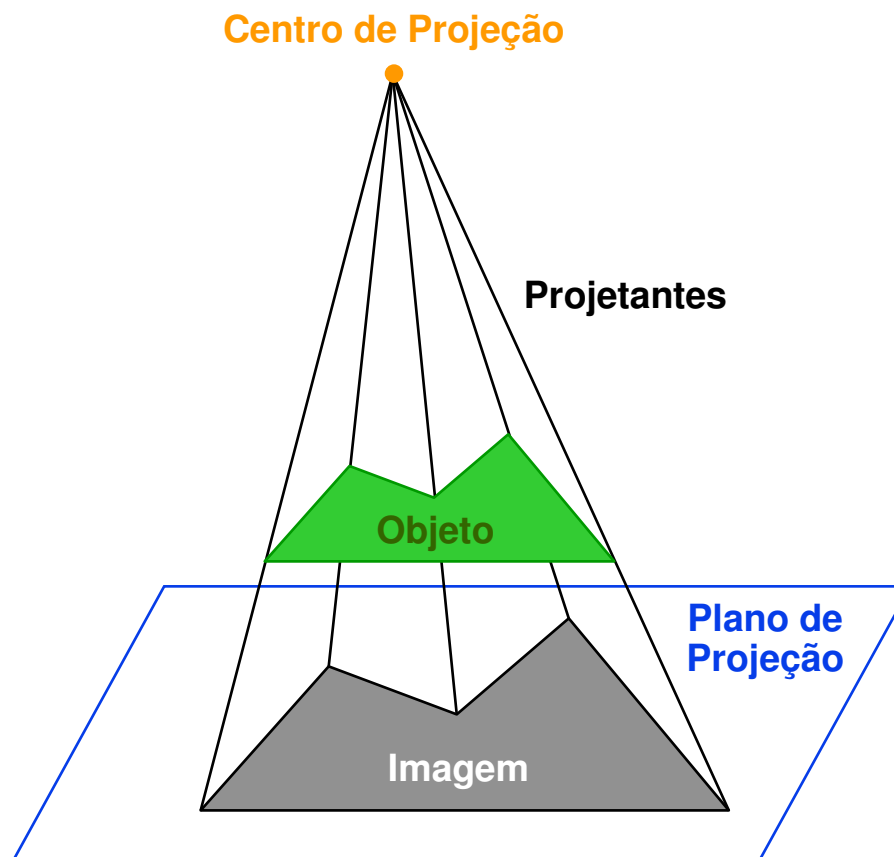
A superfície de projeção é um plano.

As projetantes são linhas retas.

A (imagem da) projeção de um ponto é a intersecção da projetante com o plano.

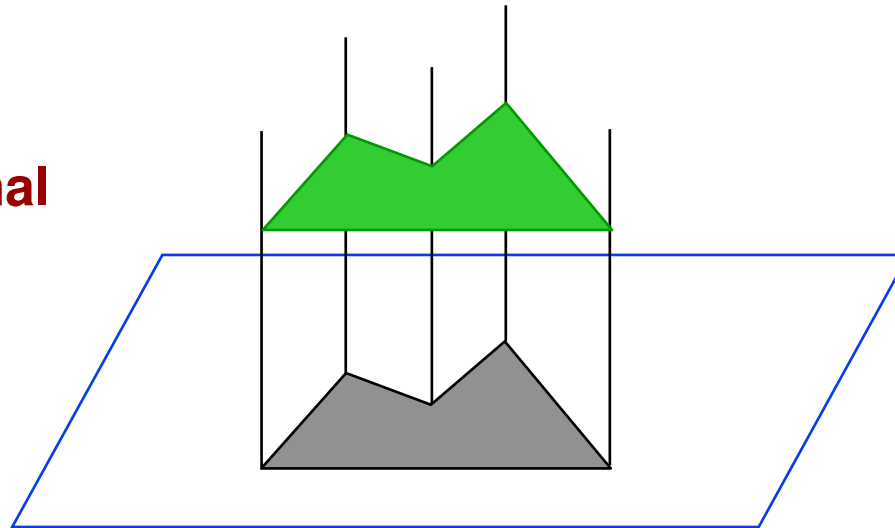


Projeção Perspectiva

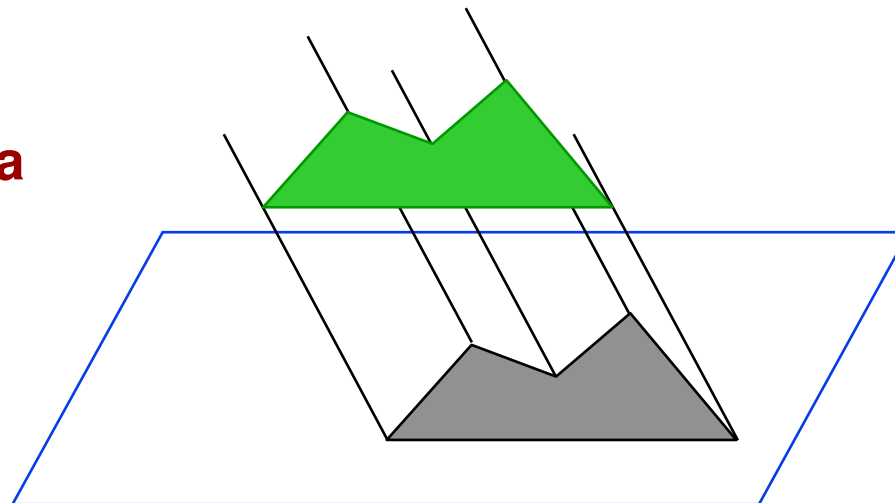


Projeção Paralela

Ortogonal



Oblíqua



UMA CLASSIFICAÇÃO DAS PROJEÇÕES NO DESENHO TÉCNICO

Central ou Cônica (Perspectiva)

Paralela

Oblíqua

Ortogonal

Simples

Axonométrica

Isométrica

Dimétrica

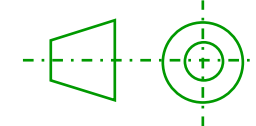
Trimétrica

Cotada

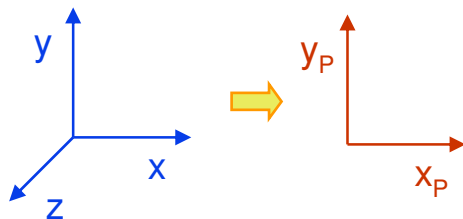
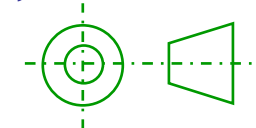
Dupla

Múltipla

Método Europeu, do 1.º Diedro ou Método E



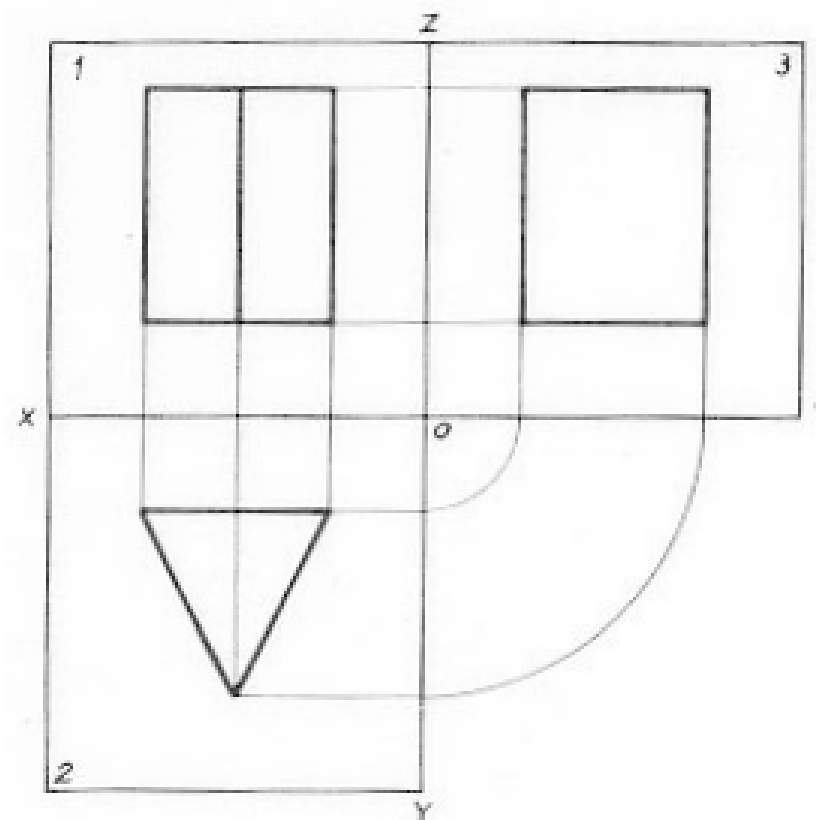
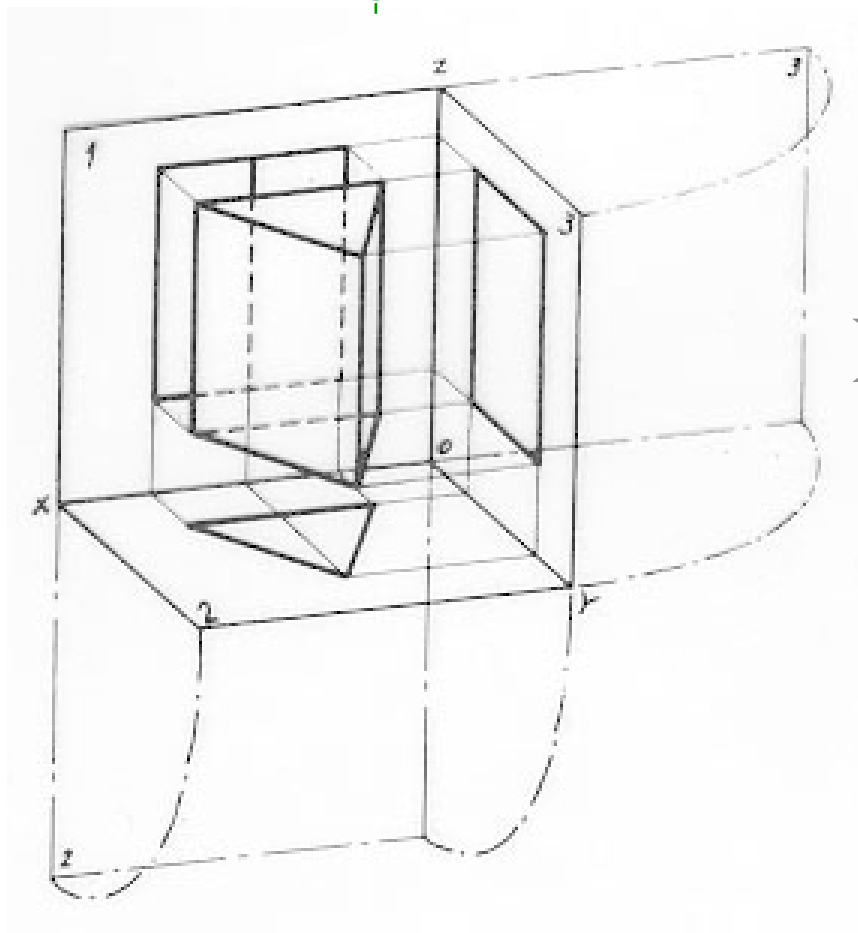
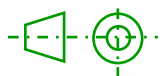
Método Americano, do 3.º Diedro ou Método A



$$\mathbf{P}' = \mathbf{P}_P = \mathbf{M}_{\text{projecção}} \cdot \mathbf{P}$$

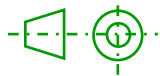
PROJEÇÕES ORTOGONAIS MÚLTIPLAS

(Método Europeu)



PROJEÇÕES ORTOGONAIS MÚLTIPLAS

(Método Europeu)



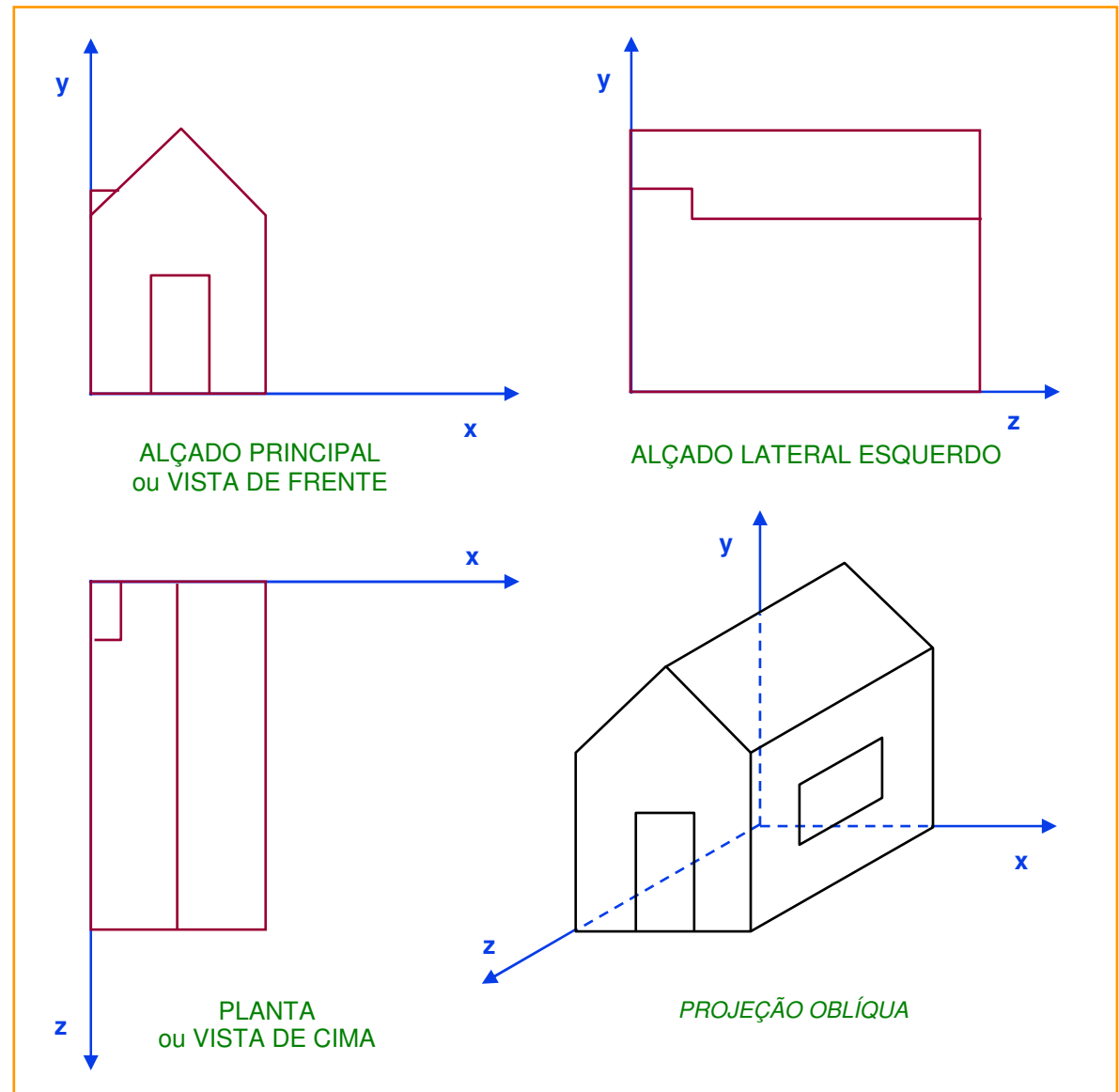
As diferenças entre os métodos A e E baseiam-se nas relações entre:

- (1) Observador
- (2) Objeto
- (3) Plano de projeção.

Método A: (3) entre (1) e (2)

Método E: (2) entre (1) e (3)

É usual aproveitar-se o quadrante livre para uma representação do objeto noutro tipo de projeção.



Matemática da PROJEÇÃO ORTOGONAL

$$P' = M_{ORT} \cdot P$$

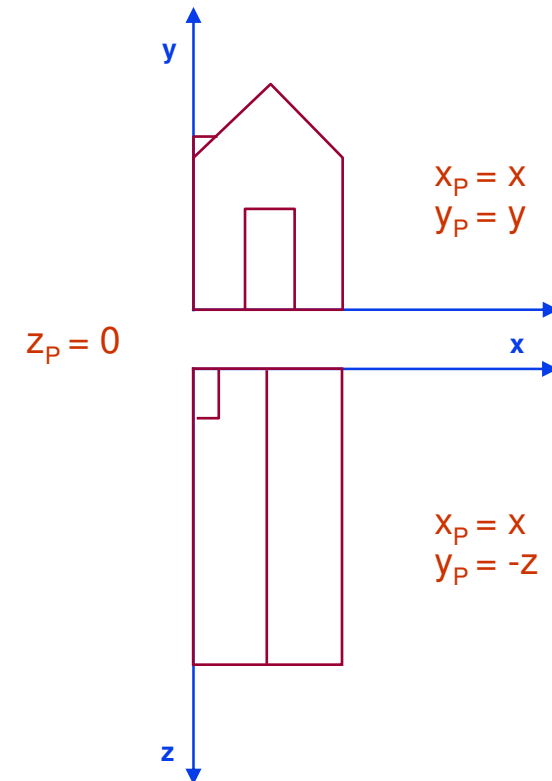
Alçado Principal:

$$M_{ORT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Planta:

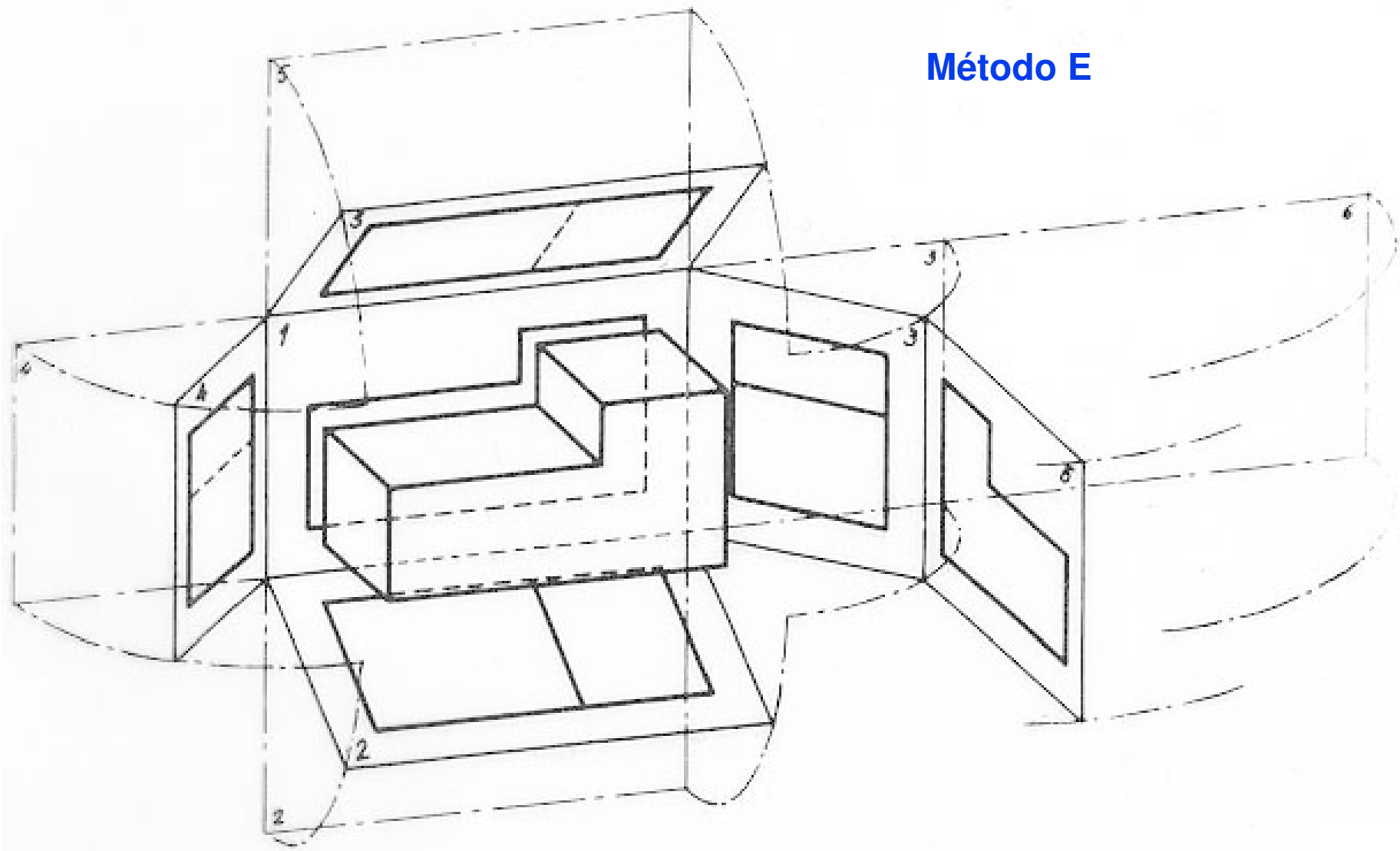
$$M_{ORT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

etc.



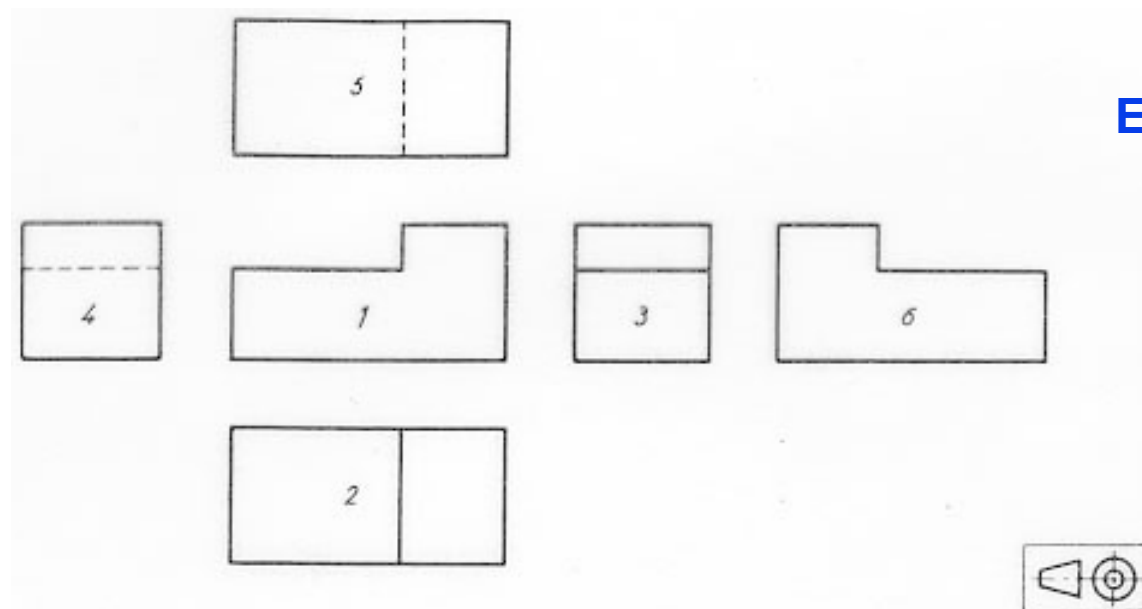
- + Mostra as dimensões exatas das faces paralelas ao plano de projeção.
- Pode ser difícil avaliar a forma tridimensional do objecto.

Método E

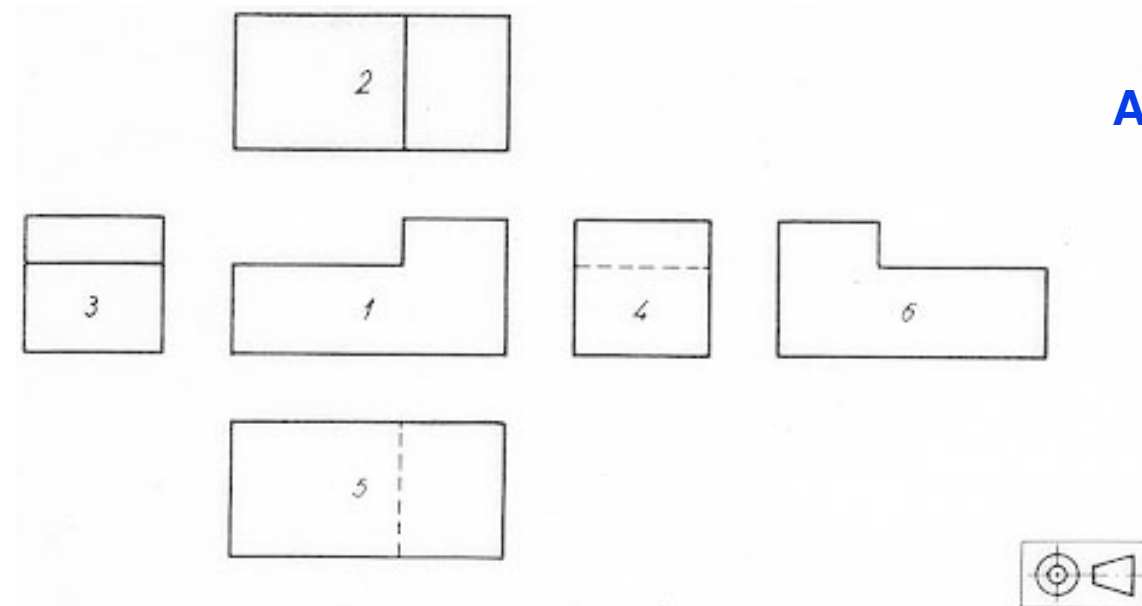


1. *Alçado principal*
2. *Planta*
3. *Alçado lateral esquerdo*
4. *Alçado lateral direito*
5. *Vista de baixo*
6. *Alçado de tardo* *

* Indiferente ficar à esquerda ou à direita

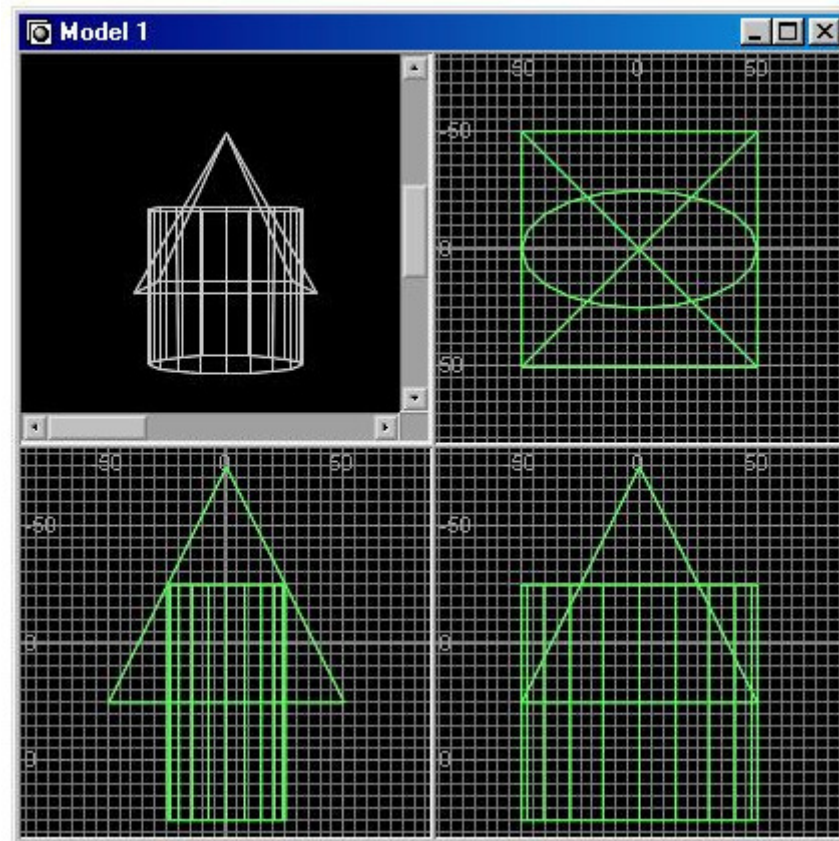


E



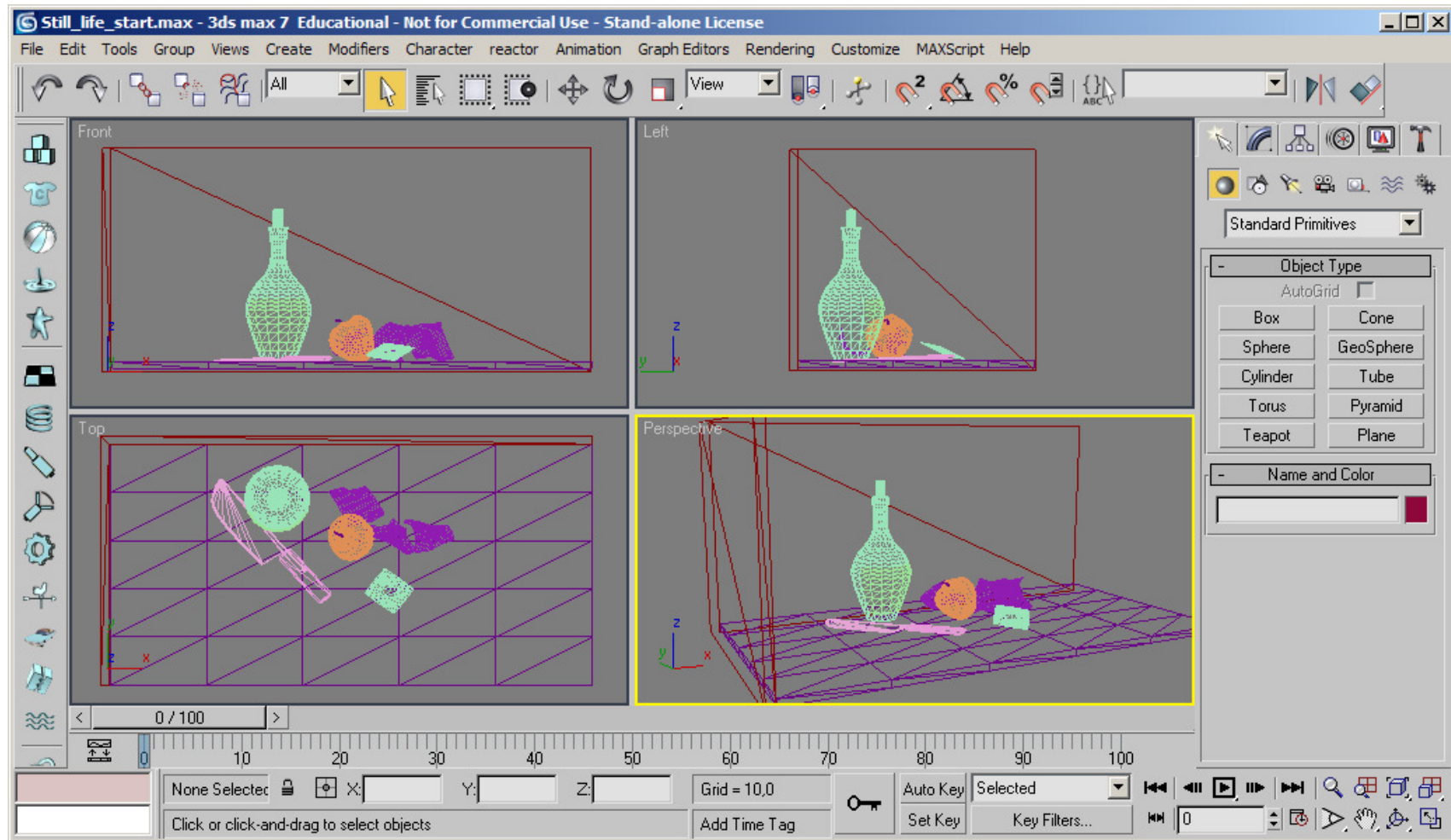
A

Método A



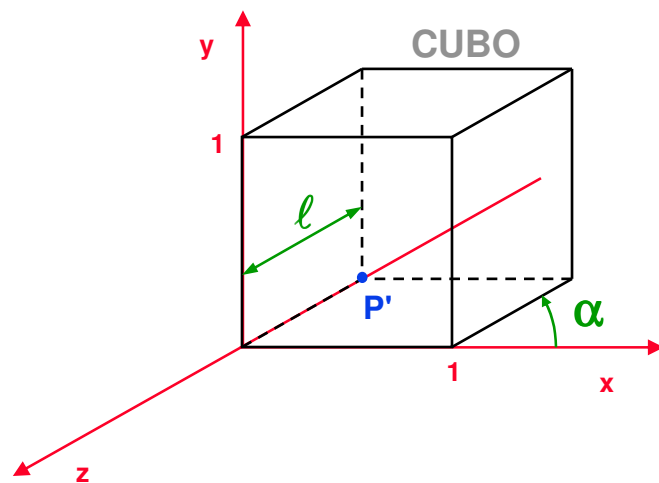
3D Flash Animator

Método E



3D Studio Max

PROJEÇÃO OBLÍQUA



ℓ – fator de redução ou de encurtamento (*Foreshortening Ratio*)

α – ângulo de fuga

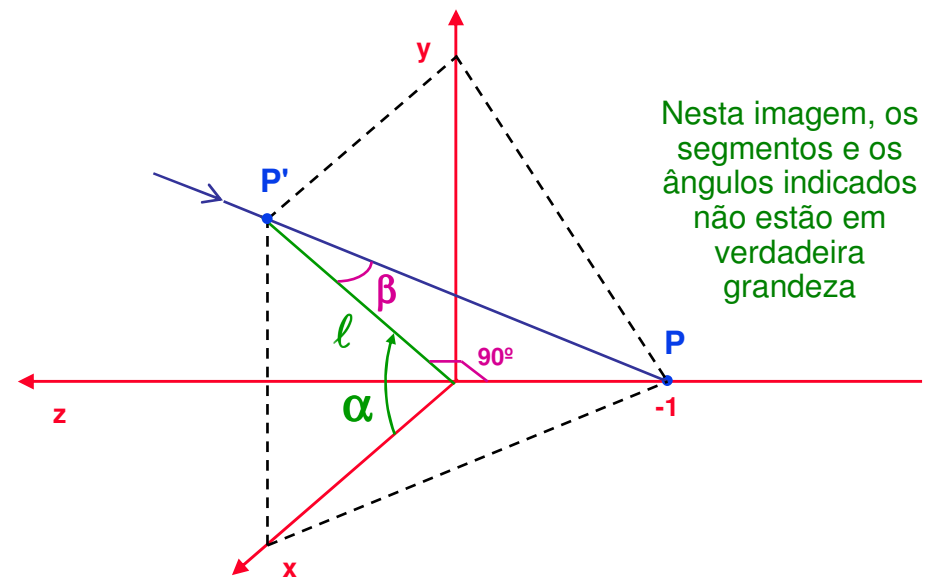
(valores medidos no espaço da imagem)

Plano de projeção: xy

$$P (0, 0, -1) \rightarrow P' (\ell \cos \alpha, \ell \sin \alpha, 0)$$

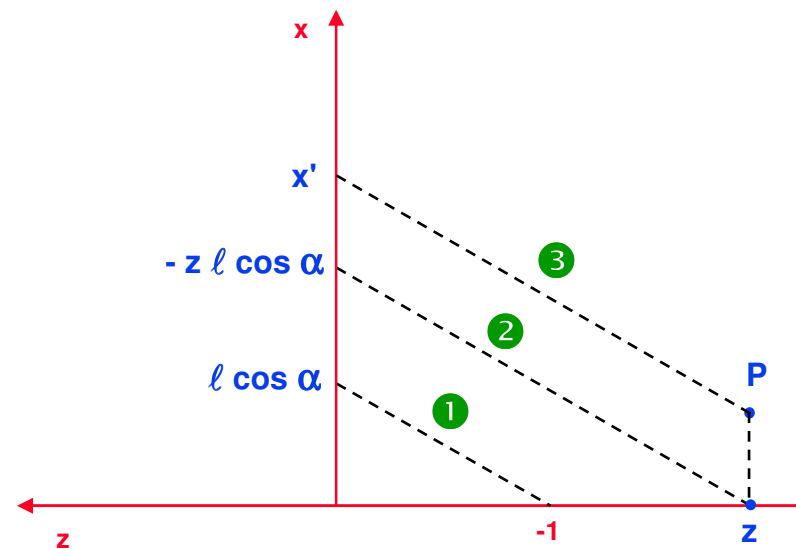
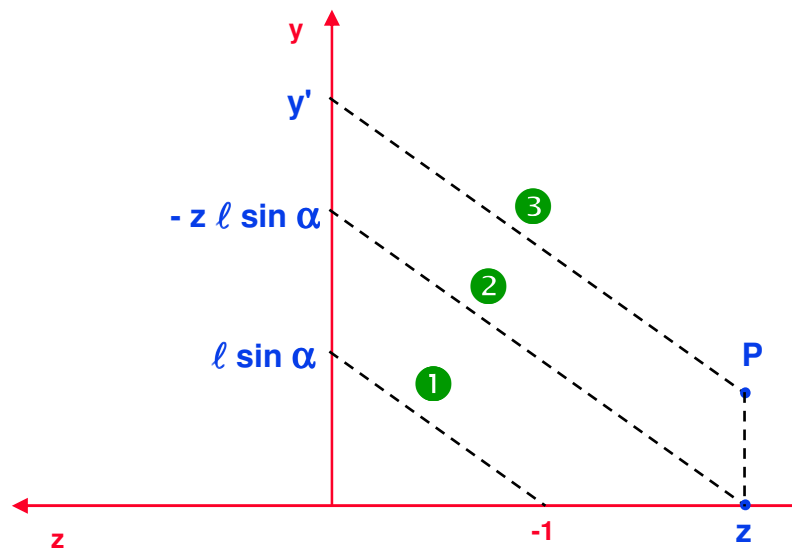
Direção de projeção:

$$DOP = P - P' = \begin{bmatrix} -\ell \cos \alpha \\ -\ell \sin \alpha \\ -1 \end{bmatrix}$$



$$\textcircled{1} \quad P(0, 0, -1) \rightarrow P'(\ell \cos \alpha, \ell \sin \alpha, 0)$$

$$\textcircled{2} \quad P(0, 0, z) \rightarrow P'(-z \ell \cos \alpha, -z \ell \sin \alpha, 0)$$



Ponto genérico:

$$\textcircled{3} \quad P(x, y, z) \rightarrow P'(x - z \ell \cos \alpha, y - z \ell \sin \alpha, 0)$$



$$M_{\text{OBL}} = \begin{bmatrix} 1 & 0 & -\ell \cos \alpha & 0 \\ 0 & 1 & -\ell \sin \alpha & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As projeções oblíquas são determinadas/caracterizadas

- Pelo ângulo β que as projetantes fazem com o plano de projecção ($z=0$) (ℓ é função de β)
- Pela orientação das projetantes, independentemente do ângulo com o plano de projecção (usam-se, habitualmente, valores de 45° ou 30° para α).

Projeção CAVALEIRA:

$$\ell = 1$$



$$\beta = 45^\circ$$

Projeção de GABINETE:

$$\ell = 0.5$$



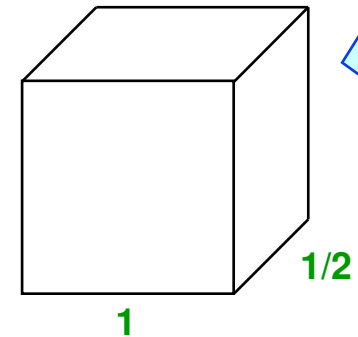
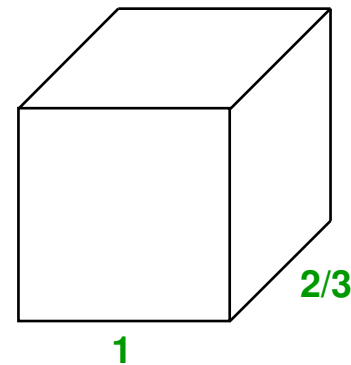
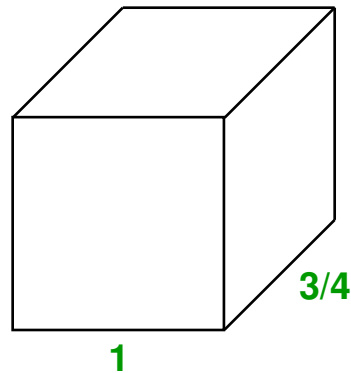
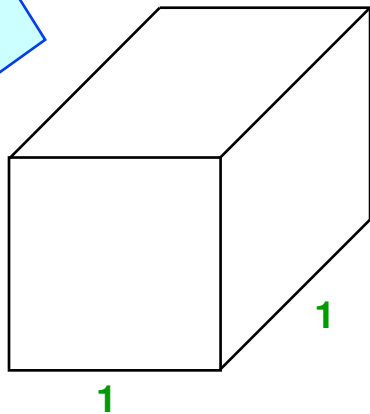
$$\beta = 63.4^\circ$$

Projeção ORTOGONAL:

$$\ell = 0$$

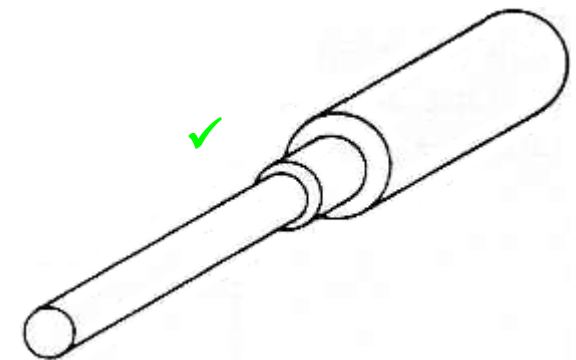
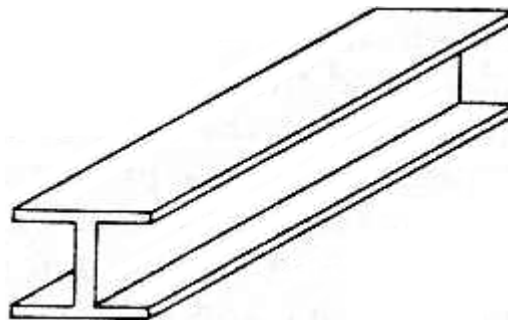
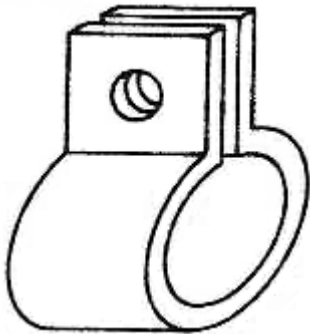
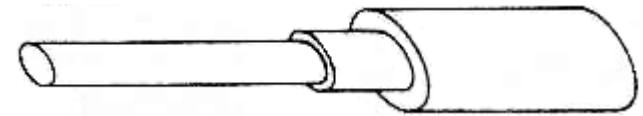
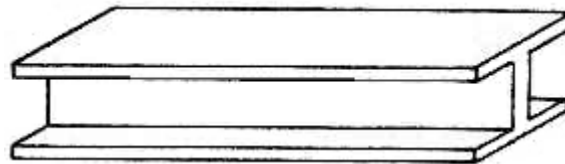
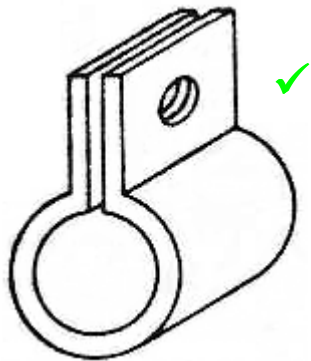


$$\beta = 90^\circ$$



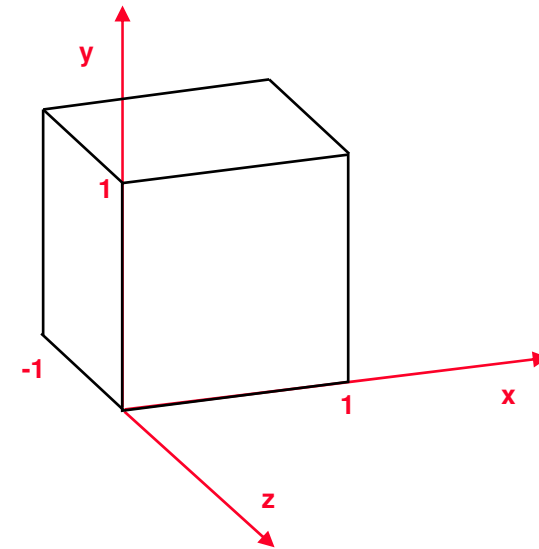
REGRAS DA PROJEÇÃO OBLÍQUA:

- R1)** O plano de projeção deverá ser paralelo às faces mais irregulares do objeto ou às que contêm formas curvas.
- R2)** O plano de projeção deverá ser paralelo à face de maior comprimento do objeto.
- R3)** A regra **R1** tem preferência sobre **R2**.

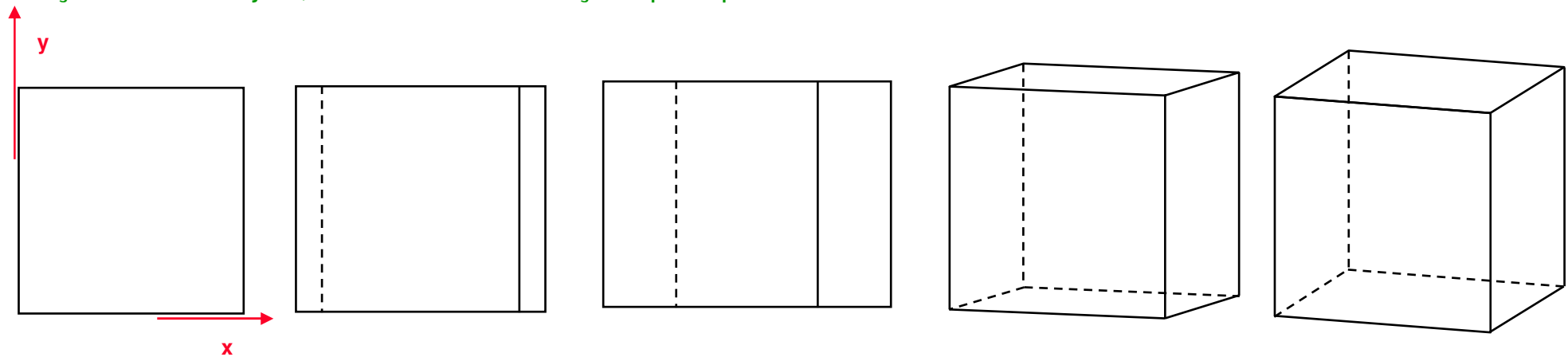


PROJEÇÃO AXONOMÉTRICA

Objeto e seu Sistema de Coordenadas (SC):



Ação sobre o objeto, visualizando-se o alçado principal:



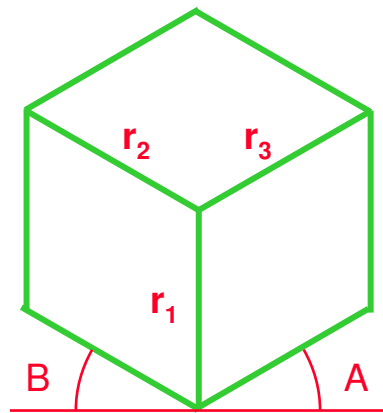
Tratamento matemático, para o caso geral:

$$\mathbf{M}_{AX} = \mathbf{M}_{ORT} \cdot \mathbf{R}_X(\gamma) \cdot \mathbf{R}_Y(\theta)$$

↪ (em $z=0$)

DESENHO AXONOMÉTRICO

ISOMETRIA

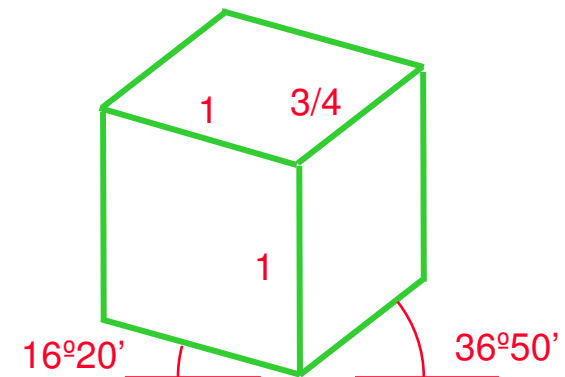
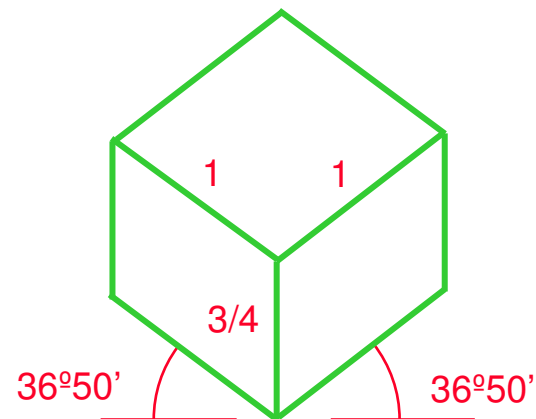


$$A = B = 30^\circ$$

Scale ratios (fatores de escala) do desenho:

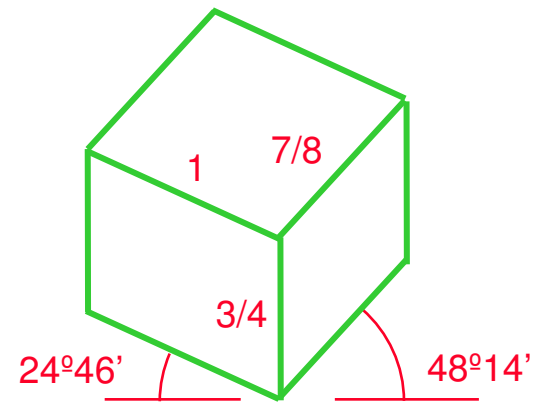
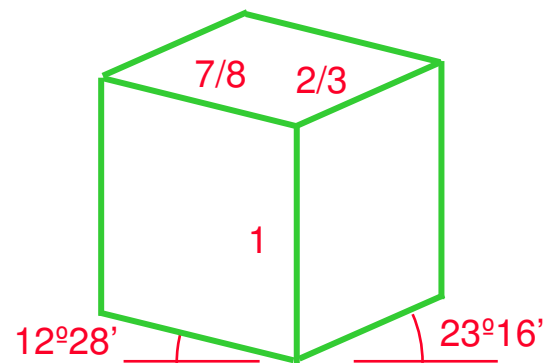
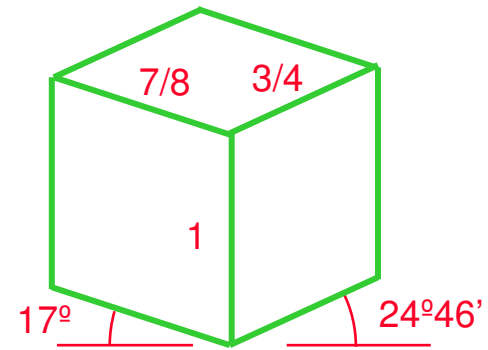
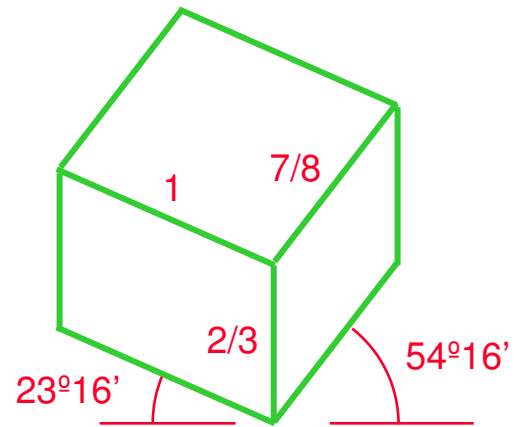
$$r_1 = r_2 = r_3 = 1 \quad (\text{ou } \underline{\text{relação das dimensões } 1:1:1})$$

DIMETRIA



DESENHO AXONOMÉTRICO

TRIMETRIA



Desenho dimétrico de relação 1:1:0.5

The image shows a yellow triangular ruler with a circular hole and a ruler scale. A red table of trigonometric values and a geometric diagram of a cube are overlaid on the ruler.

a	b	E	e	m	R	r
20	212	19	7	30	15	
25	265	24	9	38	18	
30	318	29	105	45	22	
35	37	335	125	53	26	
40	425	382	142	60	30	
45	478	43	16	68	33	
50	53	48	175	75	37	
55	583	525	195	83	41	
60	635	573	212	90	45	
65	69	62	23	98	48	
70	742	67	25	105	52	
75	80	72	265	113	56	
80	85	765	283	120	60	
85	90	81	30	128	63	
90	955	86	32	135	67	
95	101	90	34	142	71	
100	106	95	36	150	75	

The geometric diagram shows a cube with various points labeled (a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z) and lines indicating dimensions and angles. A 42° angle is marked at the bottom right corner of the cube.

PROJEÇÃO AXONOMÉTRICA

$$\theta = \arctg \sqrt{\text{tg } A / \text{tg } B} - \pi/2$$

$$\gamma = \arcsin \sqrt{\text{tg } A \cdot \text{tg } B}$$

$$r_1 = \cos \gamma$$

$$r_2 = \cos \theta / \cos B$$

$$r_3 = -\sin \theta / \cos A$$

Aplicação – Cálculo de alguns fatores de escala (ou de redução) de uma projeção:

i. $A = B = 30^\circ$

$$r_1 = r_2 = r_3 \approx 0.81650$$

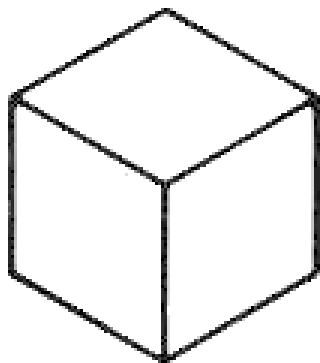
ii. $A = 36^\circ 50'$ $B = 16^\circ 20'$

$$r_1 = r_2 \approx 0.88346 \quad r_3 \approx 0.66257$$

iii. $A = 54^\circ 16'$ $B = 23^\circ 16'$

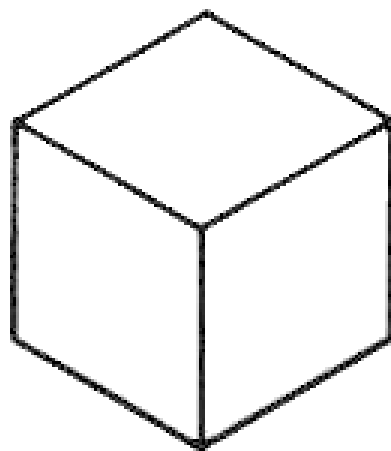
$$r_1 \approx 0.63432 \quad r_2 \approx 0.95128 \quad r_3 \approx 0.83229$$

NOTA: Todos os fatores de escala da Projeção (em *itálico*, para distinção) são inferiores aos respectivos fatores de escala no Desenho Axonométrico. Neste, o maior desses valores, em cada um dos casos, seria sempre igual a 1 (o que se justifica pela comodidade do desenho manual e da leitura de comprimentos).



81,65:100

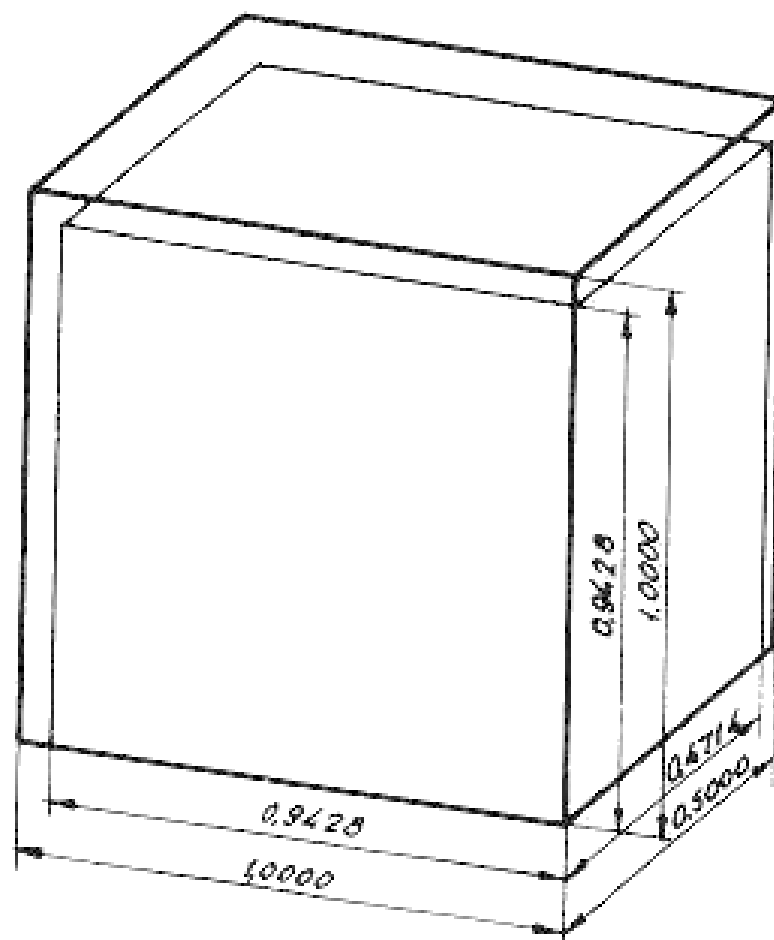
projecção isométrico



100:100

desenho isométrico

Dimetria A=42° B=7°

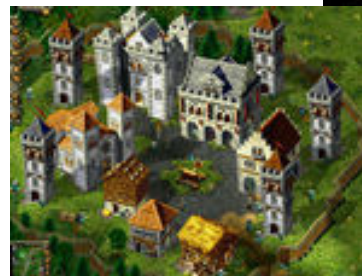


Exemplos de aplicação em jogos de estratégia

No Man's Land



Northland



Uma projeção axonométrica é determinada/caracterizada

- Pelos ângulos que os eixos coordenados locais ao objeto fazem com o plano de projeção

ou

- Pelos três fatores de escala

ou

- Pelos ângulos entre os eixos coordenados depois de projetados (na prática: pelos ângulos A e B).

CONCLUSÕES sobre a AXONOMETRIA:

- ✓ **O paralelismo de linhas é preservado...**
- ✓ **... mas os ângulos não o são;**
- ✓ **Os comprimentos são medidos usando-se fatores de escala correspondentes às 3 direções axiais*.**

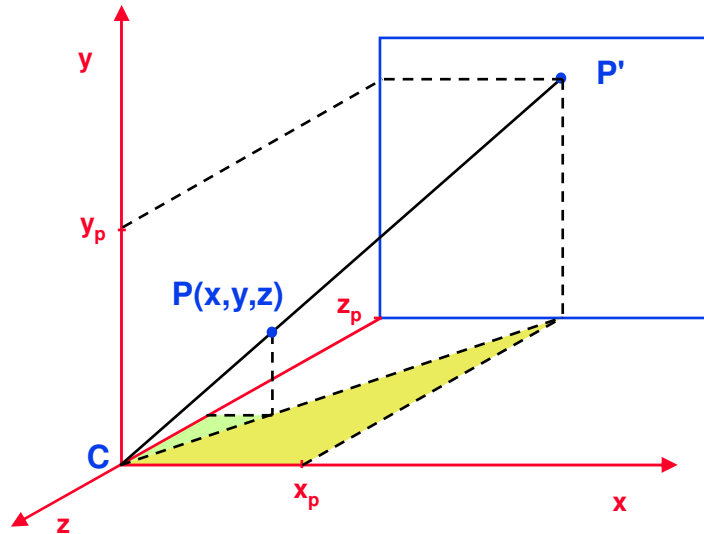
* Justificação do nome AXONOMETRIA (ie, medida segundo os eixos).

Cuidado com as justificações !!! 😊



PROJEÇÃO PERSPECTIVA (1)

Plano de projeção em $z=d \neq 0$ e centro de projeção C na origem:



$$\frac{x_p}{d} = \frac{x}{z} \quad \longrightarrow \quad x_p = \frac{x}{z/d}$$

$$\frac{y_p}{d} = \frac{y}{z} \quad \longrightarrow \quad y_p = \frac{y}{z/d}$$

$$z_p = d \quad \longrightarrow \quad z_p = \frac{z}{z/d}$$

Coordenadas da imagem de P :

$$P' = \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}$$

Coordenadas homogêneas de P' :

$$P' = \begin{bmatrix} X=x \\ Y=y \\ Z=z \\ W=z/d \end{bmatrix}$$

Estas coordenadas podem obter-se das de P pela aplicação da matriz M_{PER} :

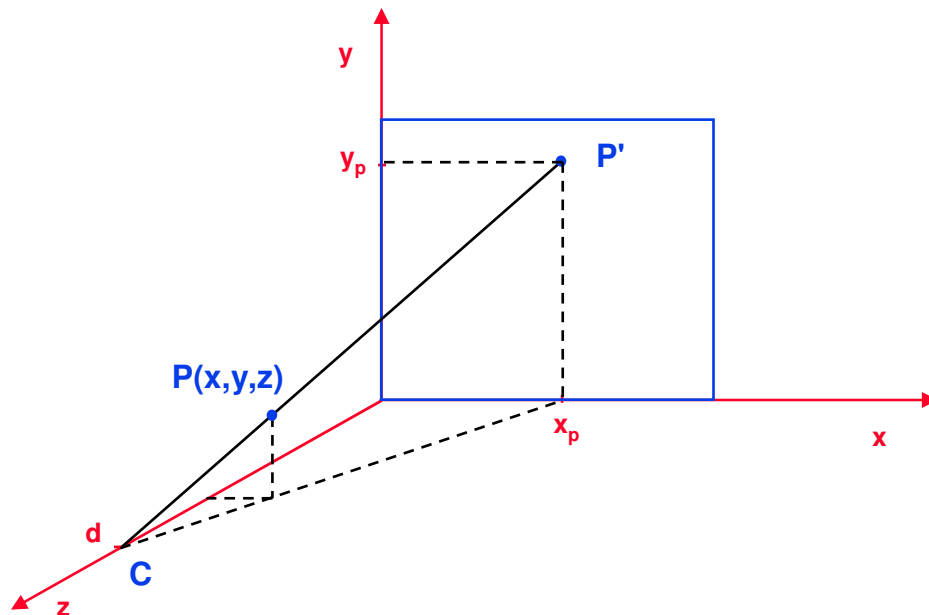
$$\begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} = M_{PER} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

em que

$$M_{PER} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

PROJEÇÃO PERSPECTIVA (2)

Plano de projeção em $z=0$ e centro de projeção C em $(0,0,d)$ com $d \neq 0$:



$$\frac{x_p}{d} = \frac{x}{d-z} \quad \longrightarrow \quad x_p = \frac{x}{1 - z/d}$$

$$\frac{y_p}{d} = \frac{y}{d-z} \quad \longrightarrow \quad y_p = \frac{y}{1 - z/d}$$

$$z_p = 0 \quad \longrightarrow \quad z_p = \frac{0}{1 - z/d}$$

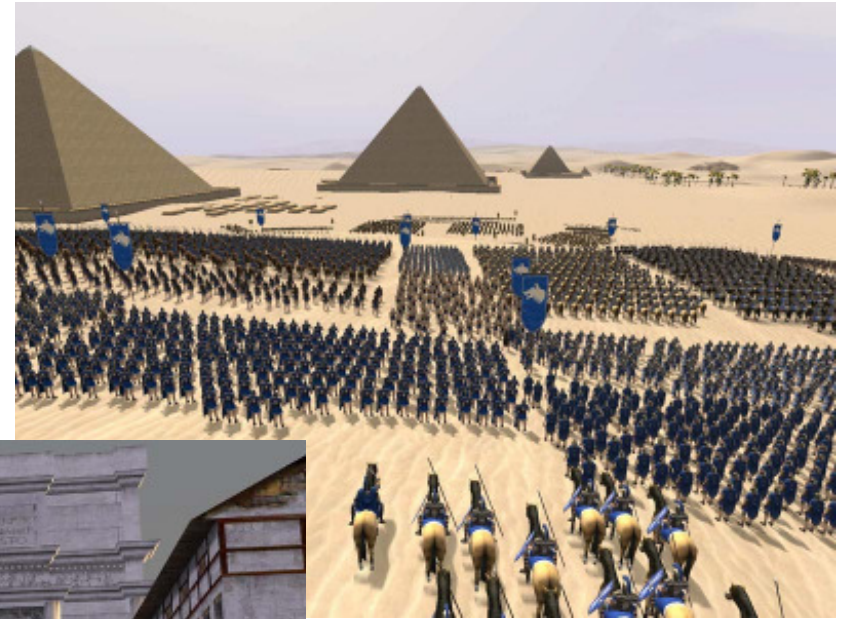
$$\begin{bmatrix} x \\ y \\ 0 \\ 1-z/d \end{bmatrix} = M'_{PER} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\Rightarrow M'_{PER} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix}$$

$$d \rightarrow \infty \Rightarrow M'_{PER} \rightarrow M_{ORT}$$

M'_{PER} aplica-se a $z=0$, como se fez nas outras projeções anteriores

Exemplo em jogo de estratégia com maior realismo



Rome Total War

"The first thing noticeable about the game is its graphical beauty"

Wargamer

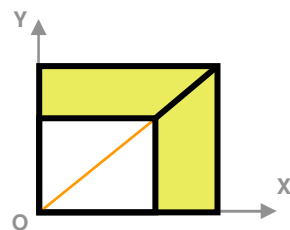
GameSpot: 91% "...realistic, cinematic-style battles."

M.Próspero

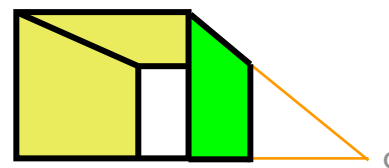
Implicações do paralelismo das direções principais do objeto com os eixos axiais

1 ponto de fuga

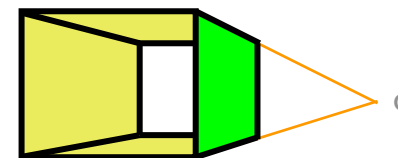
Duas famílias de arestas paralelas a XY



Modelo: Paralelepípedo alinhado com XYZ...
(caixa sem tampa nem fundo)



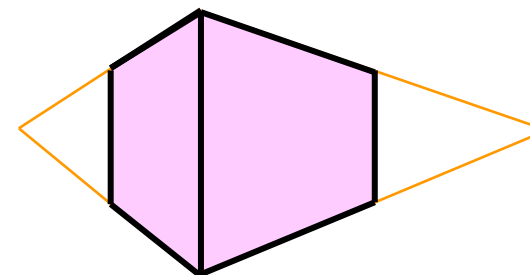
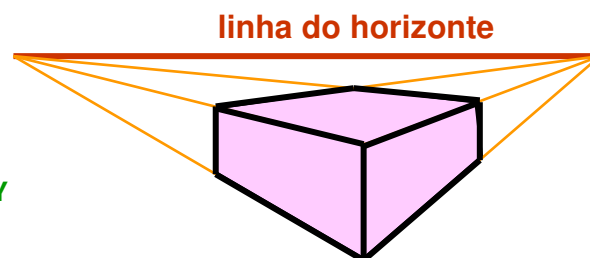
deslocado para a esquerda...



e também para baixo.

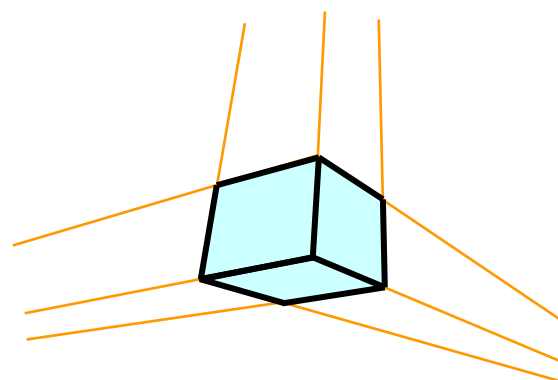
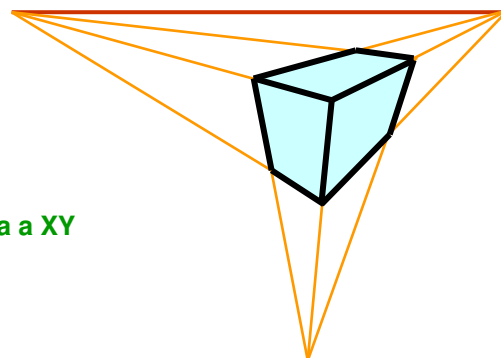
2 pontos de fuga

Uma família de arestas paralela a XY



3 pontos de fuga

Nenhuma família de arestas paralela a XY

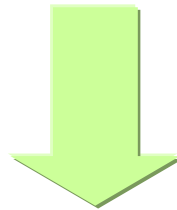


Identifique e localize os pontos de fuga:



SUPERFÍCIES BICÚBICAS NA FORMA PARAMÉTRICA

$$\begin{aligned} Q(s,t) = & a_{11} * s^3 * t^3 + a_{12} * s^3 * t^2 + a_{13} * s^3 * t + a_{14} * s^3 + \\ & a_{21} * s^2 * t^3 + a_{22} * s^2 * t^2 + a_{23} * s^2 * t + a_{24} * s^2 + \\ & a_{31} * s * t^3 + a_{32} * s * t^2 + a_{33} * s * t + a_{34} * s + \\ & a_{41} * t^3 + a_{42} * t^2 + a_{43} * t + a_{44} \end{aligned}$$



$$0 \leq s \leq 1$$

$$0 \leq t \leq 1$$

$$Q(s,t) = S \cdot A \cdot T^T$$

$$S = [s^3 \quad s^2 \quad s \quad 1]$$

$$T = [t^3 \quad t^2 \quad t \quad 1]$$

FORMA de HERMITE

Curva de Hermite:

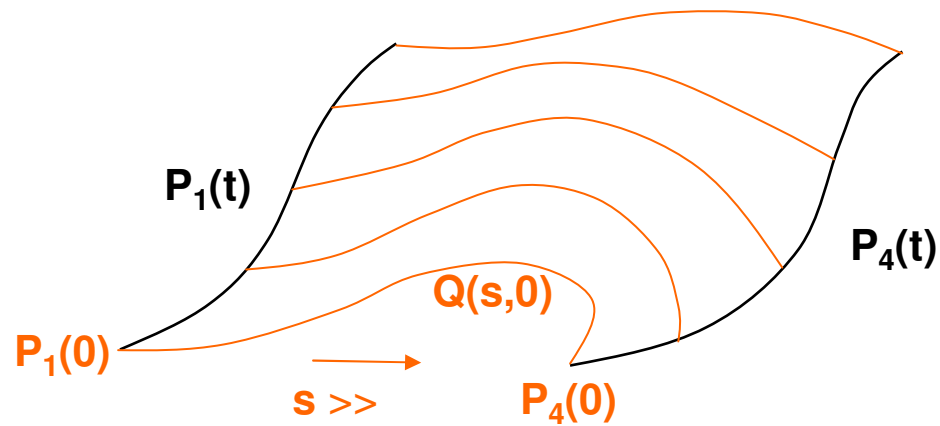
$$Q(s) = S \cdot M_H \cdot G_H$$



Superfície:

$$Q(s,t) = S \cdot M_H \cdot G_H(t) = S \cdot M_H \cdot \begin{bmatrix} P_1(t) \\ P_4(t) \\ R_1(t) \\ R_4(t) \end{bmatrix}$$

∴ A superfície $Q(s,t)$ é interpolação entre $P_1(t)$ e $P_4(t)$



Sejam os elementos de $\mathbf{G}_H(t)$ curvas de Hermite:

$$\mathbf{P}_1(t) = \mathbf{T} \cdot \mathbf{M}_H \cdot \begin{bmatrix} \mathbf{g}_{11} \\ \mathbf{g}_{12} \\ \mathbf{g}_{13} \\ \mathbf{g}_{14} \end{bmatrix}$$

$$\mathbf{P}_4(t) = \mathbf{T} \cdot \mathbf{M}_H \cdot \begin{bmatrix} \mathbf{g}_{21} \\ \mathbf{g}_{22} \\ \mathbf{g}_{23} \\ \mathbf{g}_{24} \end{bmatrix}$$

$$\mathbf{R}_1(t) = \mathbf{T} \cdot \mathbf{M}_H \cdot \begin{bmatrix} \mathbf{g}_{31} \\ \mathbf{g}_{32} \\ \mathbf{g}_{33} \\ \mathbf{g}_{34} \end{bmatrix}$$

$$\mathbf{R}_4(t) = \mathbf{T} \cdot \mathbf{M}_H \cdot \begin{bmatrix} \mathbf{g}_{41} \\ \mathbf{g}_{42} \\ \mathbf{g}_{43} \\ \mathbf{g}_{44} \end{bmatrix}$$



$$\begin{bmatrix} \mathbf{P}_1(t) & \mathbf{P}_4(t) & \mathbf{R}_1(t) & \mathbf{R}_4(t) \end{bmatrix} = \mathbf{T} \cdot \mathbf{M}_H \cdot \begin{bmatrix} \mathbf{g}_{11} & \mathbf{g}_{21} & \mathbf{g}_{31} & \mathbf{g}_{41} \\ \mathbf{g}_{12} & \mathbf{g}_{22} & \mathbf{g}_{32} & \mathbf{g}_{42} \\ \mathbf{g}_{13} & \mathbf{g}_{23} & \mathbf{g}_{33} & \mathbf{g}_{43} \\ \mathbf{g}_{14} & \mathbf{g}_{24} & \mathbf{g}_{34} & \mathbf{g}_{44} \end{bmatrix}$$

$\mathbf{G}_H(t)^T$

Por transposição:

$$\mathbf{G}_H(t) = \begin{bmatrix} \mathbf{g}_{11} & \mathbf{g}_{12} & \mathbf{g}_{13} & \mathbf{g}_{14} \\ \mathbf{g}_{21} & \mathbf{g}_{22} & \mathbf{g}_{23} & \mathbf{g}_{24} \\ \mathbf{g}_{31} & \mathbf{g}_{32} & \mathbf{g}_{33} & \mathbf{g}_{34} \\ \mathbf{g}_{41} & \mathbf{g}_{42} & \mathbf{g}_{43} & \mathbf{g}_{44} \end{bmatrix} \cdot \mathbf{M}_H^T \cdot \mathbf{T}^T = \underline{\mathbf{G}}_H \cdot \mathbf{M}_H^T \cdot \mathbf{T}^T$$

Por substituição em $\mathbf{Q}(s,t) = \mathbf{S} \cdot \mathbf{M}_H \cdot \mathbf{G}_H(t)$:

$$\mathbf{Q}(s,t) = \mathbf{S} \cdot \mathbf{M}_H \cdot \underline{\mathbf{G}}_H \cdot \mathbf{M}_H^T \cdot \mathbf{T}^T$$

Superfície de Hermite

$$\underline{G}_H = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix}$$

Interpretação da matriz de geometria de Hermite $\underline{G}_H$:

$$\underline{G}_H = \begin{bmatrix} Q(0,0) & Q(0,1) & \partial Q(s,t)/\partial t |_{s=0,t=0} & \partial Q(s,t)/\partial t |_{s=0,t=1} \\ Q(1,0) & Q(1,1) & \partial Q(s,t)/\partial t |_{s=1,t=0} & \partial Q(s,t)/\partial t |_{s=1,t=1} \\ \hline \partial Q(s,t)/\partial s |_{s=0,t=0} & \partial Q(s,t)/\partial s |_{s=0,t=1} & \partial^2 Q(s,t)/\partial s \partial t |_{s=0,t=0} & \partial^2 Q(s,t)/\partial s \partial t |_{s=0,t=1} \\ \partial Q(s,t)/\partial s |_{s=1,t=0} & \partial Q(s,t)/\partial s |_{s=1,t=1} & \partial^2 Q(s,t)/\partial s \partial t |_{s=1,t=0} & \partial^2 Q(s,t)/\partial s \partial t |_{s=1,t=1} \end{bmatrix}$$

As primeiras derivadas são os vetores tangentes
e as segundas derivadas denominam-se *twist vectors* (vetores de torção)

Superfícies de Hermite

$$\begin{bmatrix} (-5, 0, 5) & (-5, 0, -5) & (0, 0, -10) & (0, 0, -10) \\ (5, 0, 5) & (5, 0, -5) & (0, 0, -10) & (0, 0, -10) \\ (10, 0, 0) & (10, 0, 0) & (0, 0, 0) & (0, 0, 0) \\ (10, 0, 0) & (10, 0, 0) & (0, 0, 0) & (0, 0, 0) \end{bmatrix}$$

(Superfície de Ferguson)

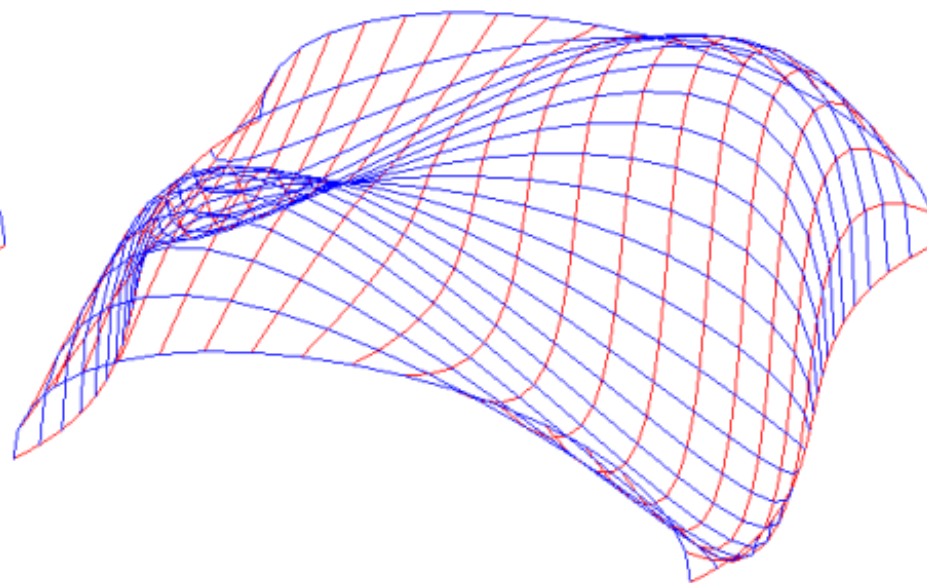
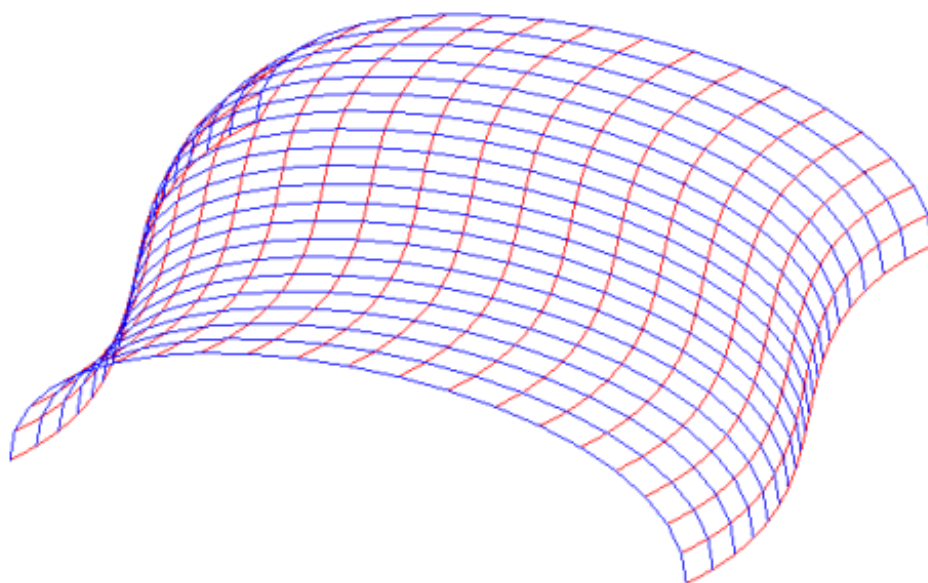
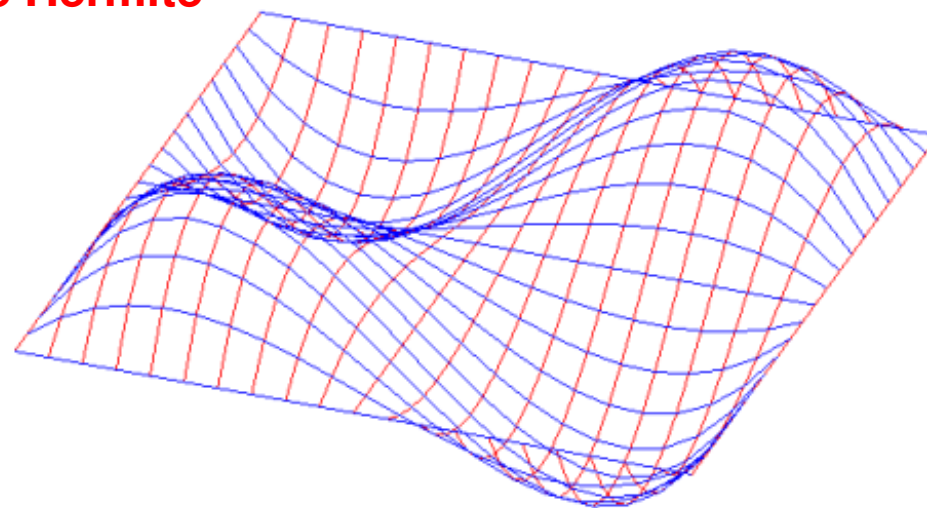
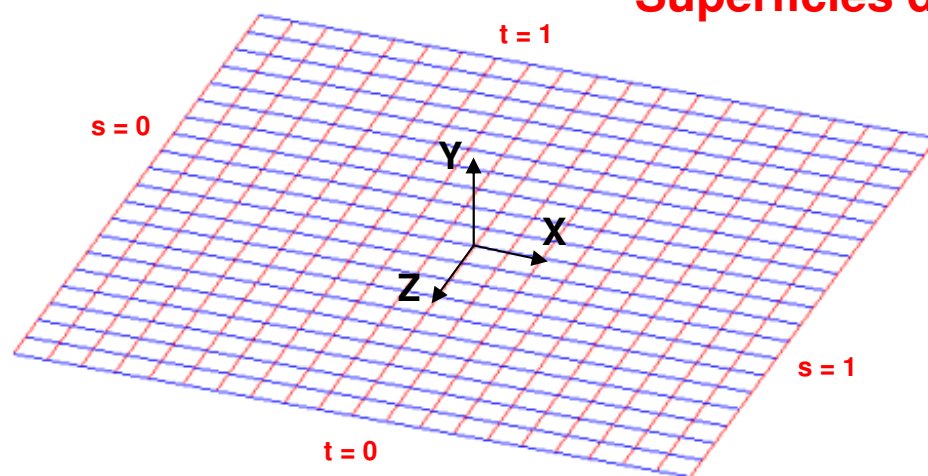
$$\begin{bmatrix} (-5, 0, 5) & (-5, 0, -5) & (0, 0, -10) & (0, 0, -10) \\ (5, 0, 5) & (5, 0, -5) & (0, 0, -10) & (0, 0, -10) \\ (10, 0, 0) & (10, 0, 0) & (0, 200, 0) & (0, 200, 0) \\ (10, 0, 0) & (10, 0, 0) & (0, 200, 0) & (0, 200, 0) \end{bmatrix}$$

$$\begin{bmatrix} (-5, 0, 5) & (-5, 0, -5) & (5, 0, -10) & (5, 0, -10) \\ (5, 0, 5) & (5, 0, -5) & (5, 0, -10) & (5, 0, -10) \\ (0, 10, 0) & (0, 10, 0) & (0, 0, 0) & (0, 0, 0) \\ (0, -10, 0) & (0, -10, 0) & (0, 0, 0) & (0, 0, 0) \end{bmatrix}$$

(Superfície de Ferguson)

$$\begin{bmatrix} (-5, 0, 5) & (-5, 0, -5) & (5, 0, -10) & (5, 0, -10) \\ (5, 0, 5) & (5, 0, -5) & (5, 0, -10) & (5, 0, -10) \\ (0, 10, 0) & (0, 10, 0) & (0, 200, 0) & (0, 200, 0) \\ (0, -10, 0) & (0, -10, 0) & (0, 200, 0) & (0, 200, 0) \end{bmatrix}$$

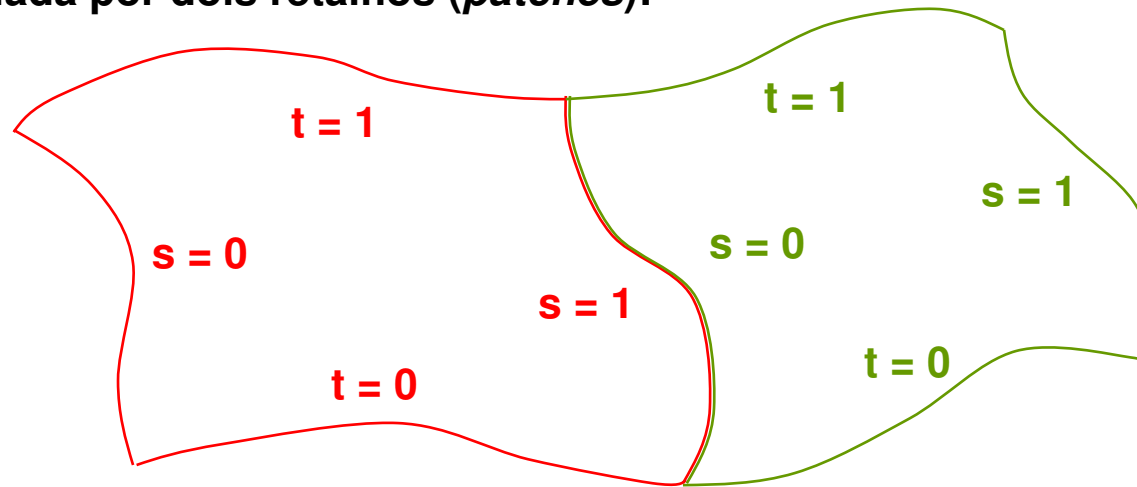
Superfícies de Hermite



Exercício: Esboçar os vetores presentes em cada matriz e interpretar geometricamente o resultado.

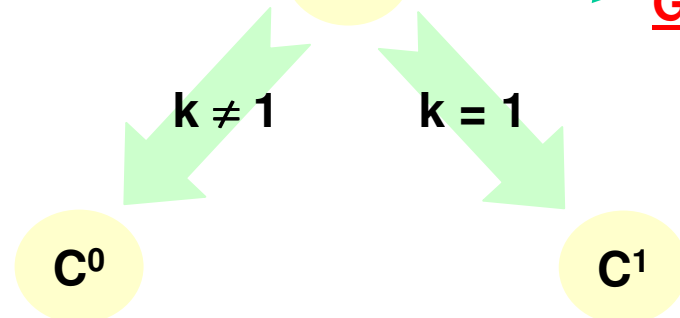
Continuidade Geométrica e Continuidade Paramétrica

Superfície formada por dois retalhos (*patches*):



Para garantir G^0 $\underline{G}_H[2, i] = \underline{G}_H[1, i]$ $i = 1..4$

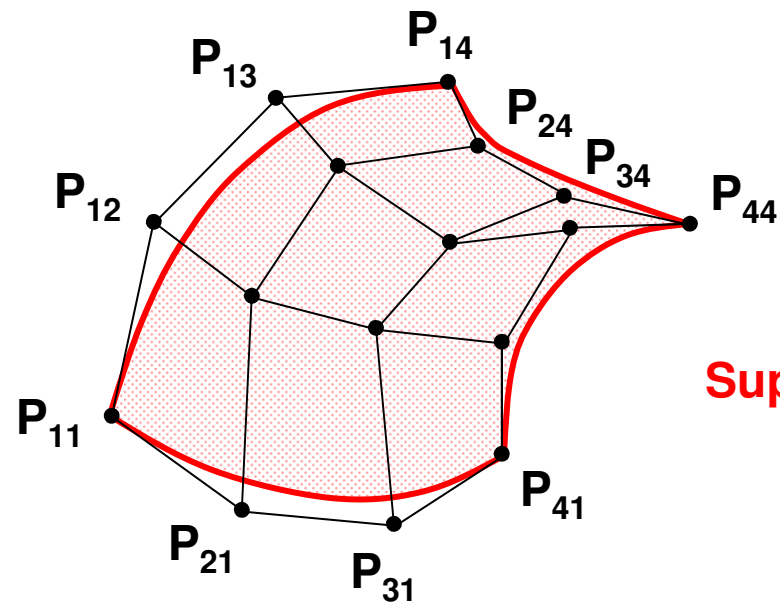
Para garantir G^1 $\underline{G}_H[4, i] = k \cdot \underline{G}_H[3, i]$



Significa que,
para cada valor de t (de
0 a 1), existe $k > 0$ tal
que $k = R_4(t) / R_3(t)$

FORMA de BÉZIER

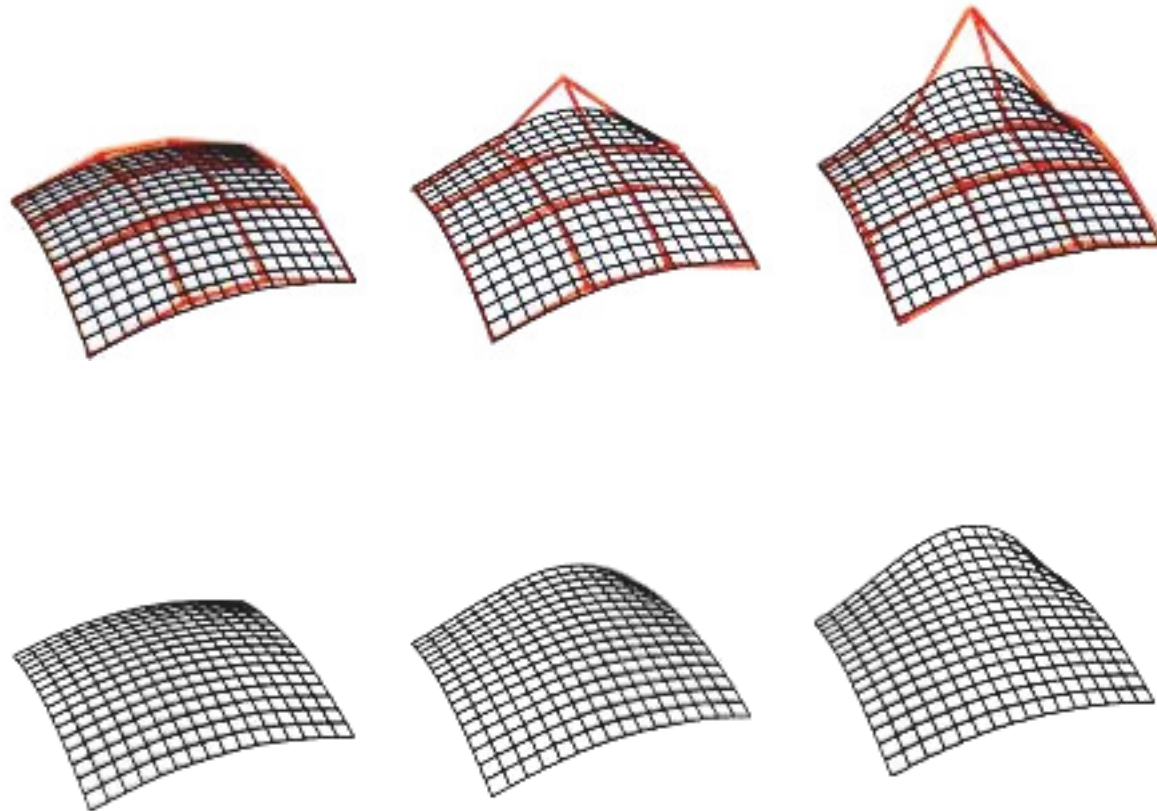
$$Q(s,t) = S \cdot M_B \cdot \underline{G} \cdot M_B^T \cdot T^T$$



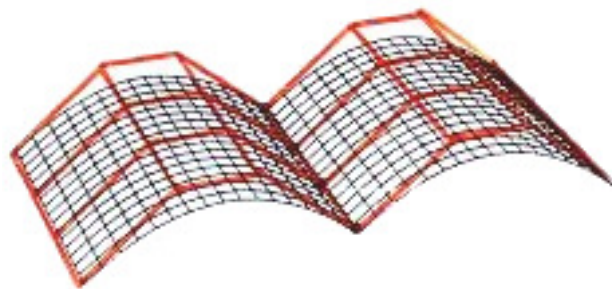
Superfície de Bézier

A matriz de geometria de Bézier $\underline{G}$ é constituída por 16 pontos.
A superfície gerada é interior ao *convex hull* dos pontos de controlo.

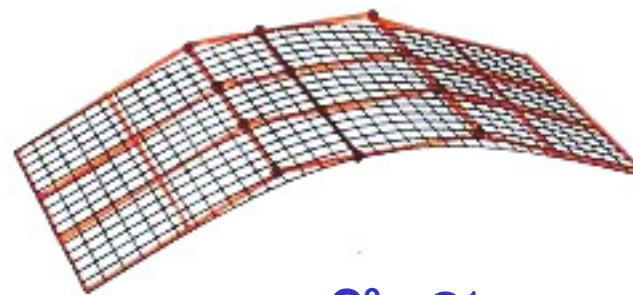
Superfícies de Bézier



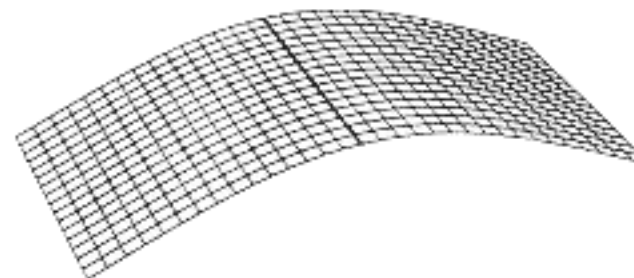
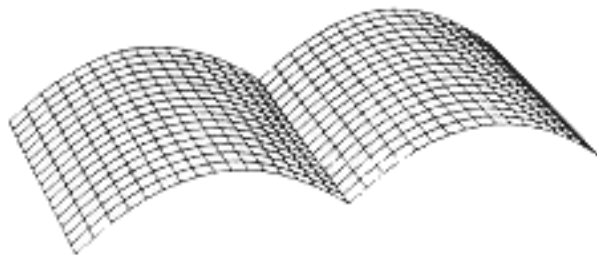
Superfícies de Bézier



$C^0 \quad G^0$



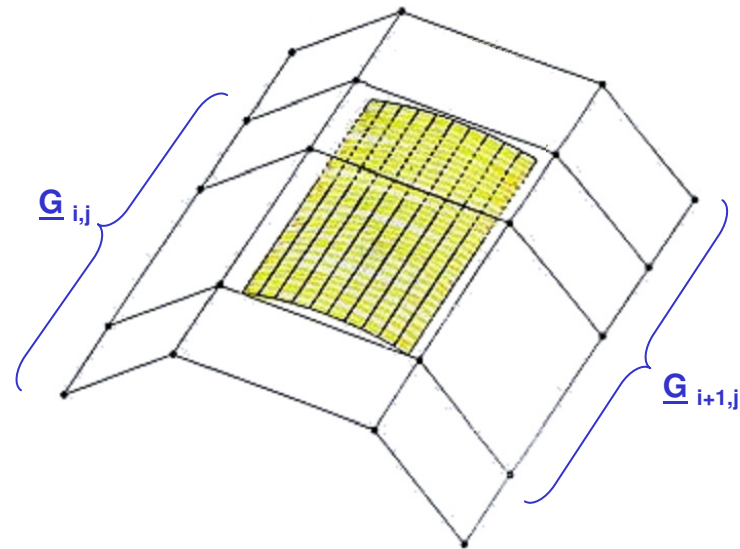
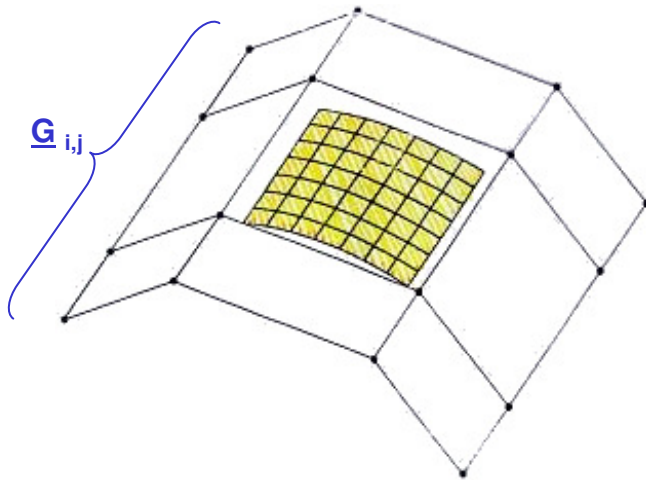
$C^0 \quad G^1$



Garante-se C^0G^0 , C^0G^1 ou C^1G^1 por construção geométrica
(via coincidência de pontos, colinearidade e controlo de distâncias)

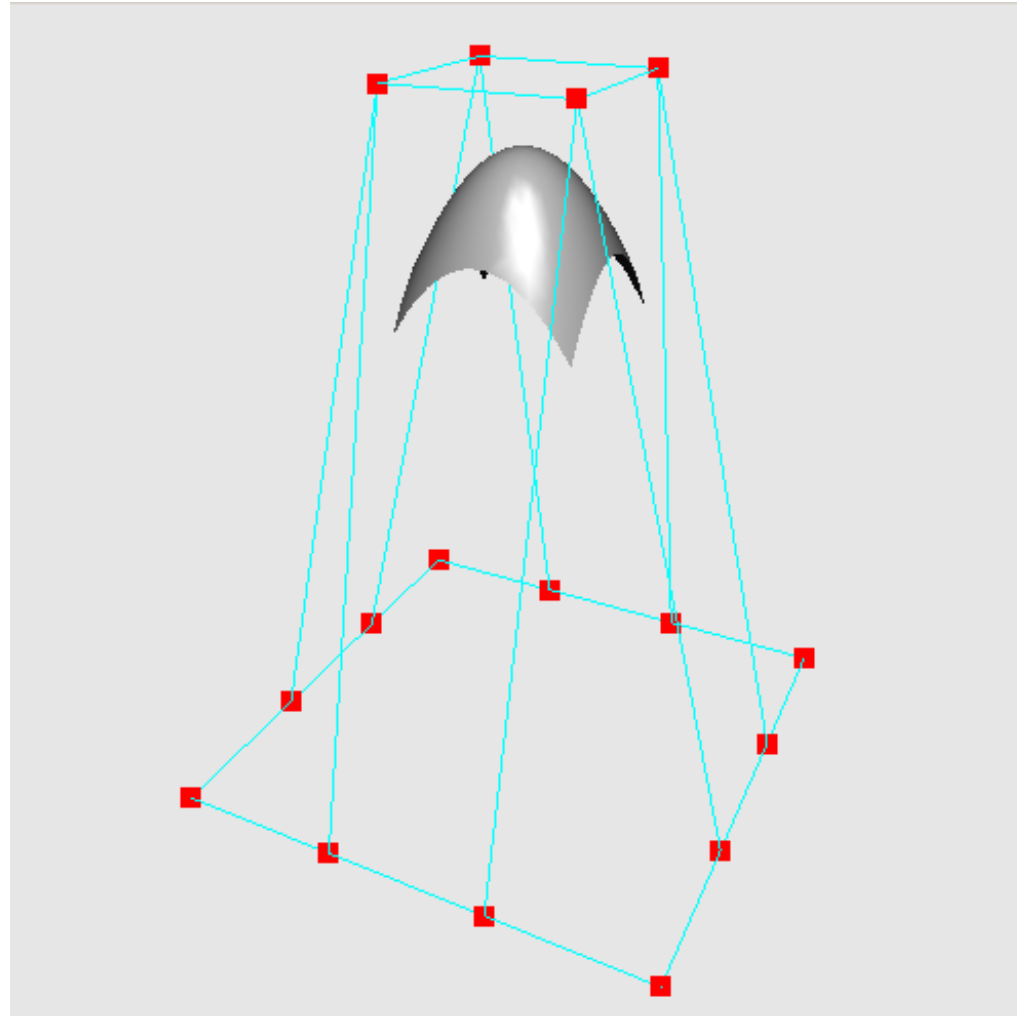
FORMA de B-SPLINE

$$Q(s,t) = S \cdot M_{Bs} \cdot \underline{G} \cdot M_{Bs}^T \cdot T^T$$

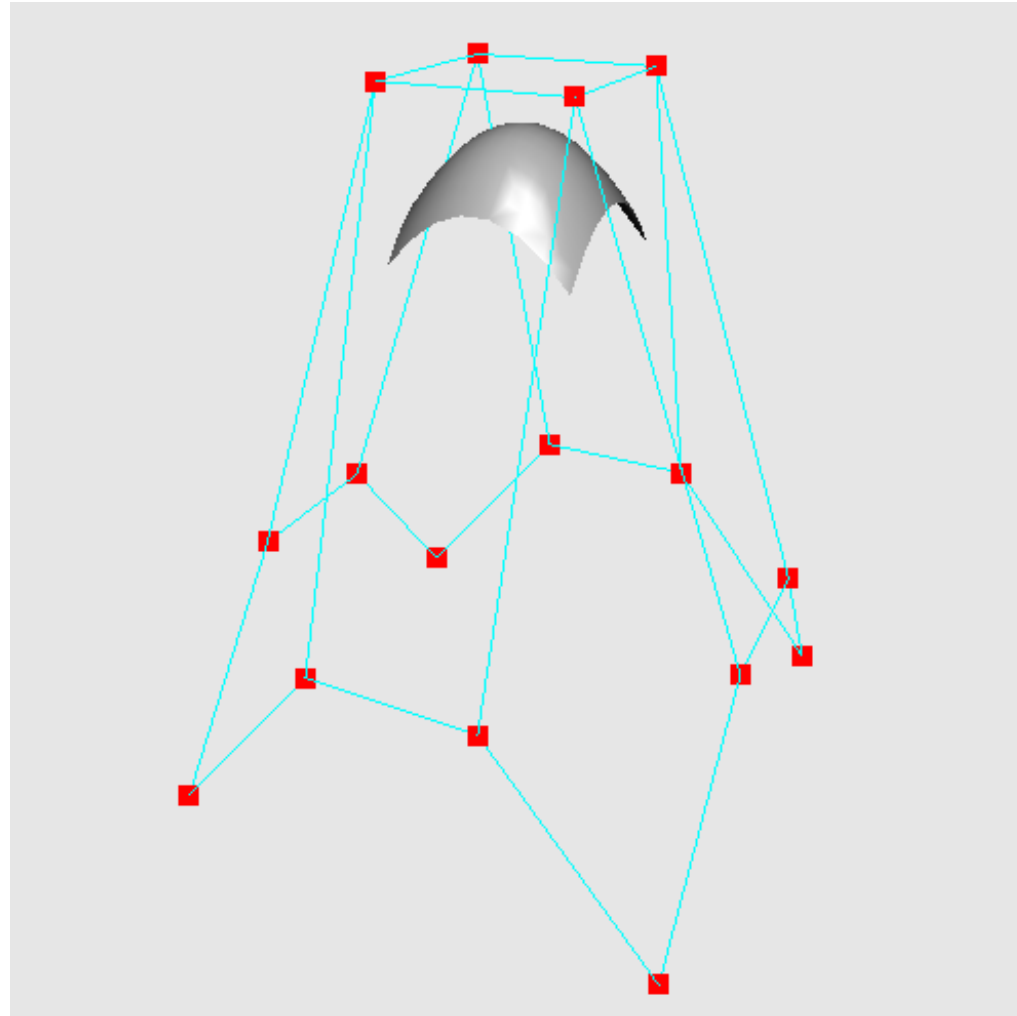


Garante-se C^2 pela utilização dos pontos de controlo via matriz de geometria $\underline{G}_{i,j}$
à semelhança das curvas B-spline com o vector de geometria G_i

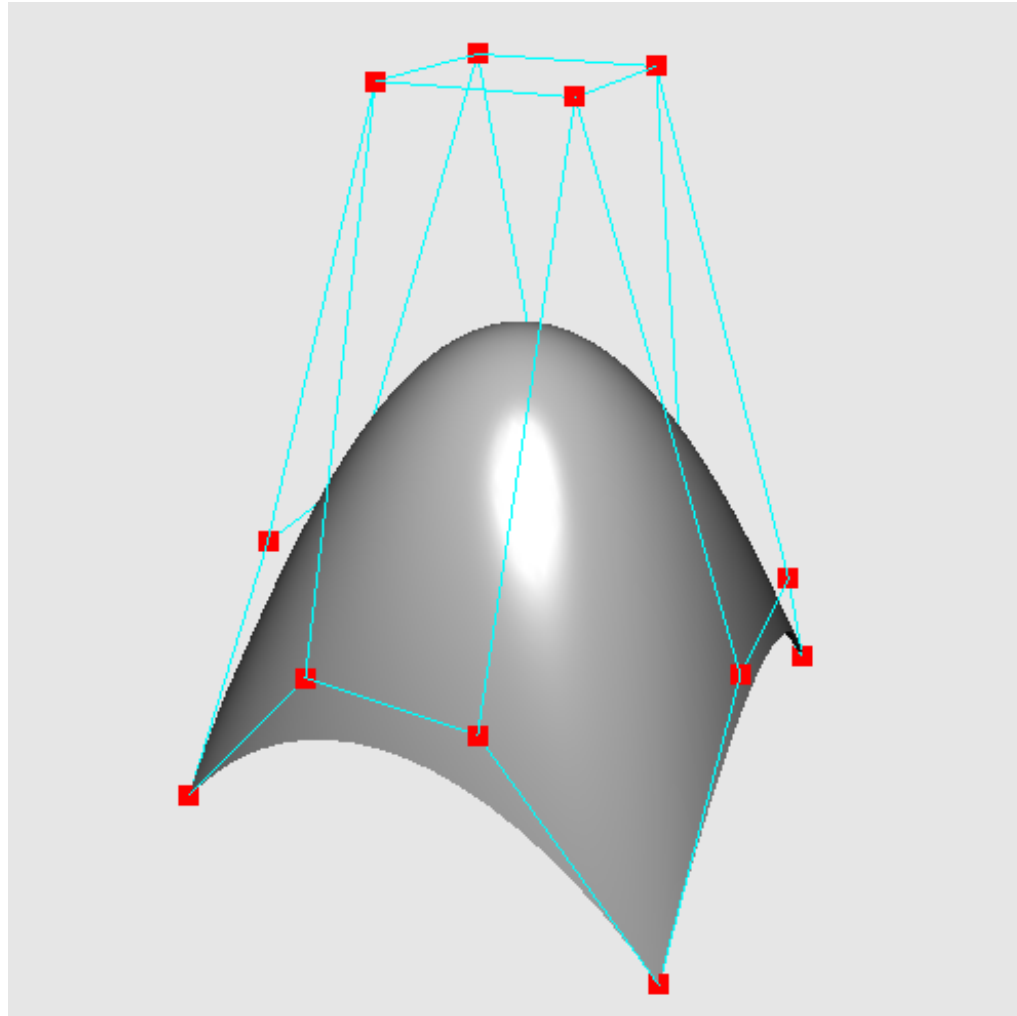
EXEMPLOS



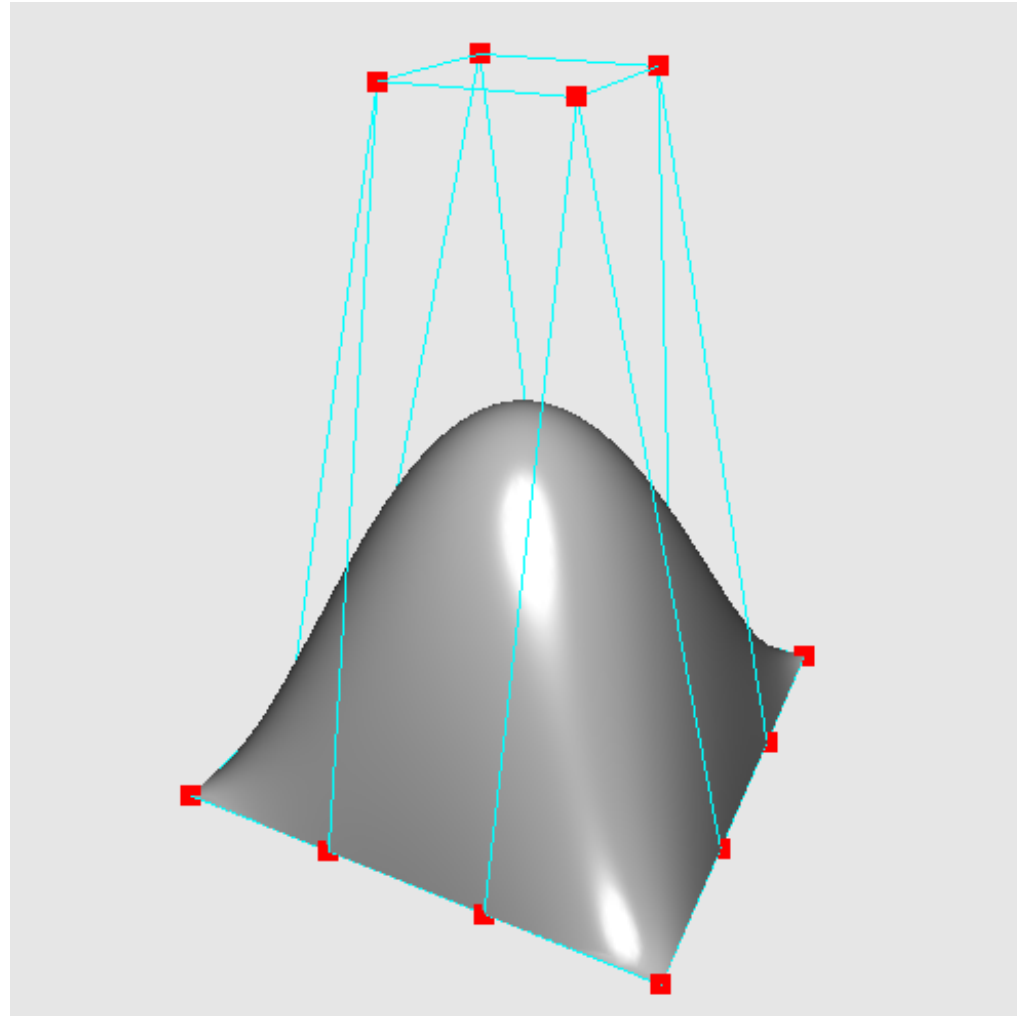
B-spline



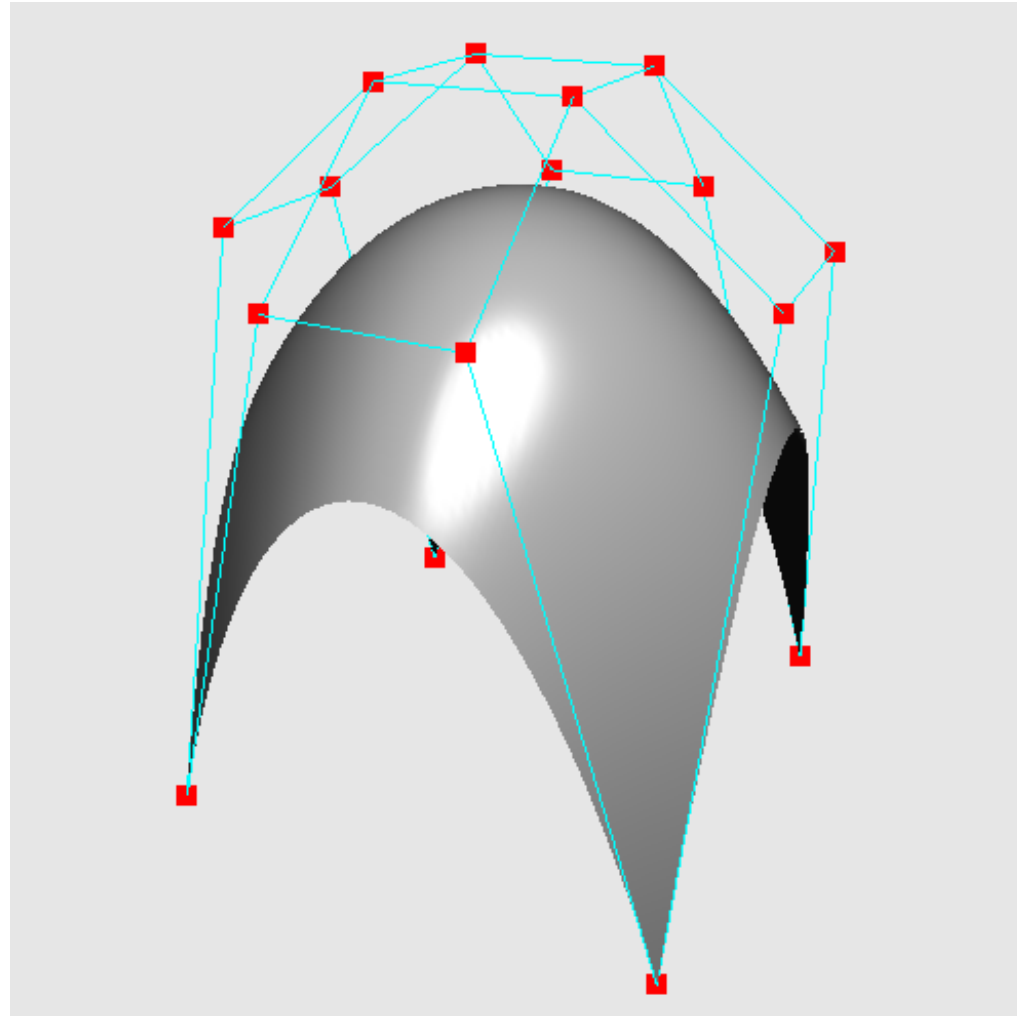
B-spline



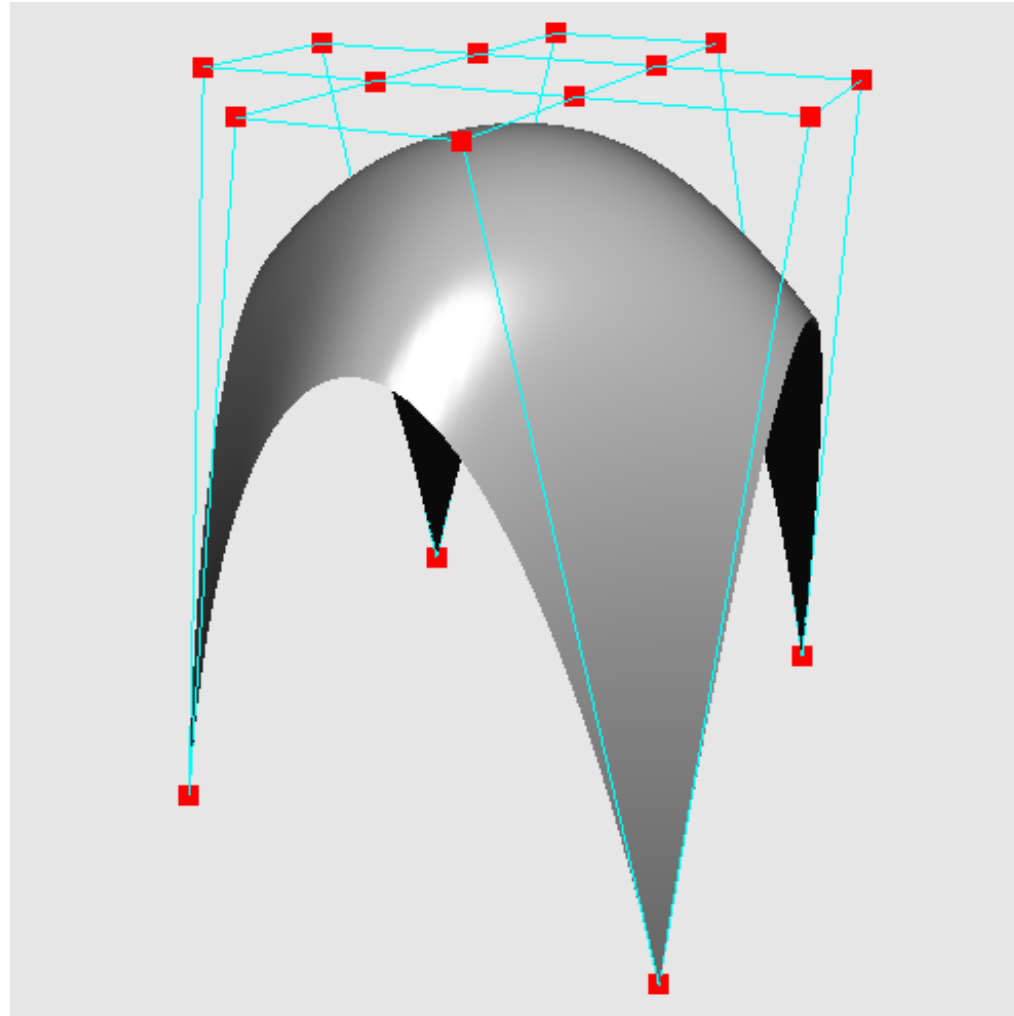
Bézier



Bézier



Bézier



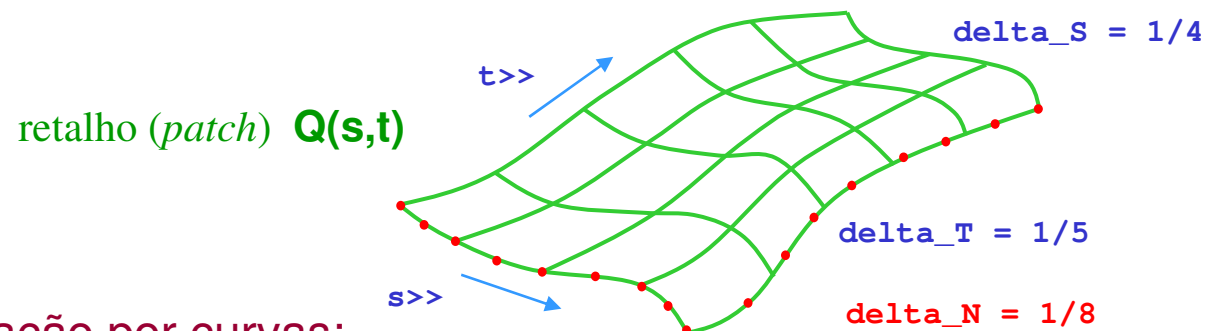
O vetor normal à superfície em cada ponto é imprescindível para a obtenção do sombreamento.

Bézier

AVALIAÇÃO DE UMA SUPERFÍCIE BICÚBICA

Recorde-se que

$$Q(s,t) = S \cdot A \cdot T^T = a_{11} * s^3 * t^3 + a_{12} * s^3 * t^2 + a_{13} * s^3 * t + a_{14} * s^3 + a_{21} * s^2 * t^3 + a_{22} * s^2 * t^2 + a_{23} * s^2 * t + a_{24} * s^2 + a_{31} * s * t^3 + a_{32} * s * t^2 + a_{33} * s * t + a_{34} * s + a_{41} * t^3 + a_{42} * t^2 + a_{43} * t + a_{44}$$



Algoritmo básico de visualização por curvas:

```
for (patch=first to last)
  for (s=0 to 1 step=delta_S)
    for (t=delta_N to 1 step=delta_N)
      Line_3D( Q(s,t-delta_N), Q(s,t) )
    for (t=0 to 1 step=delta_T)
      for (s=delta_N to 1 step=delta_N)
        Line_3D( Q(s-delta_N,t), Q(s,t) )
```

$\delta_N \leq \delta_T$

$\delta_N \leq \delta_S$

2D OpenGL Evaluators applied to bicubic patches

Examples in C

Basic procedure:

1) Define the evaluator

```
glMap2d(GL_MAP2_VERTEX_3, 0.0, 1.0, 3, 4,  
        0.0, 1.0, 12, 4, control_points);
```

2) Enable it with

```
glEnable(GL_MAP2_VERTEX_3);
```

3a) Invoke it by calling

```
glEvalCoord2d(s, t);
```

3b) or

```
glMapGrid2(1/delta_s, 0.0, 1.0, 1/delta_t, 0.0, 1.0);  
glEvalMesh2(mode, 0, 1/delta_s, 0, 1/delta_t);
```

GL_LINE
(a mesh in wireframe) or
GL_FILL
(a filled quad-mesh)

Example:

```
{ a1,a2,a3, b1,b2,b3, c1,c2,c3, d1,d2,d3,  
  e1,e2,e3, f1,f2,f3, g1,g2,g3, h1,h2,h3,  
  i1,i2,i3, j1,j2,j3, k1,k2,k3, m1,m2,m3,  
  n1,n2,n3, o1,o2,o3, p1,p2,p3, q1,q2,q3,  
  r1,r2,r3, s1,s2,s3, t1,t2,t3, u1,u2,u3 }
```

s stride

s order

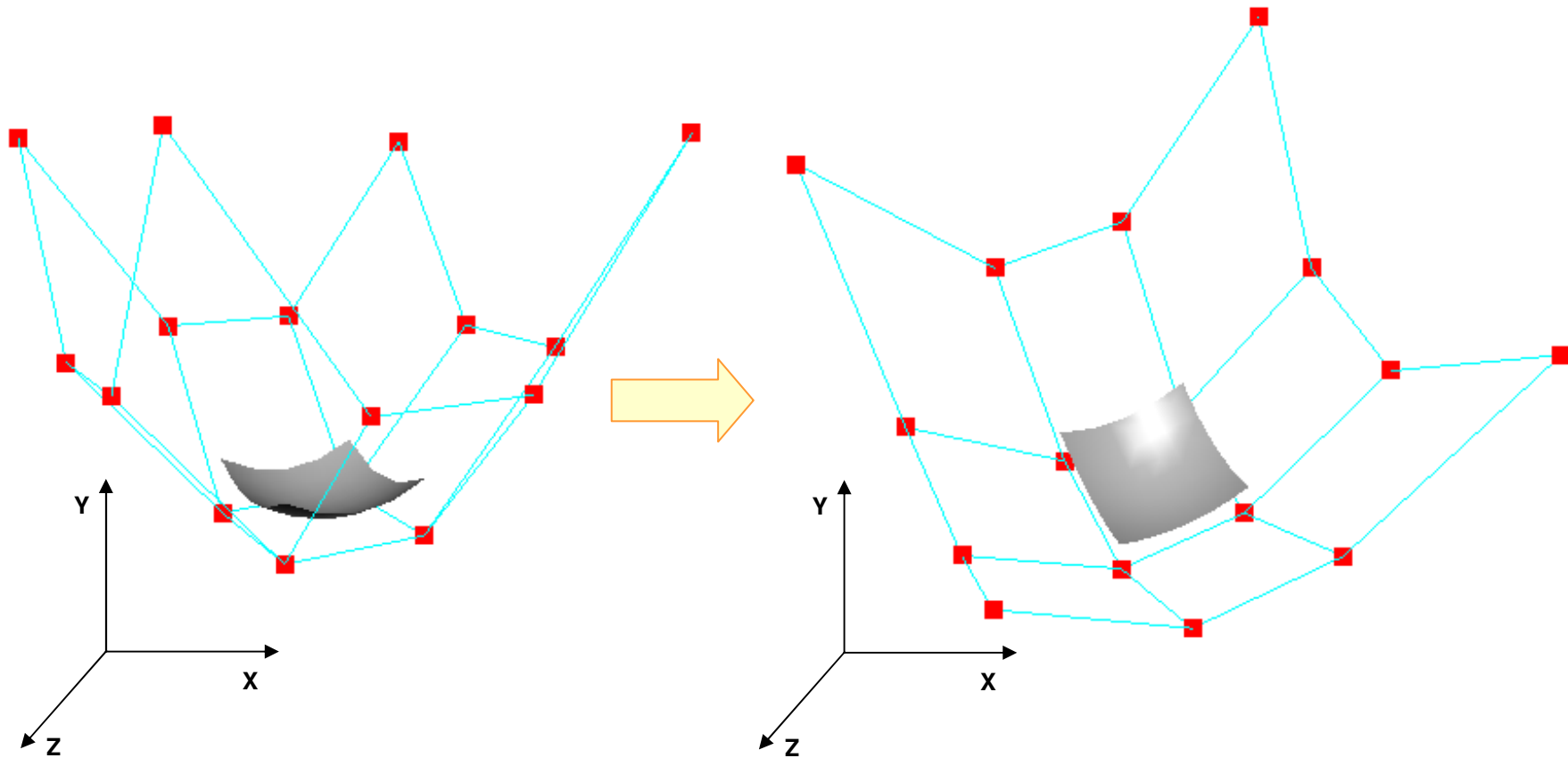
t stride

t order

To be used instead of
`glVertex3d( Q(s,t) )`

TRANSFORMAÇÕES GEOMÉTRICAS DE CURVAS E SUPERFÍCIES

S, **R** ou **T** podem aplicar-se aos coeficientes dos vetores de geometria ou das matrizes de geometria.



(As rotações apenas foram aplicadas aos 16 pontos de controlo e a superfície foi novamente gerada)

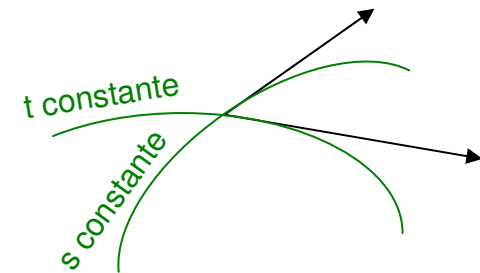
PLANO TANGENTE A UMA SUPERFÍCIE BICÚBICA

Vetor tangente segundo s :

$$\begin{aligned}\frac{\partial}{\partial s} \mathbf{Q}(s,t) &= \frac{\partial}{\partial s} (\mathbf{S} \cdot \mathbf{M} \cdot \underline{\mathbf{G}} \cdot \mathbf{M}^T \cdot \mathbf{T}^T) \\ &= [\ 3s^2 \ 2s \ 1 \ 0 \] \cdot \mathbf{M} \cdot \underline{\mathbf{G}} \cdot \mathbf{M}^T \cdot \mathbf{T}^T\end{aligned}$$

Vetor tangente segundo t :

$$\begin{aligned}\frac{\partial}{\partial t} \mathbf{Q}(s,t) &= \frac{\partial}{\partial t} (\mathbf{S} \cdot \mathbf{M} \cdot \underline{\mathbf{G}} \cdot \mathbf{M}^T \cdot \mathbf{T}^T) \\ &= \mathbf{S} \cdot \mathbf{M} \cdot \underline{\mathbf{G}} \cdot \mathbf{M}^T \cdot \begin{bmatrix} 3t^2 \\ 2t \\ 1 \\ 0 \end{bmatrix}\end{aligned}$$



VETOR NORMAL A UMA SUPERFÍCIE BICÚBICA

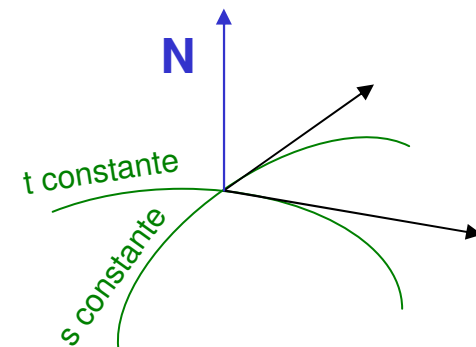
Vetor normal :

$$\frac{\partial}{\partial s} \mathbf{Q}(s,t) \times \frac{\partial}{\partial t} \mathbf{Q}(s,t) = \begin{vmatrix} \bar{i} & \bar{j} & \bar{k} \\ x_s & y_s & z_s \\ x_t & y_t & z_t \end{vmatrix}$$

produto externo

∴

Polinómios de avaliação dispendiosa
(biquintos = 2 variáveis + grau 5)



TEXTURAS

O uso de texturas pode eliminar a necessidade de uma geometria mais pormenorizada



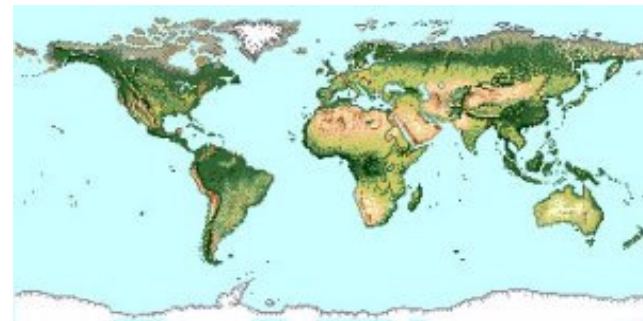
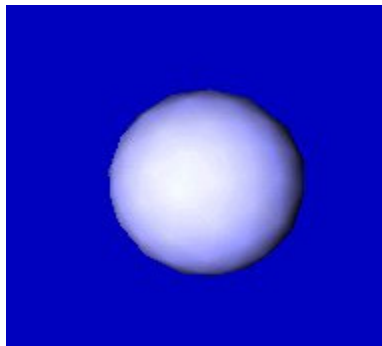
S/ texturas



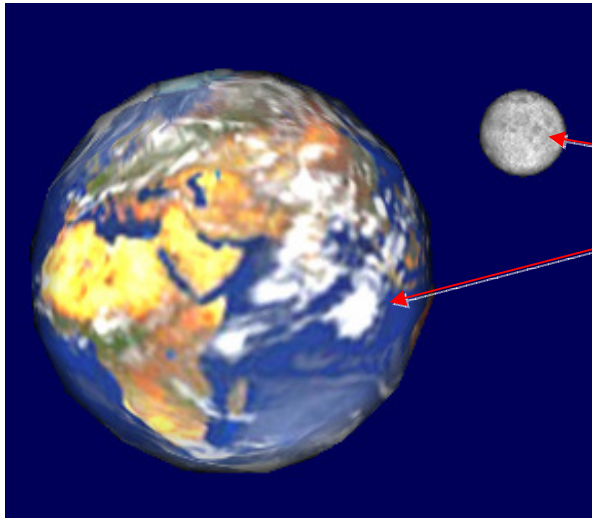
C/ texturas

Mapeamento de Texturas

(TEXTURE MAPPING)



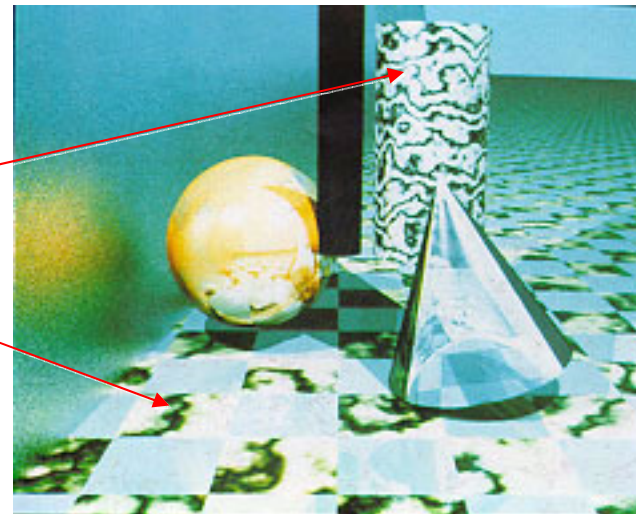
OBTENÇÃO DAS TEXTURAS



Uma textura pode ser armazenada num quadro, quer proveniente de imagem digital...

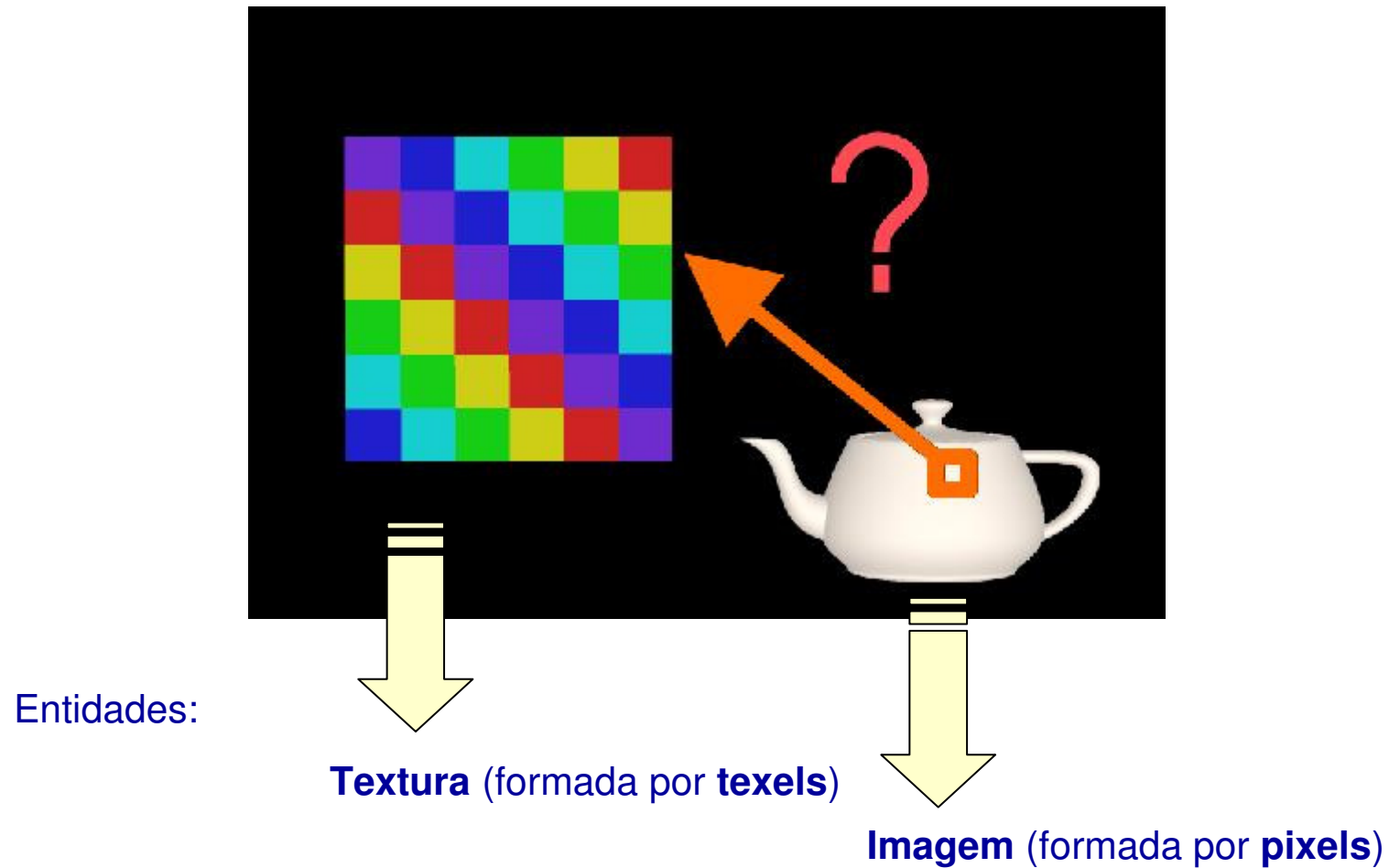
Para se utilizar um quadro como textura, em OpenGL:
`glTexImage2D()`

... quer calculada por um procedimento em tempo de execução



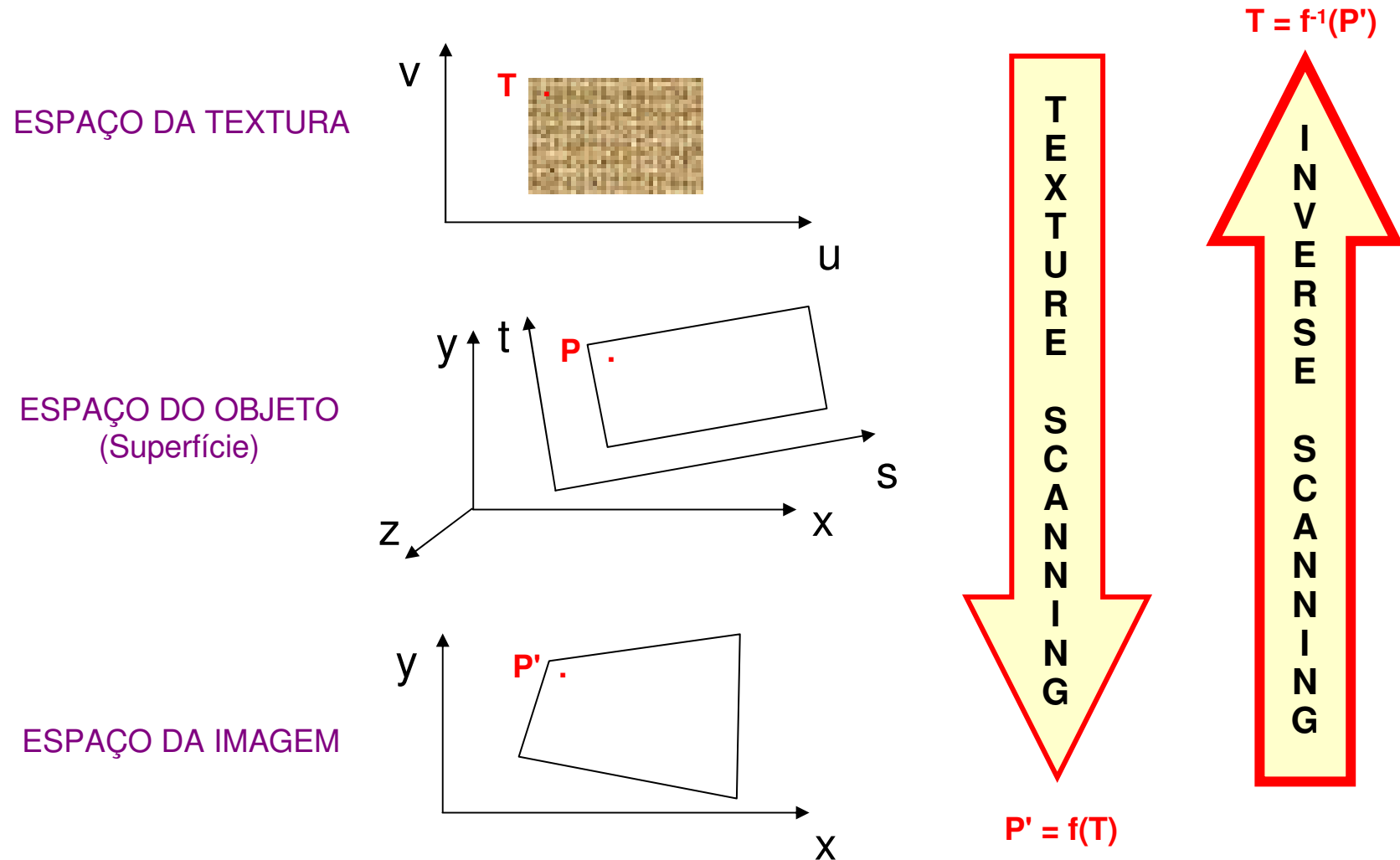
TEXTURAS

O problema da correspondência no mapeamento



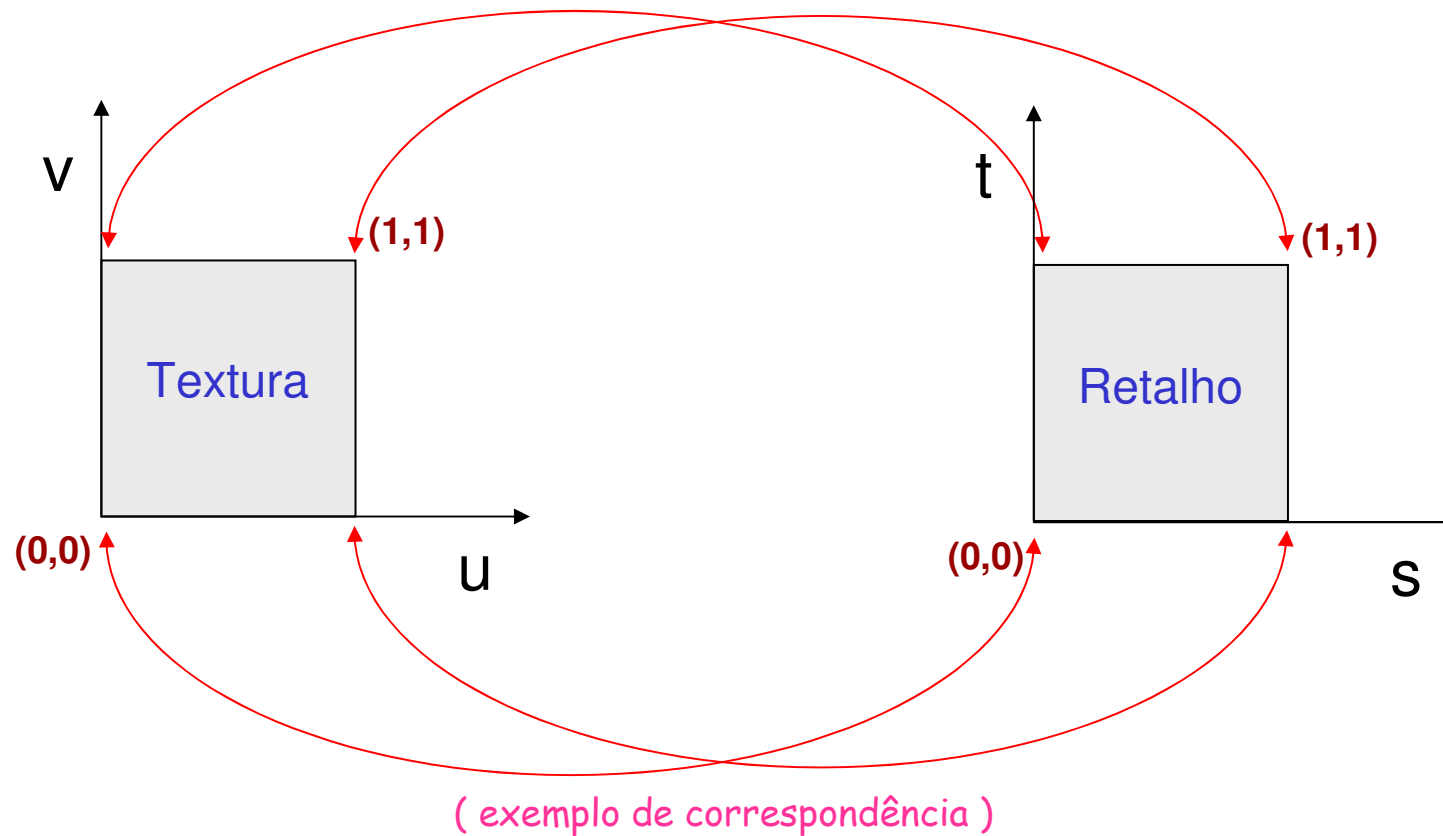
MAPEAMENTO DE UMA TEXTURA

Tratamento Matemático



scanning $\equiv$ perscrutação

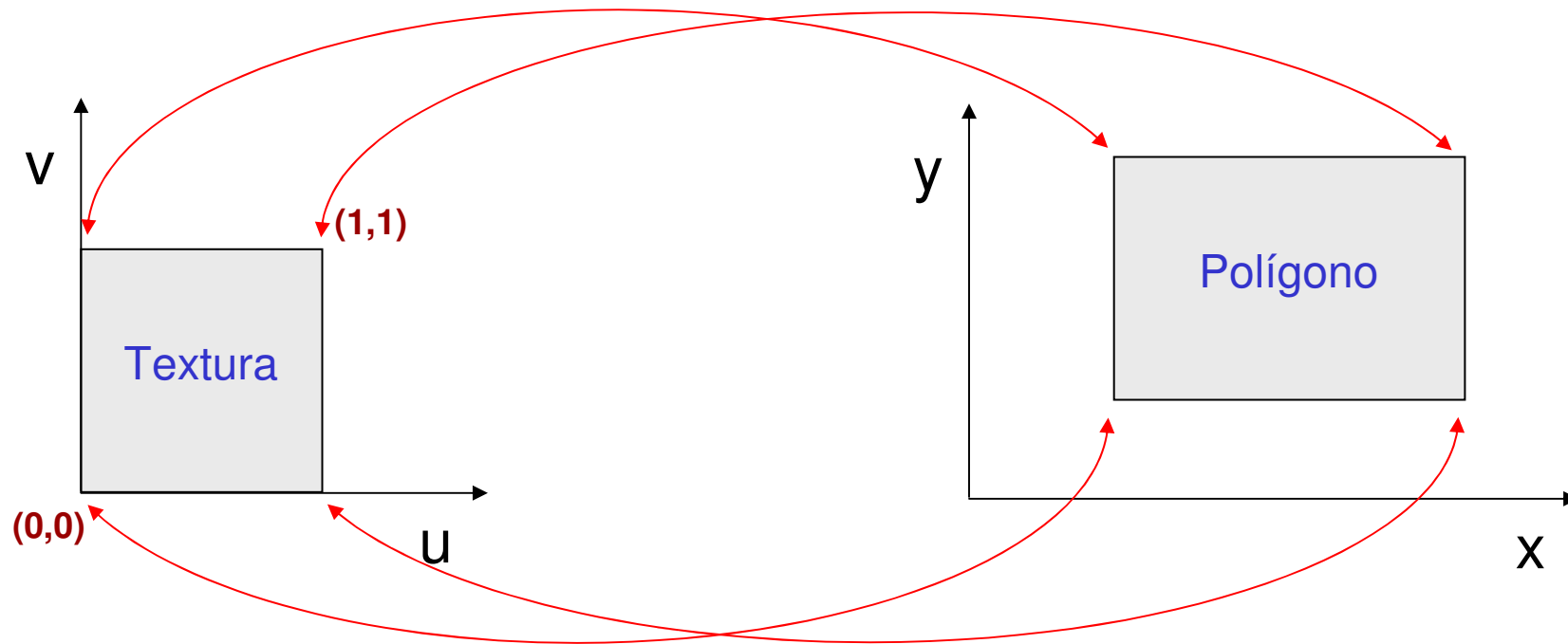
SUPERFÍCIE DEFINIDA PARAMETRICAMENTE



É habitual fazer-se com que este tipo de correspondência seja uma interpolação linear:

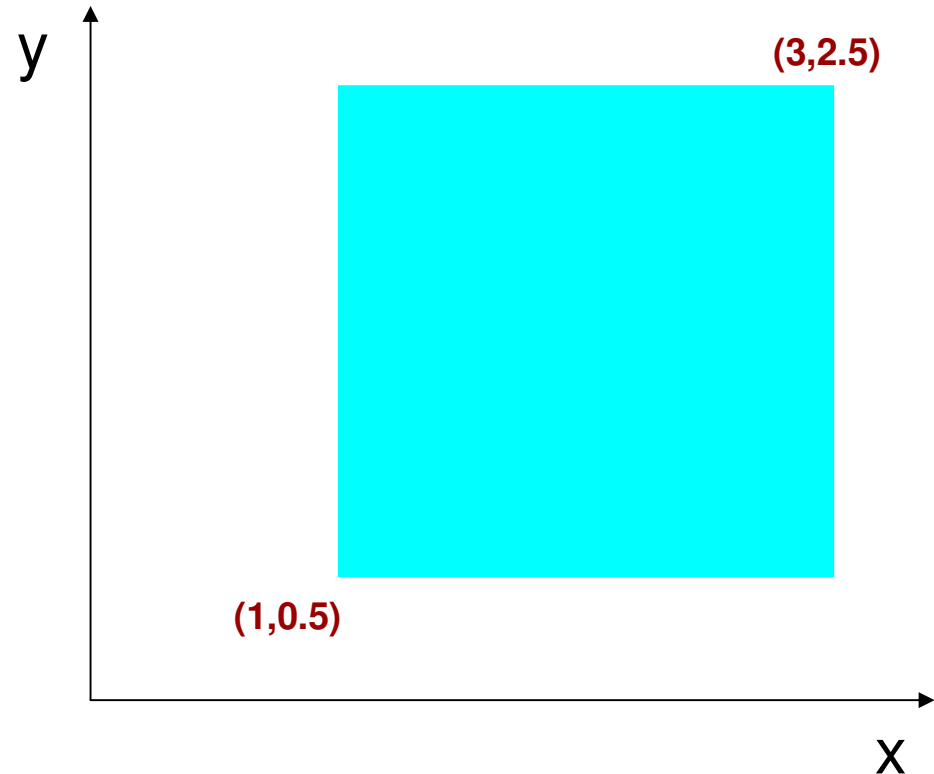
$$\begin{bmatrix} s \\ t \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

SUPERFÍCIE DEFINIDA POR UM POLÍGONO



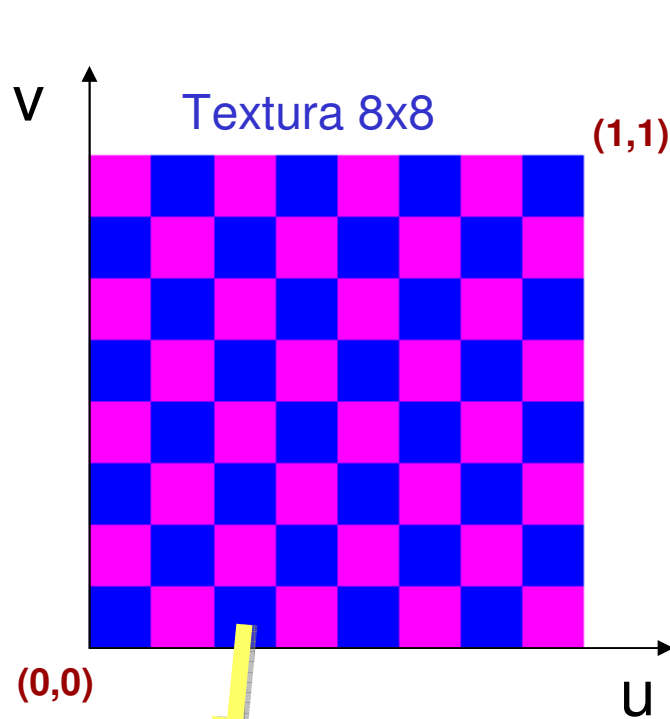
Também é habitual que a correspondência seja uma interpolação linear
(estar sempre ciente das possíveis distorções!)

Exemplos com OpenGL



```
glColor3d(0.0, 1.0, 1.0);  
glBegin(GL_QUADS);  
    glVertex3d(1.0, 0.5, 0.0);  
    glVertex3d(1.0, 2.5, 0.0);  
    glVertex3d(3.0, 2.5, 0.0);  
    glVertex3d(3.0, 0.5, 0.0);  
glEnd();
```

Nos exemplos seguintes, com texturas,
presume-se já existir antes a instrução
`glTexEnvf(GL_TEXTURE_ENV,
 GL_TEXTURE_ENV_MODE,
 GL_DECAL)`
para que cada pixel seja pintado apenas
com a cor da textura.

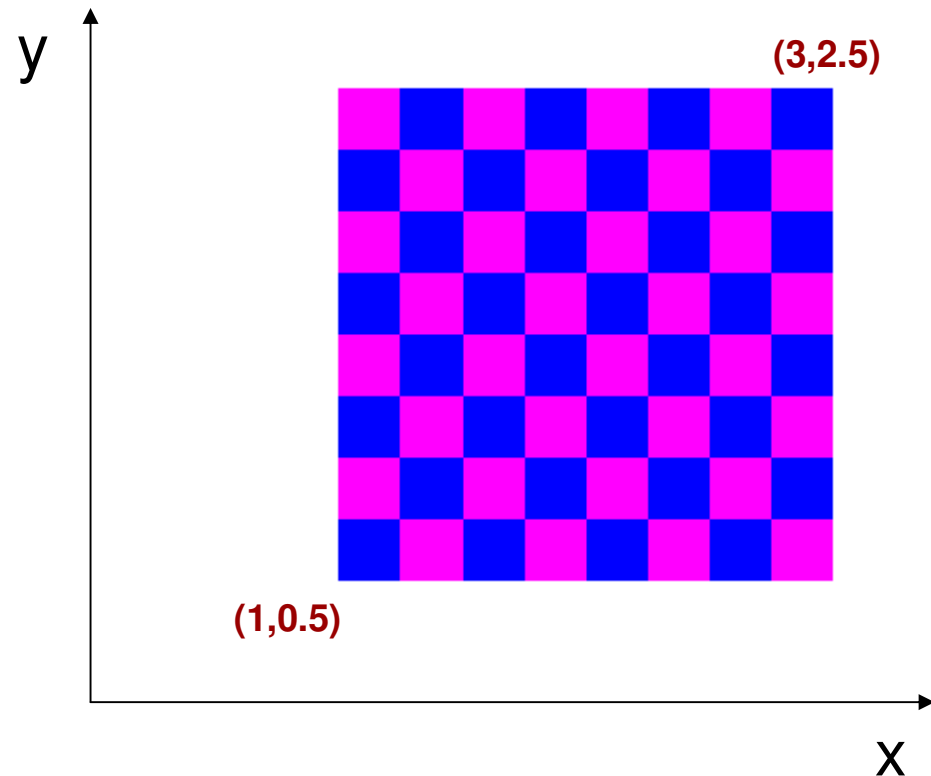


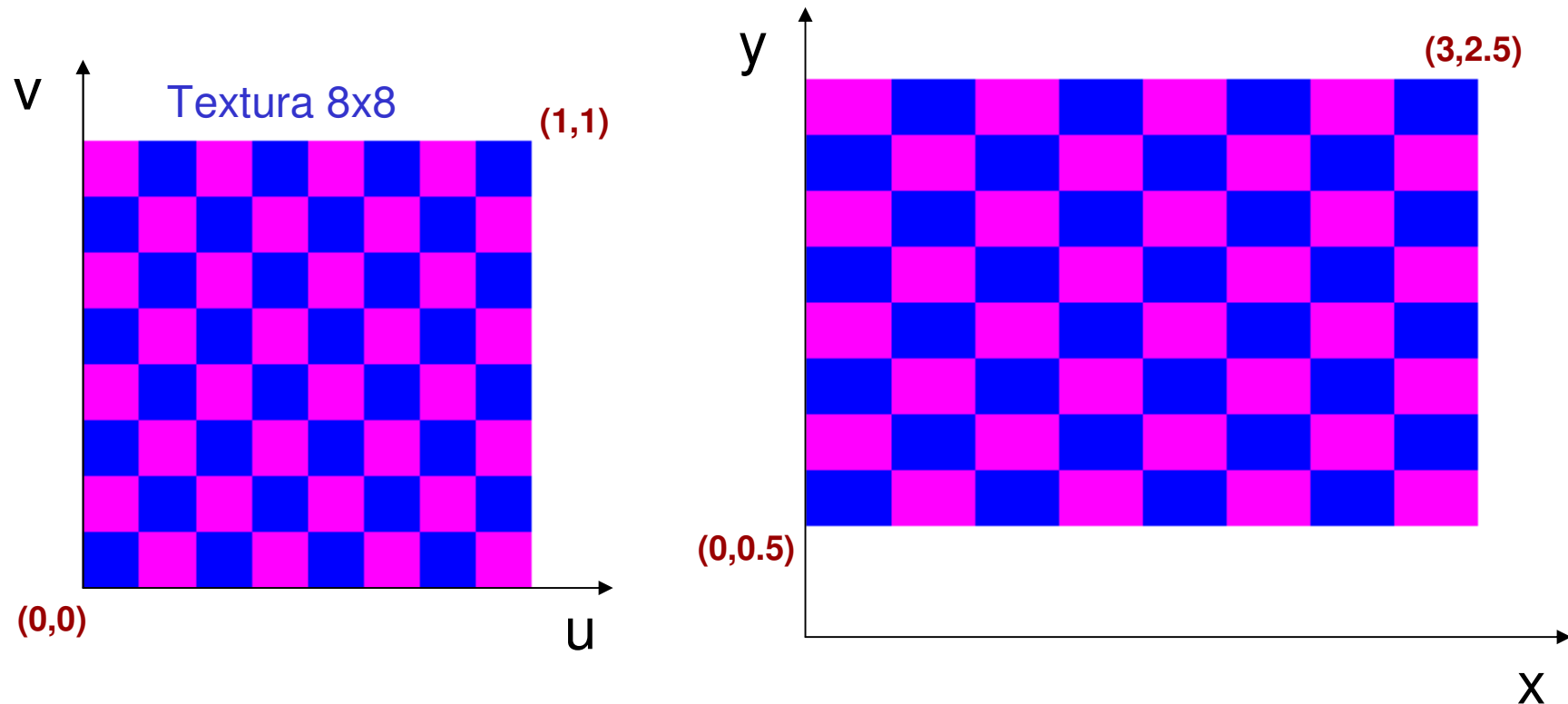
Definiu-se cada quadrícula como um único texel:

```
GLubyte textura[8][8][3];
```

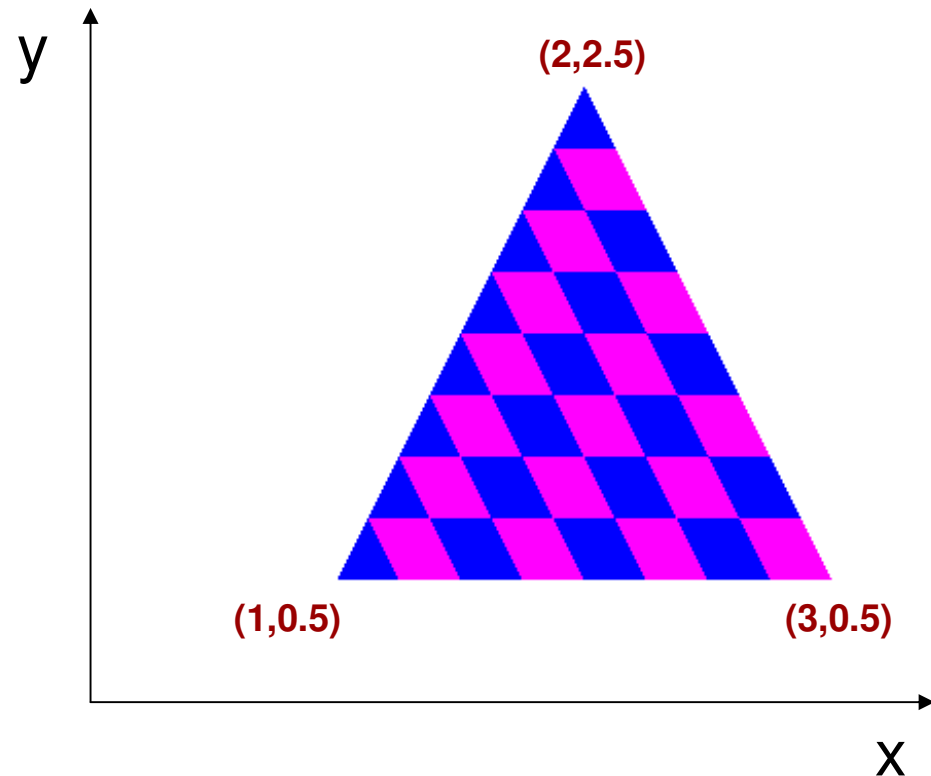
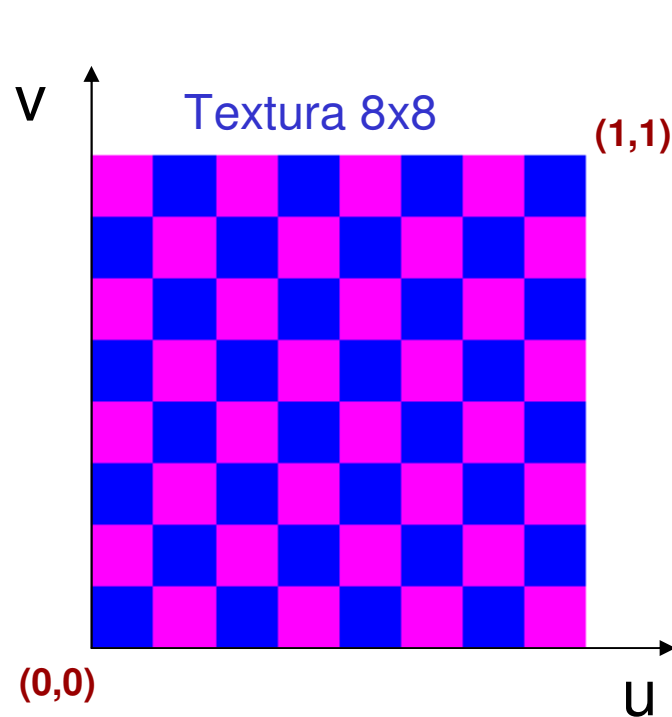
Cor RGB 24 bits

```
glBegin(GL_QUADS);
    glTexCoord2d(0.0, 0.0); glVertex3d(1.0, 0.5, 0.0);
    glTexCoord2d(0.0, 1.0); glVertex3d(1.0, 2.5, 0.0);
    glTexCoord2d(1.0, 1.0); glVertex3d(3.0, 2.5, 0.0);
    glTexCoord2d(1.0, 0.0); glVertex3d(3.0, 0.5, 0.0);
glEnd();
```



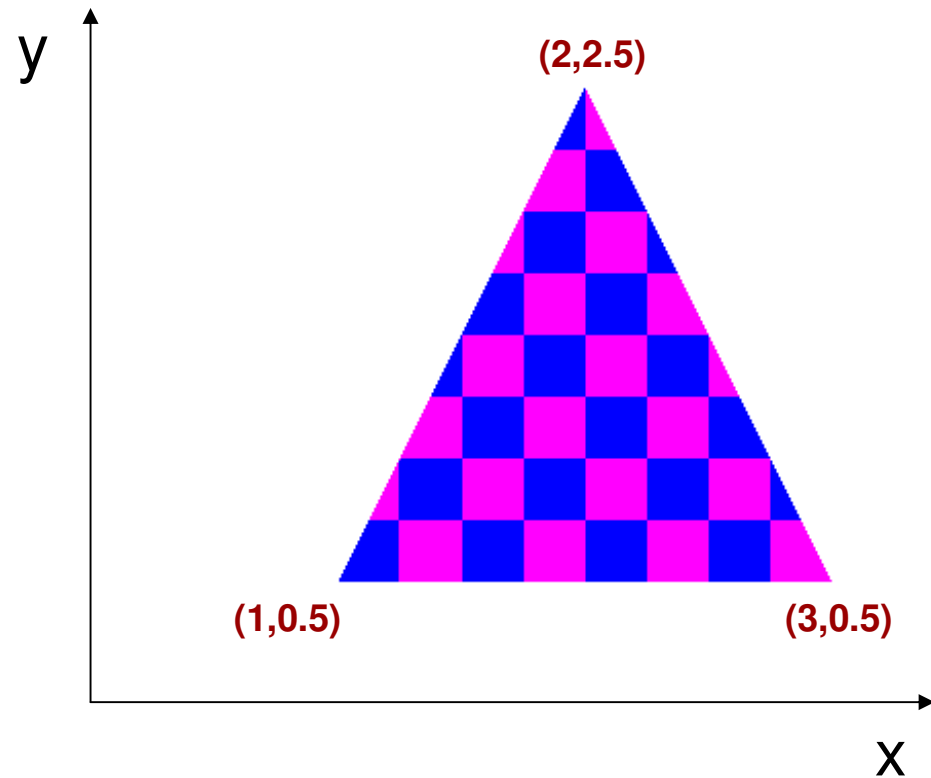
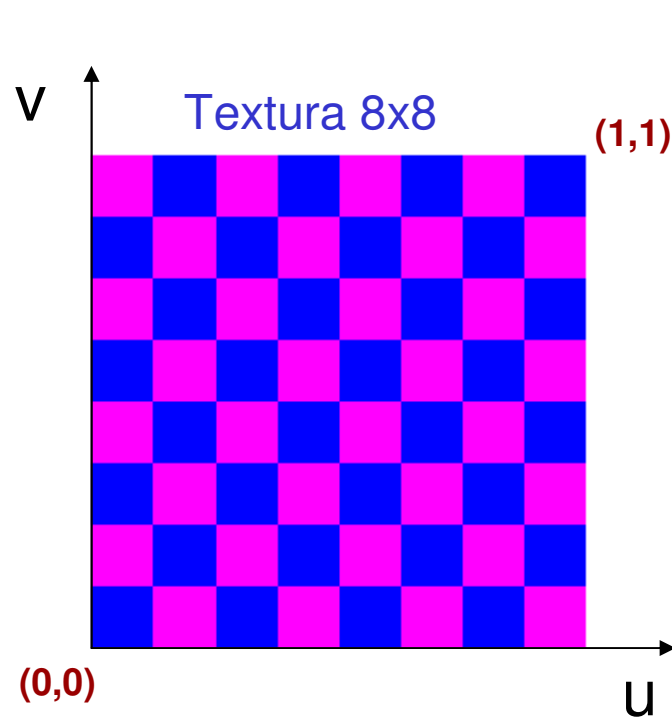


```
glBegin(GL_QUADS);
    glTexCoord2d(0.0, 0.0); glVertex3d(0.0, 0.5, 0.0);
    glTexCoord2d(0.0, 1.0); glVertex3d(0.0, 2.5, 0.0);
    glTexCoord2d(1.0, 1.0); glVertex3d(3.0, 2.5, 0.0);
    glTexCoord2d(1.0, 0.0); glVertex3d(3.0, 0.5, 0.0);
glEnd();
```

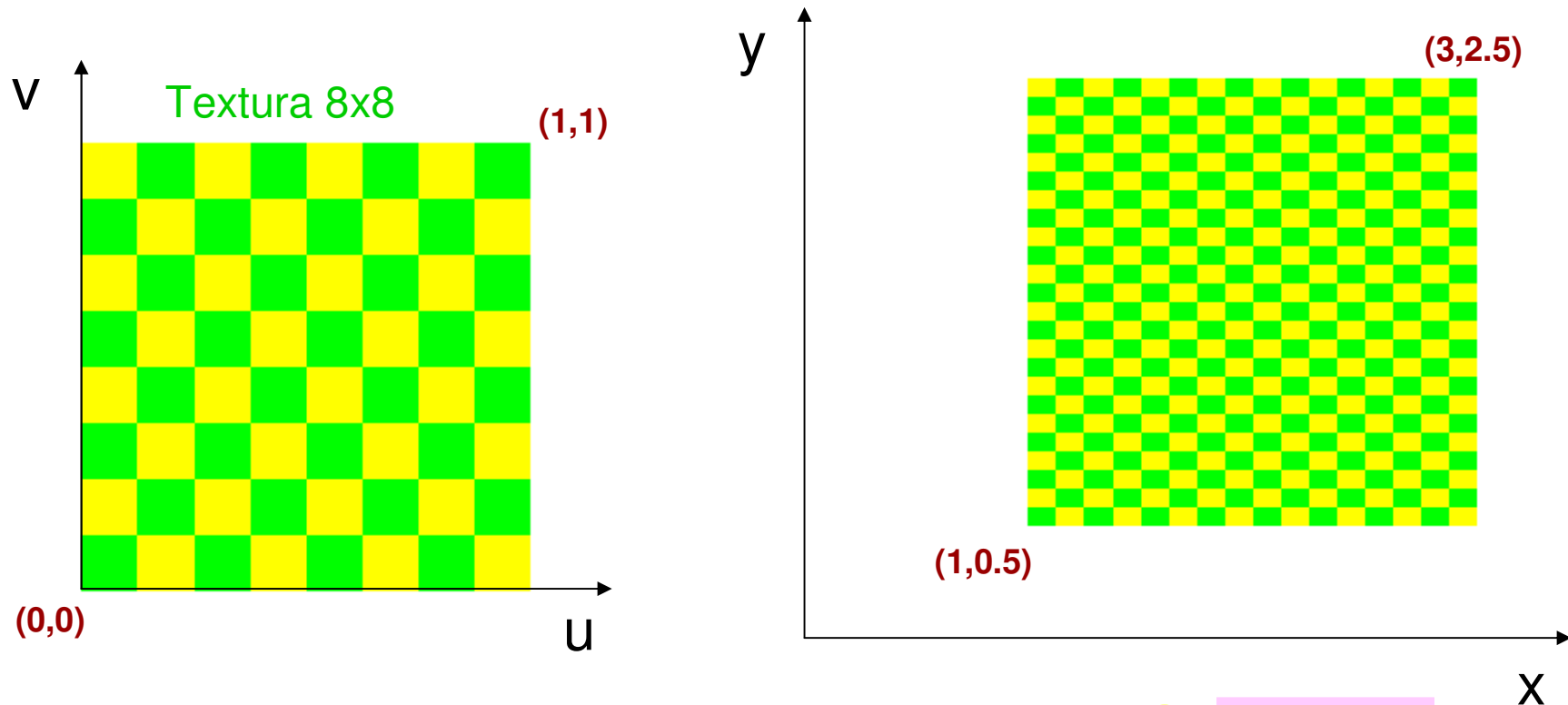



ATENÇÃO!

```
glBegin(GL_TRIANGLES);  
    glTexCoord2d(0.0, 0.0); glVertex3d(1.0, 0.5, 0.0);  
    glTexCoord2d(1.0, 1.0); glVertex3d(2.0, 2.5, 0.0);  
    glTexCoord2d(1.0, 0.0); glVertex3d(3.0, 0.5, 0.0);  
glEnd();
```



```
glBegin(GL_TRIANGLES);
    glTexCoord2d(0.0, 0.0); glVertex3d(1.0, 0.5, 0.0);
    glTexCoord2d(0.5, 1.0); glVertex3d(2.0, 2.5, 0.0);
    glTexCoord2d(1.0, 0.0); glVertex3d(3.0, 0.5, 0.0);
glEnd();
```

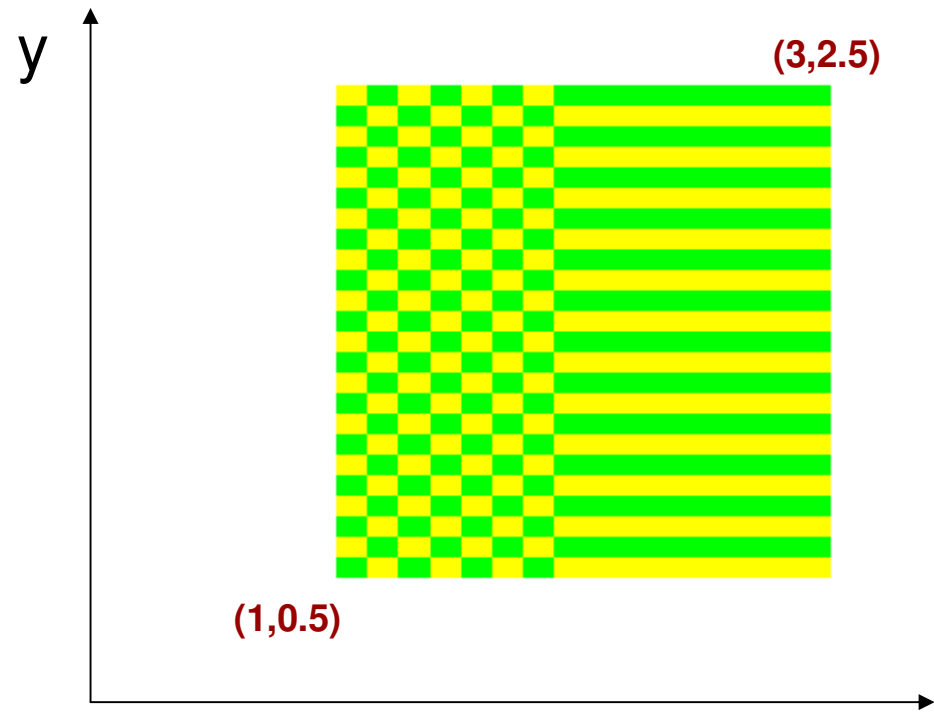
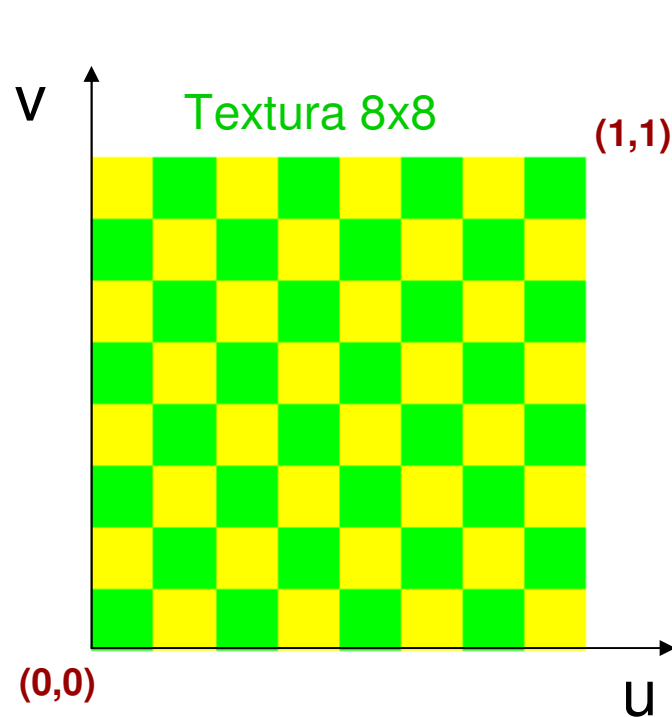


```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

```
glBegin(GL_QUADS);
    glTexCoord2d(0.0, 0.0); glVertex3d(1.0, 0.5, 0.0);
    glTexCoord2d(0.0, 3.0); glVertex3d(1.0, 2.5, 0.0);
    glTexCoord2d(2.0, 3.0); glVertex3d(3.0, 2.5, 0.0);
    glTexCoord2d(2.0, 0.0); glVertex3d(3.0, 0.5, 0.0);
glEnd();
```

Notação
das coordenadas
de textura em
OpenGL !

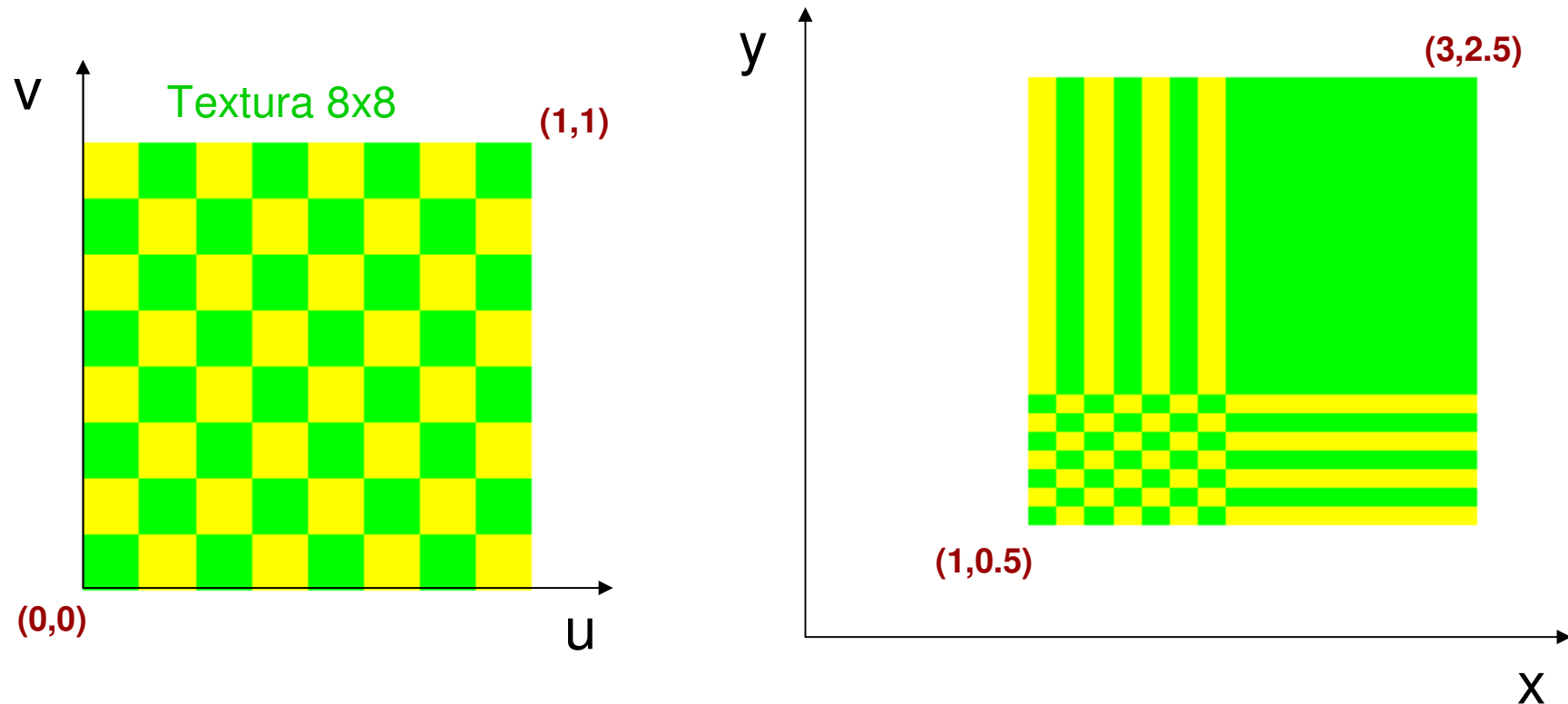
Repetição da textura se o parâmetro cair fora do intervalo



```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

```
glBegin(GL_QUADS);
    glTexCoord2d(0.0, 0.0); glVertex3d(1.0, 0.5, 0.0);
    glTexCoord2d(0.0, 3.0); glVertex3d(1.0, 2.5, 0.0);
    glTexCoord2d(2.0, 3.0); glVertex3d(3.0, 2.5, 0.0);
    glTexCoord2d(2.0, 0.0); glVertex3d(3.0, 0.5, 0.0);
glEnd();
```

$u > 1 \rightarrow u = 1$
 $u < 0 \rightarrow u = 0$



```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```

```
glBegin(GL_QUADS);
    glTexCoord2d(0.0, 0.0); glVertex3d(1.0, 0.5, 0.0);
    glTexCoord2d(0.0, 3.0); glVertex3d(1.0, 2.5, 0.0);
    glTexCoord2d(2.0, 3.0); glVertex3d(3.0, 2.5, 0.0);
    glTexCoord2d(2.0, 0.0); glVertex3d(3.0, 0.5, 0.0);
glEnd();
```

COMPARAÇÃO DOS MÉTODOS RELATIVOS À ORDEM DE EXECUÇÃO

TEXTURE SCANNING (perscrutação da textura)

- ☹️ Em geral a textura não coincide com um número inteiro de pixels, implicando cálculos para a subdivisão da área de um pixel.
- 😊 As transformações de visualização seguem-se no sentido mais natural.

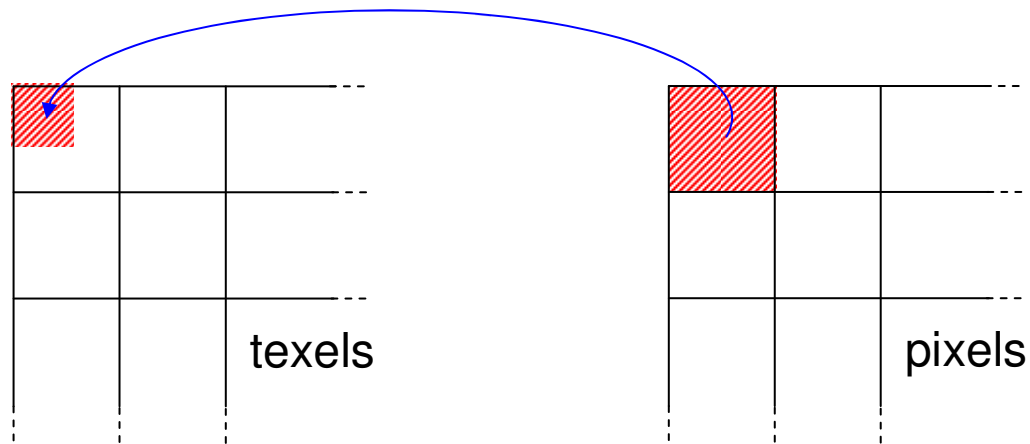
INVERSE SCANNING (perscrutação inversa)

- ☹️ Exige o cálculo das transformações de visualização inversas.
- 😊 Evita a subdivisão de pixels e permite, além disso, o uso de filtros para *antialiasing*.

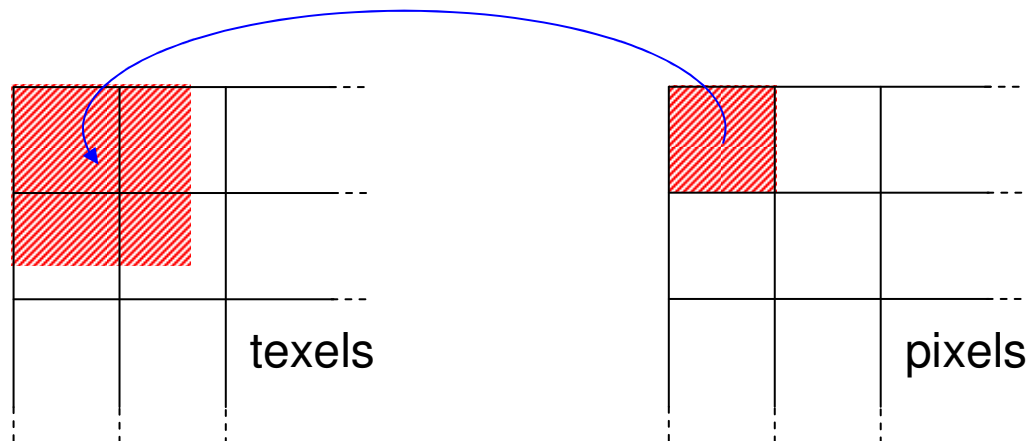
Conclusão: o método usualmente preferido é o de INVERSE SCANNING.

CORRESPONDÊNCIA ENTRE PIXELS E TEXTELS (1)

Um pixel tem uma determinada área. Quando se faz a correspondência com a textura, a área aí obtida poderá ser diferente da área de um texel.



Pixel pintado com **ampliação**
(MAGNIFICATION) da textura



Pixel pintado com **redução**
(MINIFICATION) da textura

Nestes casos, qual o valor com que se deve pintar o pixel ?

Obs.: Havendo distorção, a área na textura não será um quadrado.

CORRESPONDÊNCIA ENTRE PIXELS E TEXTELS (2)

Soluções em alternativa (que se podem misturar), tanto para **ampliação** como para **redução**:

Em OpenGL:

- Usar-se o texel mais próximo do ponto calculado pela correspondência com o (centro do) pixel.

```
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MAG_FILTER, GL_NEAREST);  
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

- Fazer-se uma média ponderada com os 2x2 texels mais próximos do ponto calculado pela correspondência com o pixel (**filtragem**).

```
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```


CORRESPONDÊNCIA ENTRE PIXELS E TEXTELS (3)

EXEMPLOS DE RESULTADOS DE APLICAÇÃO

64x64 texels



360x360 pixels

GL_NEAREST

GL_LINEAR

mais lento,
melhor resultado

Texture Mapping in OpenGL

Example in C to be used together with a vertex evaluator

Brief summary:

```
glMap2d(GL_MAP2_TEXTURE_COORD_2, 0.0, 1.0, 2, 2,  
        0.0, 1.0, 4, 2, texture_points);
```

```
glEnable(GL_MAP2_TEXTURE_COORD_2);
```

```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
```

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, filter_value);
```

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, filter_value);
```

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image_w, image_h,  
             0, GL_RGB, GL_UNSIGNED_BYTE, image);
```

```
glEnable(GL_TEXTURE_2D);
```

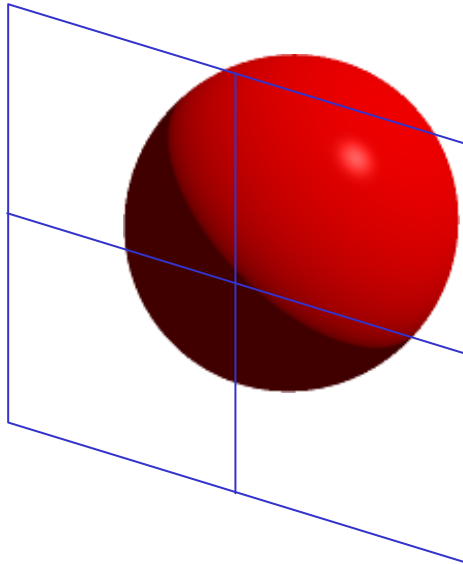
Example:
{ 0,1, 1,1,
 0,0, 1,0 }

4 corners
(texture coordinates)

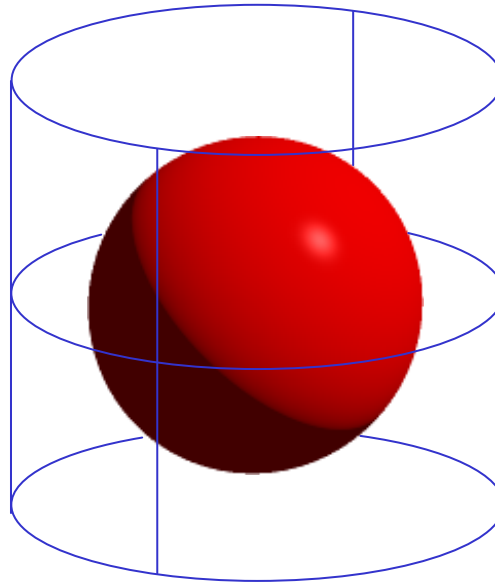
A byte buffer

TEXTURAS MAPEADAS

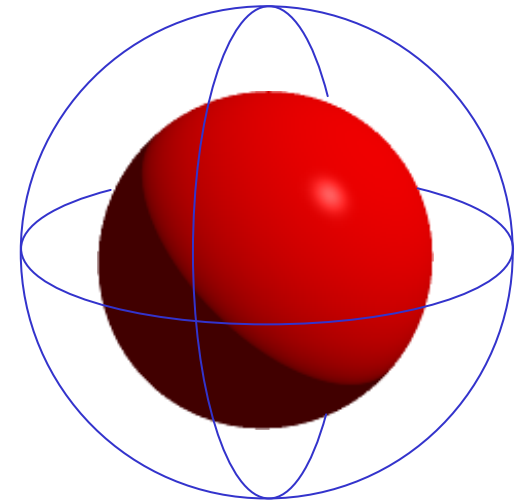
TIPOS CLÁSSICOS DE MAPEAMENTO:



Ortogonal



Cilíndrico



Esférico

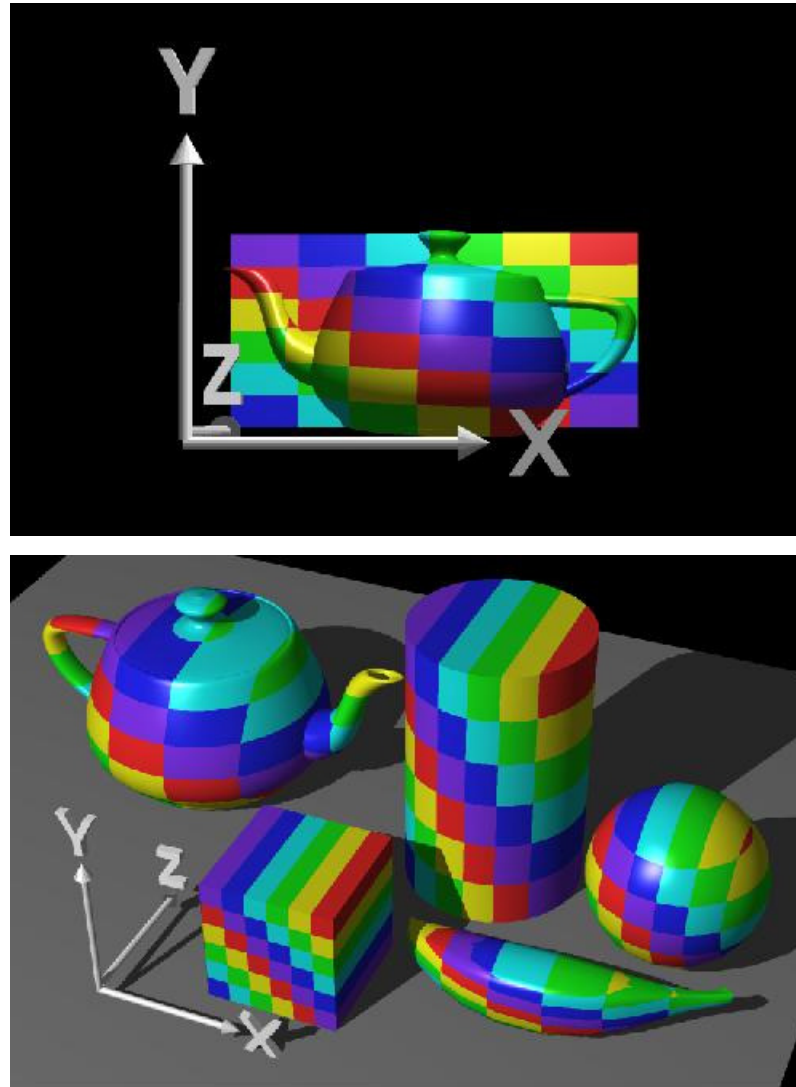
Os slides seguintes, num referencial segundo a regra da mão esquerda, são da autoria de



Rosalee Wolfe
SIGGRAPH 97 Education Slide set

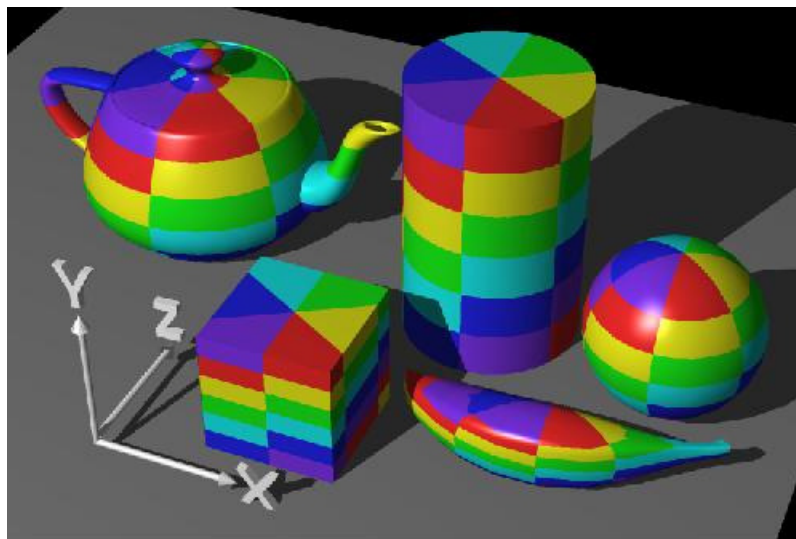
M. Próspero

Mapeamento Ortogonal



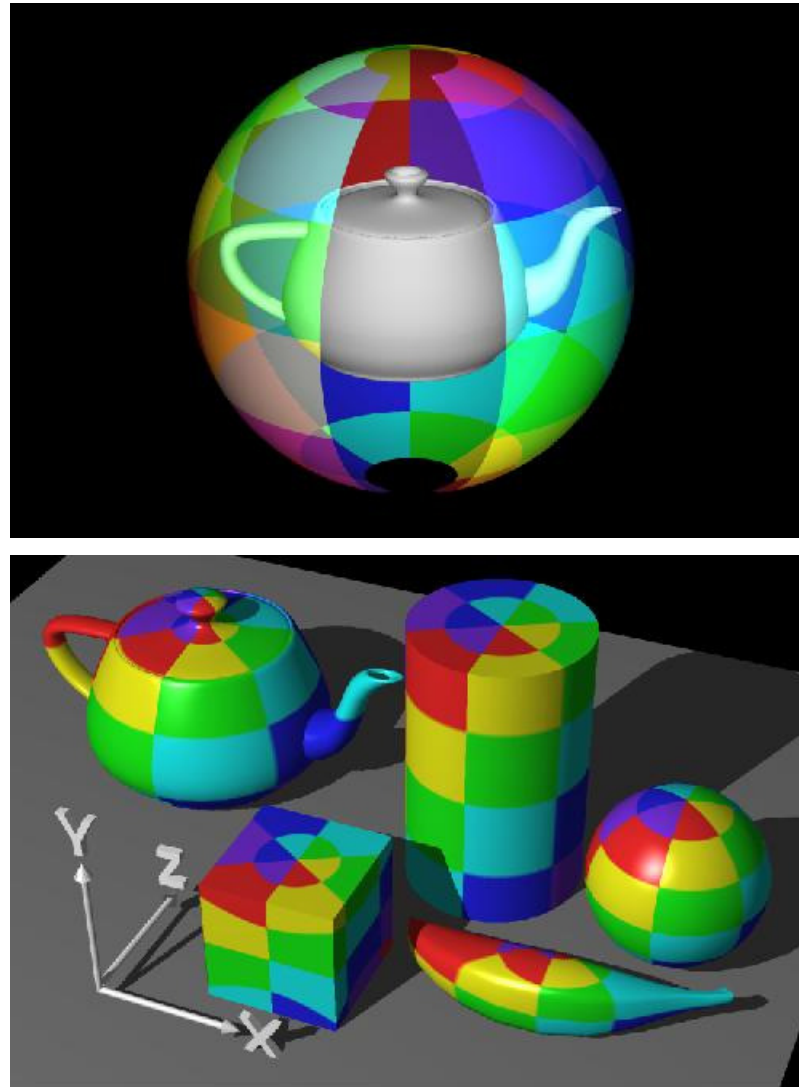
Qual a correspondência entre o espaço da textura e o espaço do objeto ?

Mapeamento Cilíndrico



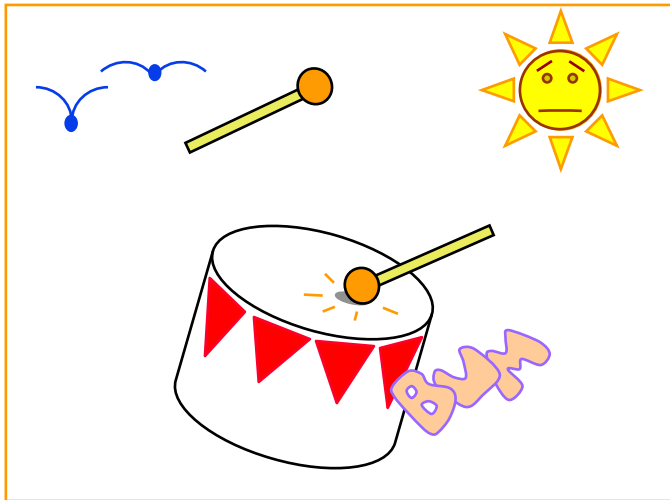
Qual a correspondência entre o espaço da textura e o espaço do objeto ?

Mapeamento Esférico



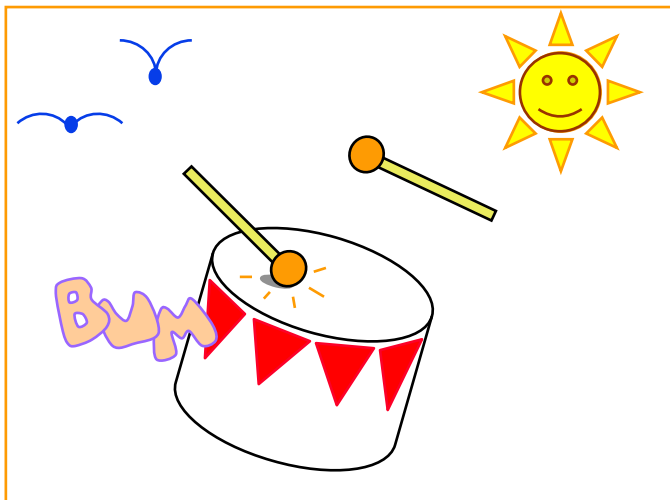
Qual a correspondência entre o espaço da textura e o espaço do objeto ?

O PRINCÍPIO DA ANIMAÇÃO

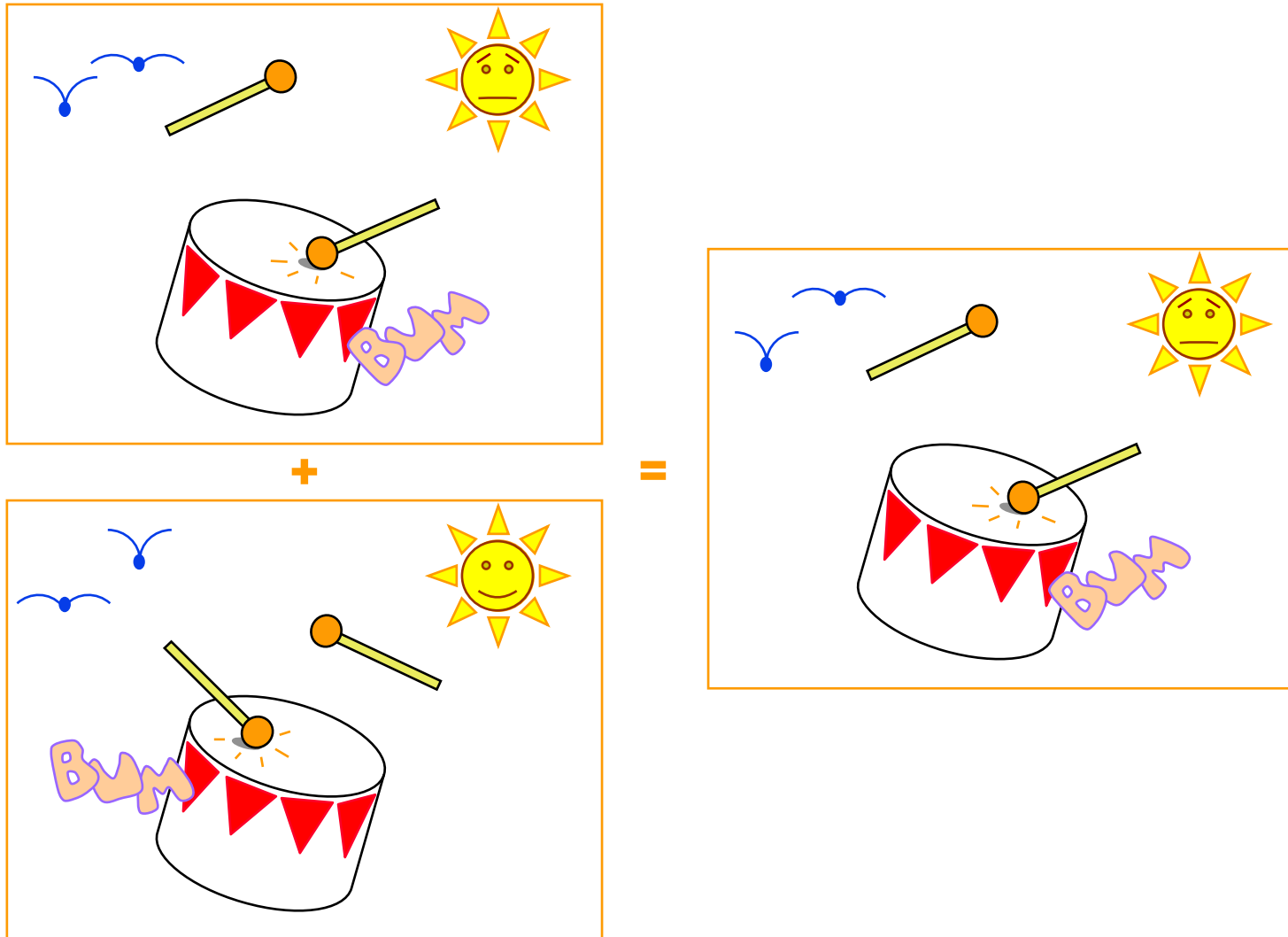


+

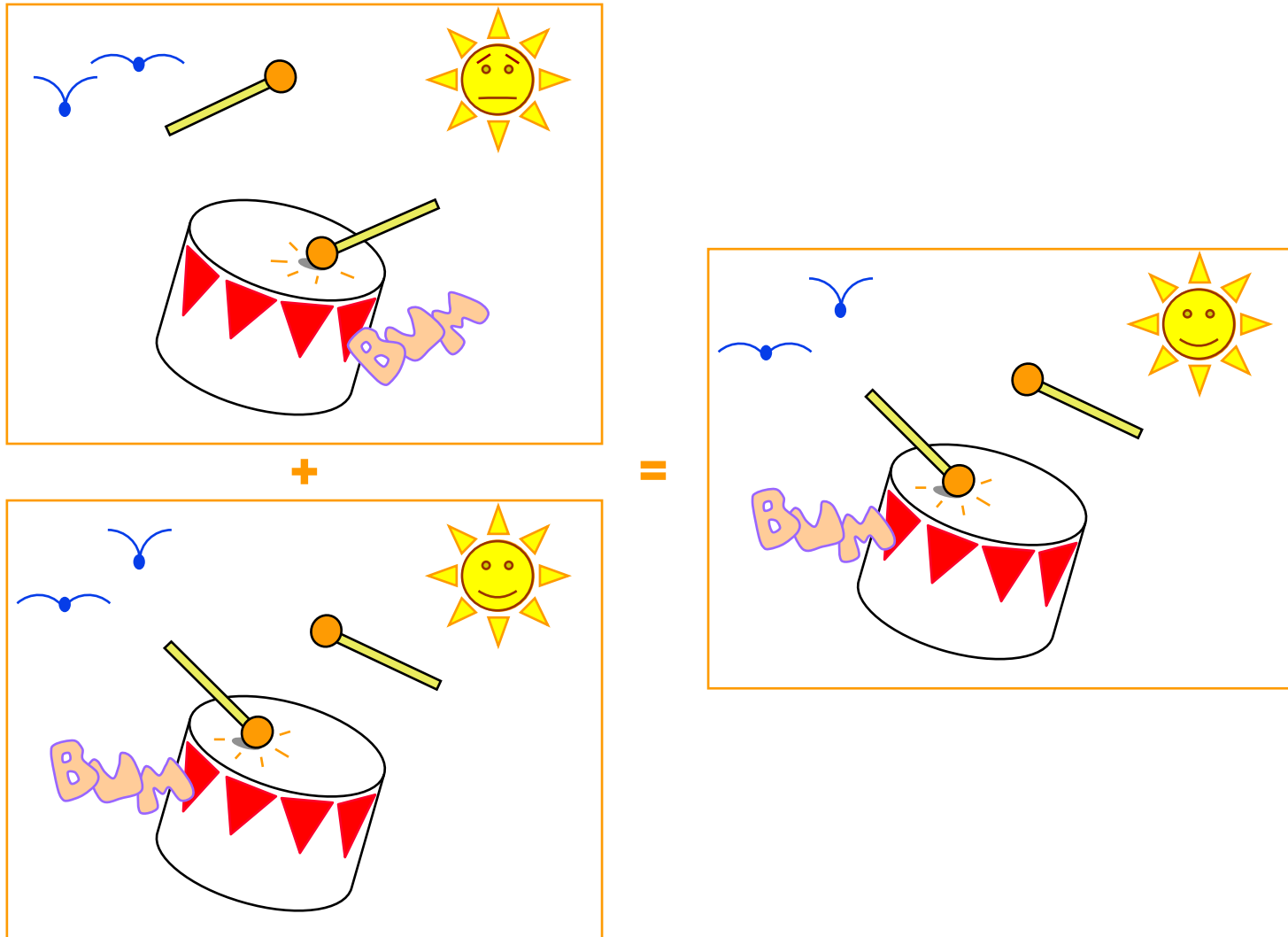
=

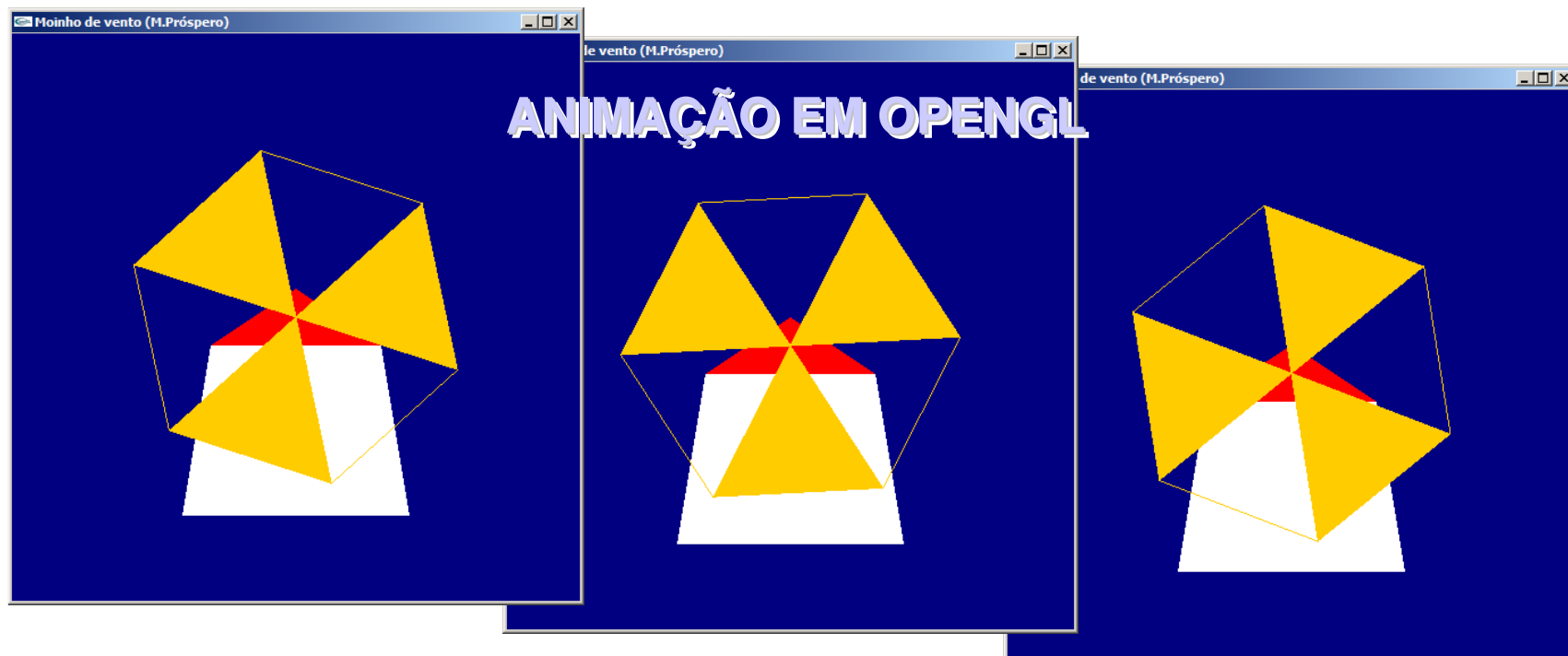


O PRINCÍPIO DA ANIMAÇÃO



O PRINCÍPIO DA ANIMAÇÃO





É necessário executar o método `display()` em ciclo, tendo-se o cuidado de ir alterando o(s) parâmetro(s) para a animação:

```
...  
    glRotated(alfa, 0.0, 0.0, 1.0);  
...  
    alfa += 1.0; // valor para a próxima chamada de display()  
    if (alfa > 360.0) alfa -= 360.0;  
...
```

ANIMATION IN JOGL

com.sun.opengl.util.Animator

com.sun.opengl.util.FPSAnimator

An Animator (or FPSAnimator) can be attached to one or more GLAutoDrawables to drive their `display()` methods in a loop.

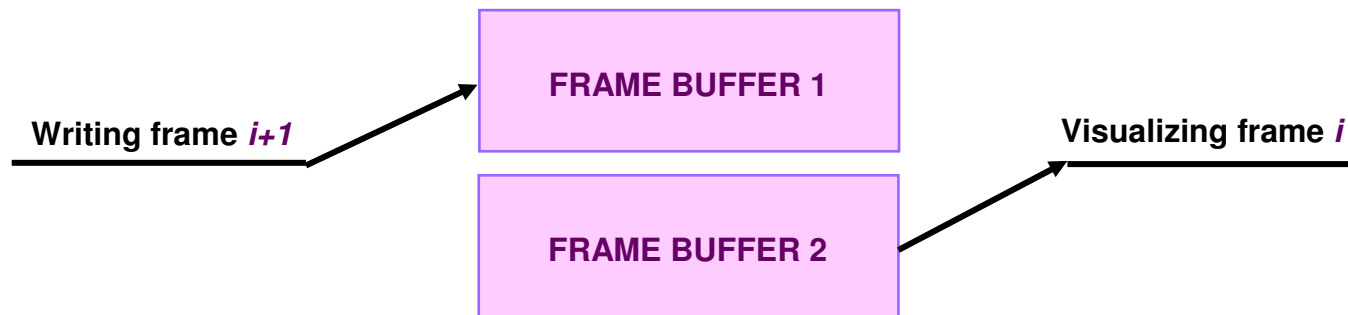
Frames Per Second, if supported by the hardware

```
...  
    final FPSAnimator animator = new FPSAnimator(canvas, 60);  
    animator.start(); // Start executing display() in a loop  
...  
    animator.stop(); // Stop animation  
...
```

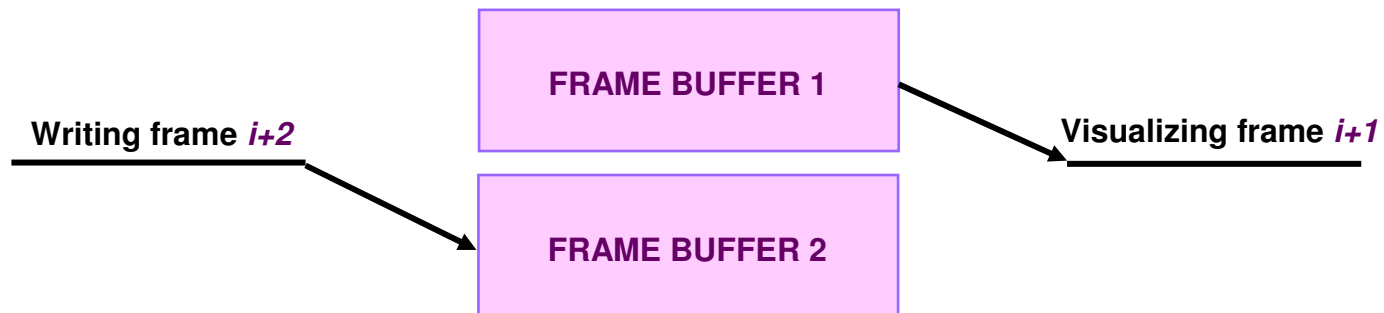
DOUBLE BUFFERING

(Frame Buffer duplo, usado em animação)

Instant t :



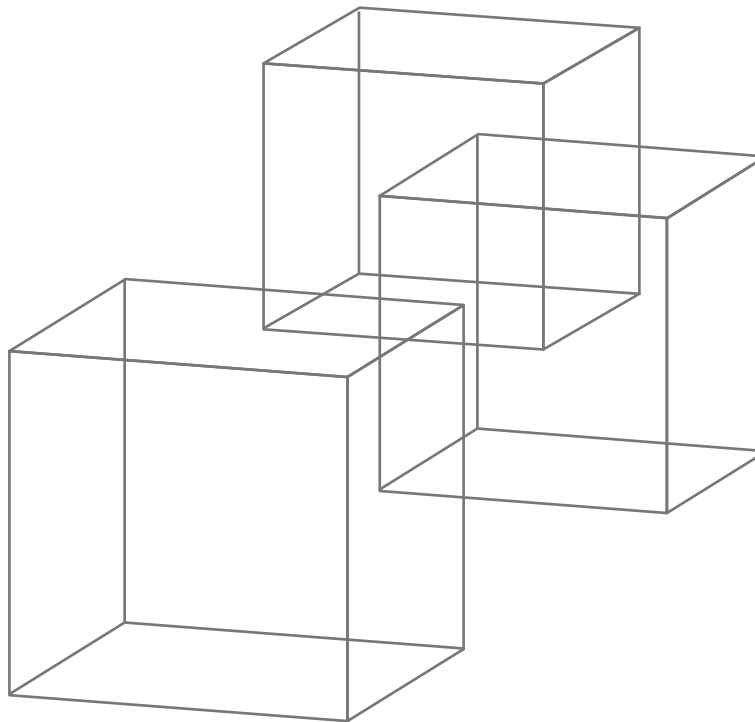
Instant $t+1$:



Default in JOGL: Double Buffer "on"

`setSwapInterval(1)` causes the wait for the next vertical refresh until swapping buffers

Necessidade de Algoritmos para Hidden-Line Hidden-Surface Removal*

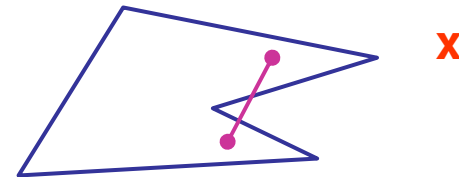
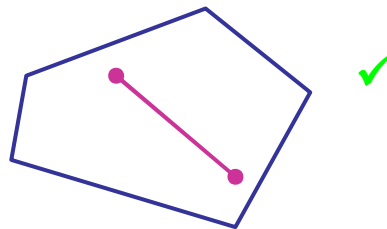


* Remoção de partes ocultas (linhas e/ou superfícies)

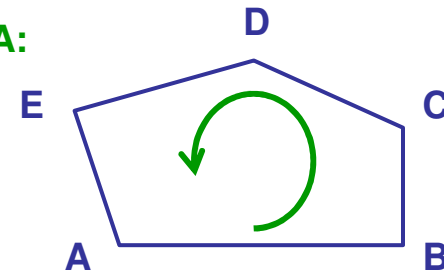
CONSIDERAÇÕES PRELIMINARES (1)

POLÍGONO CONVEXO:

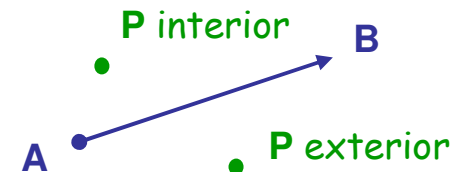
Aquele em que uma linha unindo dois quaisquer pontos interiores ao polígono esteja nele totalmente contida.



POLÍGONO (CONVEXO) COM ORIENTAÇÃO POSITIVA:

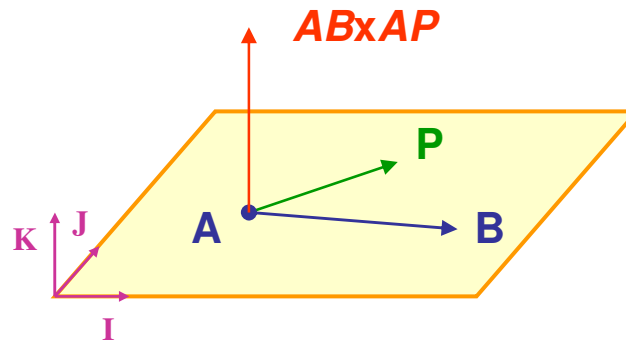


Um ponto interior a um polígono convexo com orientação positiva fica à esquerda de todas as arestas do mesmo.



CONSIDERAÇÕES PRELIMINARES (2)

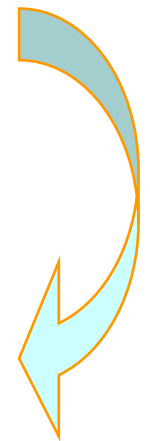
Determinação da posição de um ponto via produto externo de dois vectores:



$$AB \times AP = \begin{vmatrix} \mathbf{I} & \mathbf{J} & \mathbf{K} \\ (x_B - x_A) & (y_B - y_A) & 0 \\ (x_P - x_A) & (y_P - y_A) & 0 \end{vmatrix} = ((x_B - x_A)(y_P - y_A) - (x_P - x_A)(y_B - y_A)) \mathbf{K} = k \mathbf{K}$$

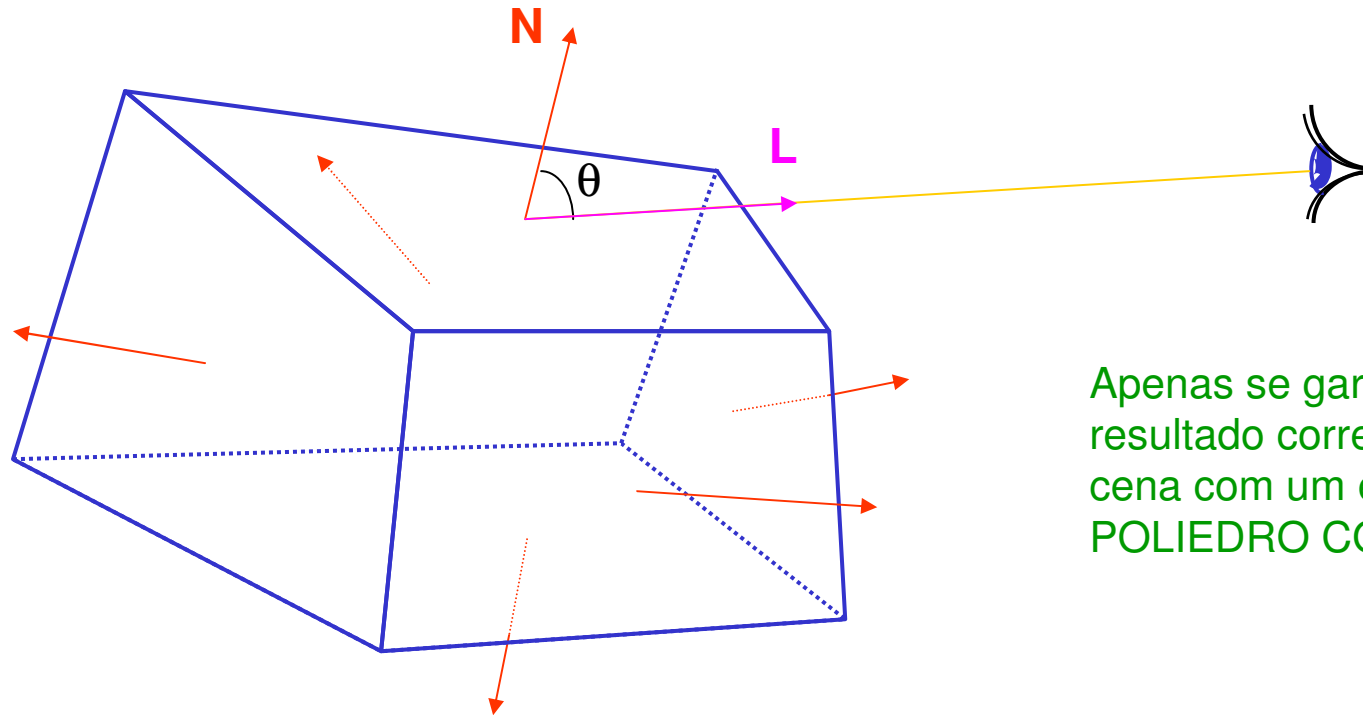
$k > 0 \Rightarrow \mathbf{P}$ à esquerda de AB

$k < 0 \Rightarrow \mathbf{P}$ à direita de AB



HLHSR

MÉTODO DO PRODUTO INTERNO (*Test for Culling Polygon Faces*)



Apenas se garante um
resultado correcto numa
cena com um e um só
POLIEDRO CONVEXO

A superfície não é visível $\longrightarrow 90^\circ \leq \theta \leq 180^\circ$

A superfície é visível $\longrightarrow 0^\circ \leq \theta < 90^\circ \longrightarrow 0 < \cos \theta \leq 1$

$$\cos \theta = \frac{L \cdot N}{|L| |N|}$$

em que

$$L \cdot N = x_L x_N + y_L y_N + z_L z_N$$

Cálculo dos vetores normais:

Polígonos com orientação positiva quando observados do exterior do poliedro

Tomando 3 vértices consecutivos:

$$N = (V_2 - V_1) \times (V_3 - V_2)$$

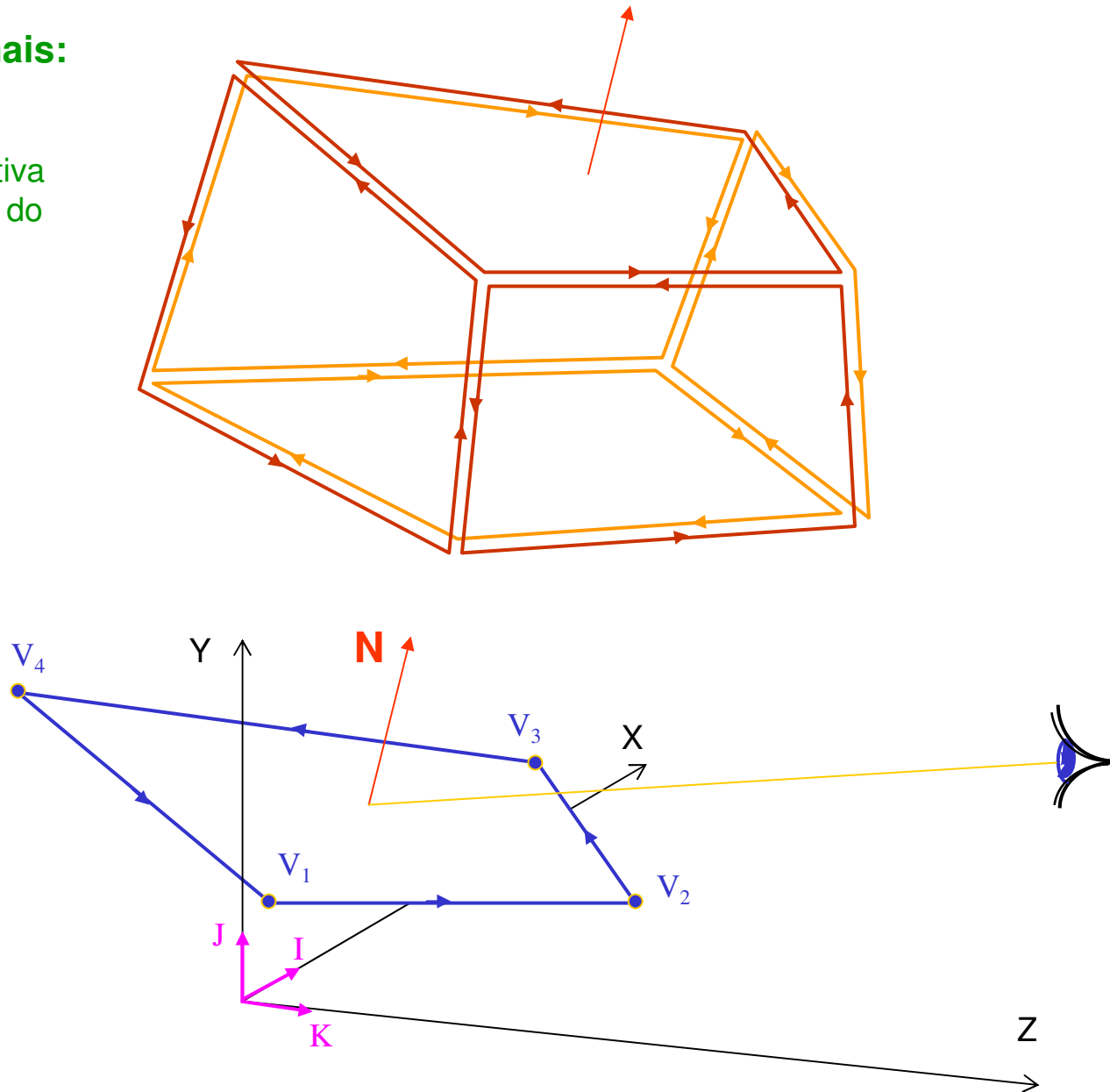
$$= \begin{vmatrix} \mathbf{I} & \mathbf{J} & \mathbf{K} \\ x_{21} & y_{21} & z_{21} \\ x_{32} & y_{32} & z_{32} \end{vmatrix}$$

em que

$$x_{ji} = x_j - x_i$$

$$y_{ji} = y_j - y_i$$

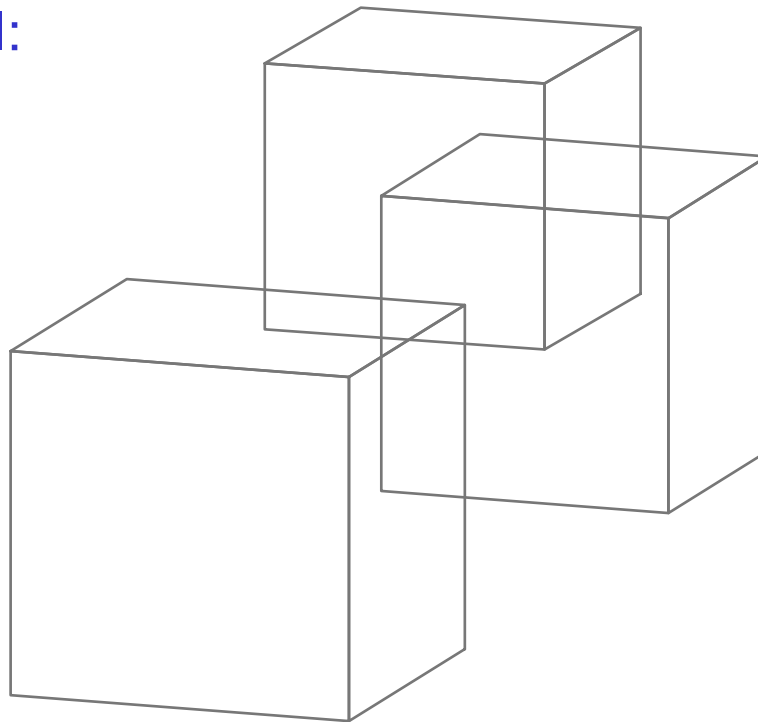
$$z_{ji} = z_j - z_i$$



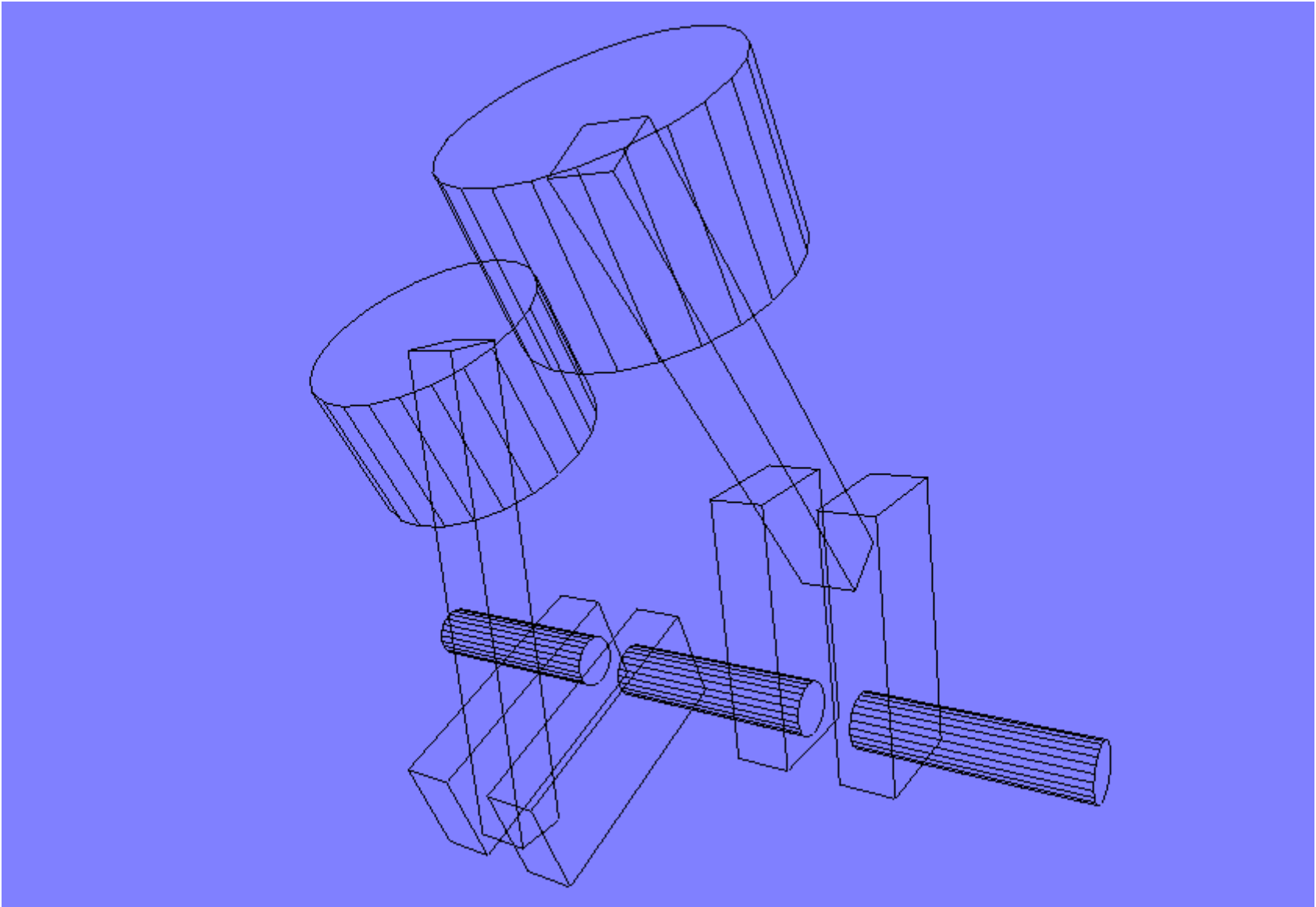
OBS.: Na **projeção ortogonal** bastará investigar o sinal de uma das componentes de N (ou seja: para a projeção em XY , se $N_z > 0$ então o polígono será visível)

Aplicação do método de *Culling* de faces para Hidden-Line Removal

Resultado final:



Em OpenGL aplica-se o método usando `glEnable(GL_CULL_FACE)`,
podendo definir-se o tipo de faces a ocultar (ie, *front* ou *back*) via `glCullFace()`

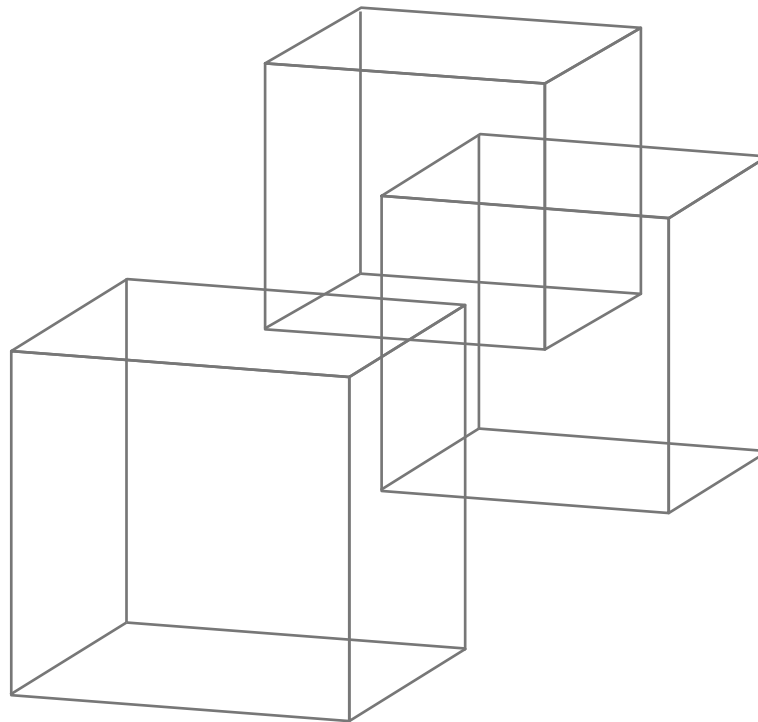


Objectos em VRML

Algoritmo mais geral e muito adequado a HSR:

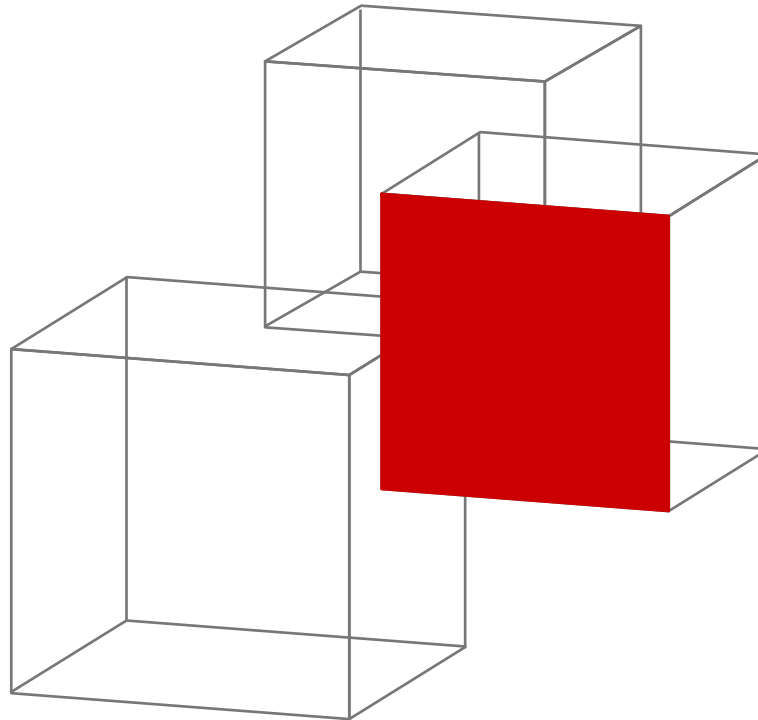
Algoritmo de Z-Buffer

Situação inicial de um
exemplo de aplicação:



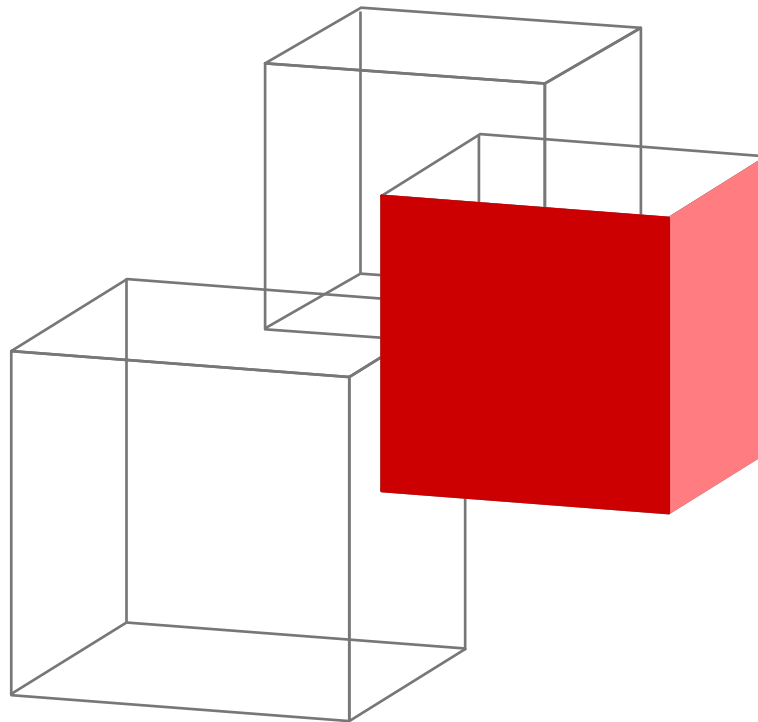
Aplicação do algoritmo de Z-Buffer

Embora os resultados intermédios possam ser diferentes, a ordem de processamento dos polígonos é irrelevante para o resultado final.



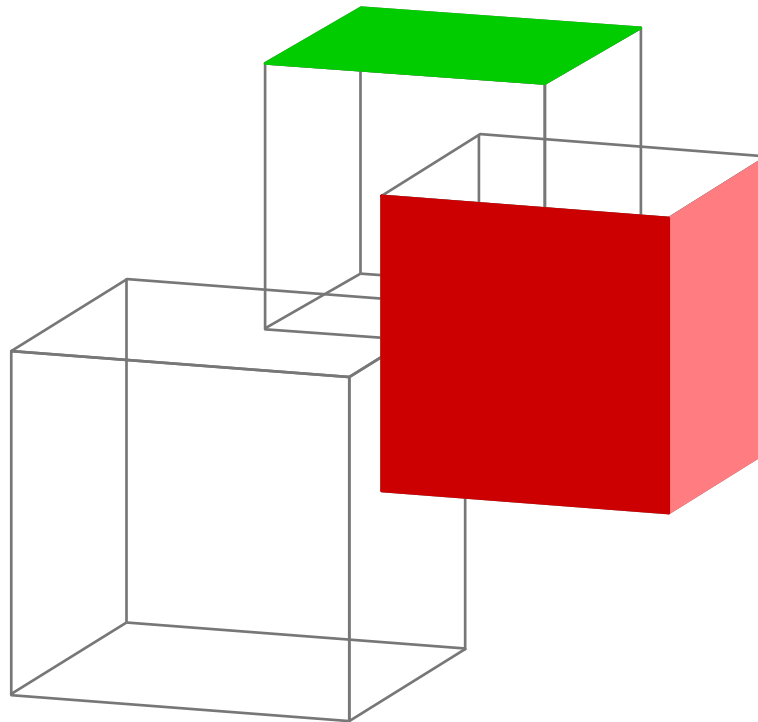
Aplicação do algoritmo de Z-Buffer

Embora os resultados intermédios possam ser diferentes, a ordem de processamento dos polígonos é irrelevante para o resultado final.



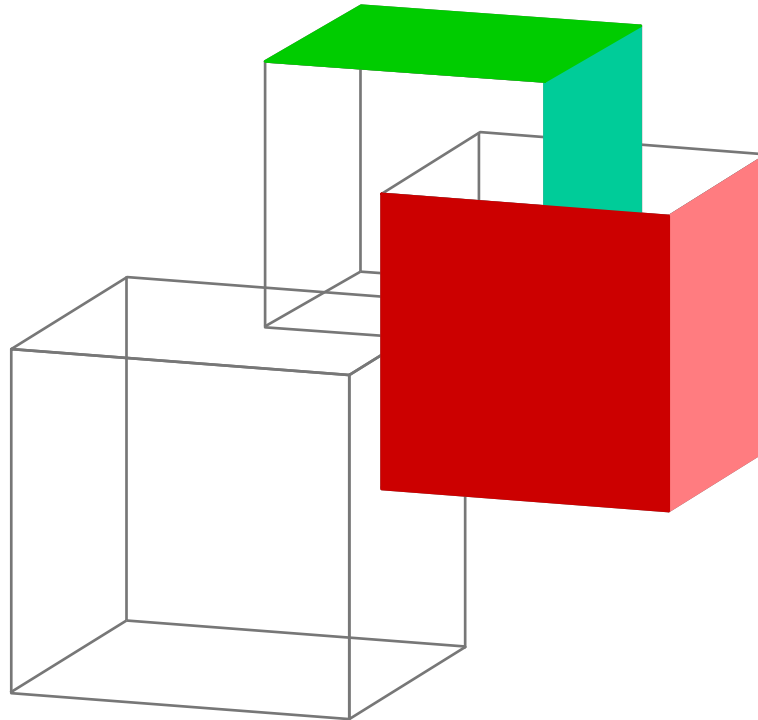
Aplicação do algoritmo de Z-Buffer

Embora os resultados intermédios possam ser diferentes, a ordem de processamento dos polígonos é irrelevante para o resultado final.



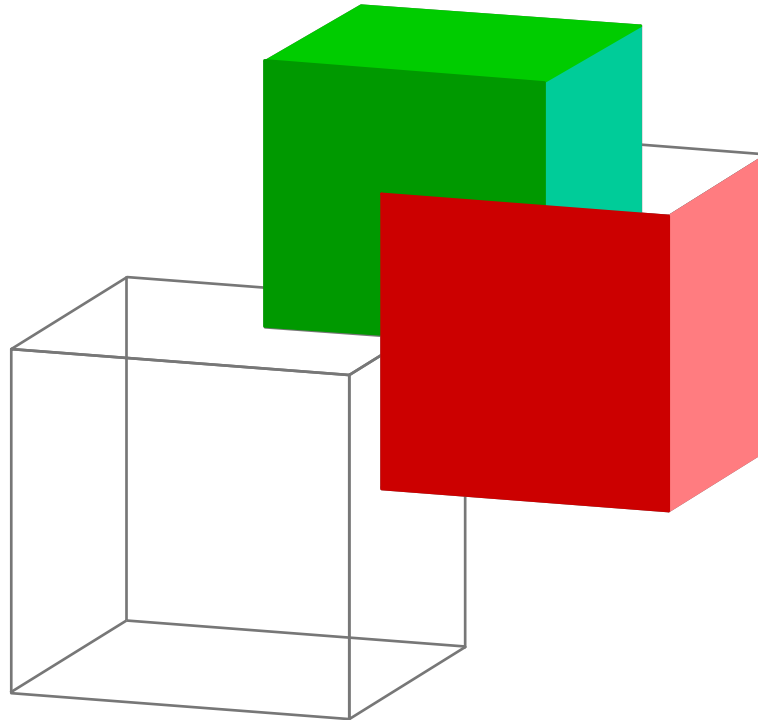
Aplicação do algoritmo de Z-Buffer

Cada pixel de um polígono só é escrito se não houver sobreposição com nenhum outro ou, caso contrário, se a sua distância ao observador for menor que a distância associada ao pixel, já escrito, pertencente a polígono tratado em fase anterior.



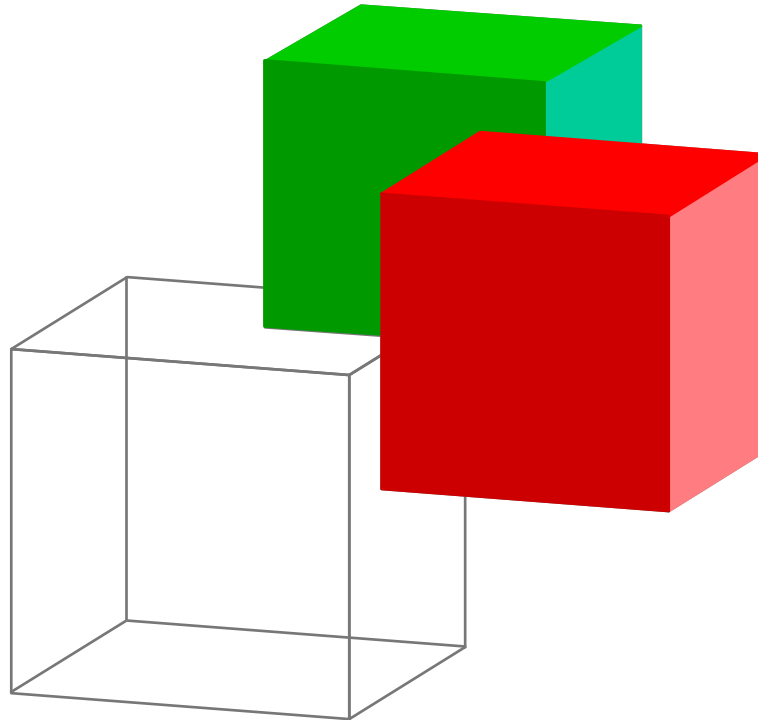
Aplicação do algoritmo de Z-Buffer

Cada pixel de um polígono só é escrito se não houver sobreposição com nenhum outro ou, caso contrário, se a sua distância ao observador for menor que a distância associada ao pixel, já escrito, pertencente a polígono tratado em fase anterior.



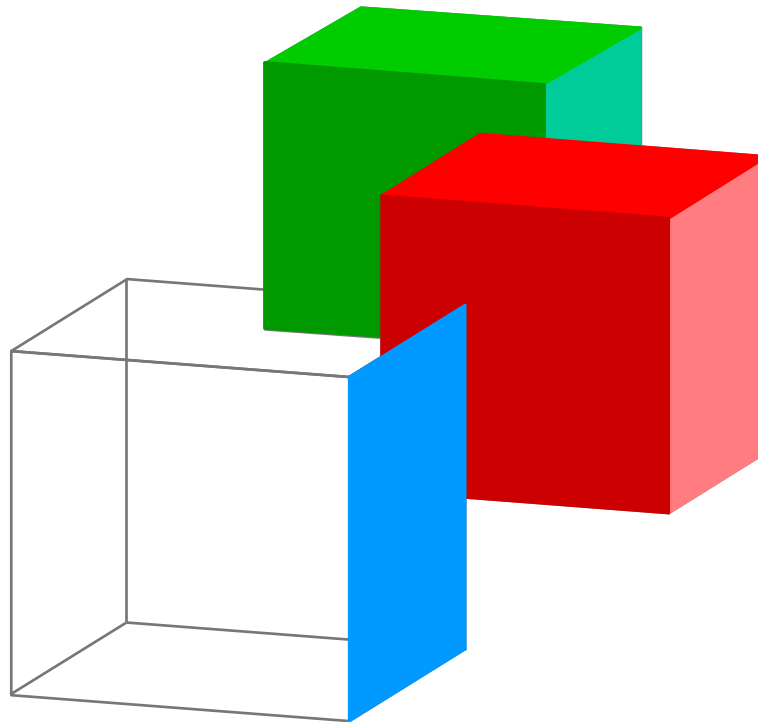
Aplicação do algoritmo de Z-Buffer

Cada pixel de um polígono só é escrito se não houver sobreposição com nenhum outro ou, caso contrário, se a sua distância ao observador for menor que a distância associada ao pixel, já escrito, pertencente a polígono tratado em fase anterior.



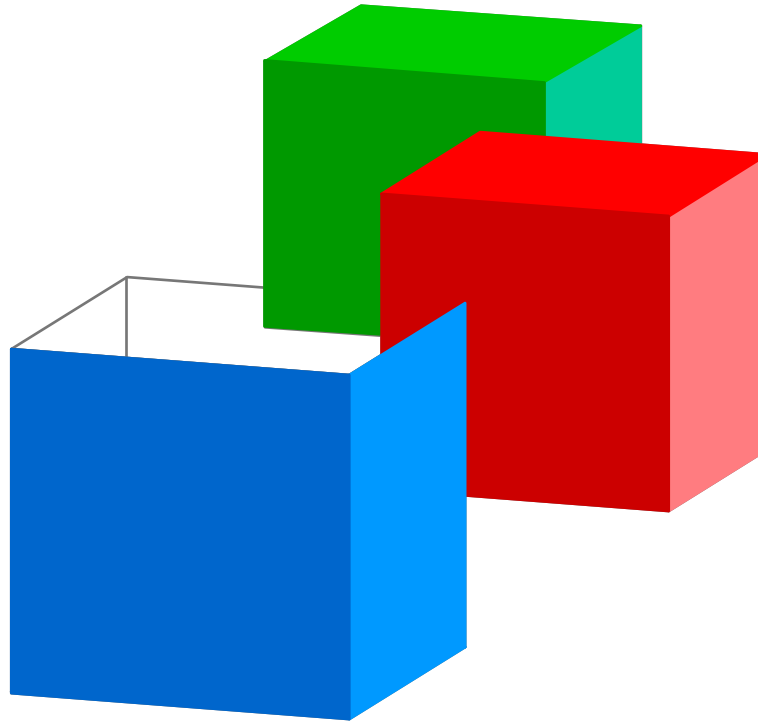
Aplicação do algoritmo de Z-Buffer

Cada pixel de um polígono só é escrito se não houver sobreposição com nenhum outro ou, caso contrário, se a sua distância ao observador for menor que a distância associada ao pixel, já escrito, pertencente a polígono tratado em fase anterior.



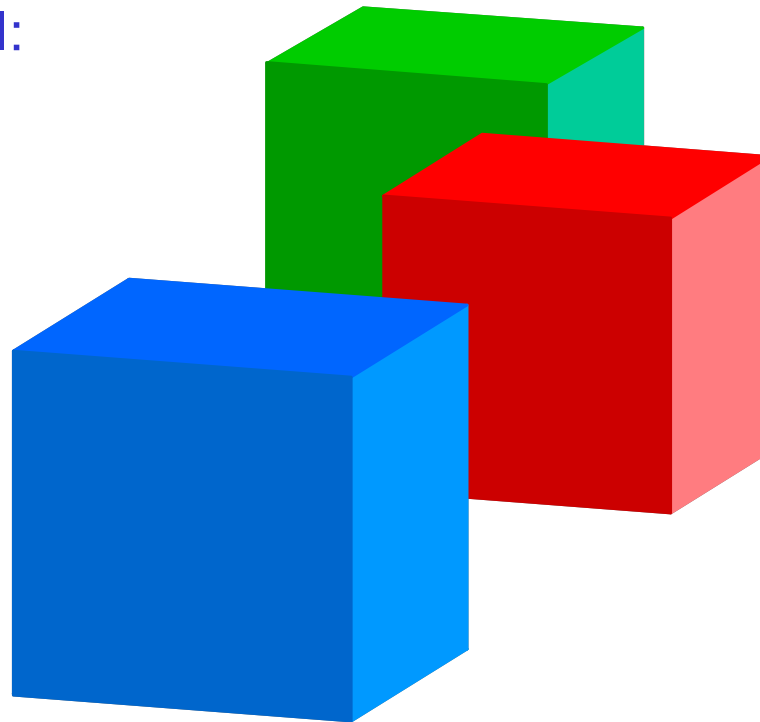
Aplicação do algoritmo de Z-Buffer

Cada pixel de um polígono só é escrito se não houver sobreposição com nenhum outro ou, caso contrário, se a sua distância ao observador for menor que a distância associada ao pixel, já escrito, pertencente a polígono tratado em fase anterior.



Aplicação do algoritmo de Z-Buffer

Resultado final:



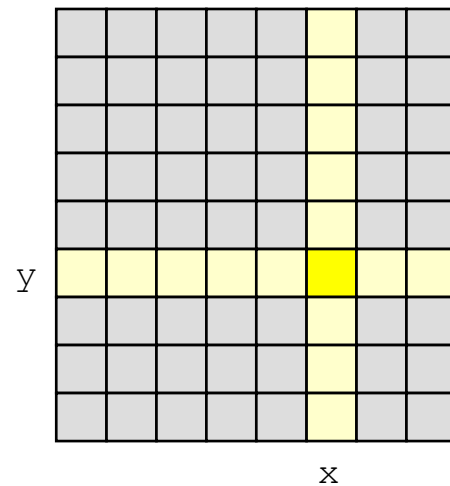
Algoritmo de Z-Buffer

Estruturas de Dados

Refresh Buffer

(RB)

$n \times m$ valores
de cor RGB

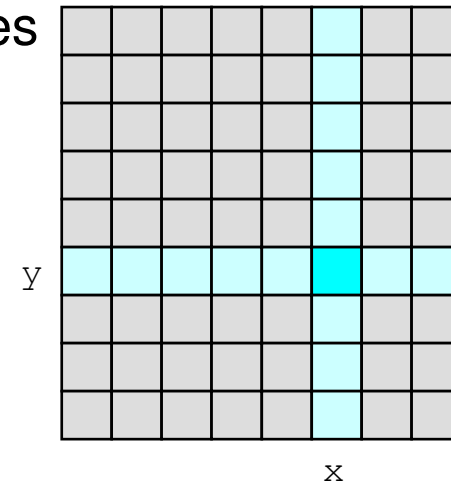


Inicialização com a cor de fundo.

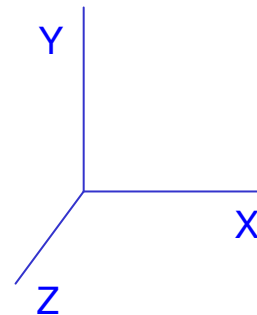
Z-Buffer

(ZB)

$n \times m$ valores
em z



Inicialização com o menor valor possível.



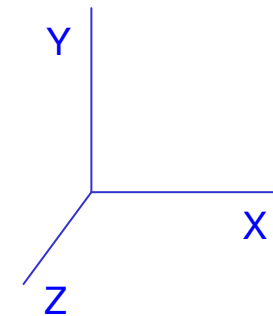
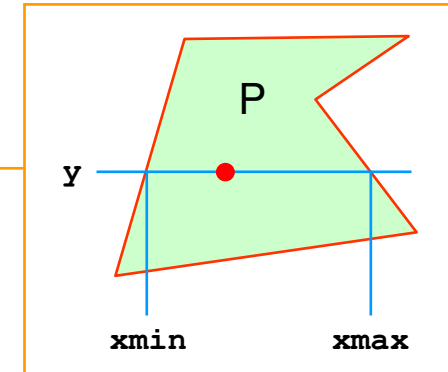
Algoritmo de Z-Buffer

DADOS:

Polígonos planos, em projeção ortogonal, e conhecidos também os valores em **z** (profundidade) dos vértices.

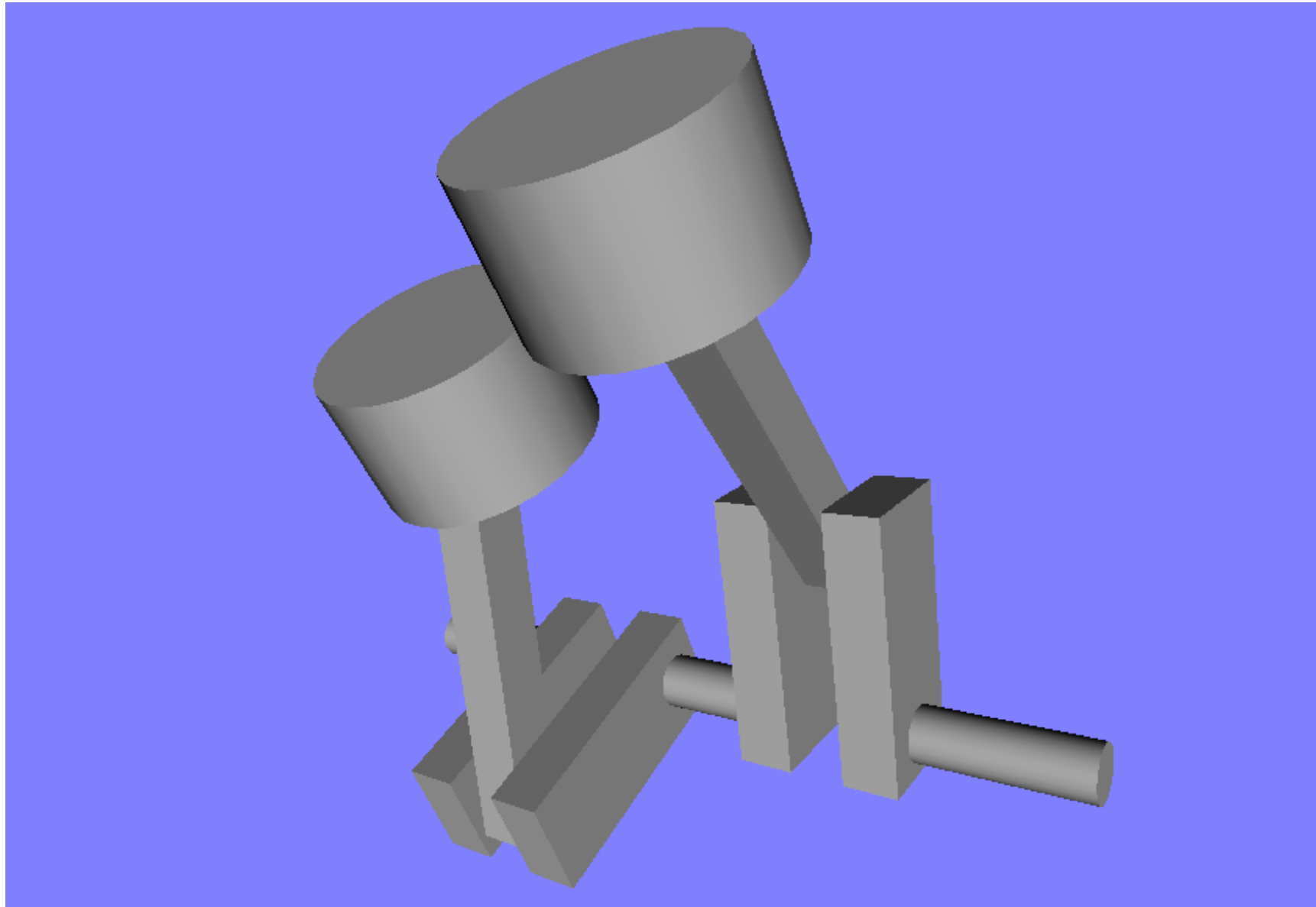
Algoritmo de varrimento, para cada polígono P:

```
for y := ymin to ymax do  
  Calcular xmin e xmax  
  for x := xmin to xmax do  
    Calcular z = f(P,x,y)  
    if z > ZB[x,y] then {o pixel (x,y) passa o teste}  
      ZB[x,y] := z  
      RB[x,y] := RGB(P,x,y)  
    endif  
  endfor  
endfor
```



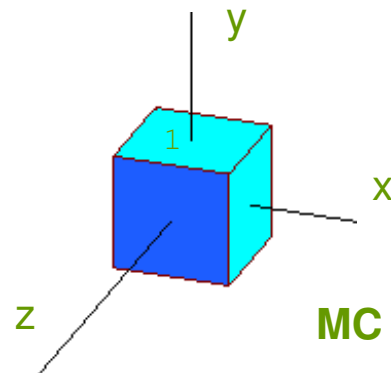
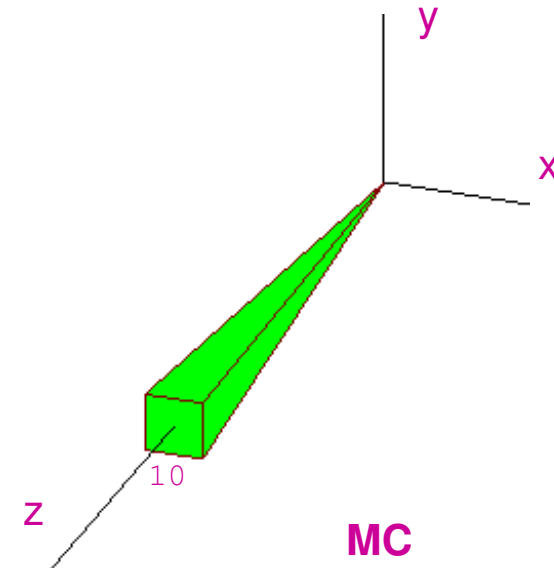
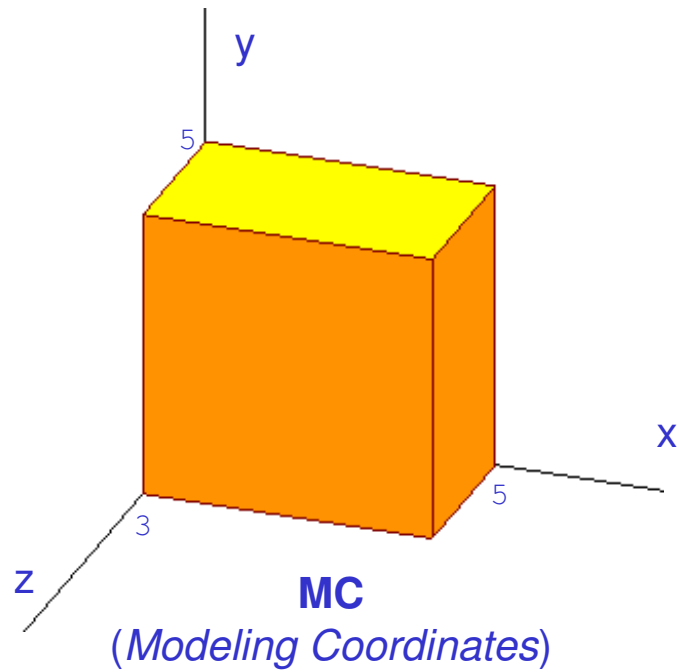
IMPORTANTE:

- Como Z-buffer se aplica depois da projeção, então esta terá de preservar a coordenada z (substituindo-se 0 por 1 na correspondente matriz da projecção anteriormente deduzida)!
- Na API de OpenGL, em vez de z usa-se a distância dz ao ponto de vista, pelo que aí o teste será $dz < ZB[x,y]$.



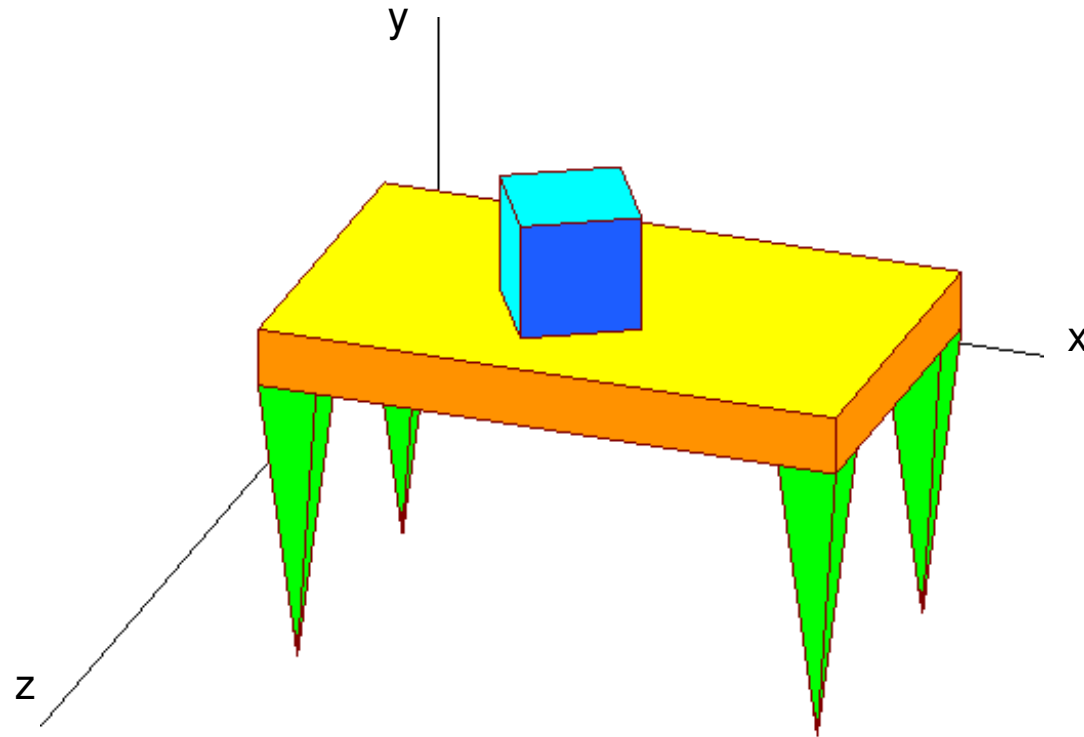
Aplicação de Z-buffer, com modelo de iluminação

COMPOSIÇÃO DE TRANSFORMAÇÕES DE MODELAÇÃO



Sistemas de Coordenadas Locais
(ou de Modelação)

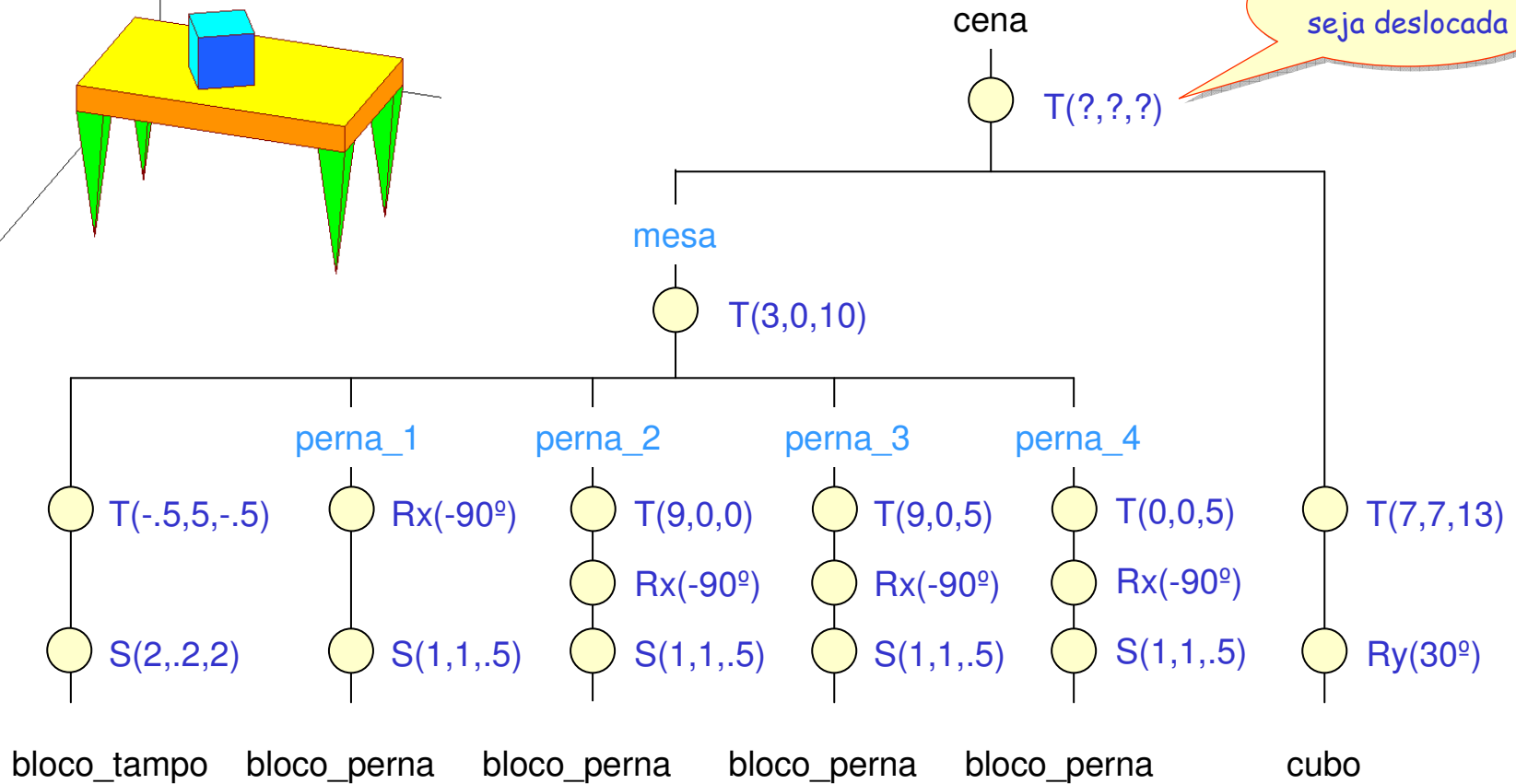
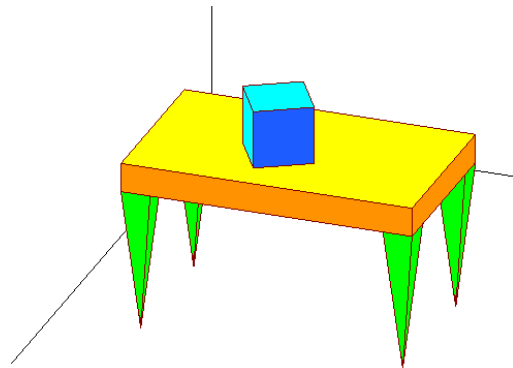
COMPOSIÇÃO DE TRANSFORMAÇÕES DE MODELAÇÃO



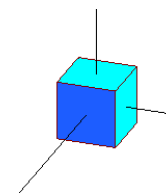
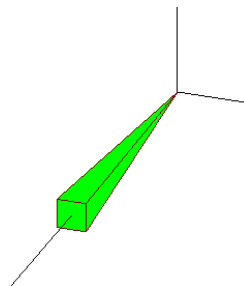
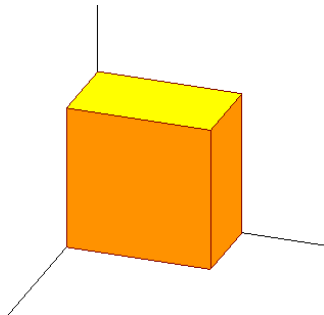
WC
(*World Coordinates*)

Sistema de Coordenadas Globais
(ou do Mundo)

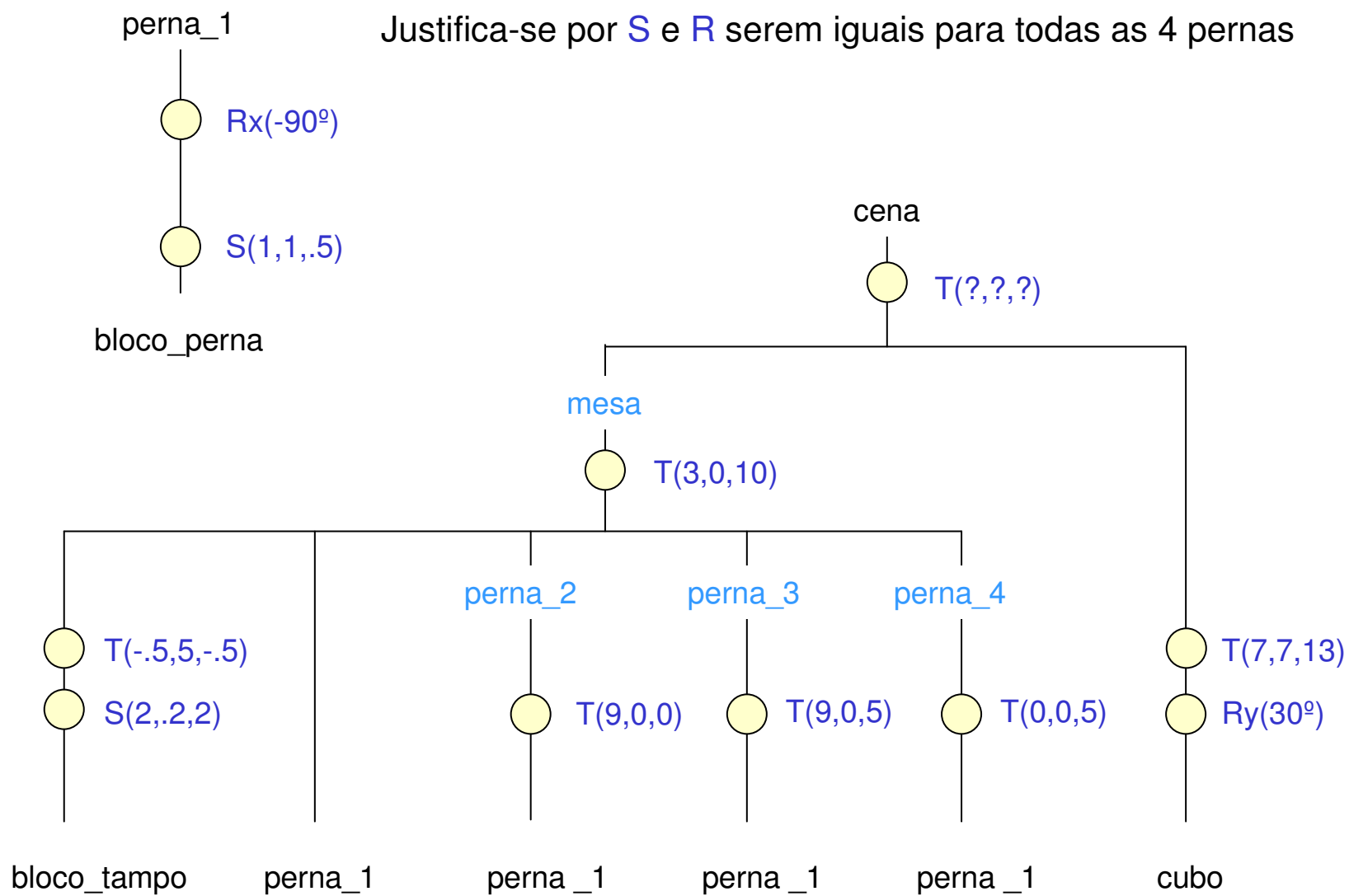
GRAFO DA CENA



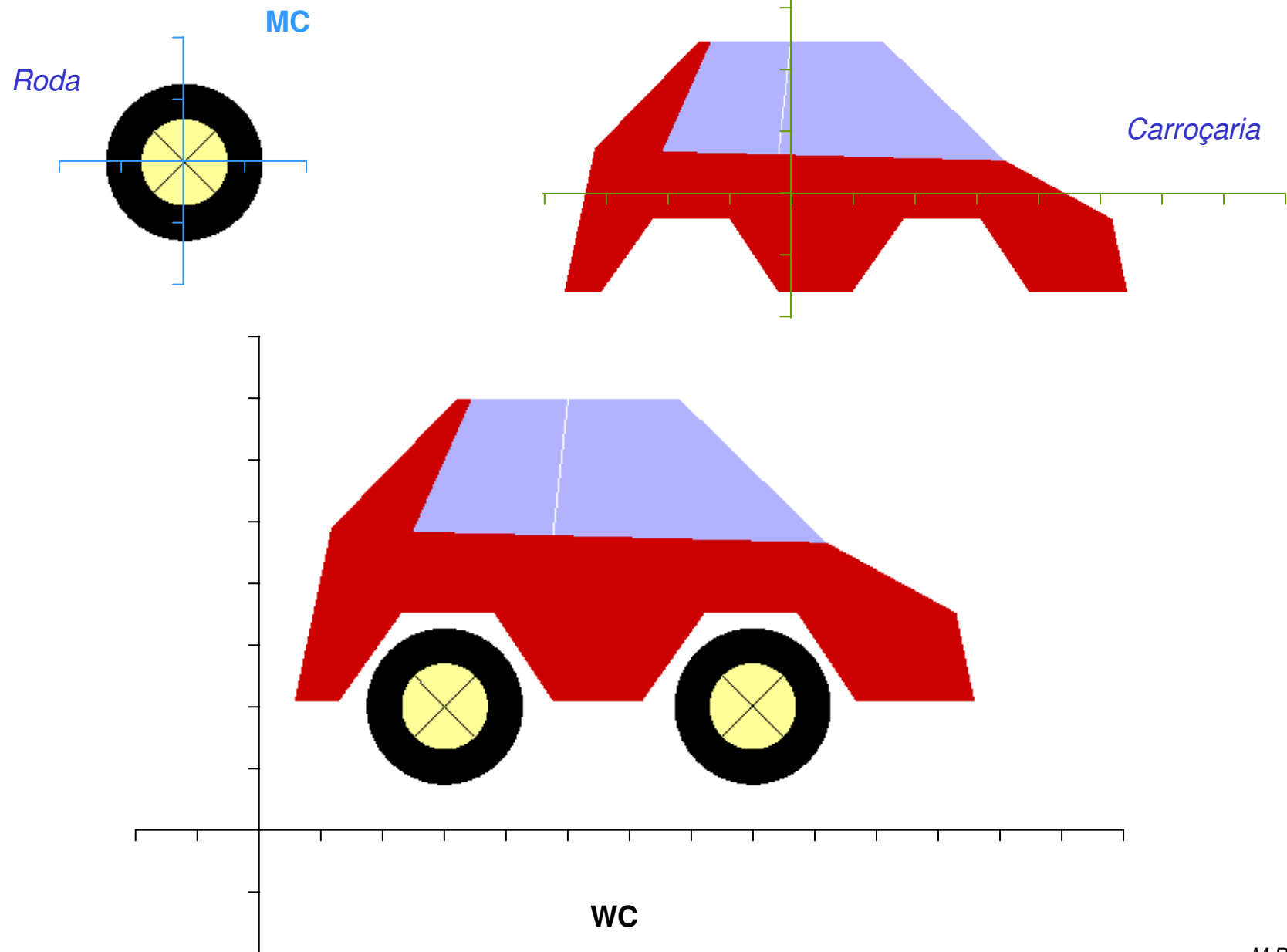
Caso a mesa
seja deslocada



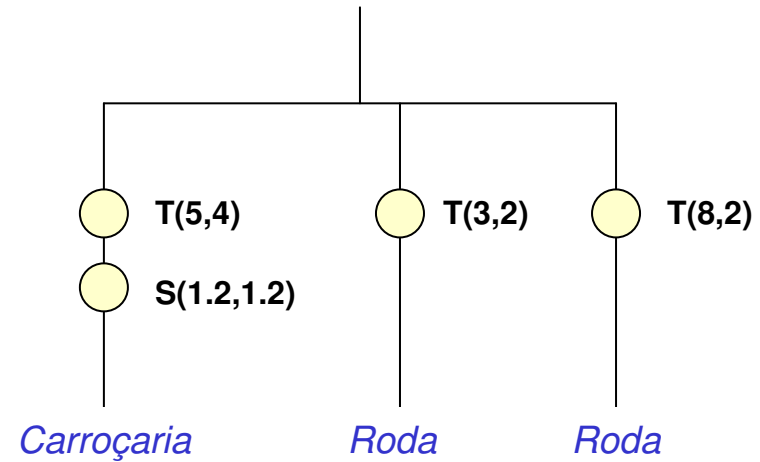
Uma alternativa possível, com base na análise do modelo, usando-se um subgrafo auxiliar:



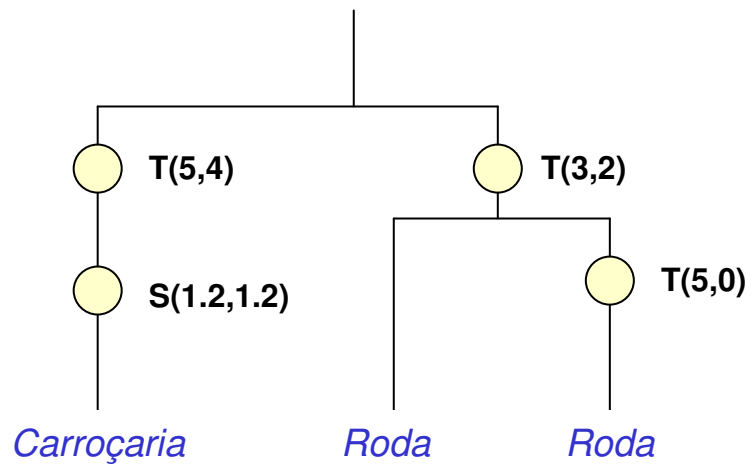
Exemplo de um carro em 2D



GRAFO DA CENA

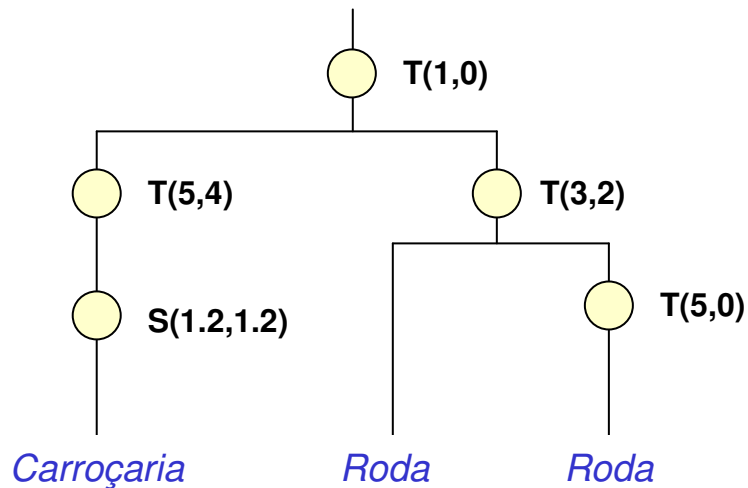


Solução alternativa:

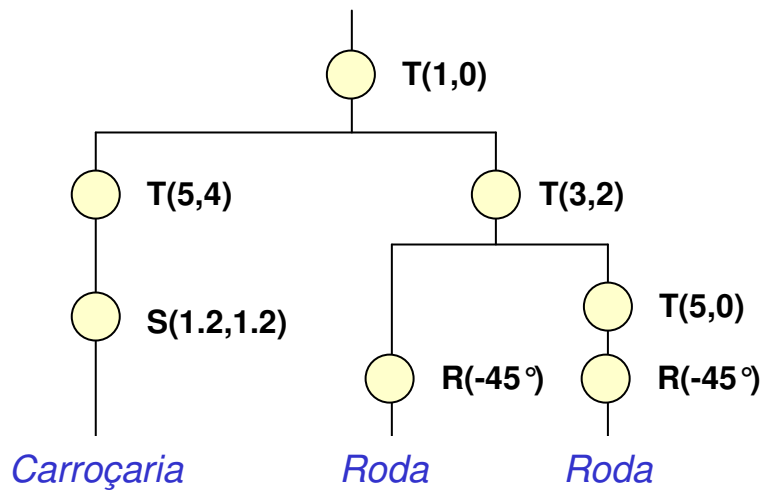


Como fazer avançar o automóvel de 1 unidade?

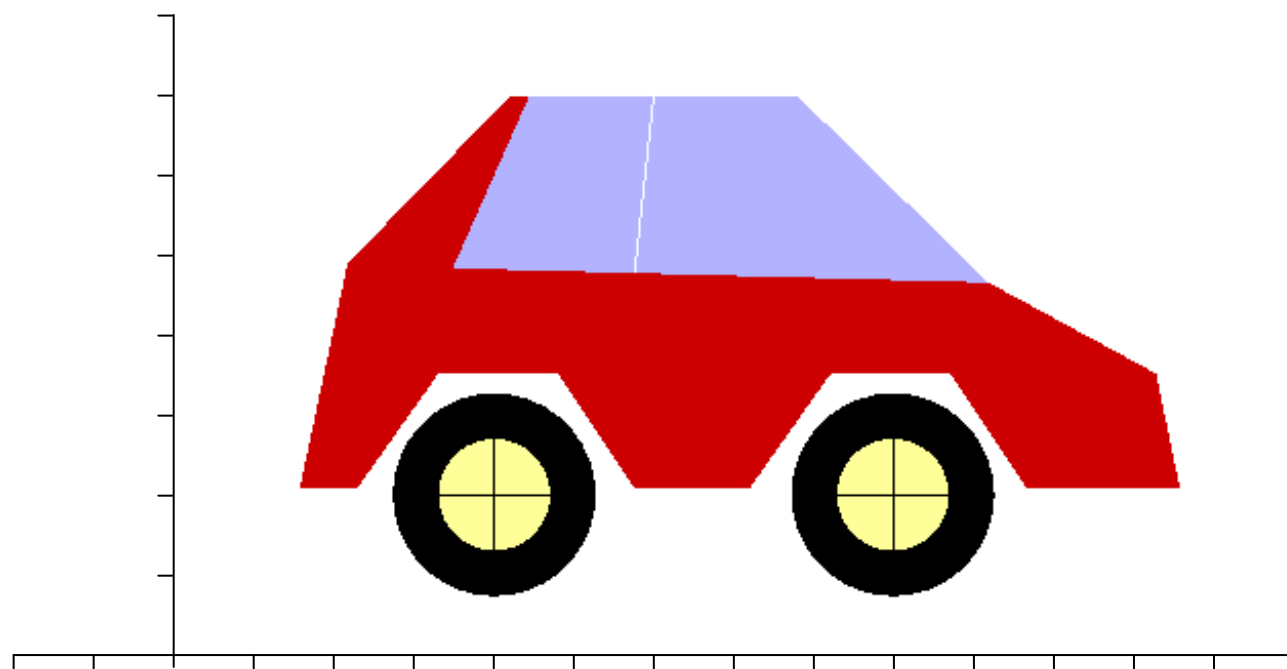
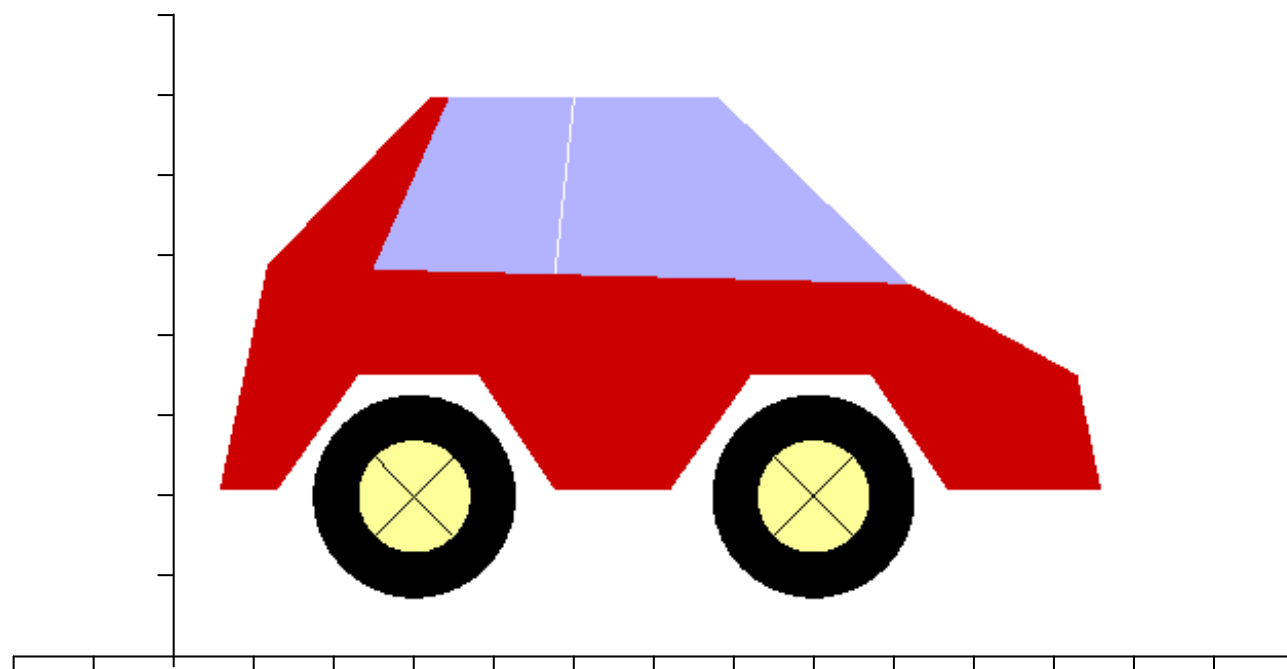
GRAFO DA CENA



E como fazer girar as rodas,
de acordo com esse movimento?

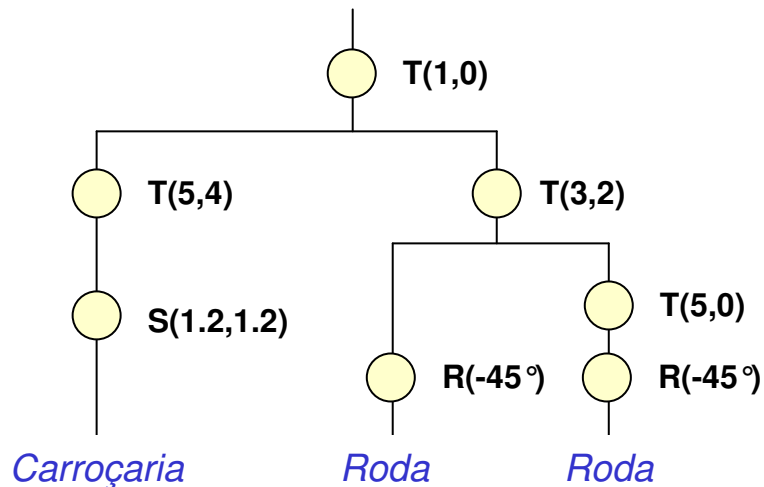


Exercício proposto: Qual seria a
diferença no caso de um trator agrícola?



Estrutura para programa OpenGL, obtida a partir do Grafo de Cena do carro

i) Tradução direta, em pseudocódigo:



(Travessia prefixada do grafo)

```
T(1, 0)
glPushMatrix()
    T(5, 4)
    S(1.2, 1.2)
    Carroçaria
glPopMatrix()
glPushMatrix()
    T(3, 2)
    glPushMatrix()
        R(-45°)
        Roda
    glPopMatrix()
    glPushMatrix()
        T(5, 0)
        R(-45°)
        Roda
    glPopMatrix()
glPopMatrix()
```

Estrutura para programa OpenGL, obtida a partir do Grafo de Cena do carro

ii) Simplificação do pseudocódigo:

```
T(1, 0)
glPushMatrix()
    T(5, 4)
    S(1.2, 1.2)
    Carroçaria
glPopMatrix()
glPushMatrix()
    T(3, 2)
    glPushMatrix()
        R(-45°)
        Roda
    glPopMatrix()
    glPushMatrix()
        T(5, 0)
        R(-45°)
        Roda
    glPopMatrix()
glPopMatrix()
```

Estrutura para programa OpenGL, obtida a partir do Grafo de Cena do carro

iii) Simplificação do pseudocódigo:

```
T(1, 0)
glPushMatrix()
    T(5, 4)
    S(1.2, 1.2)
    Carroçaria
glPopMatrix()

T(3, 2)
glPushMatrix()
    R(-45°)
    Roda
glPopMatrix()
glPushMatrix()
    T(5, 0)
    R(-45°)
    Roda
glPopMatrix()
```

Estrutura para programa OpenGL, obtida a partir do Grafo de Cena do carro

iv) Simplificação do pseudocódigo:

```
T(1, 0)
glPushMatrix()
    T(5, 4)
    S(1.2, 1.2)
    Carroçaria
glPopMatrix()

T(3, 2)
glPushMatrix()
    R(-45°)
    Roda
glPopMatrix()

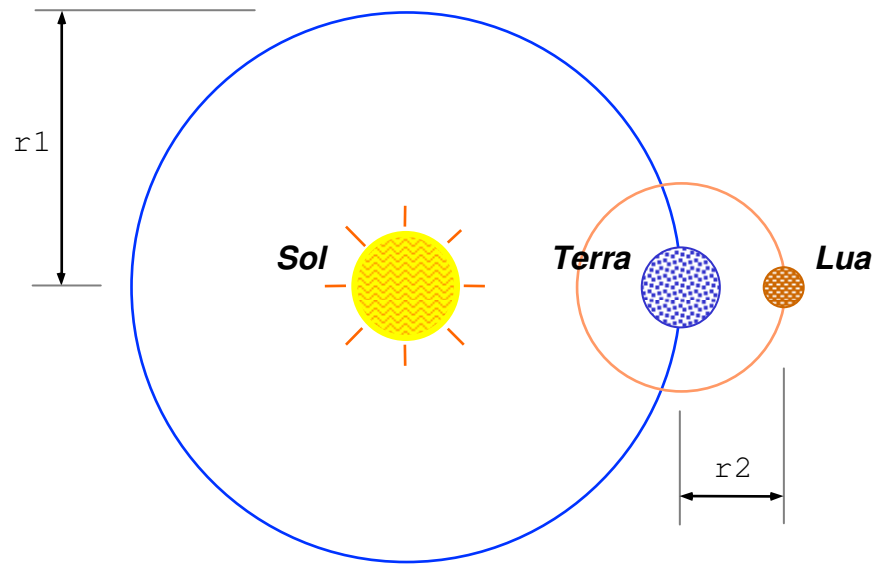
T(5, 0)
R(-45°)
Roda
```

Estrutura para programa OpenGL, obtida a partir do Grafo de Cena do carro

iv) Pseudocódigo simplificado :

```
T(1, 0)
glPushMatrix()
    T(5, 4)
    S(1.2, 1.2)
    Carroçaria
glPopMatrix()
T(3, 2)
glPushMatrix()
    R(-45°)
    Roda
glPopMatrix()
T(5, 0)
R(-45°)
Roda
```

Simulação de um Sistema Planetário simplificado

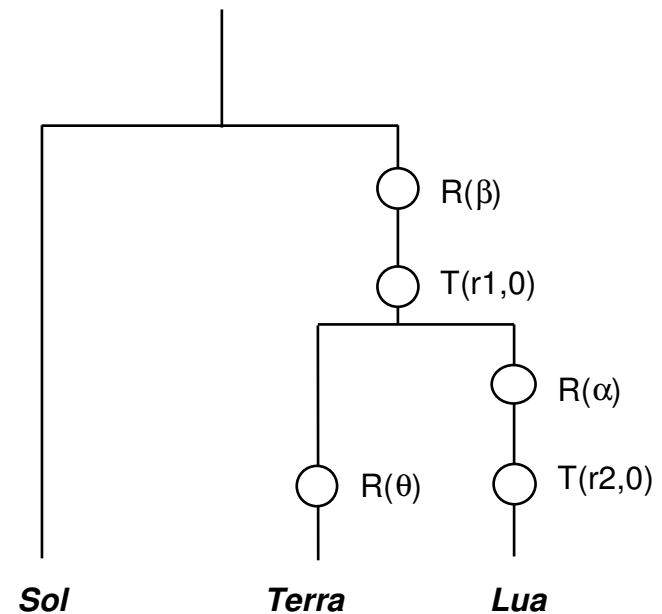


Admite-se que:

- Os objetos primitivos (2D) serão modelados com centro na origem.
- As órbitas dos planetas são circulares e encontram-se no plano XY.

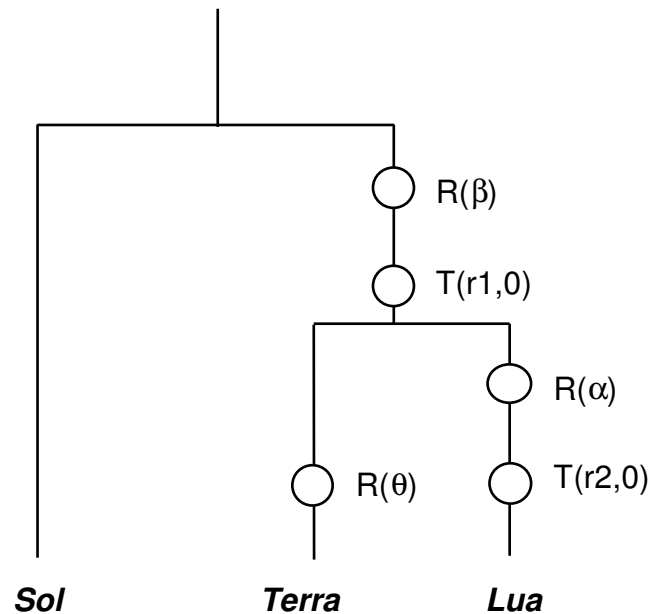
Colocado na
origem de WC

GRAFO DA CENA



Programação em OpenGL, obtida a partir do Grafo de Cena do sistema planetário

i) Tradução direta, em pseudocódigo:

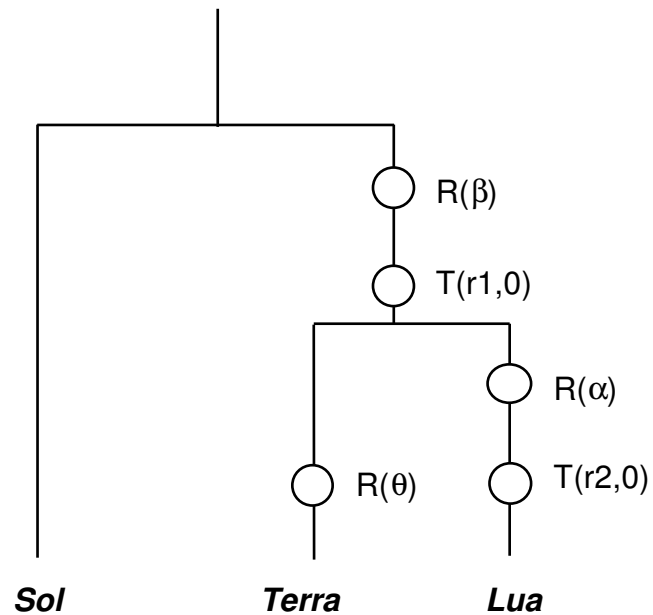


Esta solução foi escrita supondo que **nada se** conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
glPushMatrix()
    glPushMatrix()
        Sol
    glPopMatrix()
glPopMatrix()
glPushMatrix()
    R(β)
    T(r1, 0)
    glPushMatrix()
        R(θ)
        glPushMatrix()
            Terra
        glPopMatrix()
    glPopMatrix()
    R(α)
    T(r2, 0)
    glPushMatrix()
        Lua
    glPopMatrix()
glPopMatrix()
glPopMatrix()
```

Programação em OpenGL, obtida a partir do Grafo de Cena do sistema planetário

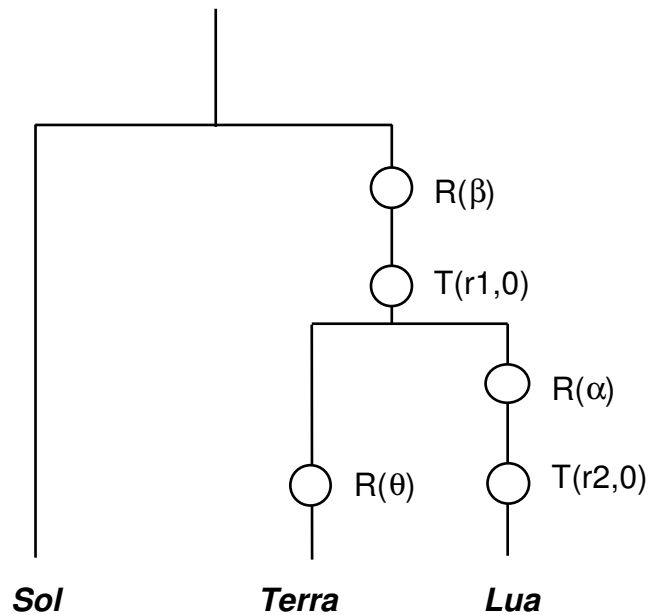
ii) Simplificação do pseudocódigo:



```
glPushMatrix()
    glPushMatrix()
        Sol
    glPopMatrix()
glPopMatrix()
glPushMatrix()
    R(β)
    T(r1, 0)
    glPushMatrix()
        R(θ)
        glPushMatrix()
            Terra
        glPopMatrix()
    glPopMatrix()
    R(α)
    T(r2, 0)
    glPushMatrix()
        Lua
    glPopMatrix()
glPopMatrix()
glPopMatrix()
```


Programação em OpenGL, obtida a partir do Grafo de Cena do sistema planetário

ii) Pseudocódigo simplificado:



```
glPushMatrix()  
    Sol  
glPopMatrix()  
R( $\beta$ )  
T(r1, 0)  
glPushMatrix()  
    R( $\theta$ )  
    Terra  
glPopMatrix()  
R( $\alpha$ )  
T(r2, 0)  
Lua
```

NOTA: Um Grafo de Cena não pode conter ciclos.

An introduction to X3D

X3D is a royalty-free open standard's file format and run-time architecture to represent and communicate 3D scenes and objects using **XML**. It is an ISO ratified standard that provides a system for the storage, retrieval and playback of real time graphics content embedded in applications, all within an open architecture to support a wide array of domains and user scenarios. X3D is also intended to be a universal interchange format for integrated 3D graphics and multimedia. The data integration capacities and the set of componentized features of X3D is expanding allowing a 3D immersive virtual experience for the Web and mobile devices across Geospacial, Augmented Reality, CAD, medical and other markets.

The development of real-time communication of 3D data across all applications and network applications has evolved from its beginnings as the Virtual Reality Modeling Language (**VRML**) to the considerably more mature and refined X3D standard. Although VRML has been superseded by the X3D standard, VRML files can still be read by any X3D-savvy application.

- VRML text files use a **.wrl** extension
- X3D text files written in XML use a **.x3d** extension (but the **.x3dv** and **.x3db** extensions are also possible for the files using the Classic VRML encoding and the Binary encoding, respectively)

X3D/VRML are supported by the Web3D Consortium (see <http://www.web3d.org/>)

We can construct X3D/VRML files using:

- **A text editor**

Pros:

- No new software to buy
- Access to all X3D/VRML features
- Detailed control of world efficiency

Cons:

- Hard to author complex 3D shapes
- Requires knowledge of X3D/VRML syntax

- **A world builder application**

Pros:

- Easy 3D drawing user interface
- Little need to learn X3D/VRML syntax

Cons:

- May not support all X3D/VRML features
- May not produce most efficient X3D/VRML

- **A shape generator**

Pros:

- Easy way to generate complex shapes
(Fractal mountains, logos, etc.)

Cons:

- Only suitable for narrow set of shapes
- Best used with other software

- **A modeler and format converter**

Pros:

- Very powerful features available
- Can make photo-realistic images too
- Easy to make shapes that are too complex

Cons:

- May not support all X3D/VRML features
- Not designed for X3D/VRML
- One-way path from modeler into X3D/VRML

- **An integrated development environment (IDE)**

Pros:

- X3D/VRML syntax validation and support
- Detailed control of world efficiency

Cons:

- Hard to author very complex 3D shapes

An X3D sample file

```
<?xml version="1.0" encoding="UTF-8"?>
```

XML header

```
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.1//EN"  
  "http://www.web3d.org/specifications/x3d-3.1.dtd">
```

The XML DTD
(Document Type
Definition) to validate
the X3D version (or an
XML SCHEMA as an
alternative to DTD)

```
<X3D version='3.1' profile='Interchange'>
```

Set of X3D features

```
<head>
```

```
  <meta name='filename' content='Cilindro1.x3d'/>
```

```
  <meta name='description' content='Flat Cylinder'/>
```

```
  <meta name='creator' content='M. Prospero dos Santos'/>
```

Optional
meta tags

```
</head>
```

X3D node

```
<Scene>
```

XML/X3D
comment

```
<Shape >
```

Field (or node
attribute)

```
  <!-- Um cilindro -->
```

```
  <Cylinder radius='0.3' height='2.0'/>
```

Attribute value

Attribute value

```
</Shape>
```

```
</Scene>
```

```
</X3D>
```

X3D node

Field (or node
attribute)

... And the equivalent VRML code

```
#VRML V2.0 utf8
# Flat Cylinder
#Autor: M. Prospero dos Santos

Shape {
    geometry
        cylinder { radius .3 height 2 }    # Um cilindro
}
```

Notes about the past:

- The first version of VRML was specified in November 1994 (known as VRML1 or VRML 1.0).
- In 1997, a new version of the format was finalized, as VRML97 (also known as VRML2 or VRML 2.0), and became an ISO standard.

The structure of an X3D file

- XML elements correspond to **X3D nodes**
- XML attributes correspond to **X3D fields**

Nodes description of scene content (shapes, lights, sounds, etc.)

Fields node attributes you can change

Values attribute values

An X3D file may contain any number of node elements.

Nodes may serve as fields to other X3D nodes.

Fields and values specify node attributes.

Fields are optional and given in any order (default value used if field not given).

Every node has:

- A *node type* (Shape , Cylinder , etc.)
- Zero or more fields

Every field has:

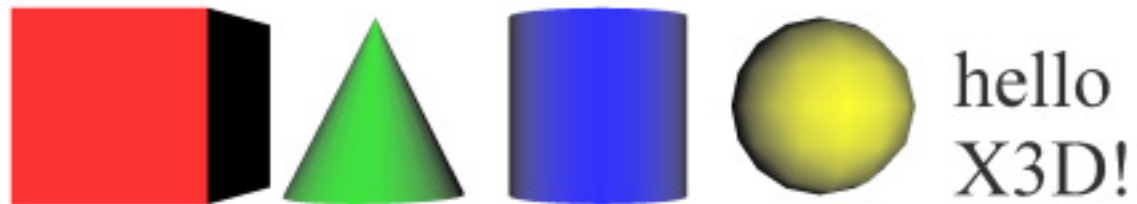
- A field name
- A *data type* (float, int, etc.)
- A default value

Shapes

Shapes are the building blocks of an X3D/VRML world.

Primitive Shapes are standard building blocks:

- **Box**
- **Cone**
- **Cylinder**
- **Sphere**
- **Text**



By default, all shapes are built at the center of the world.

Geometric Transformations

A **Transform node** enables you to

- **Scale** shapes
- **Rotate** shapes
- **Translate** (position) shapes

Note: According to the specification, scale values shall be greater than zero.

Abstract functional specification:

```
Transform : X3DGroupingNode {  
  MFNode      [in]      addChildren      [X3DChildNode]  
  MFNode      [in]      removeChildren   [X3DChildNode]  
  SFVec3f     [in,out]  center            0 0 0  (-∞,∞)  
  MFNode      [in,out]  children          []      [X3DChildNode]  
  SFNode      [in,out]  metadata          NULL    [X3DMetadataObject]  
  SFRotation  [in,out]  rotation          0 0 1 0 [-1,1] or (-∞,∞)  
  SFVec3f     [in,out]  scale             1 1 1  (-∞, ∞)  
  SFRotation  [in,out]  scaleOrientation  0 0 1 0 [-1,1] or (-∞,∞)  
  SFVec3f     [in,out]  translation       0 0 0  (-∞,∞)  
  SFVec3f     []        bboxCenter        0 0 0  (-∞,∞)  
  SFVec3f     []        bboxSize         -1 -1 -1 [0,∞) or -1 -1 -1  
}
```

Field types

Field names

Default values

Node Specification

Field type (prefix)

SF... → one value of the specified data type

MF... → a multiple-valued field ([] are used)

Interface type (it controls what access the other nodes have to the contents of the field)

[in] → write access

[out] → read access

[] → no access

[in,out] → read and write access for all nodes

Another node specification:

```
TimeSensor : X3DTimeDependentNode, X3DSensorNode {
  SFTIME [in,out] cycleInterval 1 (0,∞)
  SFBool [in,out] enabled TRUE
  SFBool [in,out] loop FALSE
  SFNode [in,out] metadata NULL [X3DMetadataObject]
  SFTIME [in,out] pauseTime 0 (-∞,∞)
  SFTIME [in,out] resumeTime 0
  SFTIME [in,out] startTime 0 (-∞,∞)
  SFTIME [in,out] stopTime 0 (-∞,∞)
  SFTIME [out] cycleTime
  SFTIME [out] elapsedTime
  SFFloat [out] fraction_changed
  SFBool [out] isActive
  SFBool [out] isPaused
  SFTIME [out] time
}
```

Animation Basics

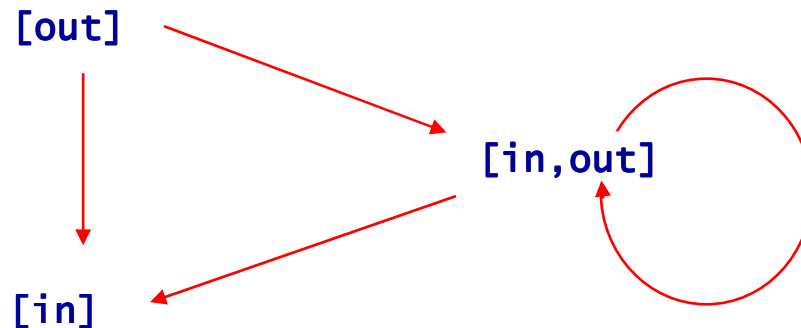
Animation is created by a sequence of events.

One event can generate any number of other events.

Connecting the output of one node to the input of another:

```
<ROUTE fromField=OutFieldName fromNode=Node1  
toField=InFieldName toNode=Node2 />
```

Admissible connections:



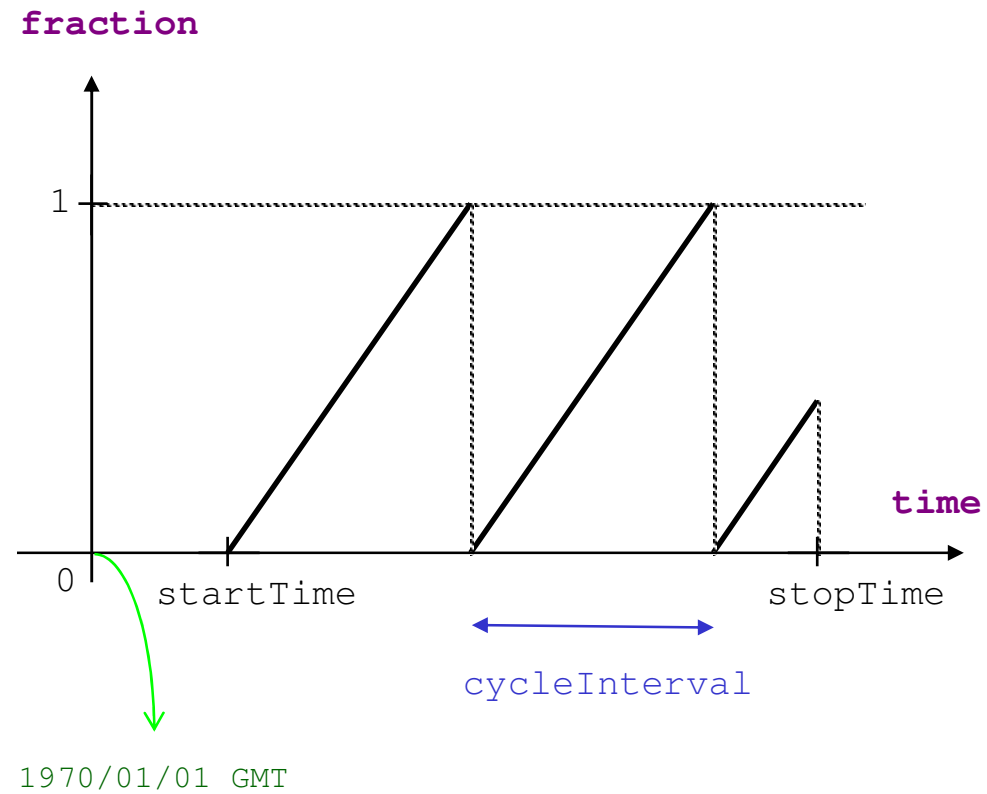
Question: What causes the first event?

Answer: An external influence, e.g.:

- the passing of time
- a user action

There is a lot of control over how events are generated !

The passing of Time...



(see **TimeSensor** node)

Interpolators

```
PositionInterpolator : X3DInterpolatorNode {  
    SFFloat [in]      set_fraction      ( $-\infty, \infty$ )  
    MFFloat [in,out] key                []    ( $-\infty, \infty$ )  
    MFVec3f [in,out] keyValue           []    ( $-\infty, \infty$ )  
    SFNode  [in,out] metadata           NULL [X3DMetadataObject]  
    SFVec3f [out]     value_changed  
}
```

```
OrientationInterpolator : X3DInterpolatorNode {  
    SFFloat [in]      set_fraction      ( $-\infty, \infty$ )  
    MFFloat [in,out] key                []    ( $-\infty, \infty$ )  
    MFRotation [in,out] keyValue        []    [-1,1] or ( $-\infty, \infty$ )  
    SFNode  [in,out] metadata           NULL [X3DMetadataObject]  
    SFRotation [out]   value_changed  
}
```

```
ScalarInterpolator : X3DInterpolatorNode {  
    SFFloat [in]      set_fraction      ( $-\infty, \infty$ )  
    MFFloat [in,out] key                []    ( $-\infty, \infty$ )  
    MFFloat [in,out] keyValue           []    ( $-\infty, \infty$ )  
    SFNode  [in,out] metadata           NULL [X3DMetadataObject]  
    SFFloat [out]     value_changed  
}
```

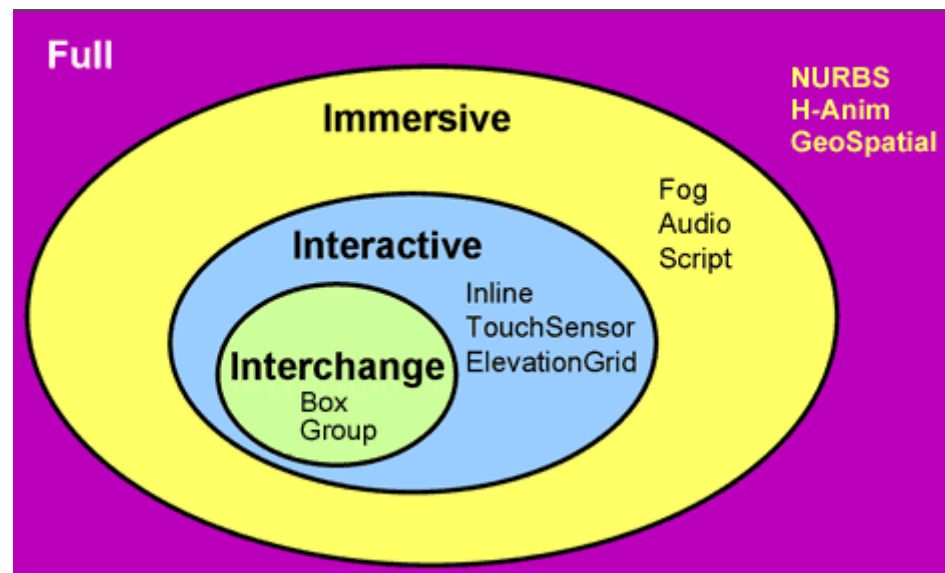
X3D Baseline Profiles

Interchange is the basic profile for communicating between applications. It support geometry, texturing, basic lighting, and animation.

Interactive enables basic interaction with a 3D environment by adding various sensor nodes for user navigation and interaction (e.g., PlaneSensor, TouchSensor, etc.), enhanced timing, and additional lighting (Spotlight, PointLight).

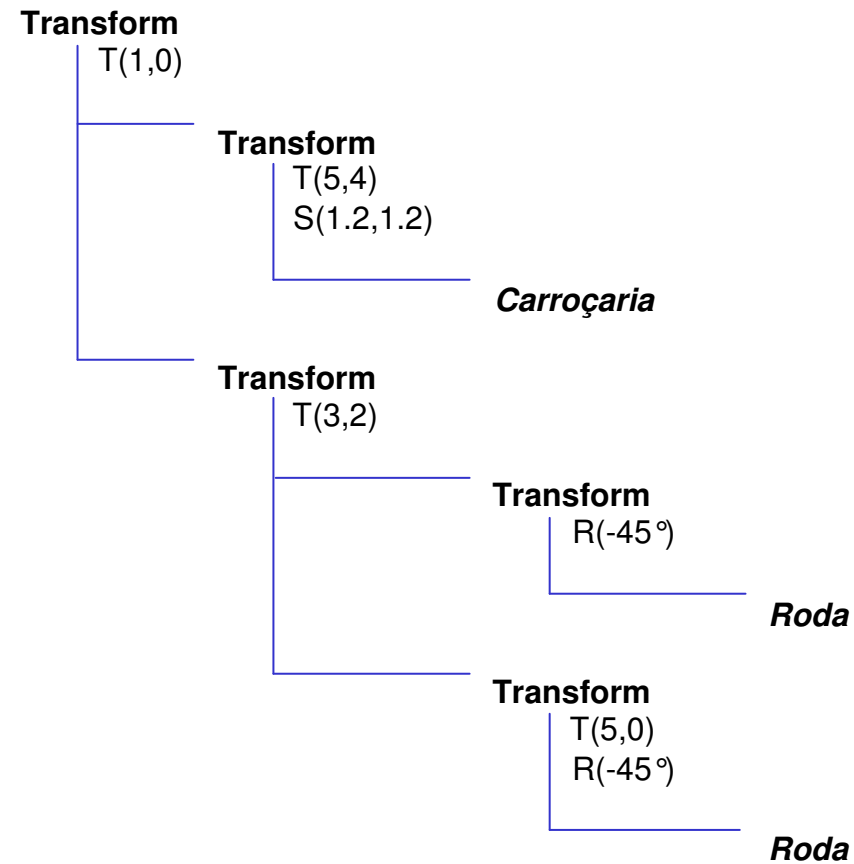
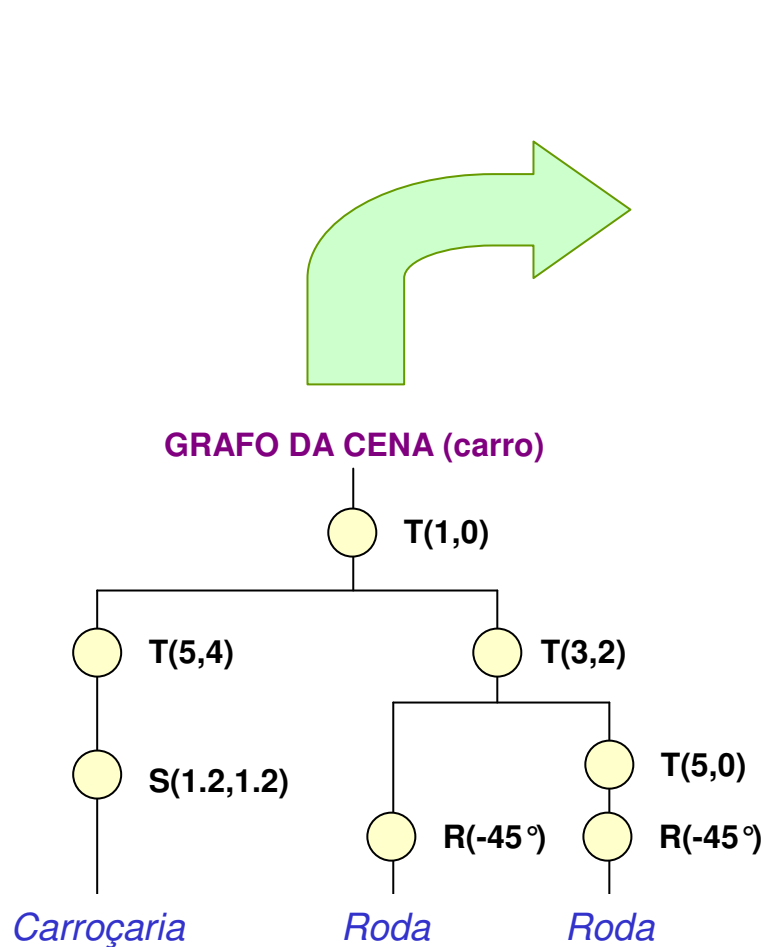
Immersive enables full 3D graphics and interaction, including audio support, collision, fog, and scripting.

Full includes all defined nodes including NURBS, H-Anim and GeoSpatial components.



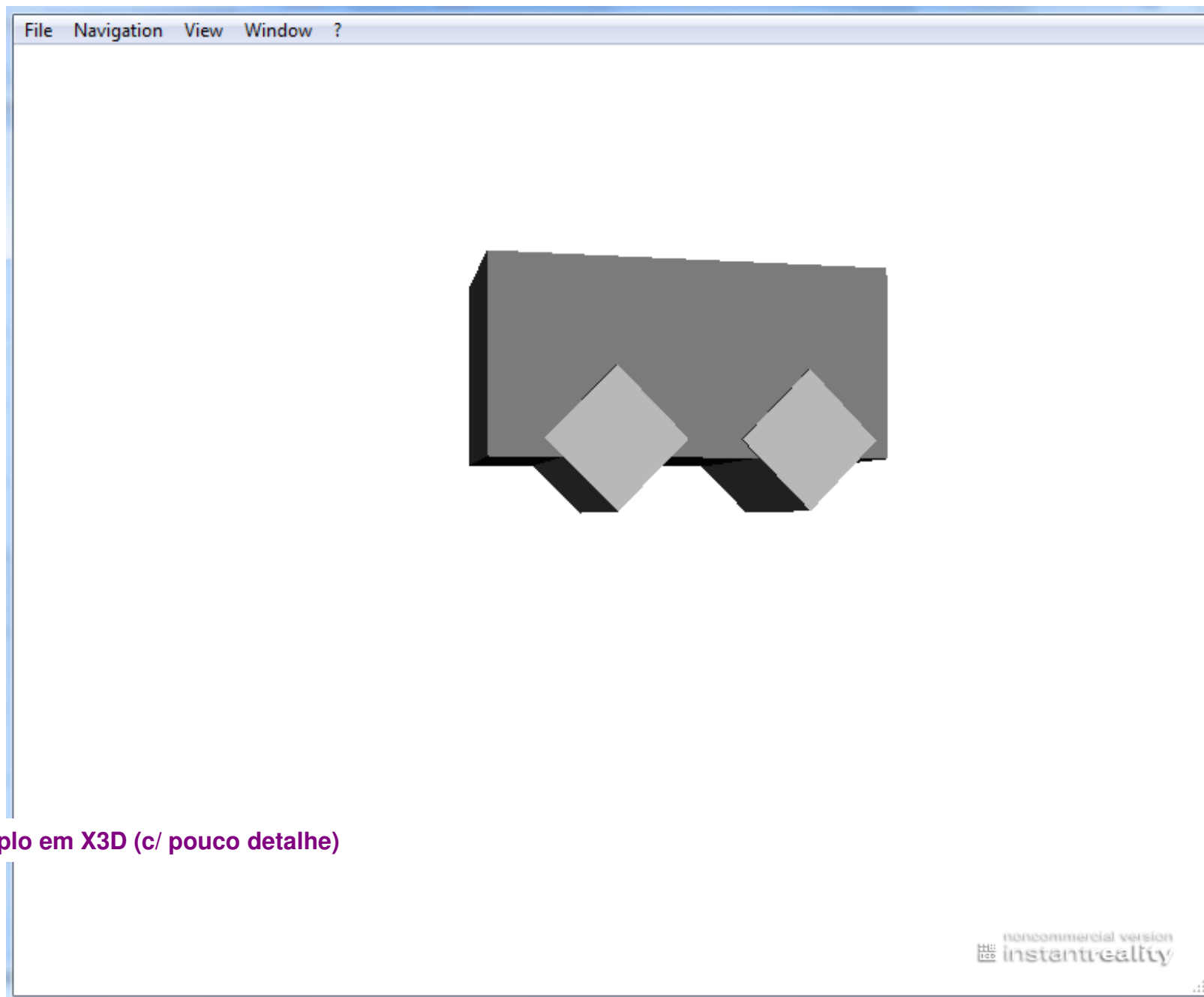
CONSTRUÇÃO DE GRAFOS DE CENA PARA USO EM X3D / VRML

Scene Graph X3D/VRML oriented

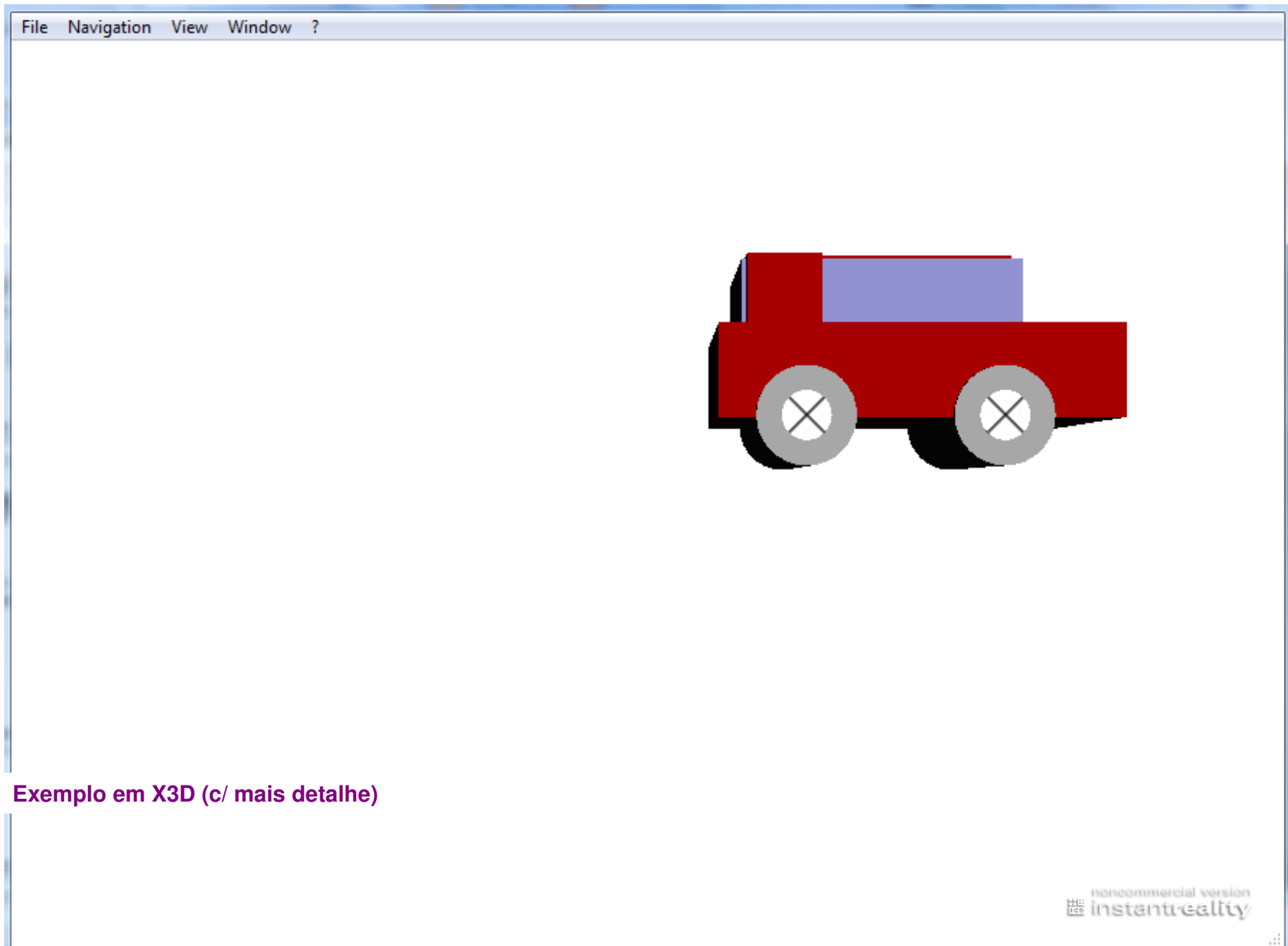


Observação:

De acordo com a norma, um nó Transform permite especificar uma combinação das 3 transformações geométricas elementares.

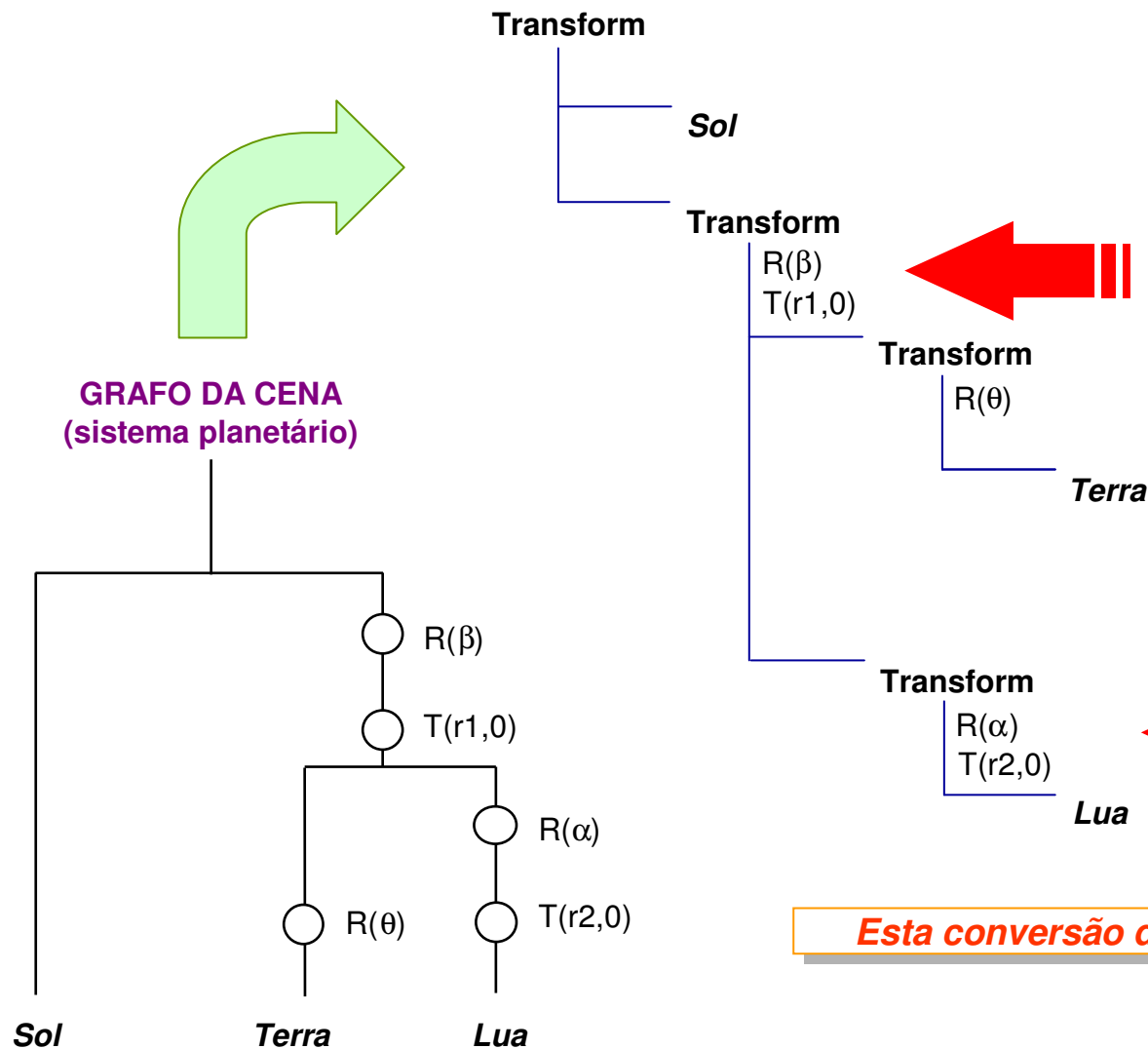


Exemplo em X3D (c/ pouco detalhe)



Exemplo em X3D (c/ mais detalhe)

GRAFO DA CENA ORIENTADO PARA X3D/VRML

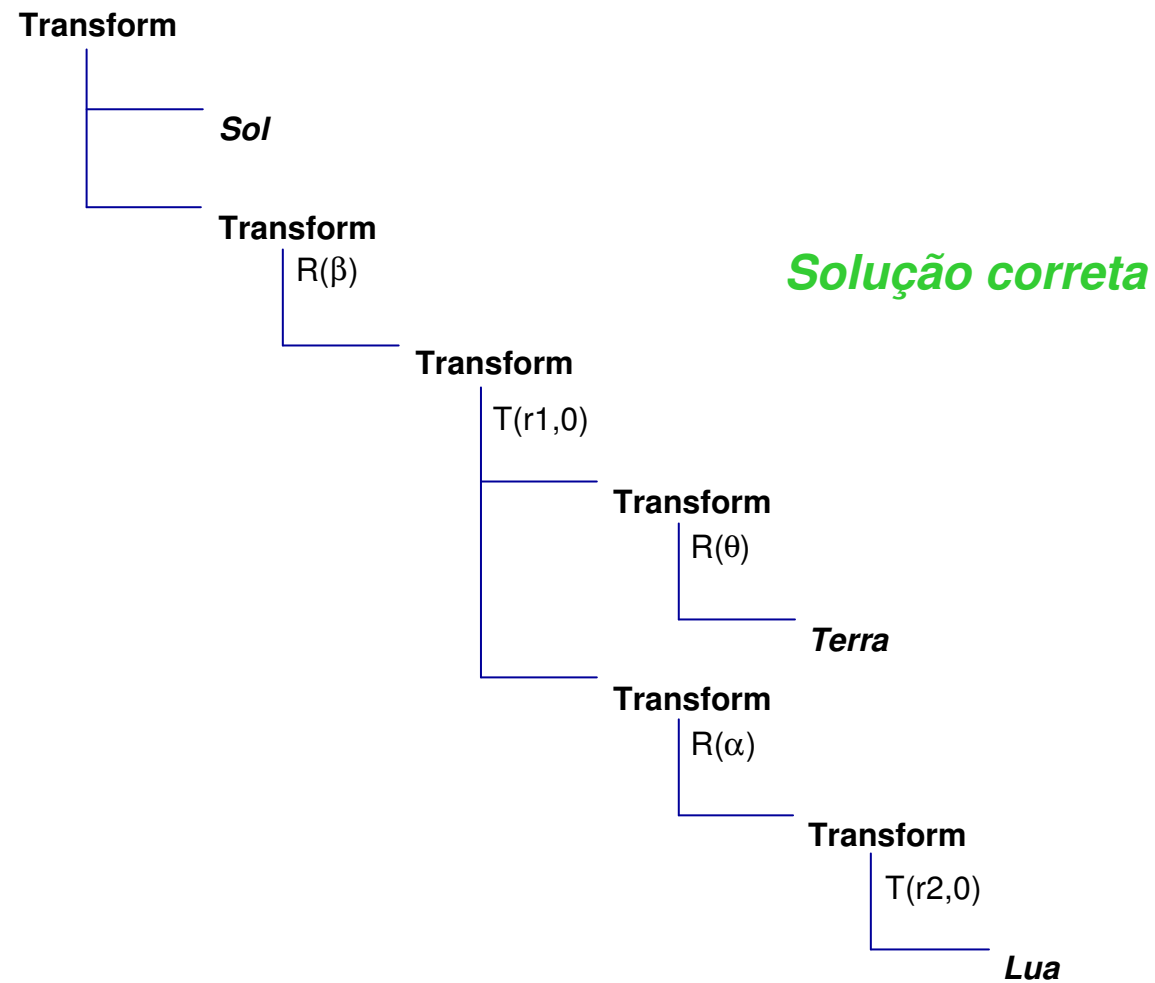


Esta conversão de representação está errada !

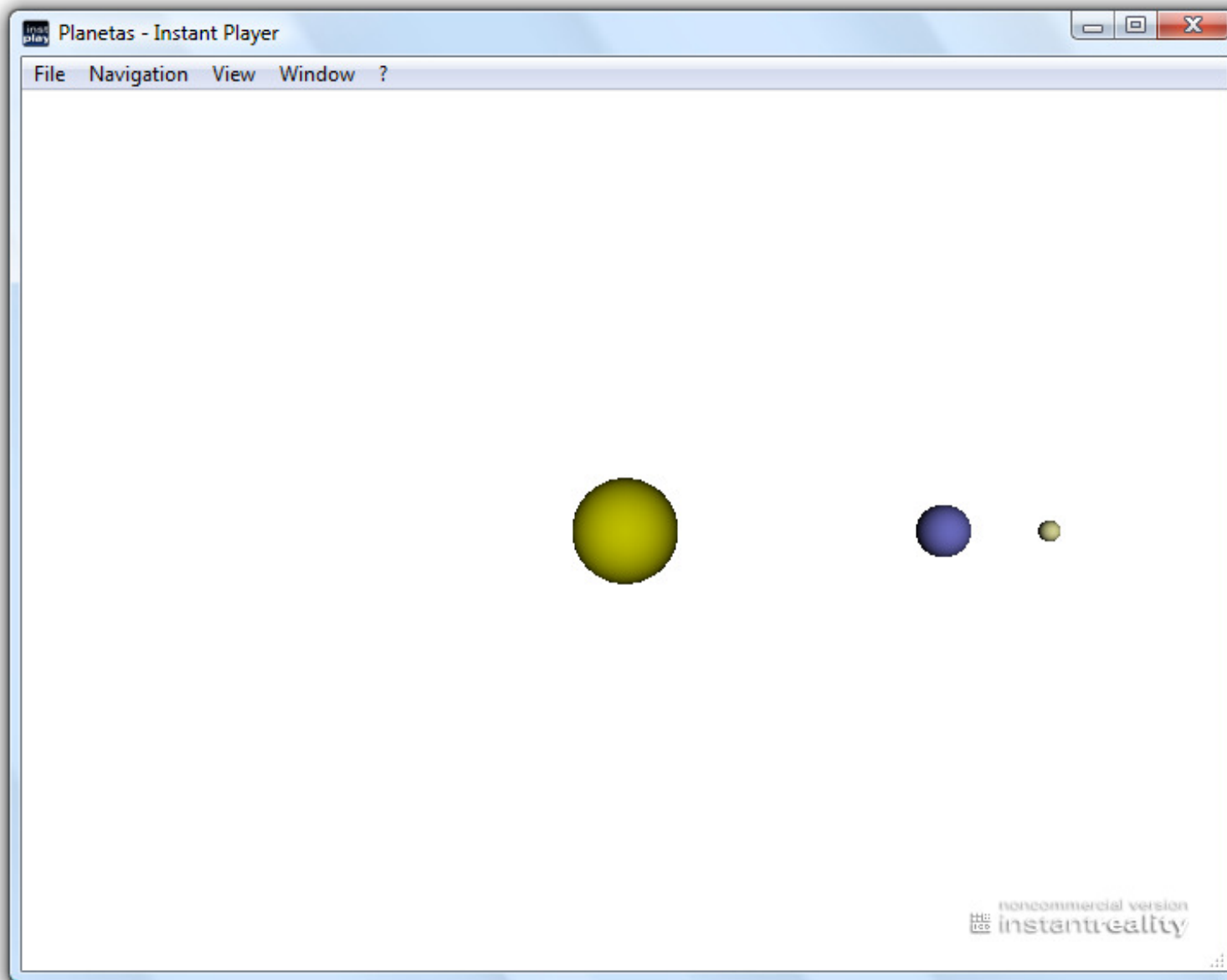
Justificação pelas características de X3D/VRML:

A ordem de execução das transformações, em cada nó, é S – R – T (independentemente da ordem de escrita no nó Transform)

GRAFO DA CENA ORIENTADO PARA X3D/VRML

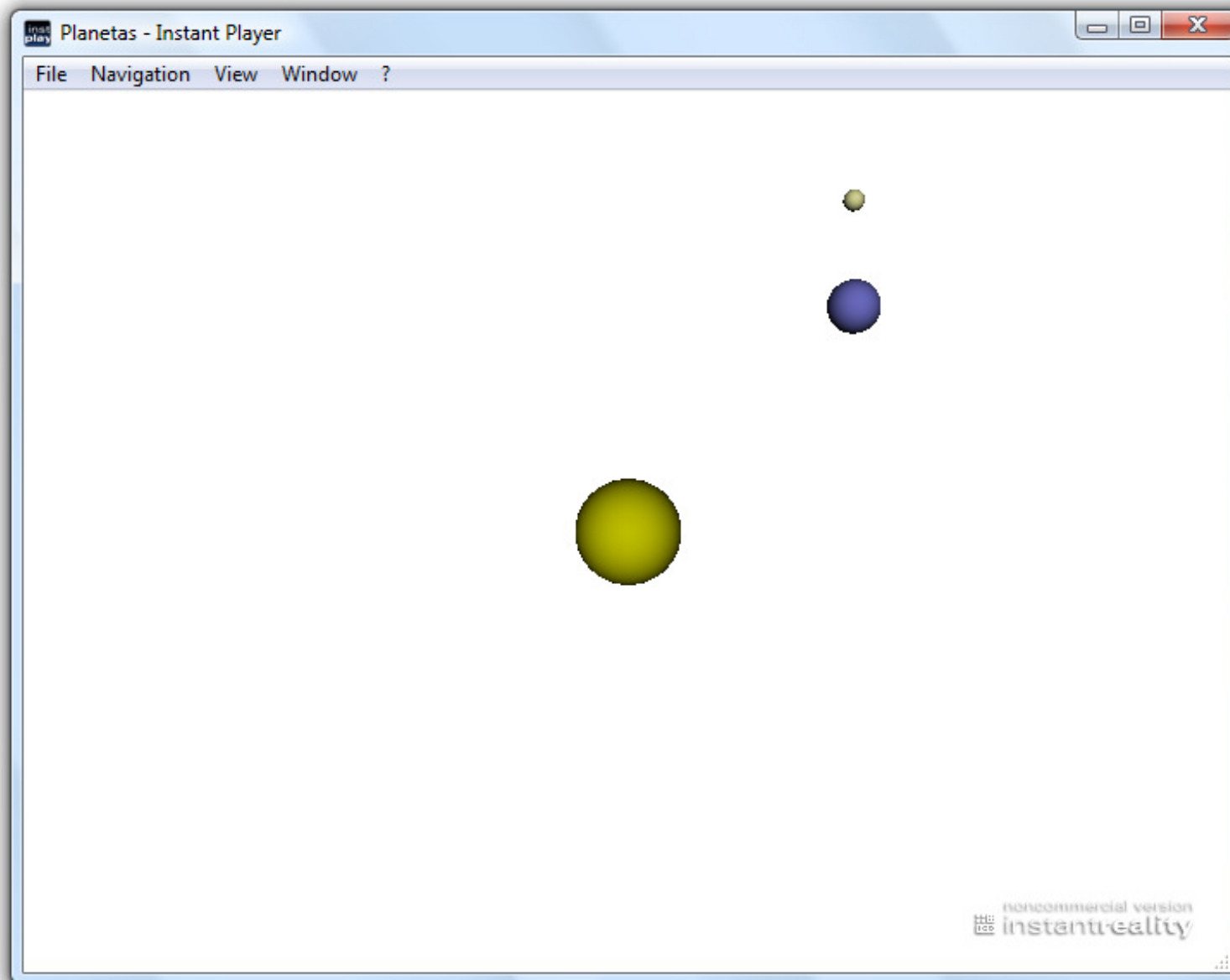


Nota: As transformações com valores por omissão não precisam de ser escritas no grafo.

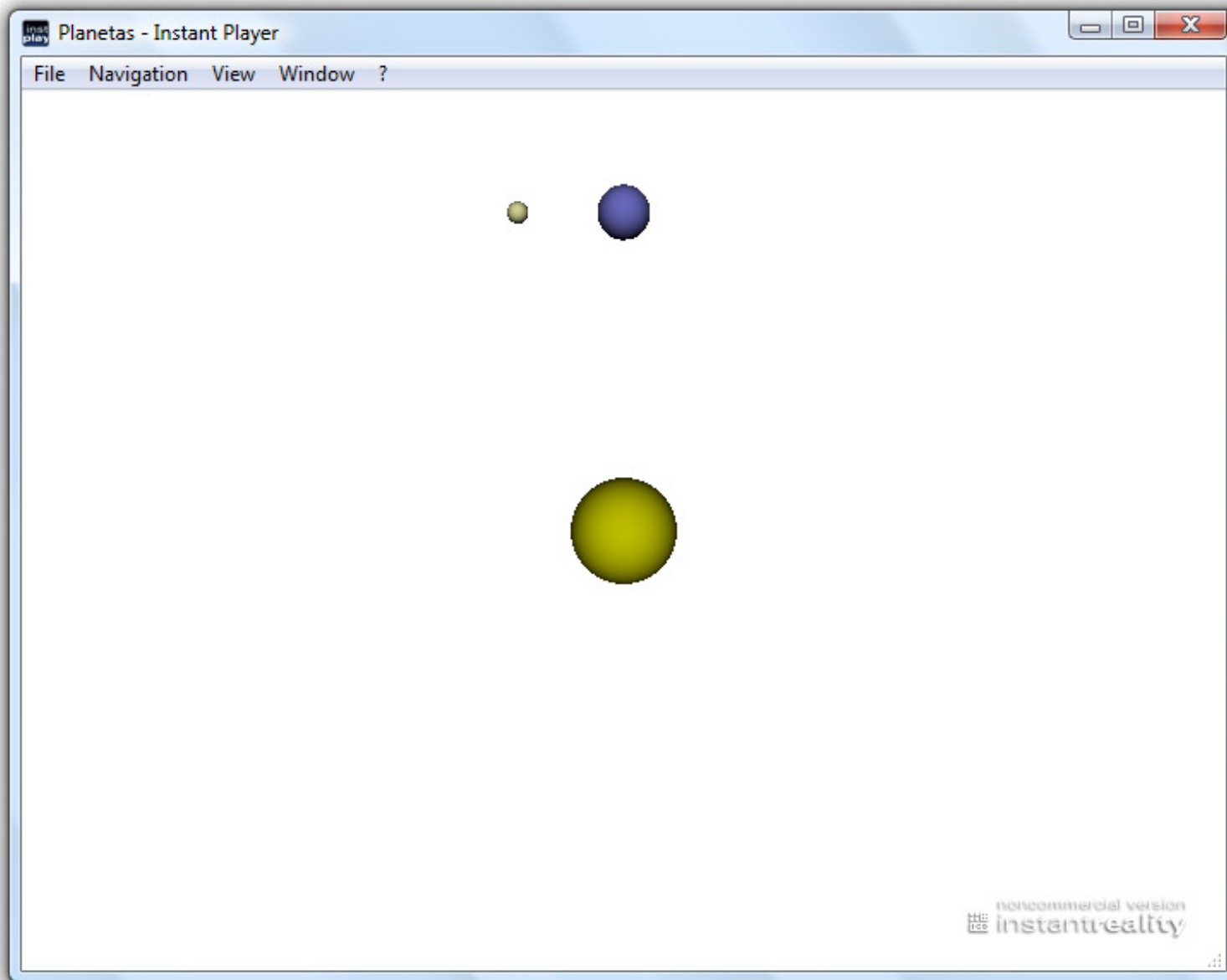


$$\beta = \theta = \alpha = 0^\circ$$

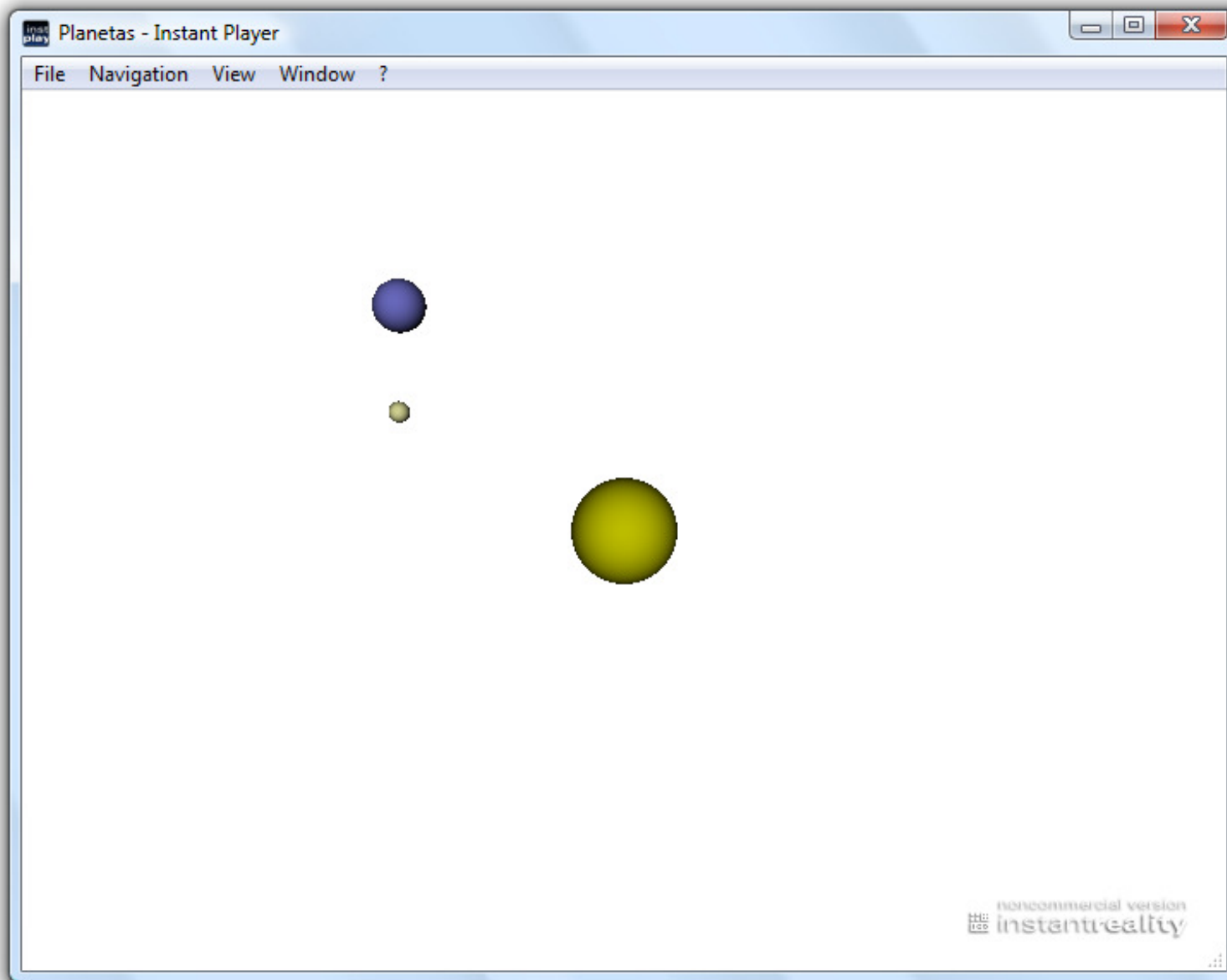
(ponto de vista no eixo Z)



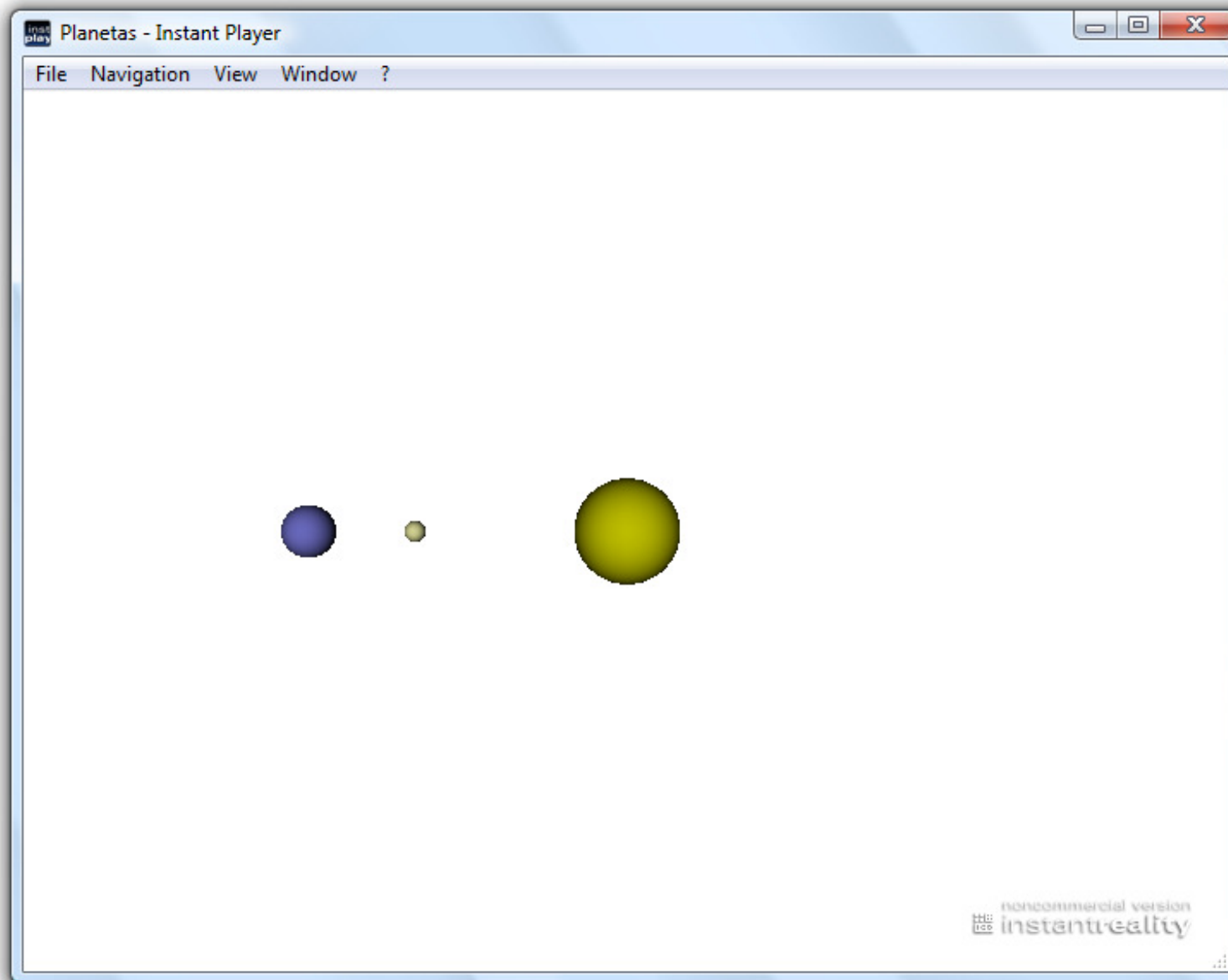
$$\beta = \theta = \alpha = 45^\circ$$



$$\beta = \theta = \alpha = 90^\circ$$

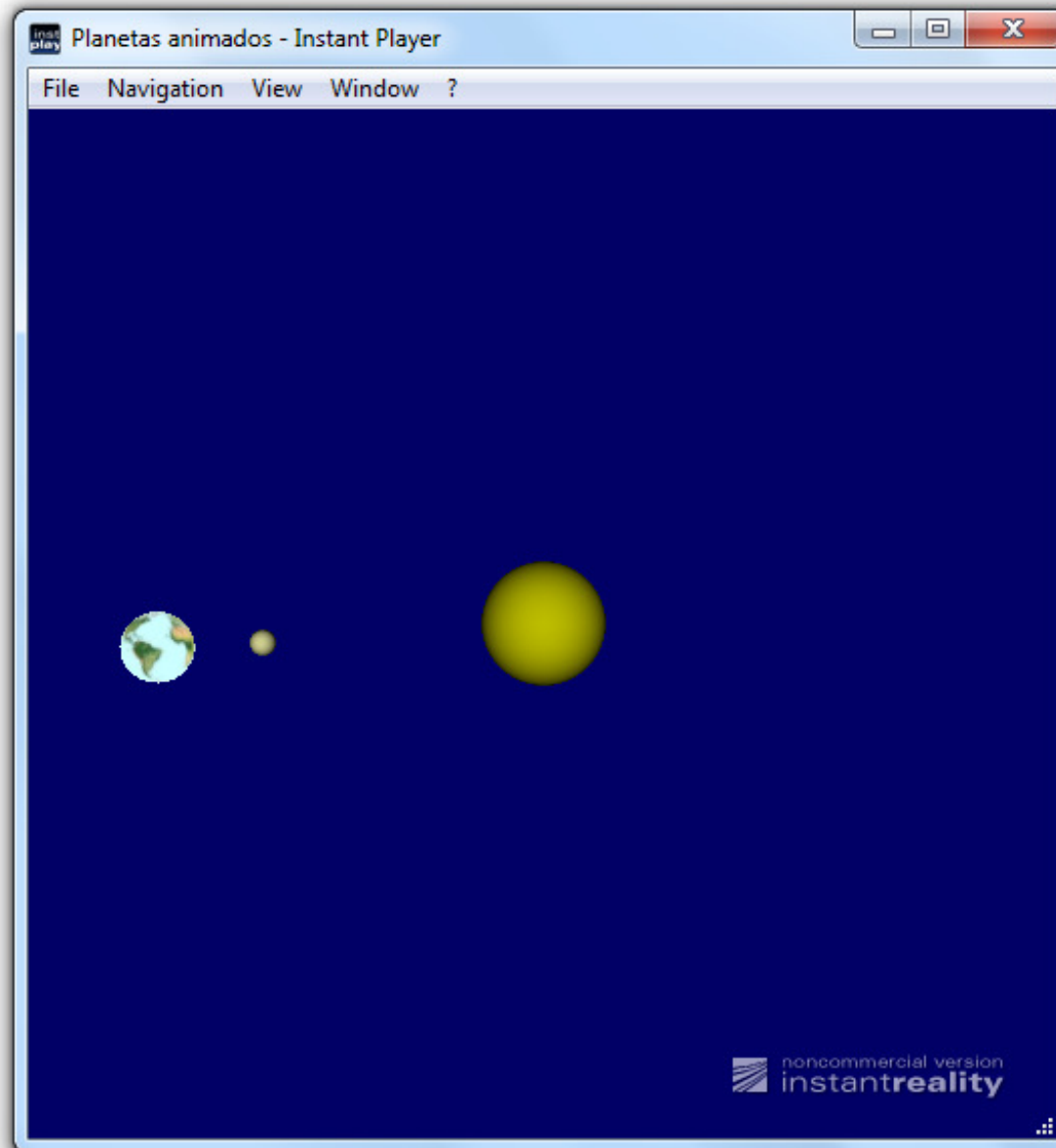


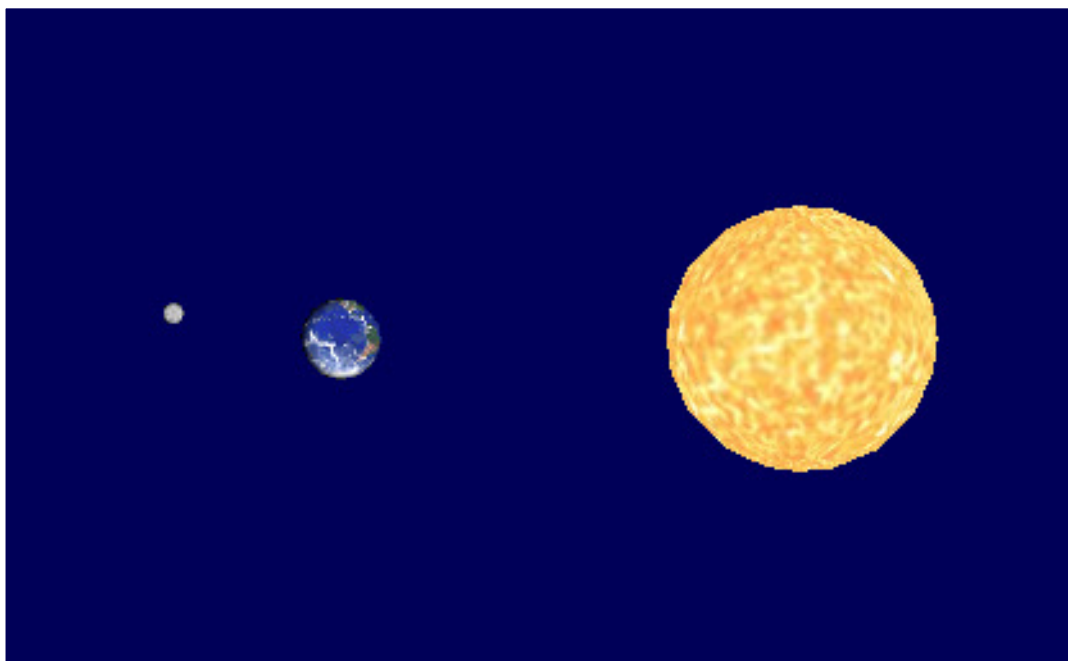
$$\beta = \theta = \alpha = 135^\circ$$



$$\beta = \theta = \alpha = 180^\circ$$

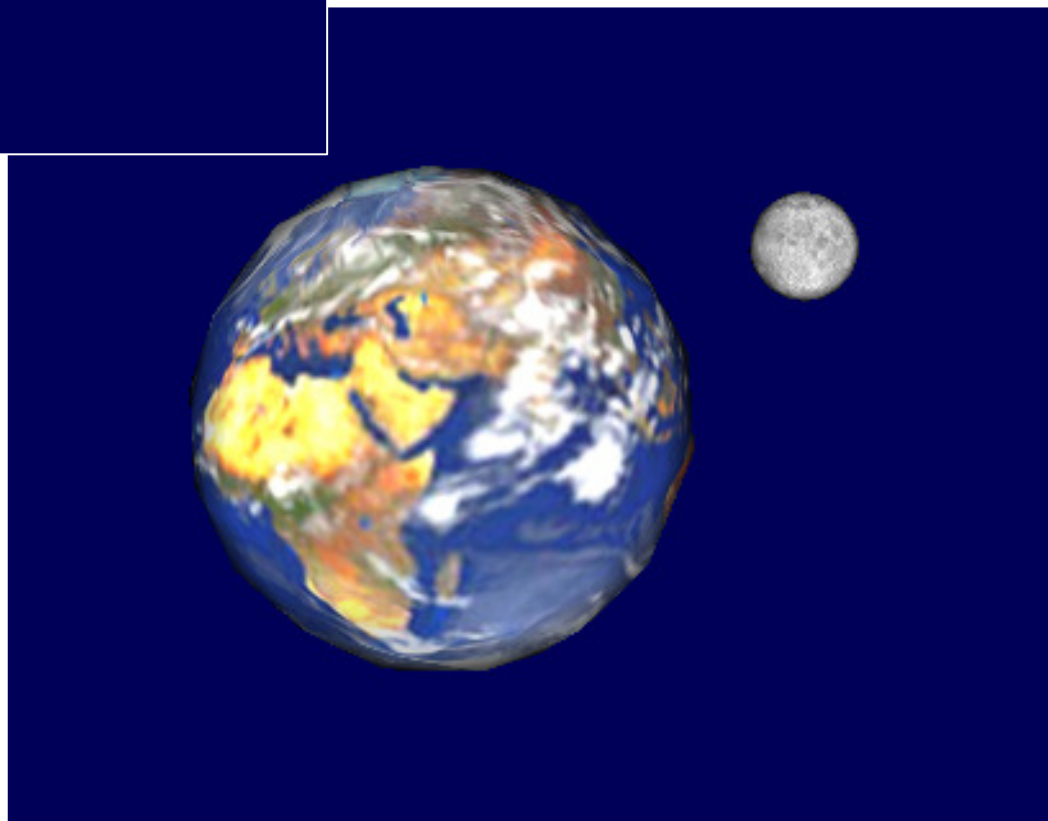
**Screenshot do
exemplo com
animação e
textura no planeta
Terra**
(ponto de vista
aproximadamente
no plano das
órbitas)





Exemplo em VRML,
com texturas e
animação

<http://ctp.di.fct.unl.pt/~ps/galeria/moon/>

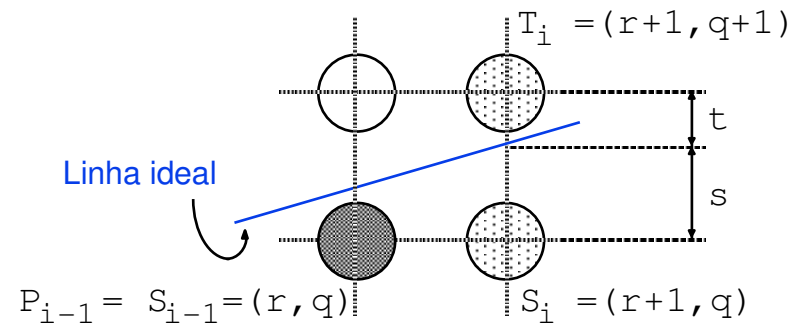


ALGORITMO DE BRESENHAM

Método eficiente [1965] por uso exclusivo de aritmética de inteiros.



Como escolher o pixel no passo i ?



```
Pi = if s-t < 0 then Si  
      else Ti
```

Hipóteses:

$$y = mx + b, \quad \text{com } 0 \leq m \leq 1$$

OBS.: O algoritmo é de grande importância histórica, mas a explicação que se segue é de carácter informativo e fora do programa de CGI.

ALGORITMO DE BRESENHAM

Escolher a designação das coordenadas tal que: $x_1^2 + y_1^2 < x_2^2 + y_2^2$

Mudança de coordenadas:

$$\overline{(x_1, y_1) \quad (x_2, y_2)} \rightarrow \overline{(0, 0) \quad (dx, dy)} \quad \text{em que} \quad \begin{cases} dx = x_2 - x_1 \\ dy = y_2 - y_1 \end{cases}$$

Dedução das fórmulas finais:

$$y = x \, dy/dx \quad \begin{matrix} P_{i-1} = (r, q) \\ \Rightarrow \end{matrix} \quad \begin{aligned} s + q &= (r+1) \, dy/dx \\ t &= 1 - s = 1 + q - (r+1) \, dy/dx \end{aligned}$$

$\Downarrow$

$$s - t = 2 \, (r+1) \, dy/dx - 2q - 1$$

Variável de decisão

$$d_i = dx \, (s - t) = 2 \, dy \, (r+1) - 2 \, q \, dx - dx$$

$$\text{e como} \quad \begin{cases} r = x_{i-1} \\ q = y_{i-1} \end{cases} \quad \text{virá:}$$

$$d_{i+1} = 2 \, dy \, (x_i + 1) - 2 \, y_i \, dx - dx$$

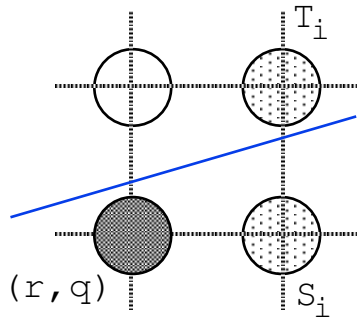
$$d_{i+1} - d_i = 2 \, dy \, (x_i - x_{i-1}) - 2 \, (y_i - y_{i-1}) \, dx$$



$$d_{i+1} = d_i + 2 \, dy - 2 \, (y_i - y_{i-1}) \, dx$$

ALGORITMO DE BRESENHAM

$$d_{i+1} = d_i + 2 dy - 2 (y_i - y_{i-1}) dx$$



$P_i = \text{if } d_i < 0 \text{ then } S_i$

$$y_i = y_{i-1} \quad \wedge \quad d_{i+1} = d_i + 2 dy - 0 = d_i + k_s$$

$\text{else } T_i$

$$y_i = y_{i-1} + 1 \quad \wedge \quad d_{i+1} = d_i + 2 dy - 2 dx = d_i + k_t$$

De $d_{i+1} = 2 dy (x_i + 1) - 2 y_i dx - dx$ resulta o valor inicial ($r=q=0$):

$$\begin{aligned} d_{i=1} &= 2 dy (0+1) - 0 - dx \\ &= 2 dy - dx \end{aligned}$$

ALGORITMO DE BRESENHAM

```
procedure LINE_BRESENHAM (x1,y1,x2,y2,value: integer);

    { Condição de aplicabilidade:  $0 \leq m \leq 1$ 
      Para resolver nos outros octantes faz-se uma conversão para este! }

var    dx,dy,ks,kt,d,x,y,x_end: integer;

begin
    dx := abs(x2-x1);    dy := abs(y2-y1);    d := 2*dy - dx;
    ks := 2*dy;    kt := 2*(dy - dx);
    if x1 > x2 then begin x := x2;    y := y2;    x_end := x1    end
                    else begin x := x1;    y := y1;    x_end := x2    end;
    WRITE_PIXEL(x,y,value);
    while x < x_end do
        begin
            x := x + 1;
            if d < 0 then d := d + ks
                        else begin y := y + 1;    d := d + kt    end;
            WRITE_PIXEL(x,y,value)
        end
    end;
end;
```

ALGORITMO DO PONTO MÉDIO

“MIDPOINT ALGORITHM” [Pitteway 65 / Van Aken 85]

Justificação: O algoritmo de Bresenham não é de generalização fácil para as cónicas em geral.

Aplicação ao caso de uma reta, cuja equação é

$$F(x, y) = a x + b y + c = 0$$

Comparando com a forma explícita ($0 \leq m \leq 1$)

$$y = m x + B = (dy/dx) x + B$$

determinam-se os coeficientes da forma implícita:

$$F(x, y) = dy x - dx y + B dx = 0$$

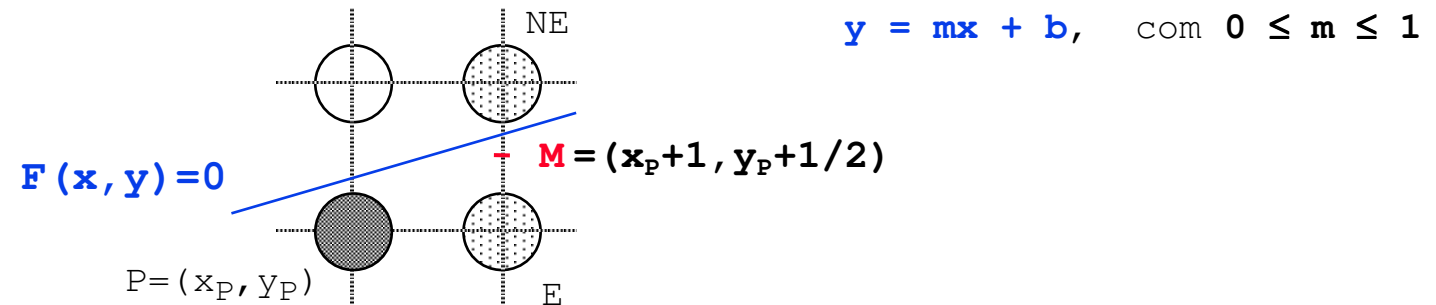


$a = dy$
$b = - dx$
$c = B dx$

Hipótese: $0 \leq m \leq 1$ e $dy \geq 0$, por escolha da ordem entre os pontos que definem o segmento de reta.

Exercício: Qual o sinal de F para um ponto acima da reta? E abaixo?

ALGORITMO DO PONTO MÉDIO



Comparação no ponto médio **M** para se conhecer o novo ponto **P** com abscissa $x_P + 1$:

```
P = if F(M) < 0 then E
      else NE
```

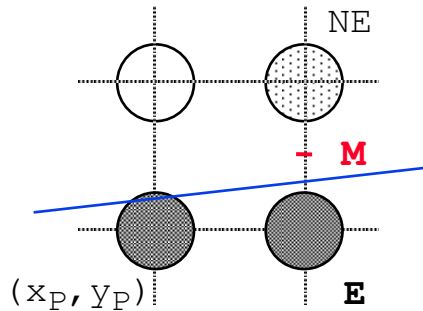
$$F(M) = a(x_P + 1) + b(y_P + 1/2) + c = d'$$

Variável de decisão

```
P = if d' < 0 then E
      else NE
```

ALGORITMO DO PONTO MÉDIO

Como $d' = F(M)$, no passo seguinte e quando

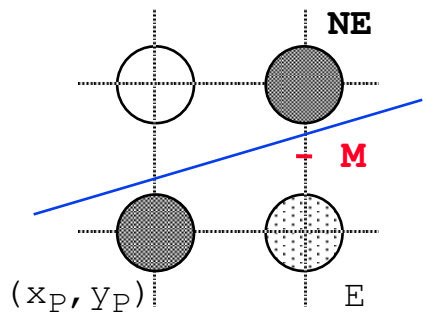


... a escolha for E (novo M , para x_P+2 , manterá a ordenada):

$$d' = F(x_P+2, y_P+1/2) = a(x_P+2) + b(y_P+1/2) + c$$

ou seja, numa instrução de atribuição:

$$d' := d' + a$$



... a escolha for NE (a ordenada do novo M , para x_P+2 , será incrementada):

$$d' = F(x_P+2, y_P+3/2) = a(x_P+2) + b(y_P+3/2) + c$$

ou seja, numa instrução de atribuição:

$$d' := d' + a + b$$

Como, por hipótese, as coordenadas das extremidades do segmento de reta são valores inteiros, a e b também o são.

Inicialização da variável de decisão no ponto $P(x_1, y_1)$:

$$\begin{aligned} F(x_1+1, y_1+1/2) &= a(x_1+1) + b(y_1+1/2) + c \\ &= a x_1 + b y_1 + c + a + b/2 \\ &= F(x_1, y_1) + a + b/2 \\ &= a + b/2 \end{aligned}$$

ALGORITMO DO PONTO MÉDIO

O valor inicial da variável de decisão seria:

$$d' = F(x_1+1, y_1+1/2) = a + b/2$$

$$\begin{aligned} a &= dy \\ b &= -dx \end{aligned}$$

Mas, para se trabalhar exclusivamente com valores inteiros, multiplique-se por 2 e substitua-se $2d'$ por d :

$$d = 2a + b = 2dy - dx$$

Conclusão:

```
P := if d < 0 then { E }
```

```
  x := x + 1 ^ d := d + 2 * dy
```

```
  else { NE }
```

```
  x := x + 1 ; y := y + 1 ^ d := d + 2 * (dy - dx)
```

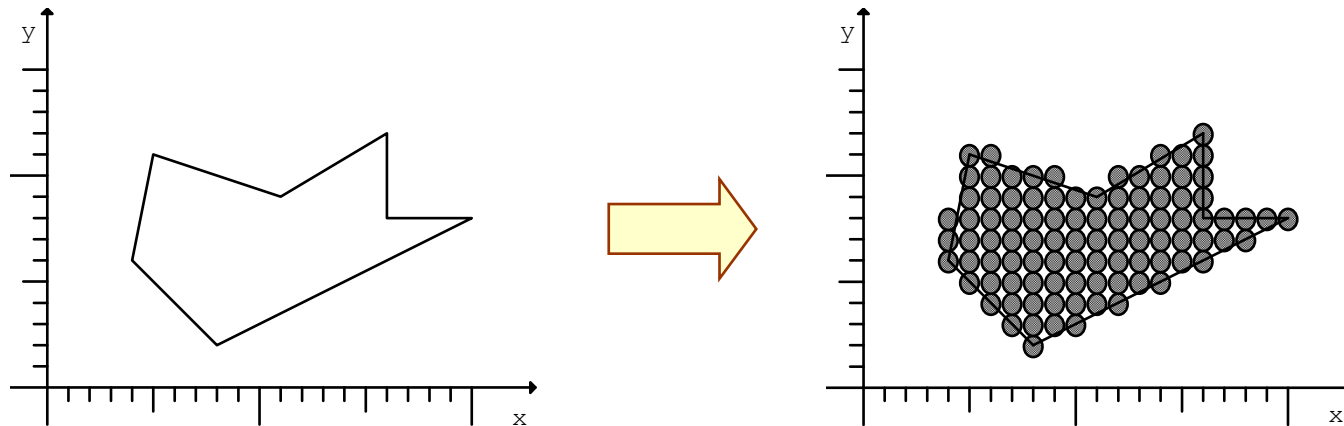
```
WRITE_PIXEL(x, y, value)
```

POLÍGONOS

FILL AREA

Objetivo: Preenchimento de um polígono definido pela sequência dos respectivos vértices (pontos 2D), independentemente dos valores pré-existentes na memória de refrescamento.

Um exemplo concreto, numa 1.^a aproximação executada por preenchimento manual “a olho”:



[mais adiante referido como **exemplo de referência** e que se verá não estar inteiramente correto!]

FILL AREA

ALGORITMO PAR-ÍMPAR (*Even-Odd*)

É um algoritmo de VARRIMENTO por linhas (*scan-lines*), em que os pixels são testados, não individualmente, mas tirando partido da...

coerência por linha de varrimento

Se um dado pixel numa linha pertence ao polígono, é muito provável que os pixels vizinhos também pertençam.

coerência por aresta do polígono

Se uma aresta intersecta uma dada linha de varrimento, é muito provável que intersecte também as linhas vizinhas.

Principais etapas do algoritmo de FILL AREA

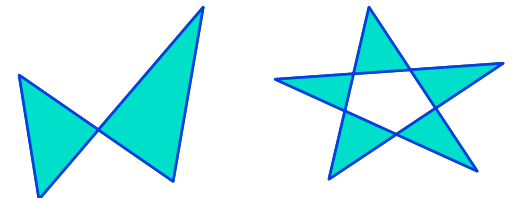
Para cada linha de varrimento

- 1 calcular todas as intersecções com as arestas do polígono.

As arestas horizontais são excluídas do algoritmo

- 2 ordenar essas intersecções segundo os valores crescentes da abcissa.
- 3 preencher todos os pixels entre pares de intersecções.

NB: O algoritmo também funciona para polígonos com arestas que se cruzem (os pontos fronteira destes dois exemplos não foram gerados pelo algoritmo).



Is a given point **P** inside or outside a Polygon ?

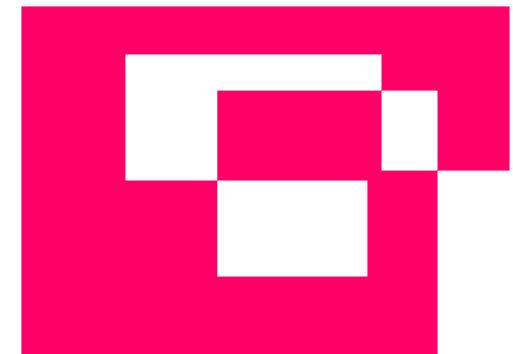
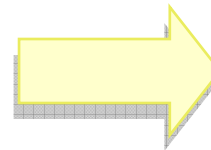
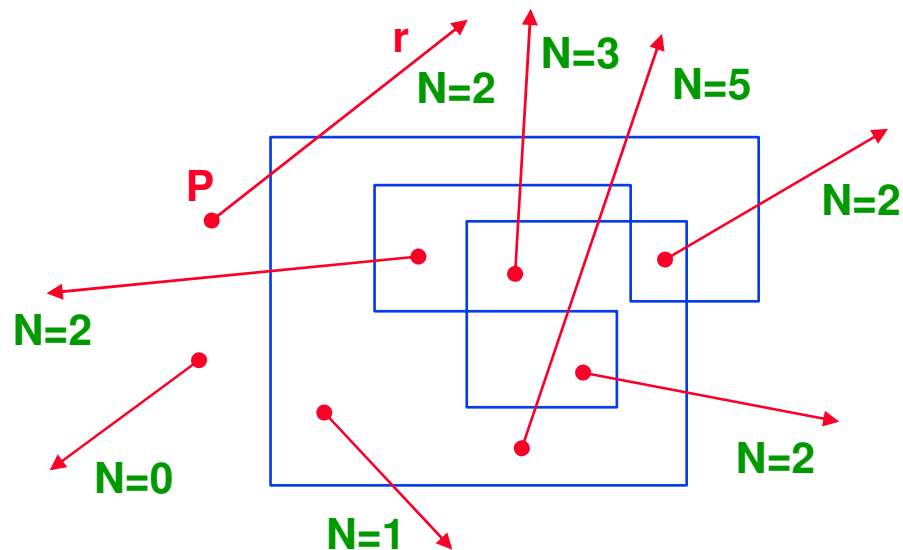
- 1 Let **r** be a half of an arbitrary straight line starting at **P**.

Note: avoid passing **r** trough a vertex.

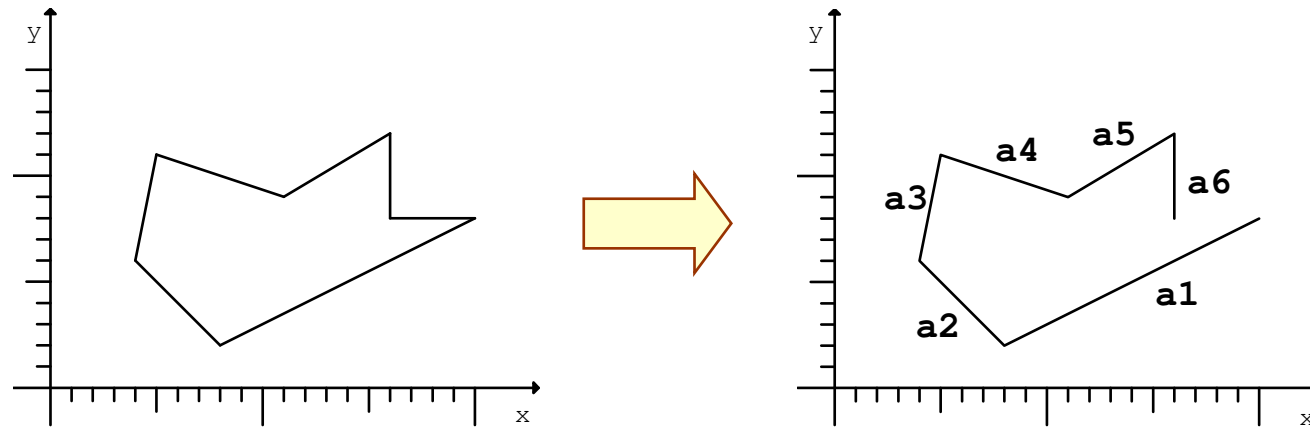
- 2 Count the total number **N** of intersections with the edges of the polygon.

- 3 If **N** is even or **0** then **P** is outside the polygon
else **P** is inside the polygon.

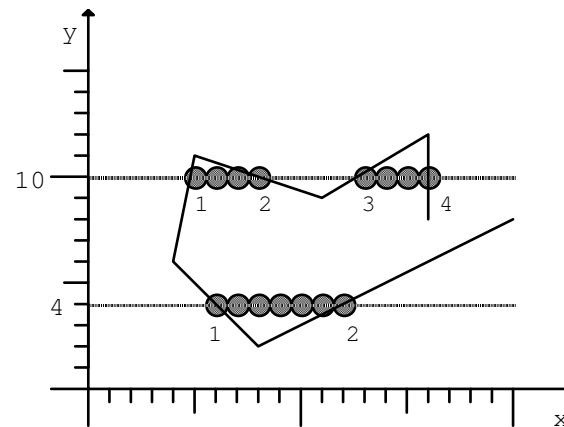
Examples:

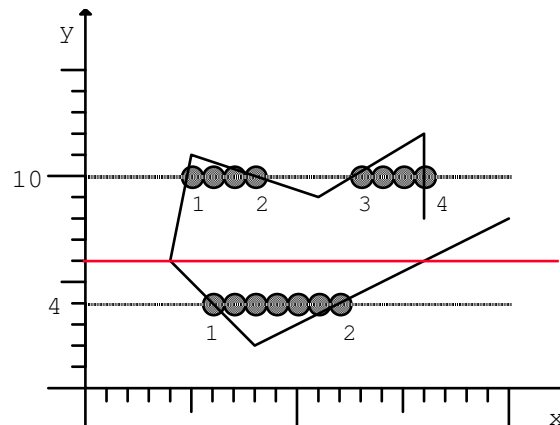


Exemplo de exclusão de uma aresta, por ser horizontal:



O preenchimento ao longo das linhas $y=4$ e $y=10$:





PROBLEMA NA PASSAGEM POR ALGUNS VÉRTICES

A linha $y=6$ dá origem a um número ímpar de intersecções,
pelo que o preenchimento não dará o resultado pretendido!

OBS: Nos mínimos e máximos locais não se coloca tal problema (p.ex. $y=9$).

RESOLUÇÃO

O preenchimento faz-se entre cada par de intersecções, mas excluídas as arestas
para as quais a ordenada da linha de varrimento seja a sua ordenada máxima.

OBS: Para $y=6$, a aresta **a2** ficaria excluída, reduzindo-se as intersecções a duas.

ESTRUTURAS DE DADOS

REGISTOS DAS ARESTAS

aresta	y _{max}	x	1/m	apontador
a1	8	8	2	
a2	6	8	-1	
a3	11	4	1/5	
a4	11	11	-3	
a5	12	11	5/3	
a6	12	16	0	

↑
Chave de
identificação

↑
Abcissa da intersecção da aresta com a linha
de varrimento (informação não estática,
inicializada para a linha de ordenada mínima
que intersecta essa aresta)

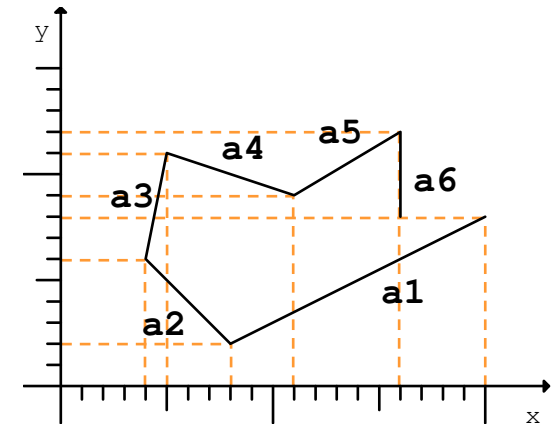
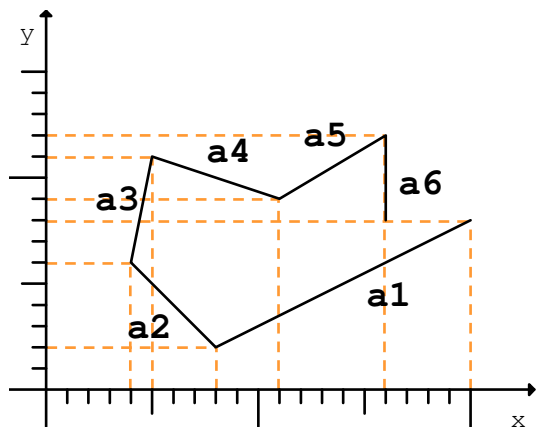


TABELA DE ARESTAS (TA)



Linha y	Novas arestas intersectadas
0	-
1	-
2	a1 → a2
3	-
4	-
5	-
6	a3
7	-
8	a6
9	a4 → a5
10	-
11	-
12	-
...	-

Havendo mais do que um elemento em cada entrada, a respectiva lista de arestas ordena-se crescentemente segundo as abcissas (Em cada um dos dois casos do exemplo acontece as abcissas terem valores iguais por partilha de vértice entre arestas, pelo que as duas listas poderiam ter ordem contrária).

TABELA DE ARESTAS ATIVAS (TAA)

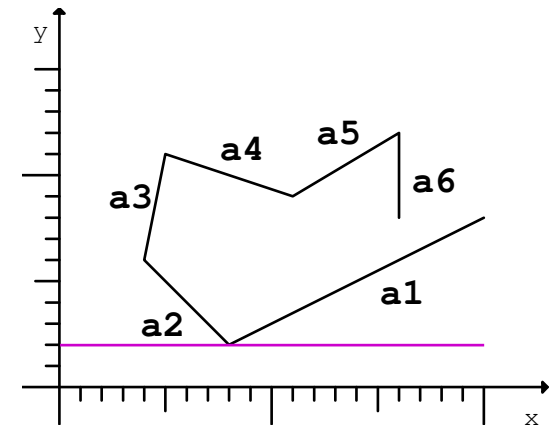
Estrutura dinâmica, sempre ordenada pelos valores de abcissa, para cada linha de varrimento.

Partindo de $y = mx + b$, a atualização para a linha seguinte (coerência por arestas) pode obter-se com uma fórmula de recorrência:

$$y_{i+1} - y_i = mx_{i+1} - mx_i = 1 \quad \rightarrow \quad x_{i+1} = x_i + 1/m$$

TAA
y=2

a1	8	8	2	
a2	6	8	-1	



Introduzem-se na TAA as arestas lidas na TA para a linha de varrimento corrente.

TABELA DE ARESTAS ATIVAS (TAA)

Estrutura dinâmica sempre ordenada pelos valores de abcissa, para cada linha de varrimento.

Passa-se para a linha seguinte (coerência por arestas) atualizando-se a abcissa

$$x_{i+1} = x_i + 1/m$$

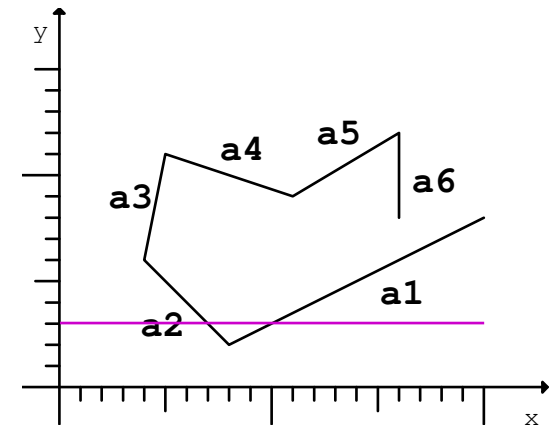
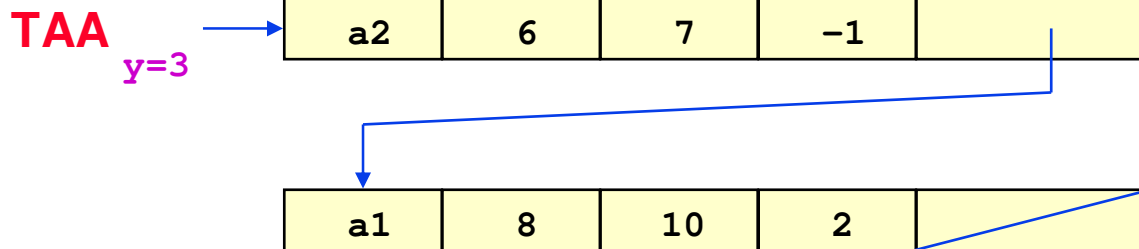


TABELA DE ARESTAS ATIVAS (TAA)

Estrutura dinâmica sempre ordenada pelos valores de abcissa, para cada linha de varrimento.

Passa-se para a linha seguinte (coerência por arestas) atualizando-se a abcissa

$$x_{i+1} = x_i + 1/m$$

TAA $y=4$ →

a2	6	6	-1	
a1	8	12	2	

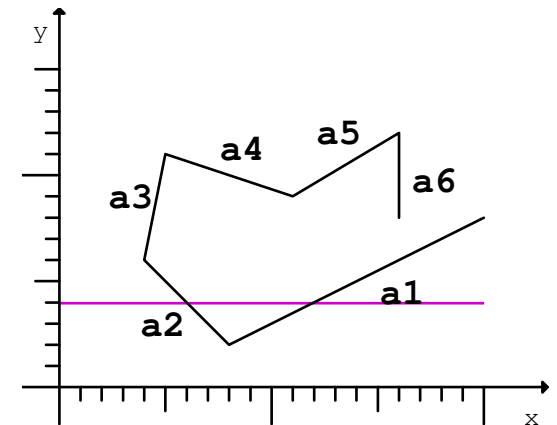


TABELA DE ARESTAS ATIVAS (TAA)

Estrutura dinâmica sempre ordenada pelos valores de abcissa, para cada linha de varrimento.

Passa-se para a linha seguinte (coerência por arestas) atualizando-se a abcissa

$$x_{i+1} = x_i + 1/m$$

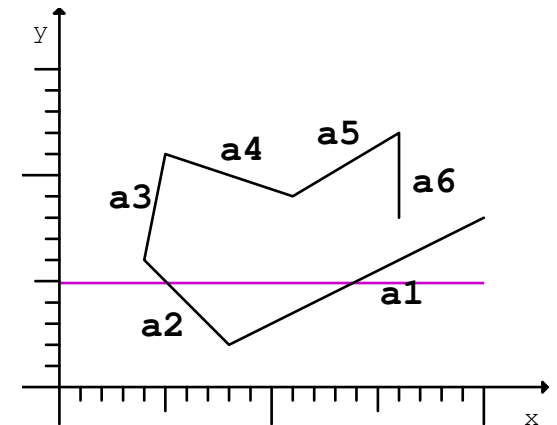
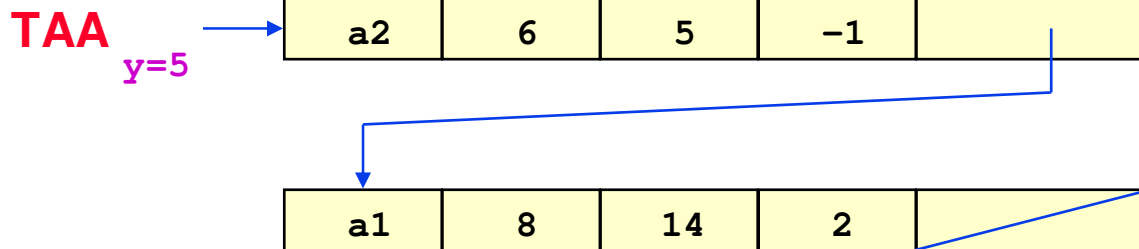


TABELA DE ARESTAS ATIVAS (TAA)

Estrutura dinâmica sempre ordenada pelos valores de abcissa, para cada linha de varrimento.

Passa-se para a linha seguinte (coerência por arestas) atualizando-se a abcissa

$$x_{i+1} = x_i + 1/m$$

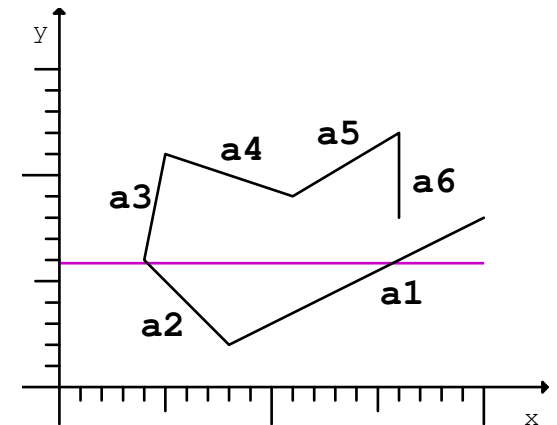
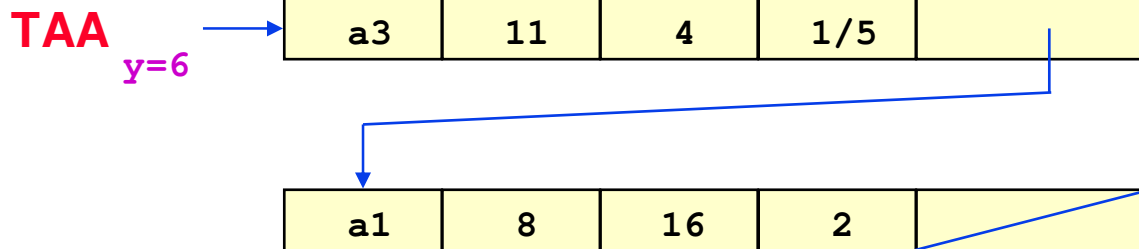


TABELA DE ARESTAS ATIVAS (TAA)

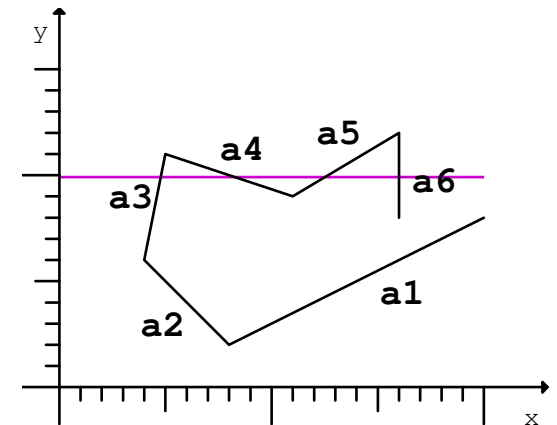
Estrutura dinâmica sempre ordenada pelos valores de abcissa, para cada linha de varrimento.

Passa-se para a linha seguinte (coerência por arestas) atualizando-se a abcissa

$$x_{i+1} = x_i + 1/m$$

TAA $y=10$ →

a3	11	24/5	1/5	
a4	11	8	-3	
a5	12	38/3	5/3	
a6	12	16	0	



Algoritmo de FILL AREA (*Even-Odd*)

- Construir TA (Seja Max a última entrada não vazia)
- Inicializar TAA com null
- Inicializar y com a primeira entrada na TA não vazia, caso exista, senão com Max+1

Enquanto TAA $\neq$ null ou $y < \text{Max}+1$

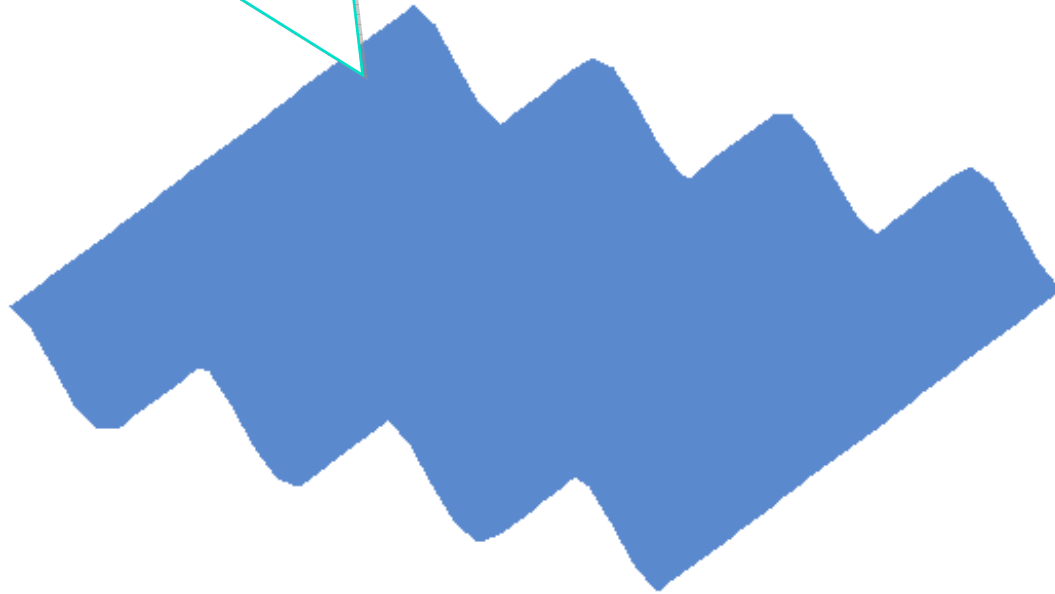
- Eliminar de TAA as arestas com $y_{\text{max}} == y$
- Atualizar TAA com as arestas referidas em TA[y] e ordenar a lista em x
- Percorrer TAA para preencher as carreiras de pixels entre os valores de x de cada par de arestas
- Incrementar y
- Para cada aresta em TAA, fazer $x = x + 1/m$

OBS.:

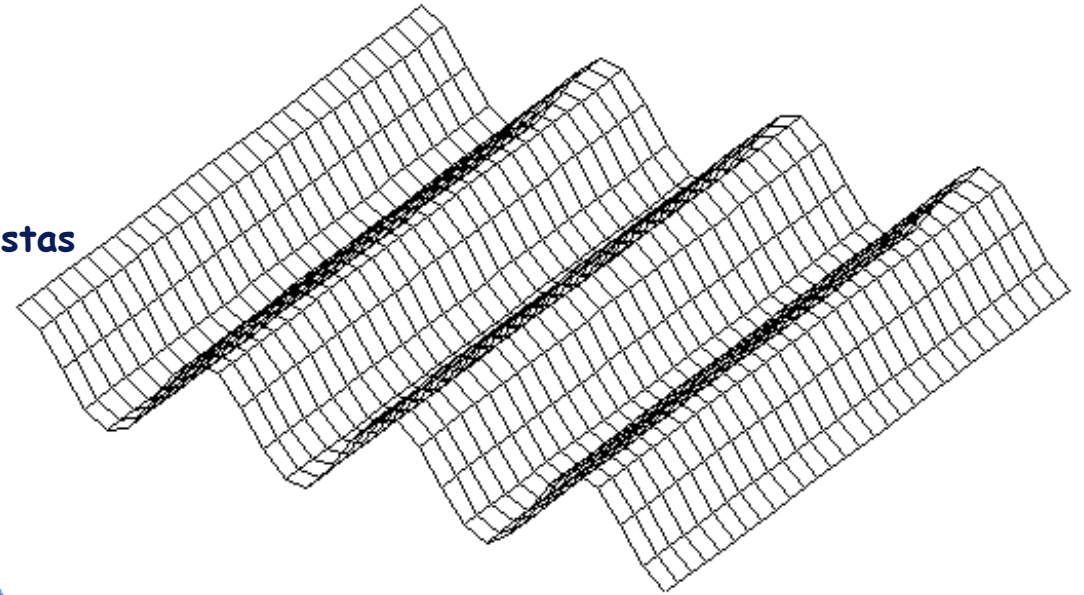
O algoritmo de FILL AREA atua sobre os pixels que se considera serem o “*interior*” do polígono. Pretendendo-se visualizar também as arestas, elas deverão ser rasterizadas como segmentos de reta imediatamente a seguir à aplicação de FILL AREA.

Aplicação em Malhas de Polígonos

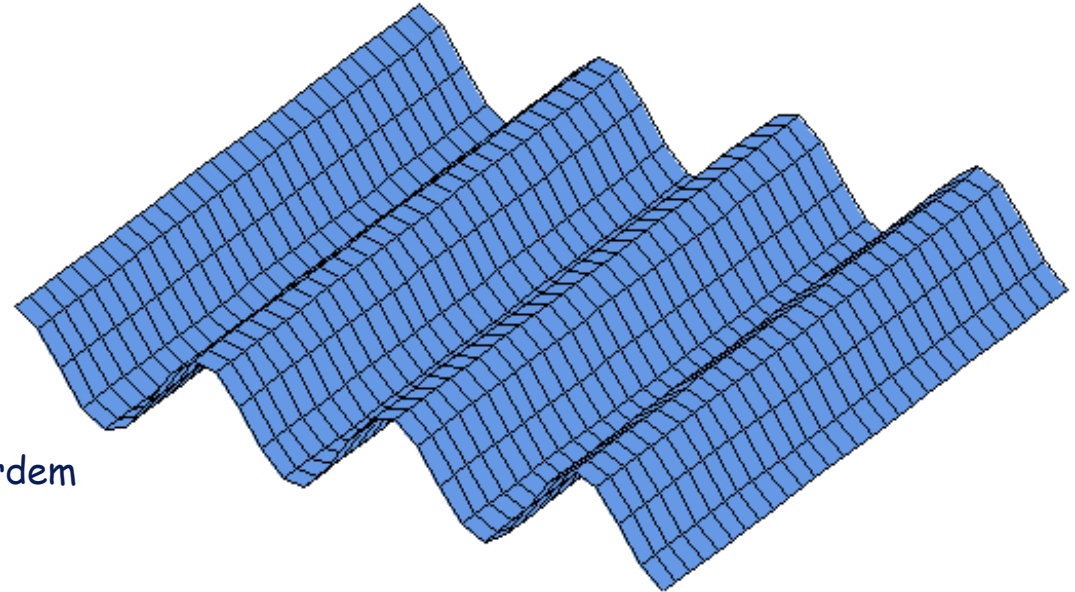
Para se obter este resultado, o algoritmo não só deverá pintar o interior de cada polígono, como também parte da sua fronteira!



Só Arestas

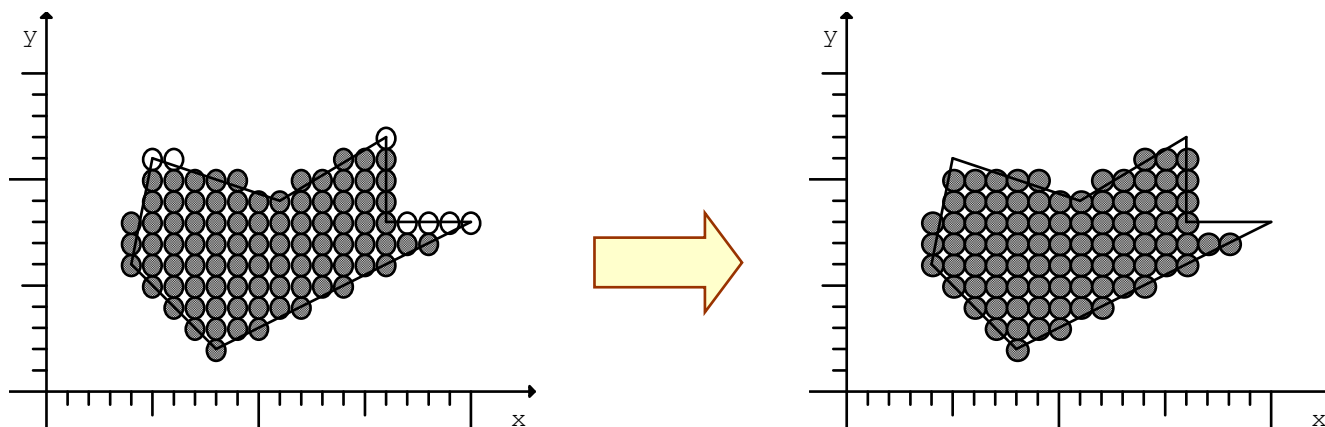


Só FILL AREA



FILL AREA + Arestas
(tratamento repartido, por ordem
não arbitrária)

Primeira correção do exemplo de referência, de acordo com o algoritmo dado:



Problemas com a fronteira do preenchimento

- Acumulação de erros por $x+1/m$ nem sempre ser um inteiro.

Resolução:

Trabalhar com os números na forma de fracção.

Exemplos com arestas para vários valores de linhas de varrimento:

a3

y	6	7	8	9	10	11
x	4	$4+1/5$	$4+2/5$	$4+3/5$	$4+4/5$	5

m=5

a5

y	9	10	11	12
x	11	$11+5/3 = 12+2/3$	$12+7/3 = 14+1/3$	$14+6/3 = 16$

m=3/5

Problemas com a fronteira do preenchimento

- Garantir que os limites das carreiras de pixels não são pontos exteriores ao polígono.

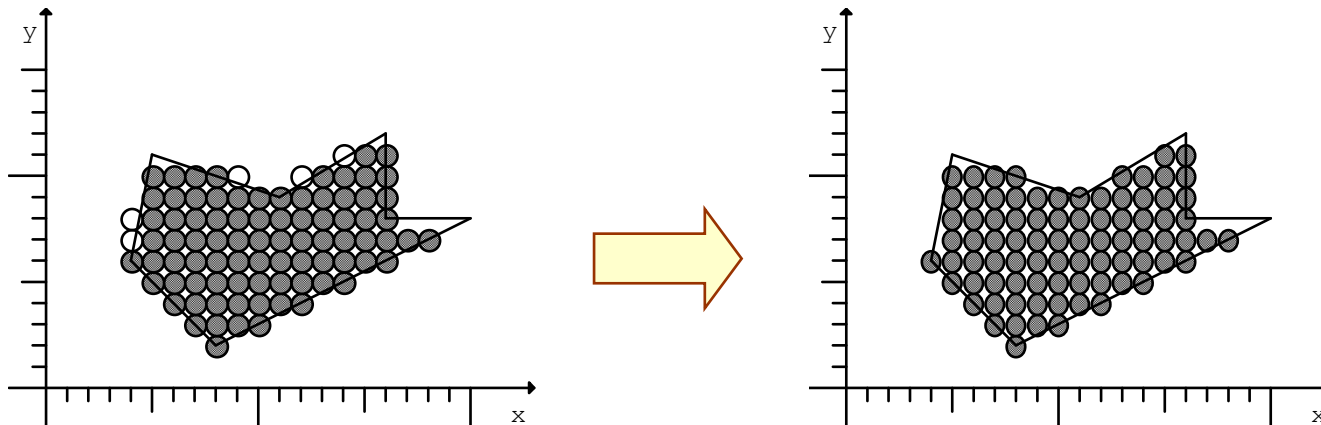
Não sendo x inteiro e caminhando-se, na linha de varrimento, do...

... exterior para o interior do polígono (Ex.: a3)

→ [arredondamento para cima](#)

... interior para o exterior do polígono

→ [truncatura](#)

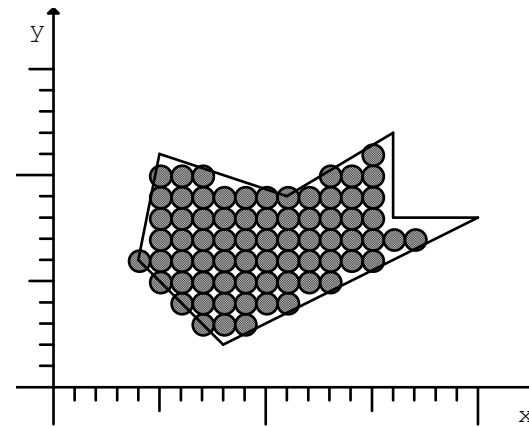
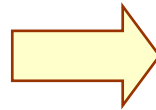
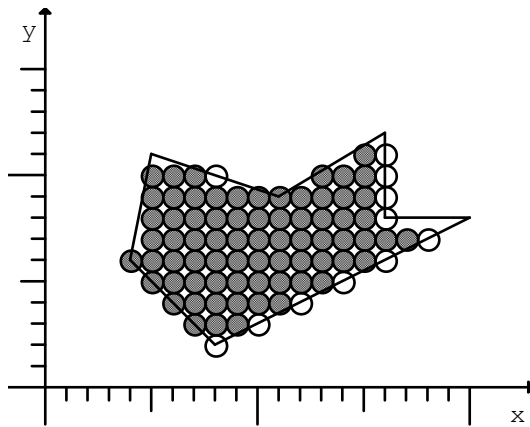


[Repare-se nos valores de x arredondados para 5 em a3, mas todas as intersecções em a1 dão valores inteiros a x]

Problemas com a fronteira do preenchimento

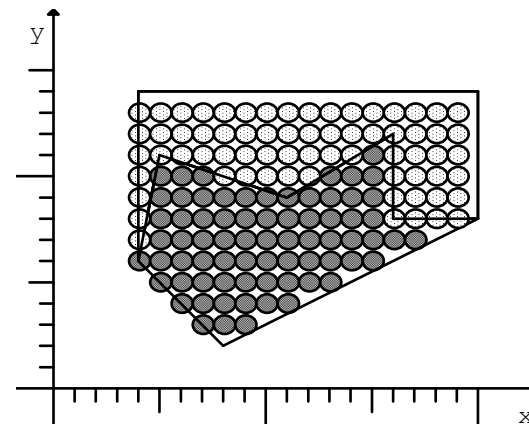
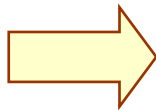
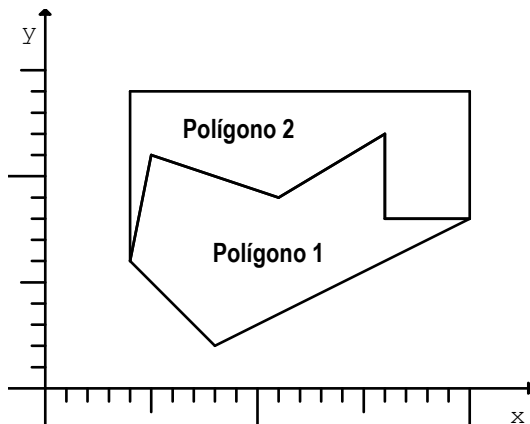
- E se a intersecção der um valor inteiro para x ? É preciso evitar sobreposição com polígonos vizinhos!

Uma convenção possível: { Se é o pixel à esquerda na carreira (Ex.: **a2**) → o pixel é considerado 'interior'
Se é o pixel à direita na carreira (Ex.: **a1**) → o pixel é considerado exterior



Prioritário em relação ao anterior

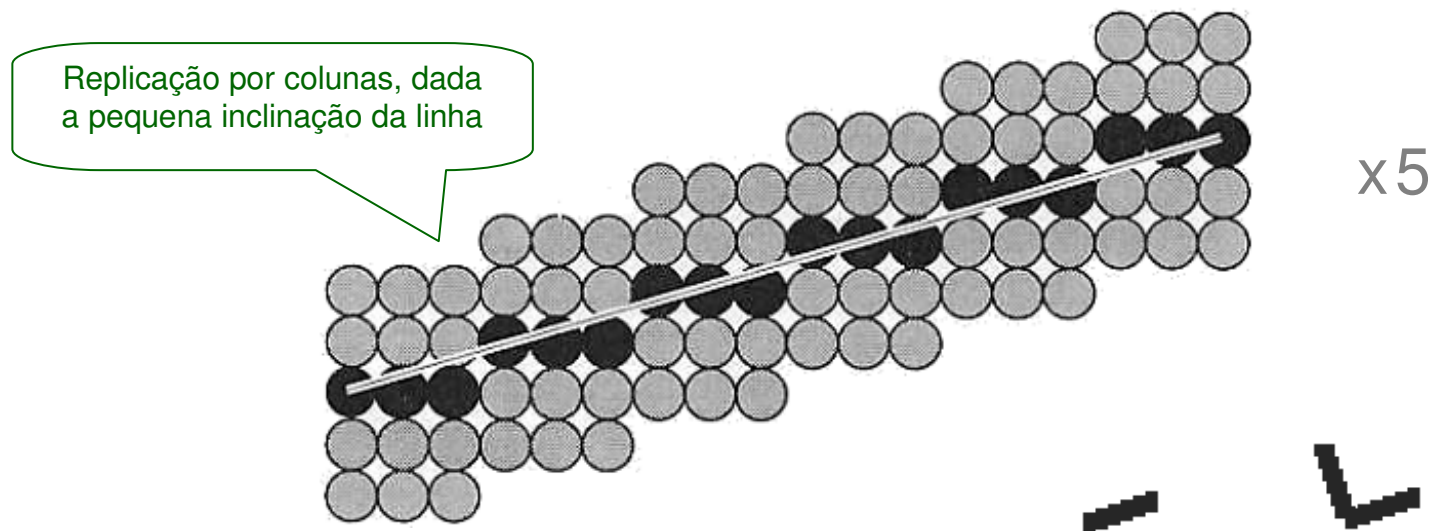
Aplicação: As malhas de polígonos são bem e completamente preenchidas, sem sobreposições.



Grossura de Linhas (e de Curvas, em geral)

(A) REPLICAÇÃO DE PIXELS (POR COLUNAS OU POR LINHAS)

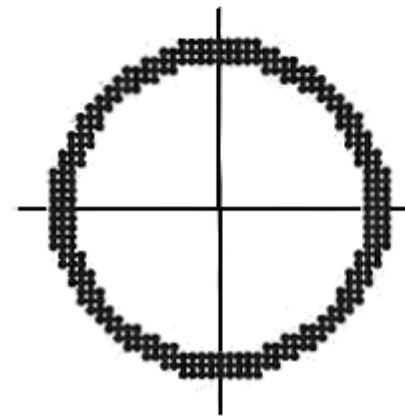
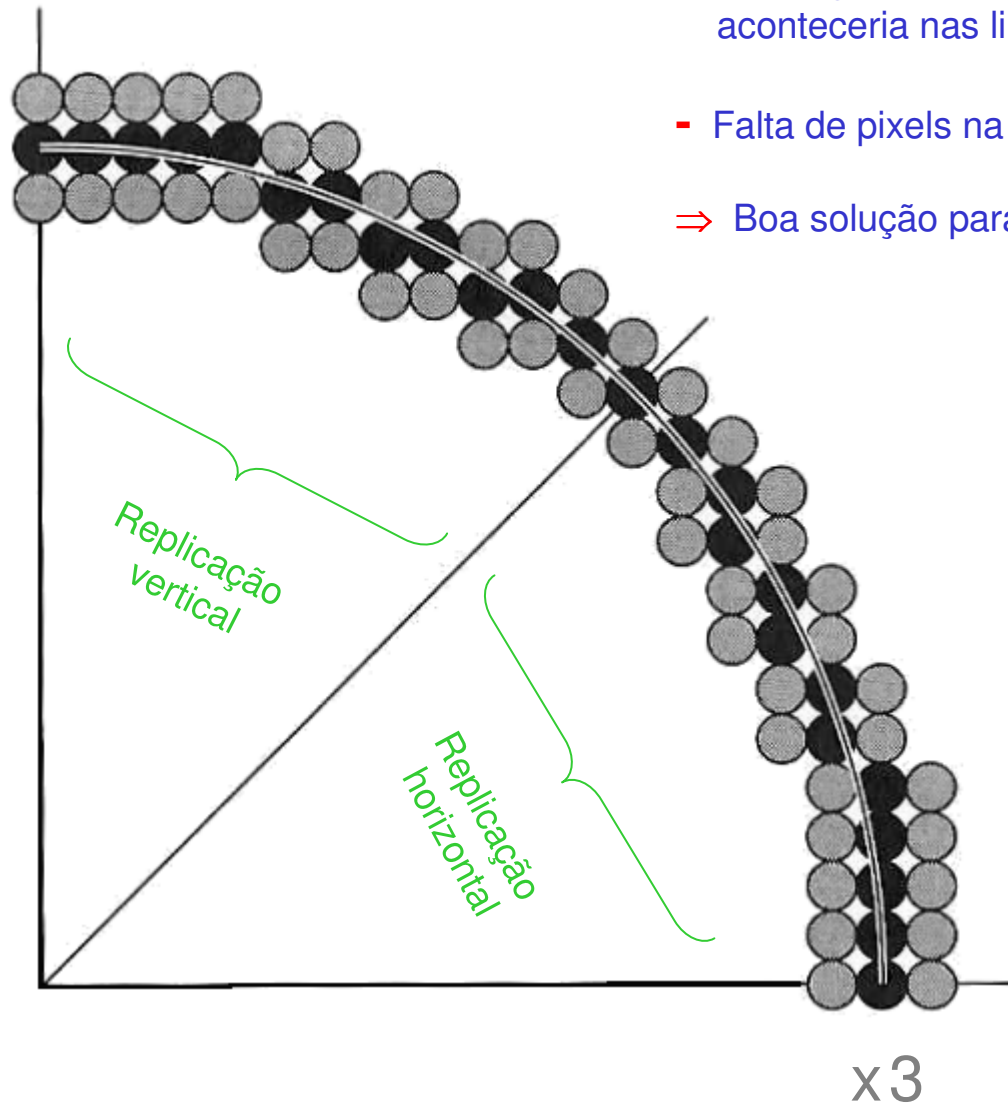
+ Eficiente e facilmente adaptável aos algoritmos de conversão raster.



- As linhas terminam sempre vertical ou horizontalmente.
- Pode implicar a existência de vazios nos pontos de ligação de troços.
- É necessário decidir quando a grossura tiver um valor par (Comum a qualquer técnica!).

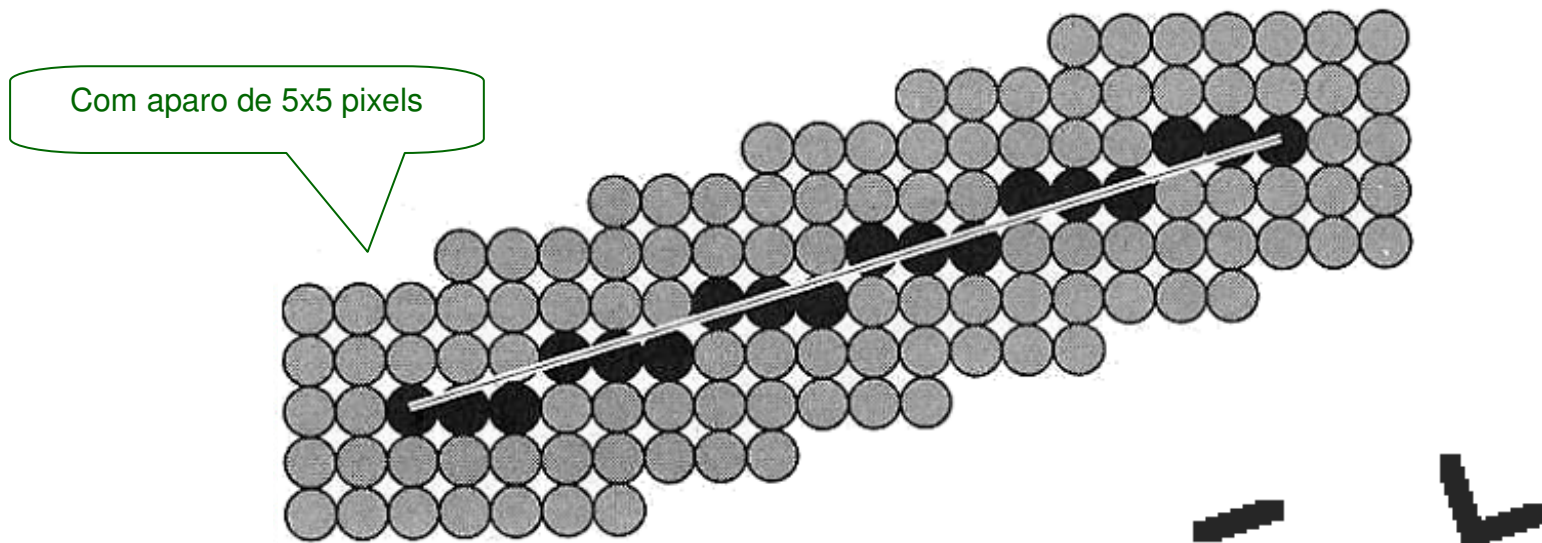
Para algoritmos de conversão raster de arcos de circunferência:

- Menor grossura ($\times 1 / \sqrt{2}$) à aproximação de 45 graus (tal como aconteceria nas linhas a 45°).
 - Falta de pixels na passagem de replicação vertical a horizontal.
- ⇒ Boa solução para linhas pouco grossas.



(B) FORMA DO APARO

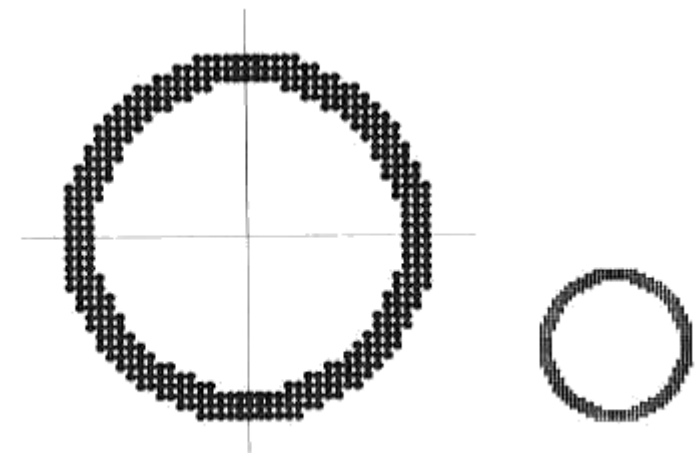
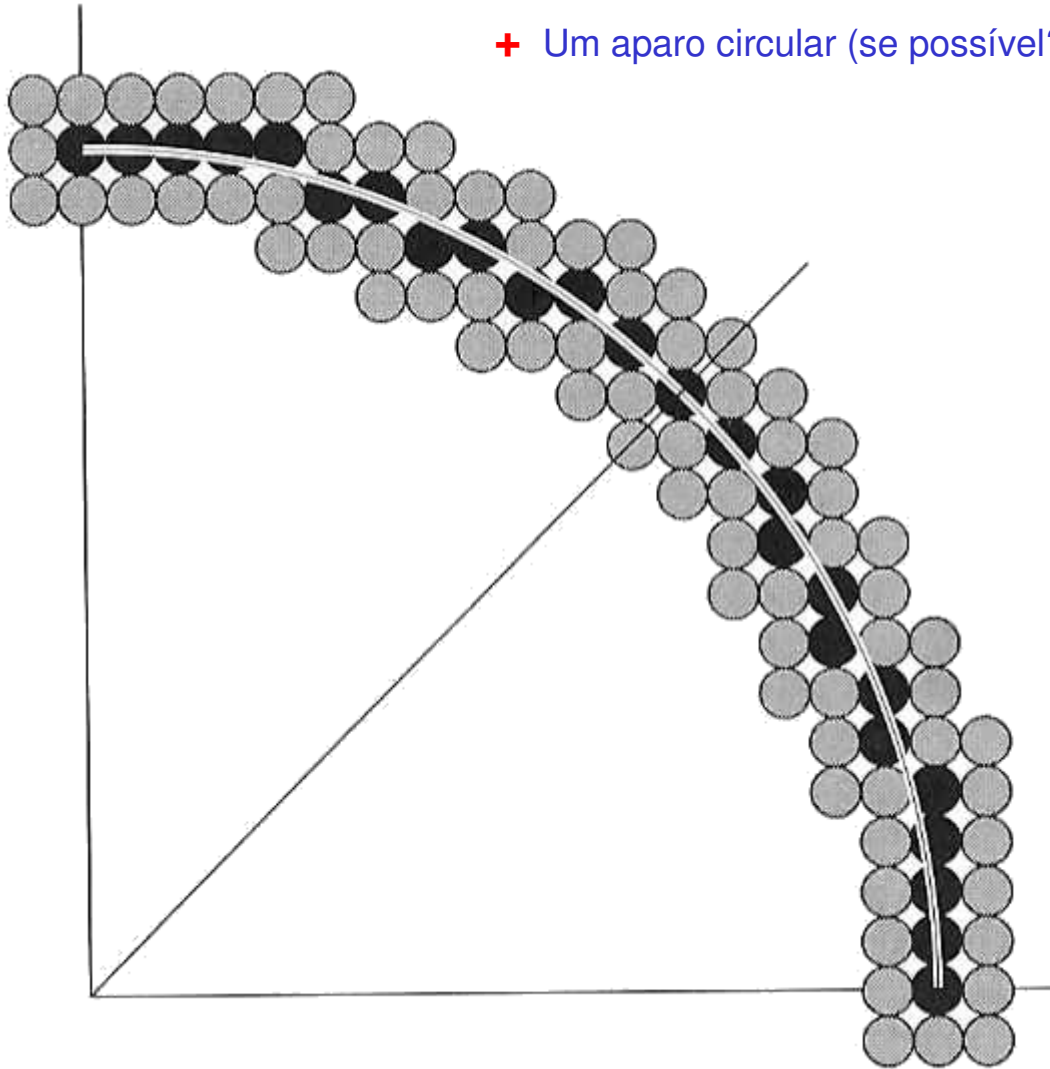
+ Comparando com (A), um aparo quadrado melhora a qualidade dos extremos duma linha.



- Embora igualmente bem adaptável aos algoritmos de conversão raster, a execução é mais lenta.
- Há situações em que não se deve permitir sobreposição na escrita de pixels (e.g. modo XOR).
- Altera o comprimento duma linha (e não resolve o caso de espessura par).

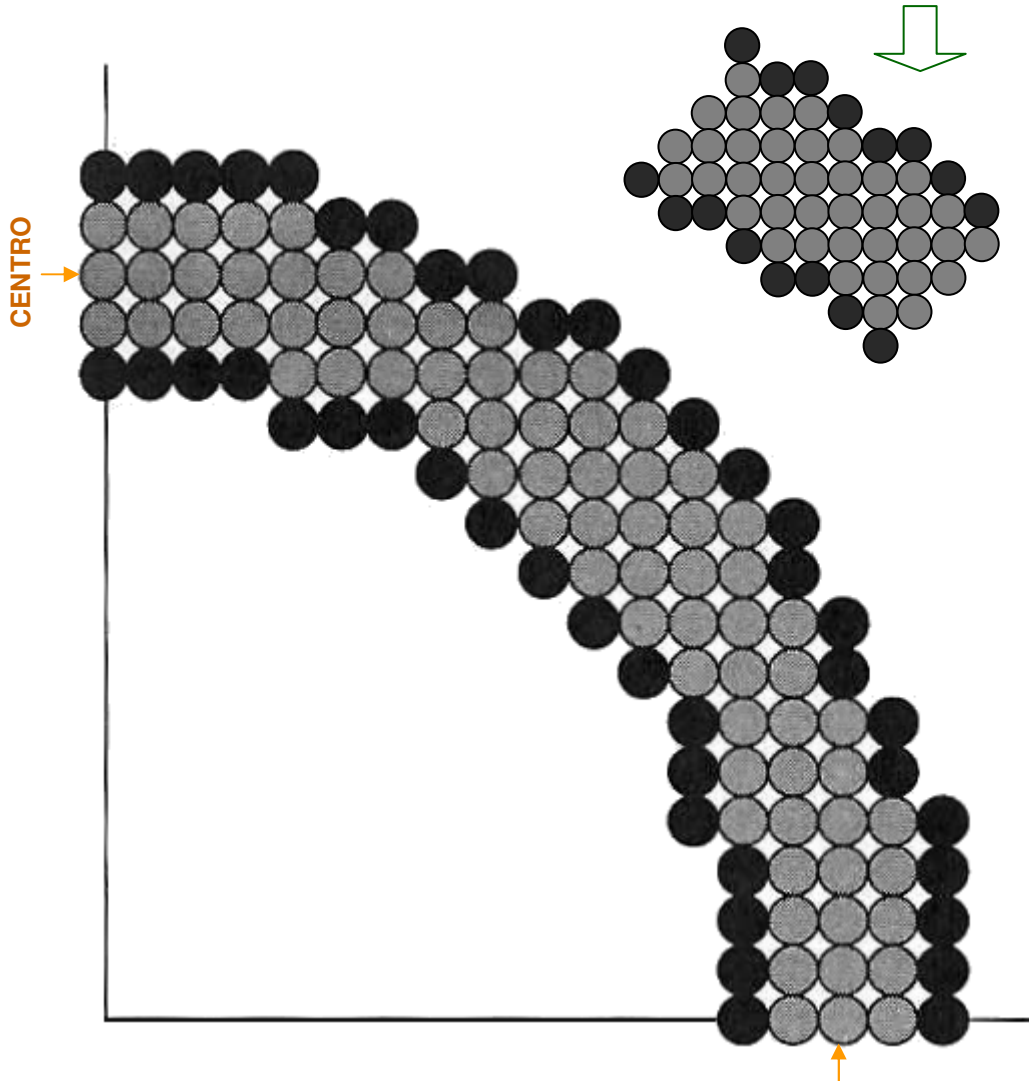
Para arcos de circunferência:

- A grossura média varia com o declive da linha, sendo maior ($\times \sqrt{2}$) a 45 graus.
- + Um aparo circular (se possível?!) evitará o problema.

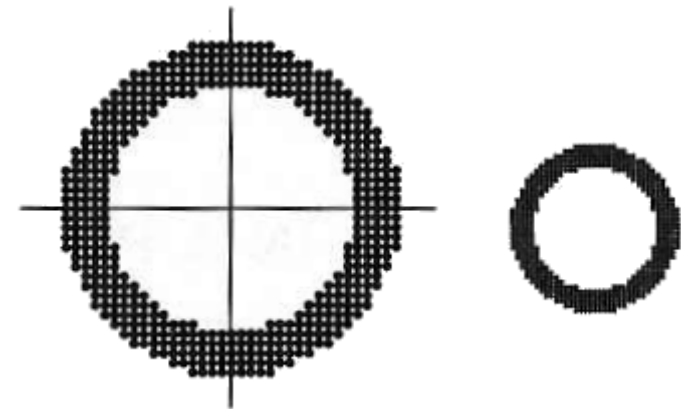


(C) PREENCHIMENTO DE ÁREAS

Criação de duas fronteiras, igualmente afastadas para cada lado, e preenchimento do espaço entre elas:
a um segmento de recta corresponderá um rectângulo.



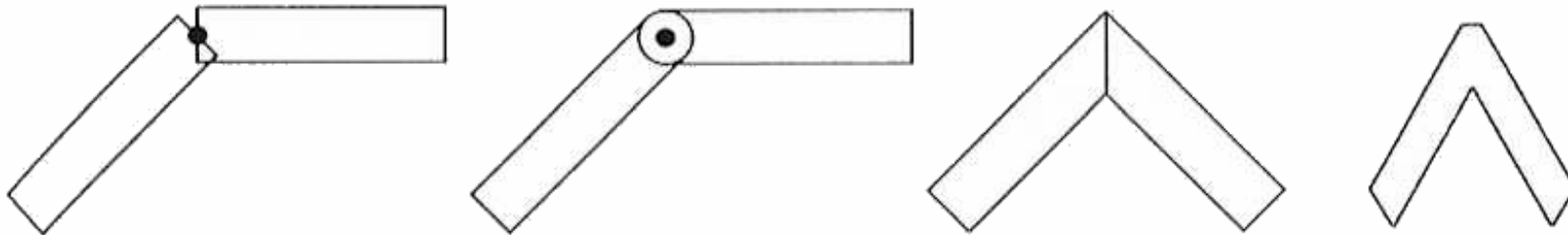
- + Não altera o comprimento duma linha.
- Dificuldades de realização nalguns casos (elipses).
- + Pode aplicar-se o algoritmo de FILL AREA...
- ...mas também se poderá notar desfasamento em relação à linha central.



(D) Para LINHAS POLIGONAIS* (GROSSAS)

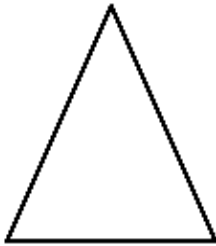
Uma **curva** é aproximada por **troços rectilíneos**, usando-se depois os algoritmos de conversão de linhas e polígonos.

- + Eficiência na geração (e recorte).
- Dificuldades com a aparência dos pontos de junção: **



* Para além da construção de polígonos, são muito usadas para aproximação de linhas curvas

** Problema geral para as linhas poligonais, tanto as abertas como as fechadas



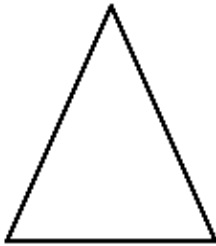
Programação em PostScript

```
%!PS
% Triangle_2
% Triângulo no topo da página
/cm {28.35 mul} def
5 cm 20 cm moveto
7.5 cm 25 cm lineto
10 cm 20 cm lineto
closepath
stroke
showpage
```



Programação em PostScript

```
%!PS
% Triangle_4
% Polígono c/ traço de 1 cm
/cm {28.35 mul} def
5 cm 20 cm moveto
7.5 cm 25 cm lineto
10 cm 20 cm lineto
closepath
1 cm setlinewidth
stroke
showpage
```



Programação em PostScript

```
%!PS
% Triangle_3
% A mesma figura que Triangle_2
/cm {28.35 mul} def
5 cm 20 cm moveto
7.5 cm 25 cm lineto
10 cm 20 cm lineto
5 cm 20 cm lineto
stroke
showpage
```



Programação em PostScript

```
%!PS
% Triangle_5
% Poligonal c/ traço de 1 cm
/cm {28.35 mul} def
5 cm 20 cm moveto
7.5 cm 25 cm lineto
10 cm 20 cm lineto
5 cm 20 cm lineto
1 cm setlinewidth
stroke
showpage
```

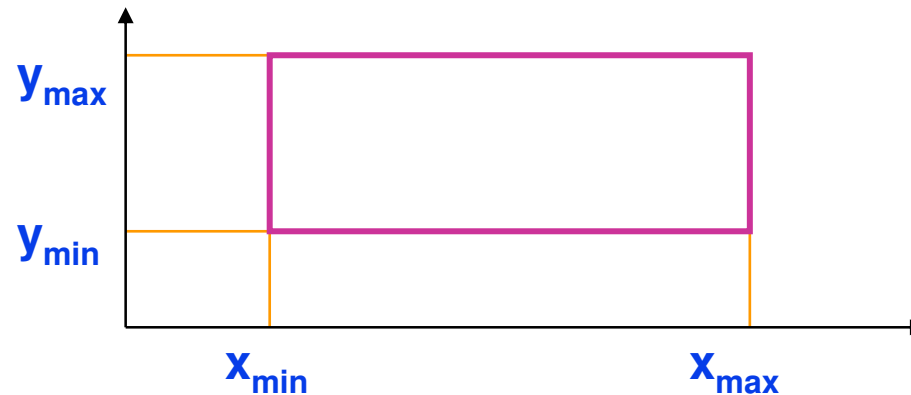


Programação em PostScript

```
%!PS
% Polyline
% Poligonal c/ traço de 1 cm
/cm {28.35 mul} def
5 cm 20 cm moveto
7.5 cm 25 cm lineto
10 cm 20 cm lineto

1 cm setlinewidth
stroke
showpage
```


RECORTE (*CLIPPING*) por janelas retangulares



i. PONTOS

$P(x,y)$ é visível se não for exterior à janela

$$x \leq x_{\max} \wedge x \geq x_{\min} \wedge y \leq y_{\max} \wedge y \geq y_{\min}$$

ii. LINHAS (segmentos de reta)

PQ é visível se P for visível e Q for visível
(condição trivial de aceitação sem cálculos de recorte)

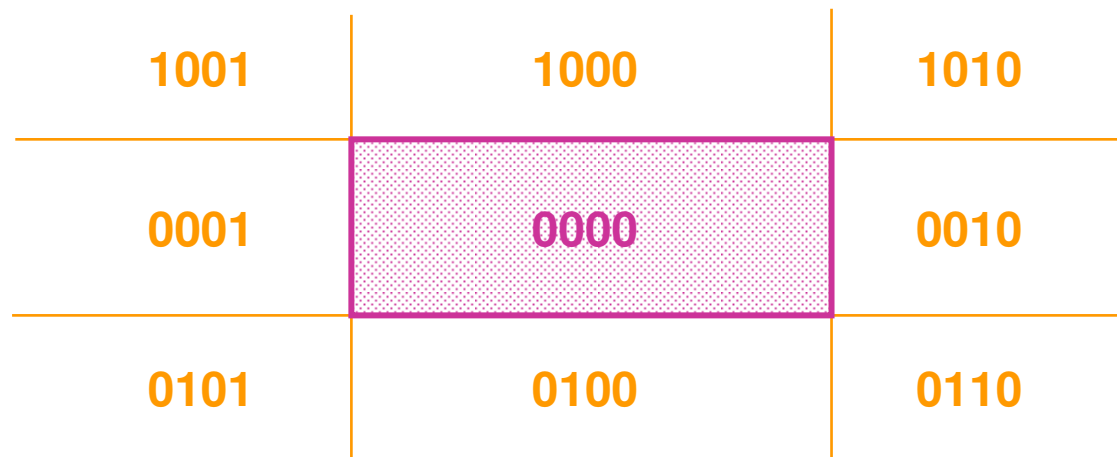
RECORTE de LINHAS

- Método da força bruta ☹️

- Teste de paralelismo
- Resolução de um sistema de duas equações
- O ponto de intersecção das retas pertence aos segmentos?

- Algoritmo de Cohen-Sutherland

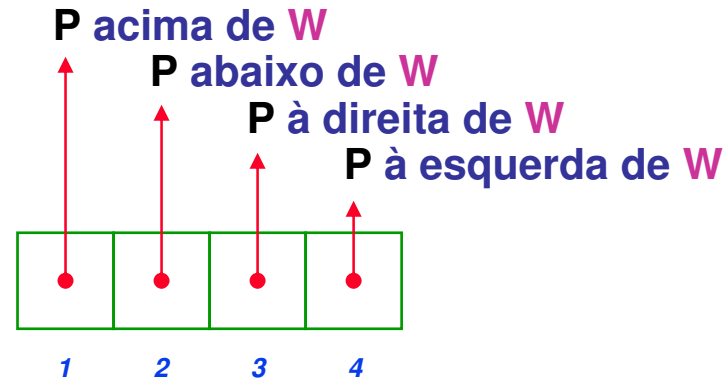
Baseia-se na definição de regiões de teste com relação à janela **W** e atribuição de um código binário a cada extremidade de uma linha, e.g.:



Algoritmo de Cohen-Sutherland

Convenção para cada ponto P: $\text{bit}_i = 1$ se

Esta correspondência
pode ser fixada por
escolha arbitrária



- **PQ é trivialmente aceite se**
 $(\text{código}(P) \text{ or código}(Q)) = 0000$
- **PQ é trivialmente rejeitado se**
 $(\text{código}(P) \text{ and código}(Q)) \neq 0000$
- **Dado existir pelo menos uma intersecção nos restantes casos, usa-se uma estratégia iterativa para a procurar.**

Algoritmo de Cohen-Sutherland

- Para implementar essa estratégia iterativa, escolher-se-á a seguinte ordem para efectuação dos testes

Bit 1 → Bit 2 → Bit 3 → Bit 4

aplicando-se então as regras decorrentes da convenção que se use:

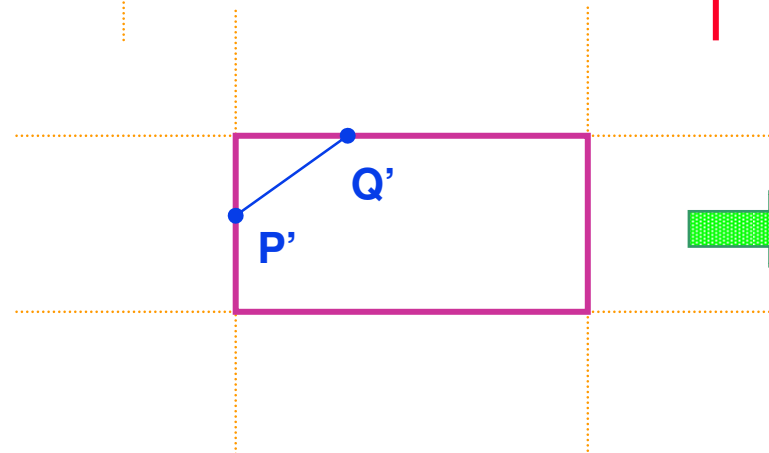
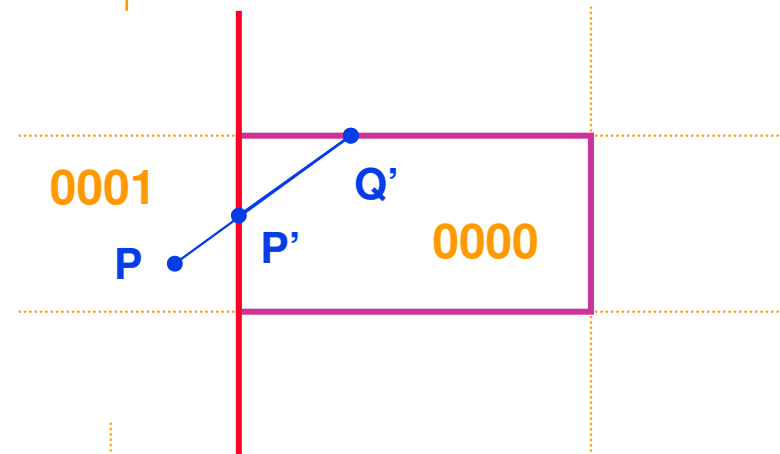
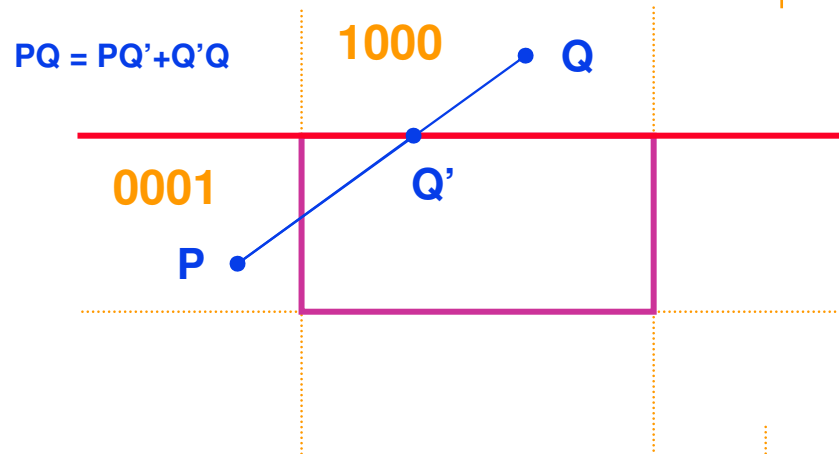
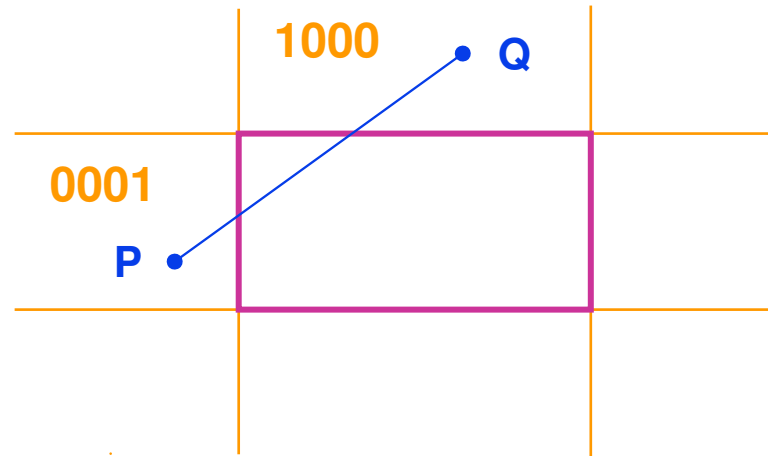
Bits 1 diferentes → rejeitam-se as linhas acima de **W** e recomeça-se o algoritmo

Bits 2 diferentes → rejeitam-se as linhas abaixo de **W** e recomeça-se o algoritmo

Bits 3 diferentes → rejeitam-se as linhas à direita de **W** e recomeça-se o algoritmo

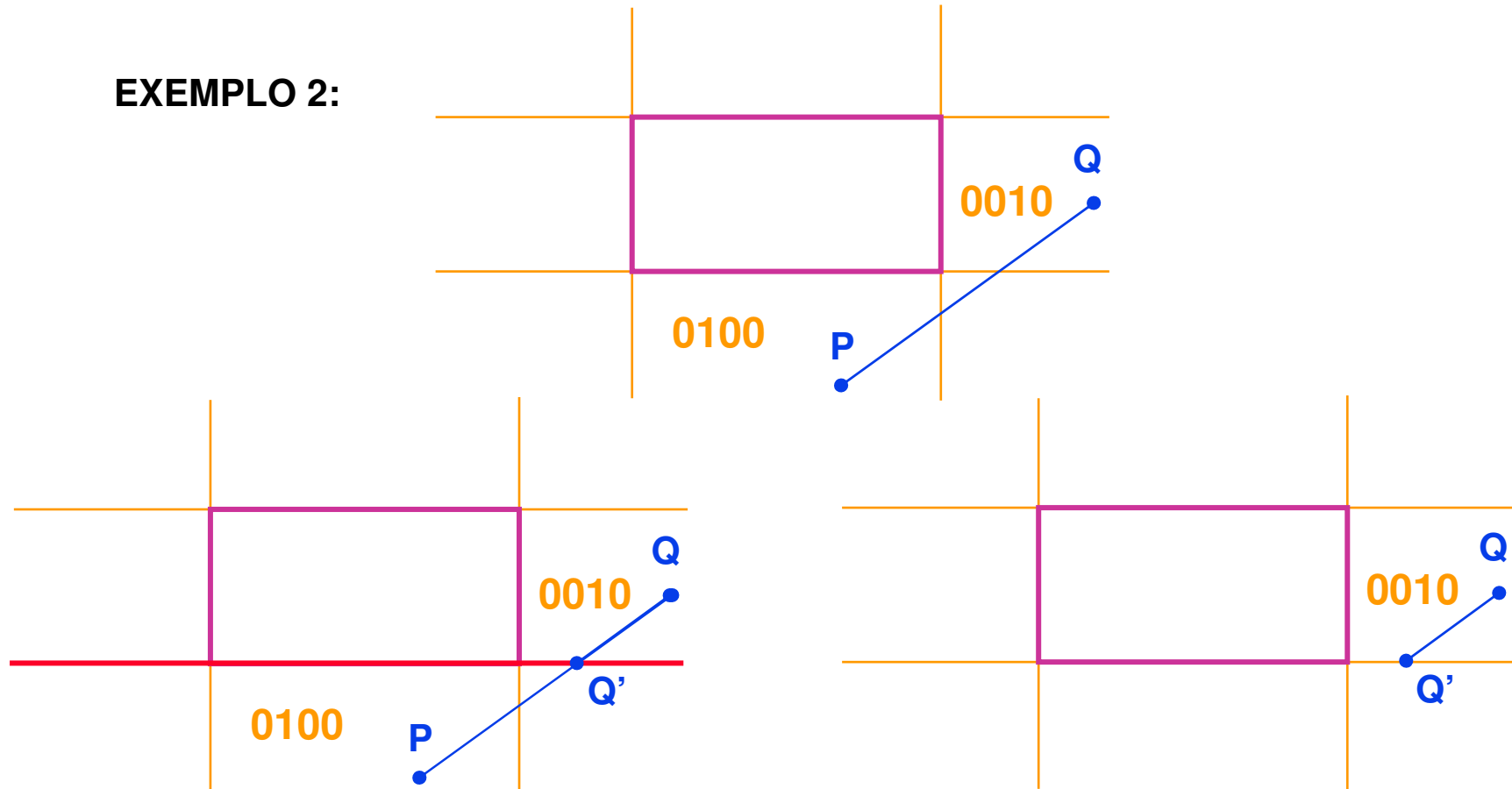
Bits 4 diferentes → rejeitam-se as linhas à esquerda de **W** e recomeça-se o algoritmo

EXEMPLO 1:



$P'Q'$ trivialmente aceite

EXEMPLO 2:



QQ' trivialmente rejeitado

Algoritmo de Cohen-Sutherland

➤ Métodos alternativos para resolução da intersecção:

- a) Resolução de um sistema de equações incluindo, conforme o caso, uma das quatro seguintes

$$X = X_{\max} \quad X = X_{\min} \quad Y = Y_{\max} \quad Y = Y_{\min}$$

(obriga a multiplicações e divisões)

- b) Substituição de P ou Q pelo ponto médio

$$x_{\text{med}} = (x_P + x_Q) / 2 \quad y_{\text{med}} = (y_P + y_Q) / 2$$

Aplicado iterativamente, este algoritmo de pesquisa dicotómica necessita, no máximo, de $\log_2 M_x$ subdivisões, com

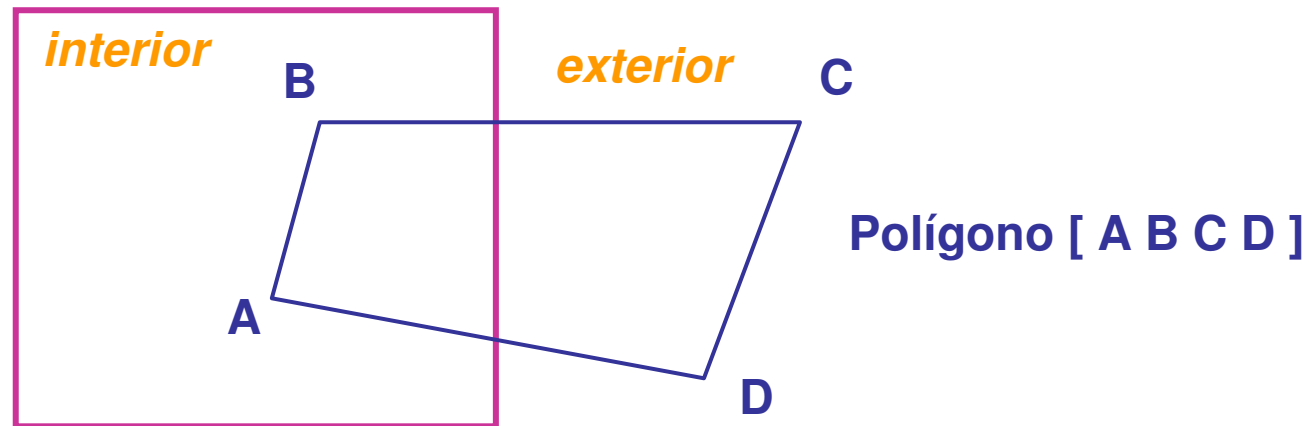
M_x = número máximo de pixels de uma linha

➤ Aplicação ao recorte em 3D:

Extensão de 4 para 6 bits + Planos (de recorte) em vez de retas

RECORTE de POLÍGONOS por janelas retangulares

Algoritmo de Sutherland-Hodgman *



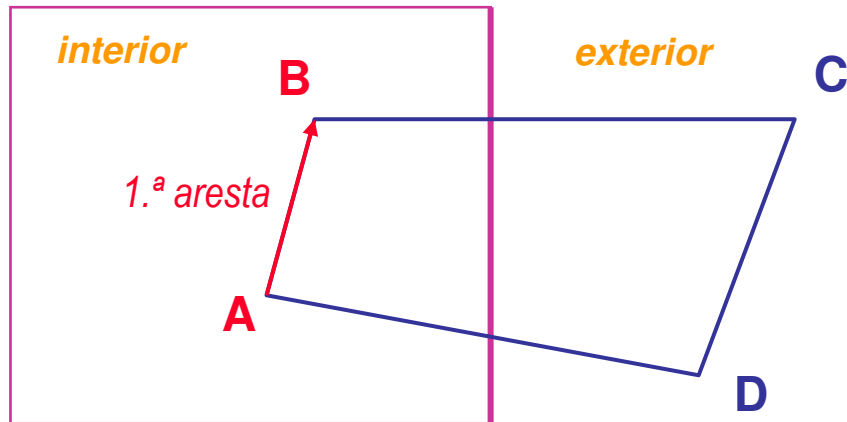
Inicializações:

V_i = Lista dos vértices de entrada = [A B C D]

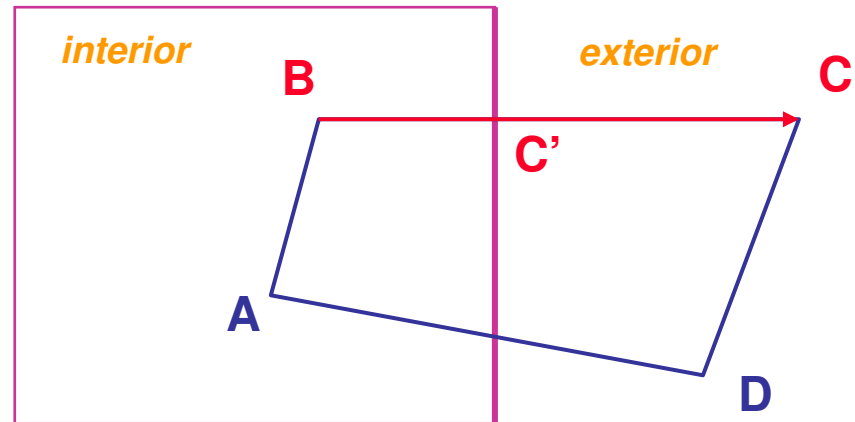
V_o = Lista dos vértices de saída = nil

* Este algoritmo pode generalizar-se a qualquer janela poligonal convexa

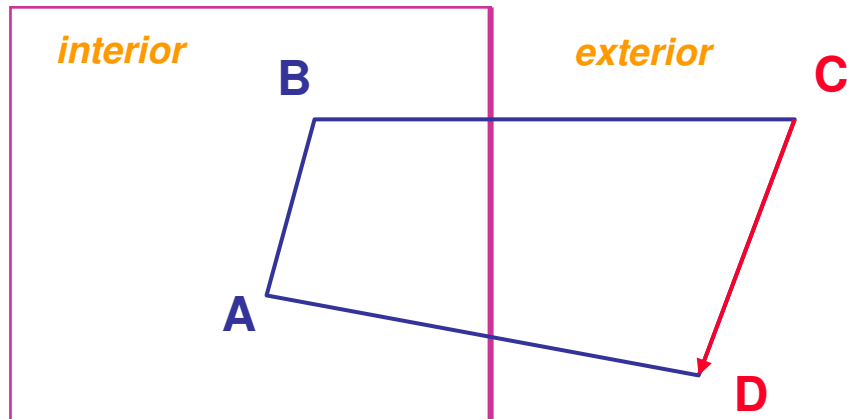
Algoritmo de Sutherland-Hodgman : as 4 regras exemplificadas



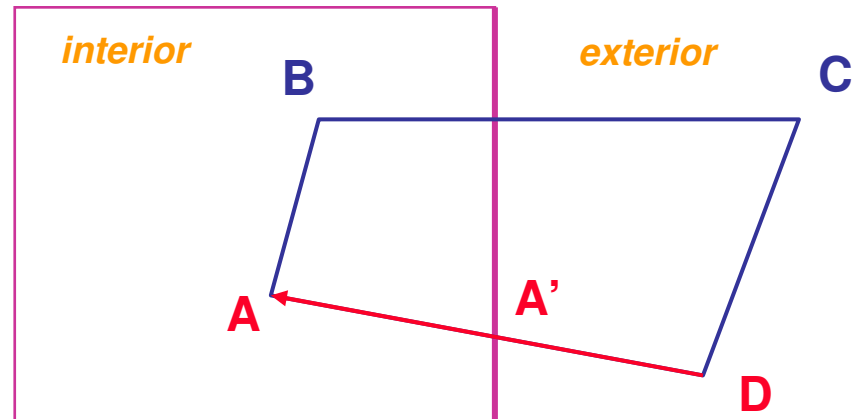
A e B no interior $\Rightarrow$ juntar B a V_o
 $V_o = [B]$



B interior e C exterior $\Rightarrow$ calcular e juntar C'
 $V_o = [B C']$



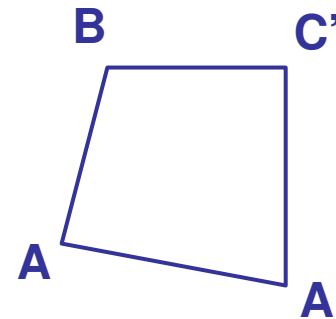
C e D no exterior $\Rightarrow$ continuar
 $V_o = [B C']$



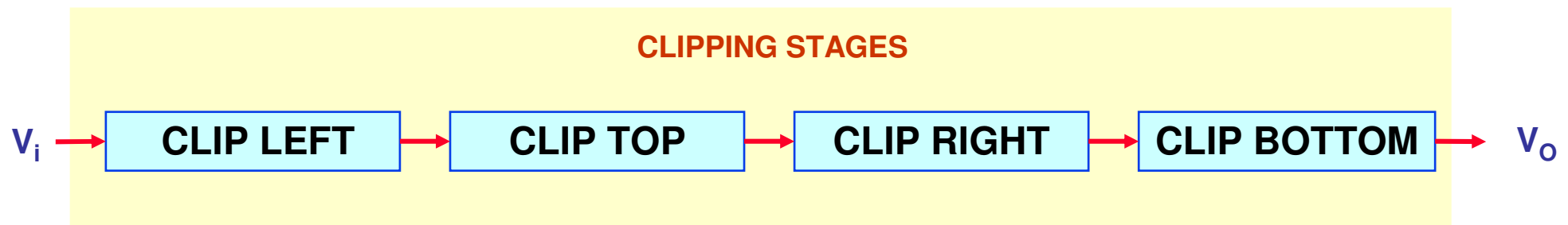
D exterior e A interior $\Rightarrow$ calcular A' e juntar A' e A
 $V_o = [B C' A' A]$

Algoritmo de Sutherland-Hodgman

O polígono [A B C D] atrás exemplificado foi transformado em [B C' A' A] pela operação de recorte:



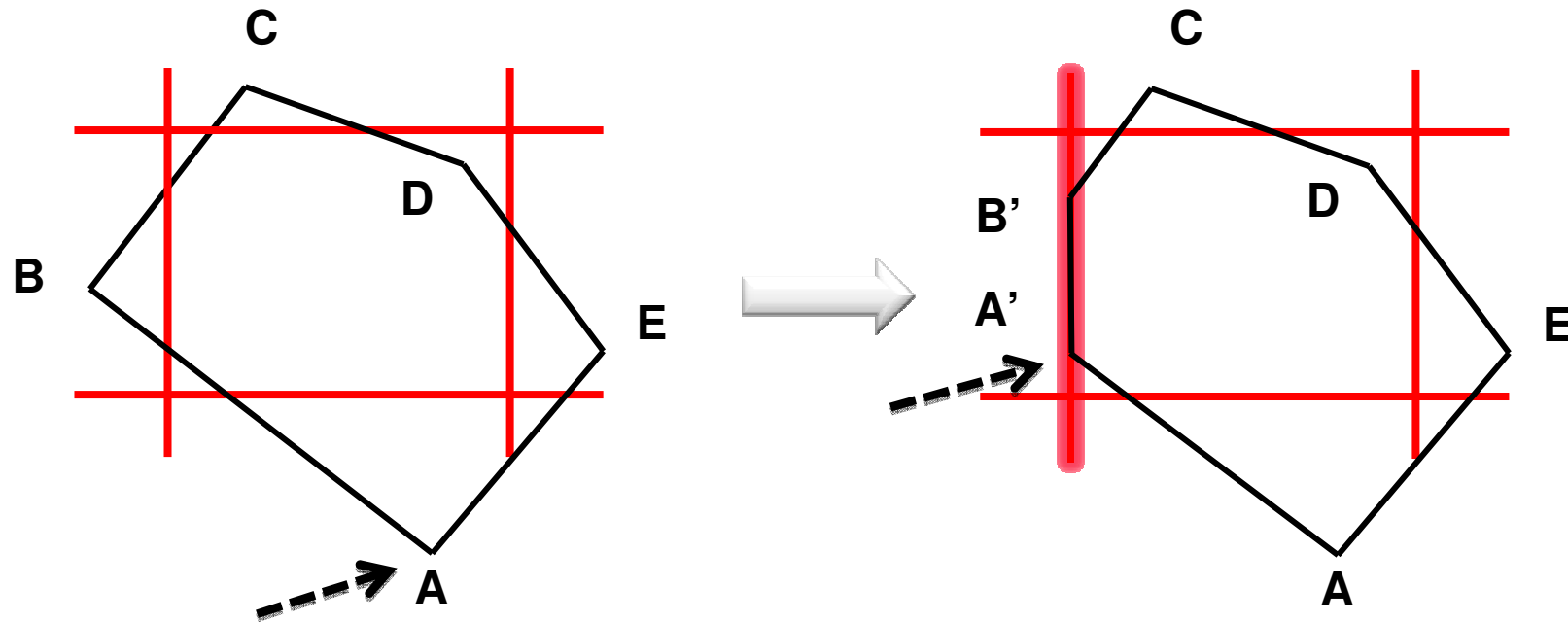
Para um polígono arbitrário, o algoritmo aplica-se por fases sequenciais:



Nota: Esta ordem é arbitrária!

Questão: Como se poderia generalizar este algoritmo aos casos de janelas não retangulares?

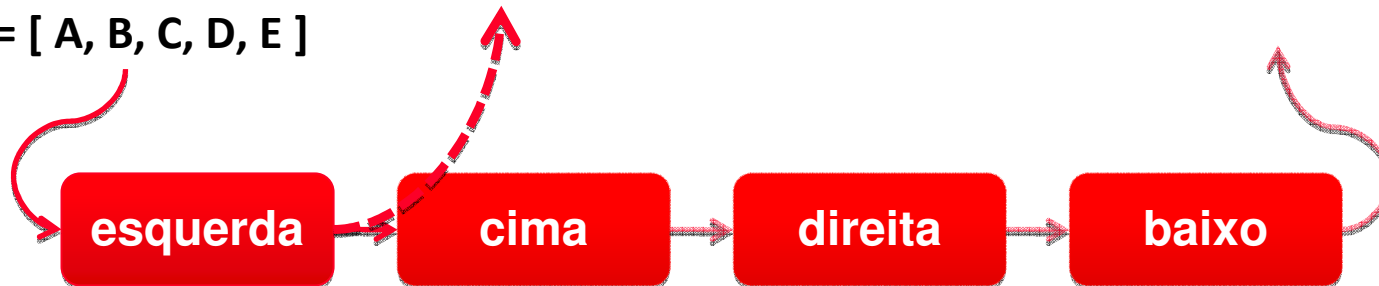
Exemplo



~~$Vo_1 = [A, A', B', C, D, E]$~~

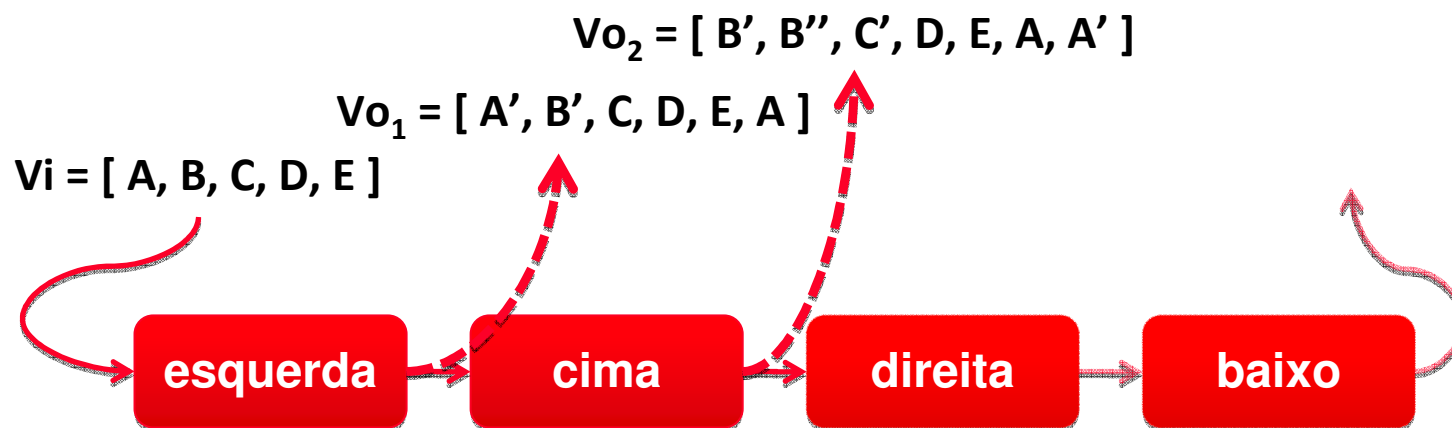
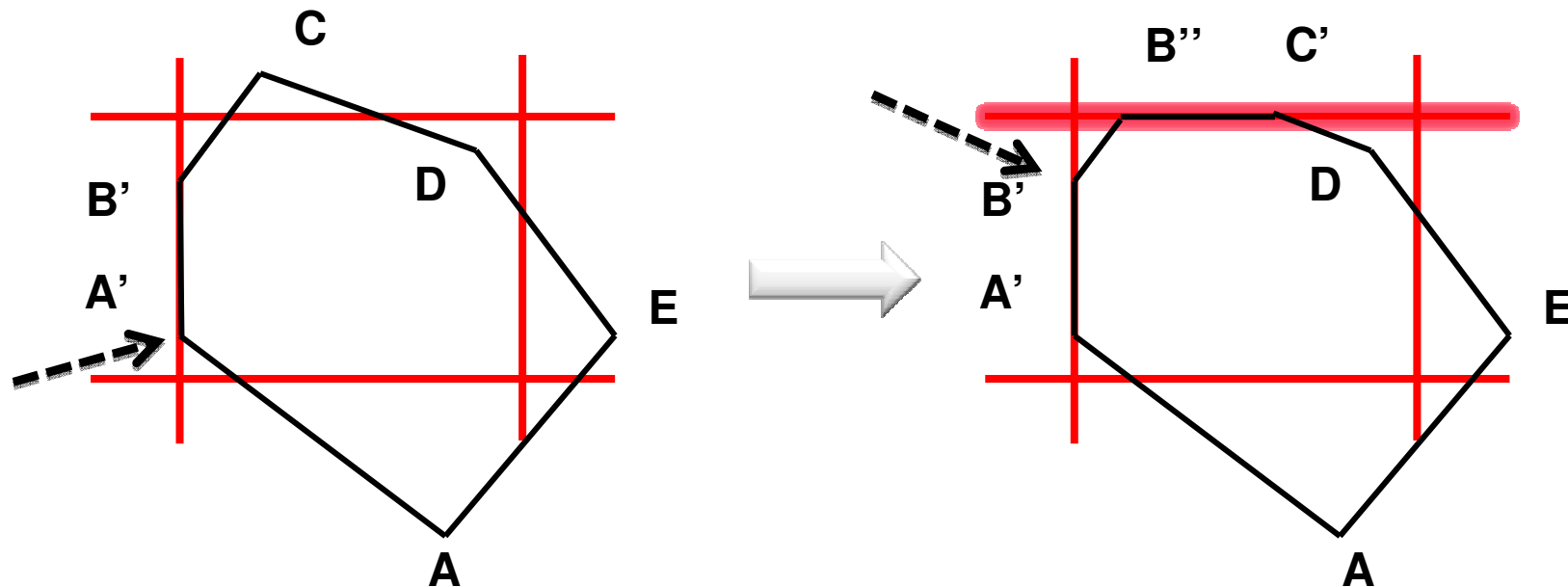
$Vo_1 = [A', B', C, D, E, A]$

$Vi = [A, B, C, D, E]$



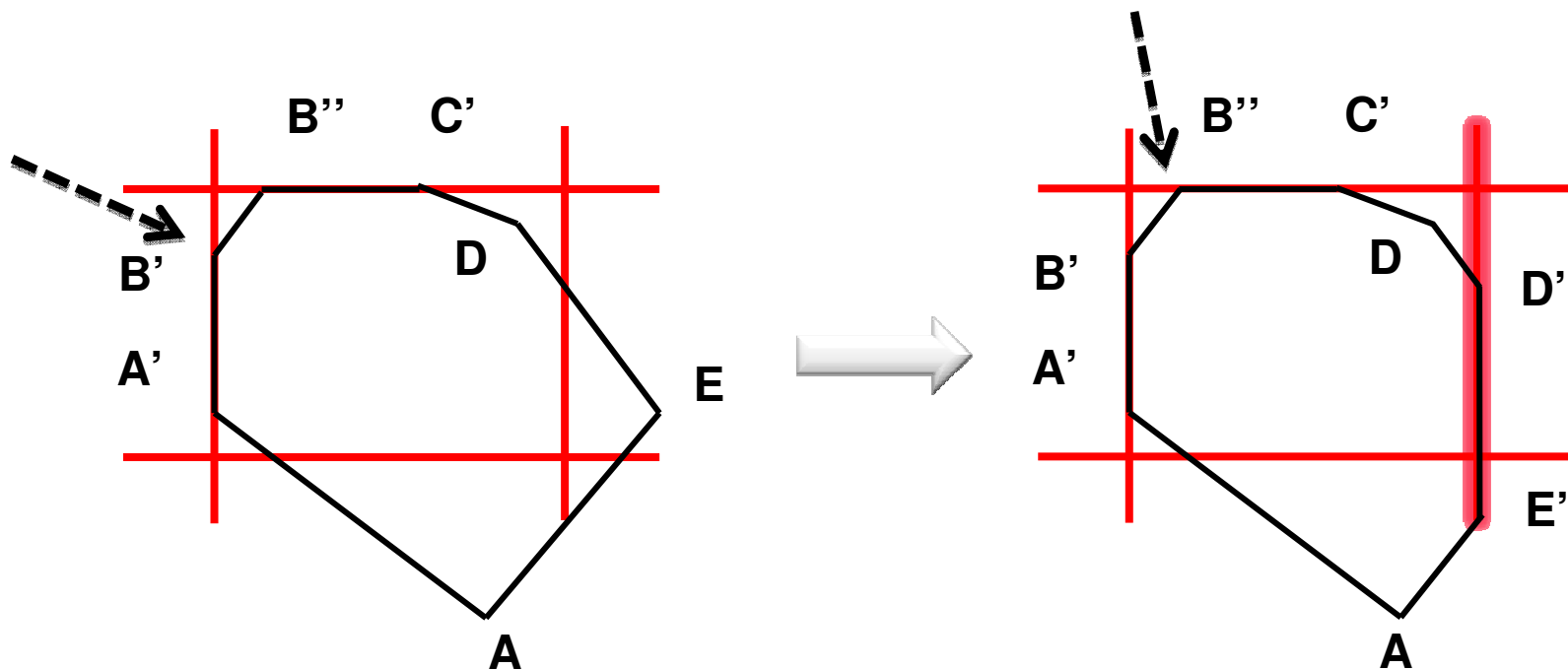
Autor: Adriano Lopes

M.Próspero



Autor: Adriano Lopes

M.Próspero

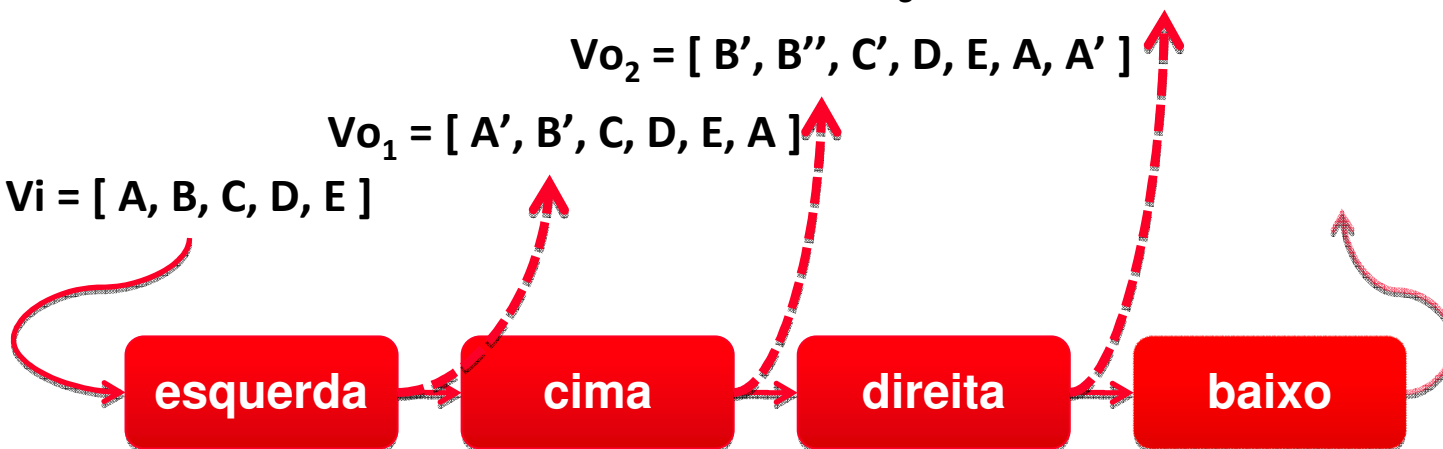


$$Vo_3 = [B'', C', D, D', E', A, A', B']$$

$$Vo_2 = [B', B'', C', D, E, A, A']$$

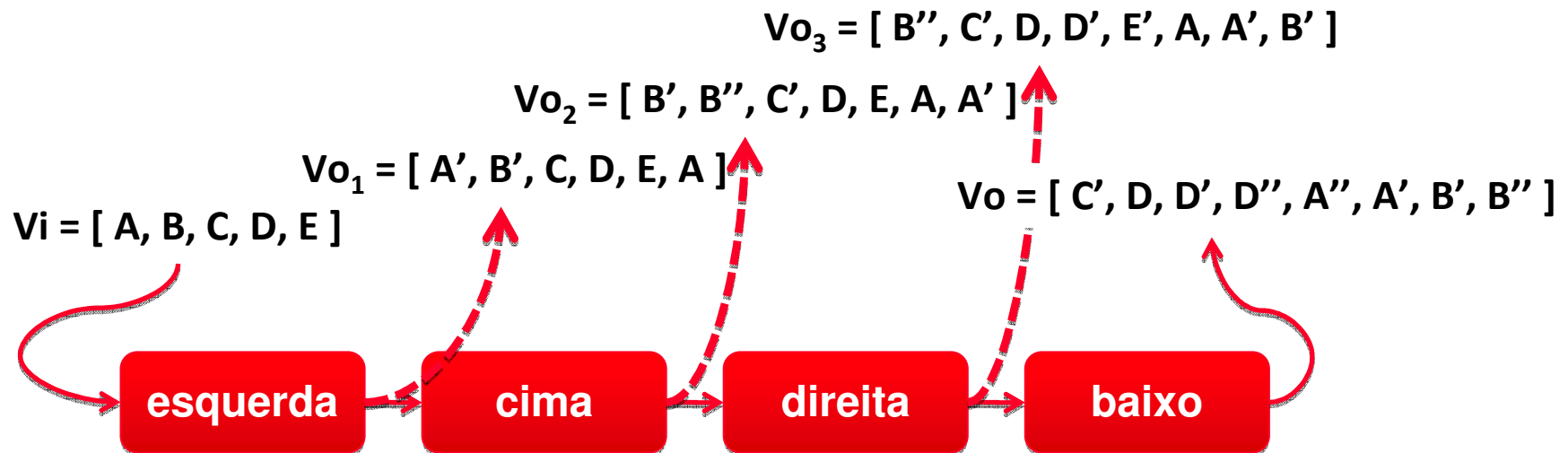
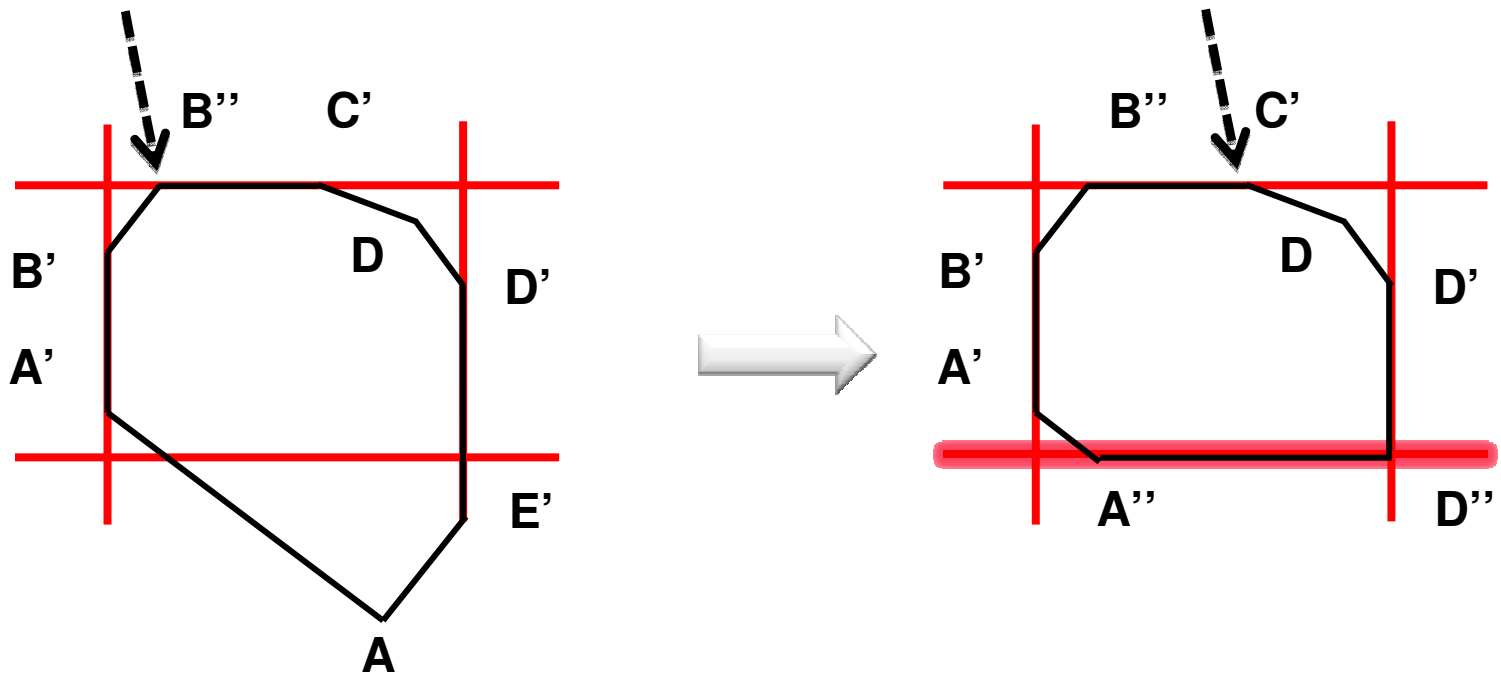
$$Vo_1 = [A', B', C, D, E, A]$$

$$Vi = [A, B, C, D, E]$$



Autor: Adriano Lopes

M.Próspero

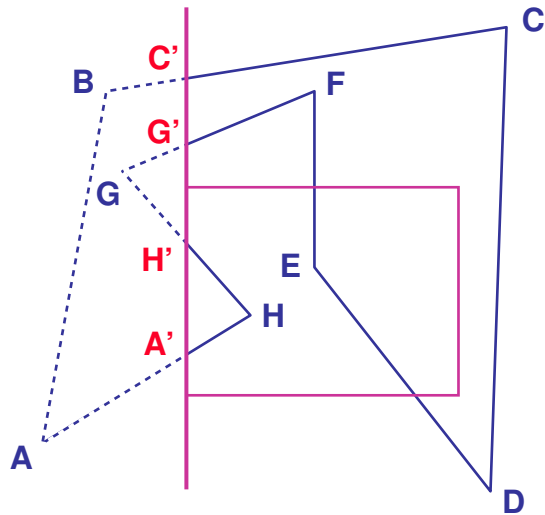
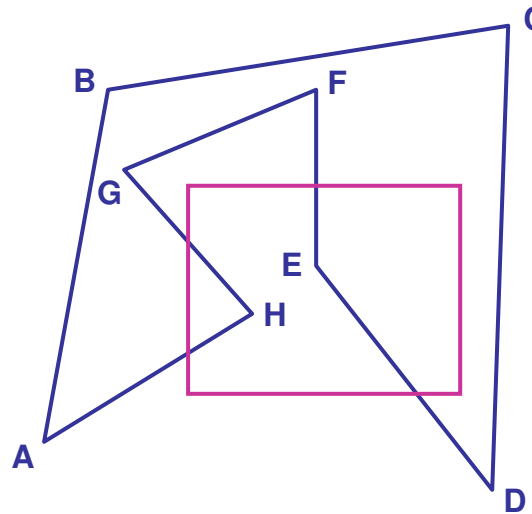


Autor: Adriano Lopes

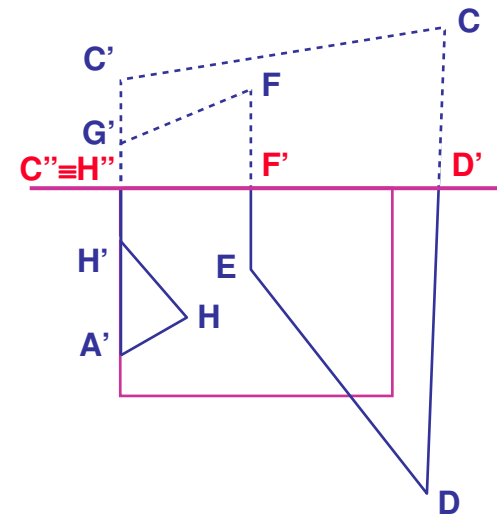
M. Próspero

Algoritmo de Sutherland-Hodgman - EXEMPLO MAIS COMPLEXO (*Polígono não convexo*) :

$V_i = [A B C D E F G H]$



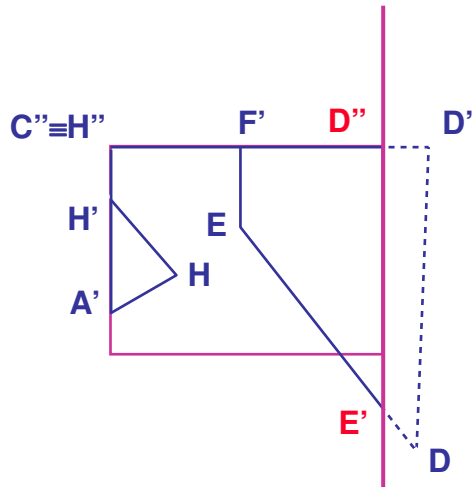
$V_{01} = [C' C D E F G' H' H A'] = V_{i2}$



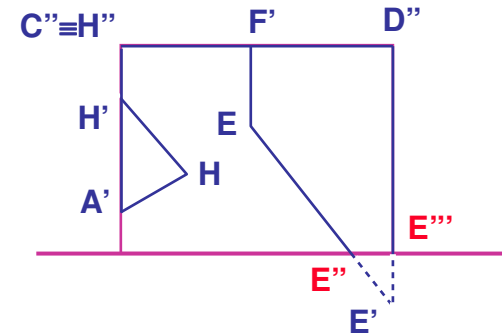
$V_{02} = [D' D E F' H'' H' H A' C'] = V_{i3}$



Algoritmo de Sutherland-Hodgman - (conclusão)

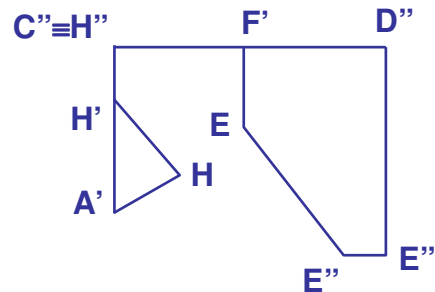


$$\longrightarrow V_{O3} = [E' E F' H'' H' H A' C'' D''] = V_{i4}$$



$$\longrightarrow V_{O4} = [E'' E F' H'' H' H A' C'' D'' E'''] = V_o$$

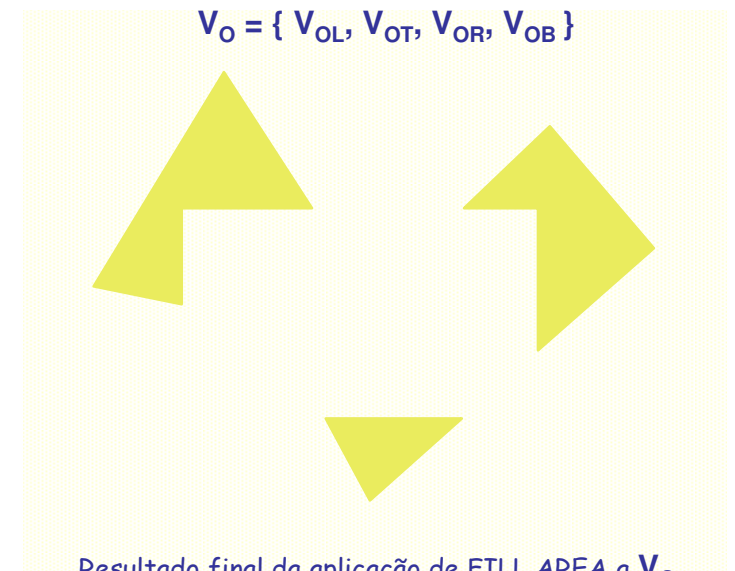
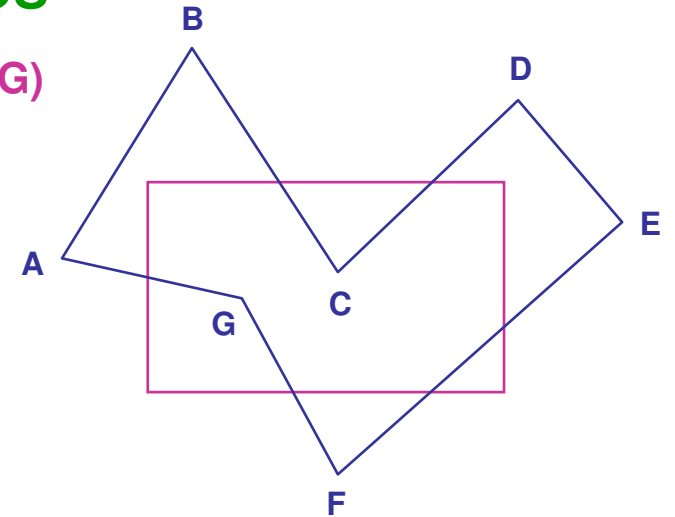
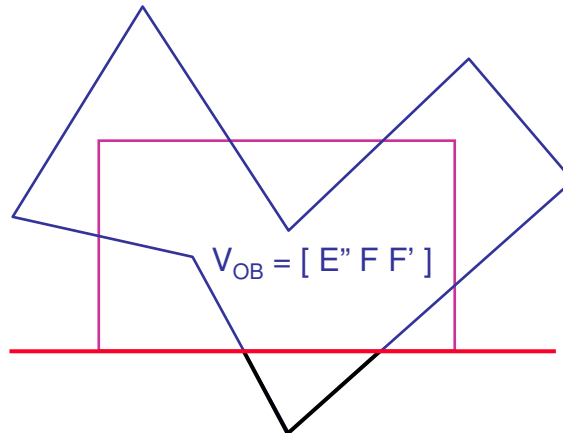
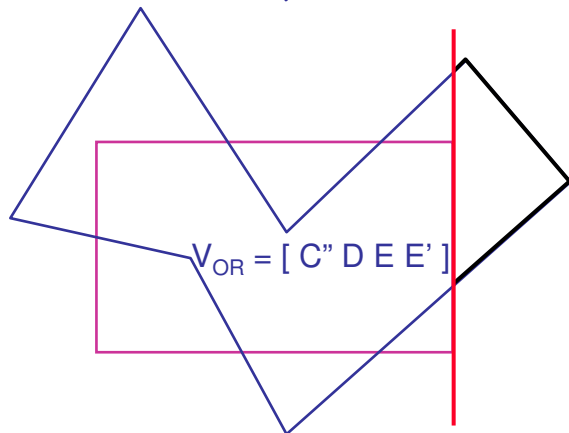
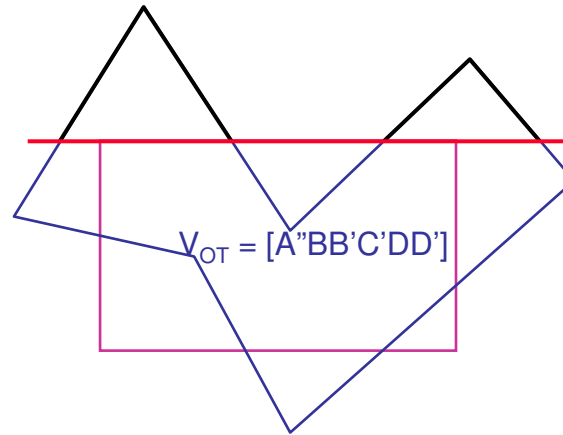
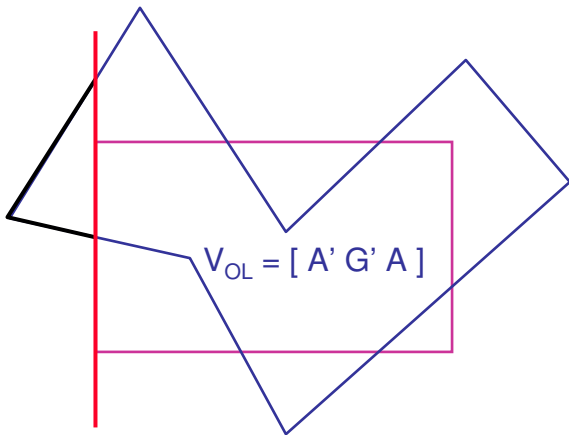
Polígono recortado:



O algoritmo de Sutherland-Hodgman pode gerar arestas estranhas (como $F'H''$ ou $H''H'$) aos polígonos não convexos, com possibilidade de serem eliminadas apenas em pós-processamento. No entanto, não haverá problemas desse género se se recortar um qualquer polígono convexo.

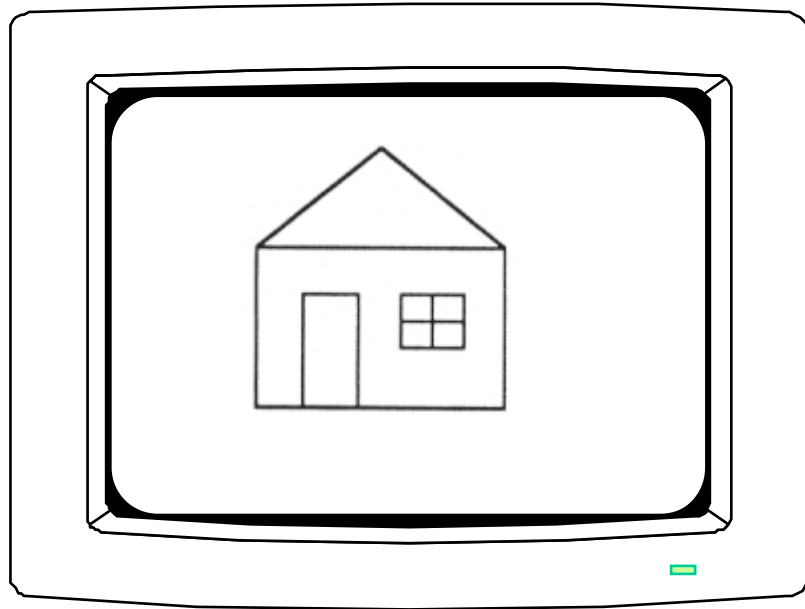
RECORTE de POLÍGONOS

Outra aplicação: RECORTE EXTERIOR (SHIELDING or BLANKING)



Resultado final da aplicação de FILL AREA a V_o
(as sobreposições não são evitadas!)

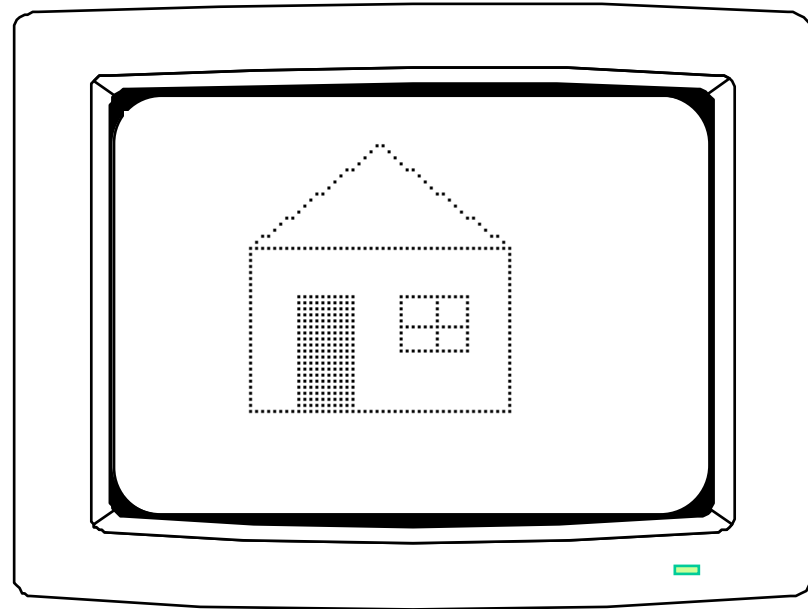
Vector Display



Pen plotter (*traçador*)

DVST

Raster Display



Laser printer

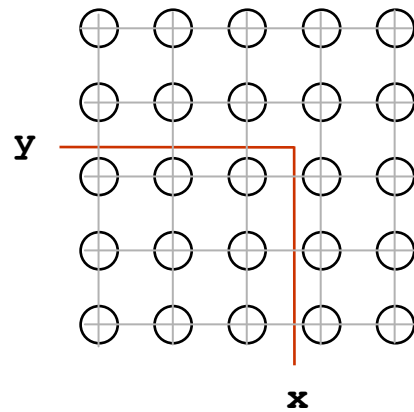
Inkjet printer

Scan Conversion problems

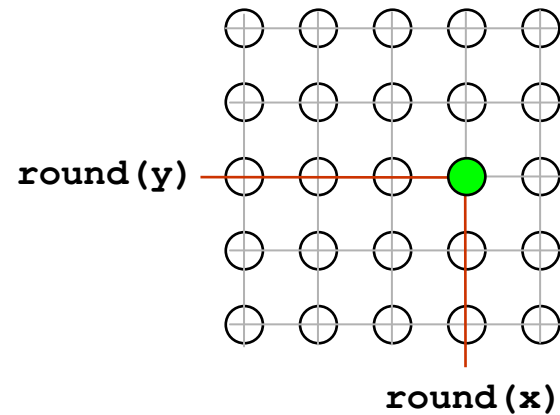
The most obvious attempt to overcome almost all the following side-effects is to deal with a higher resolution.

Aliasing

Plotting a point in a location other than its true location.



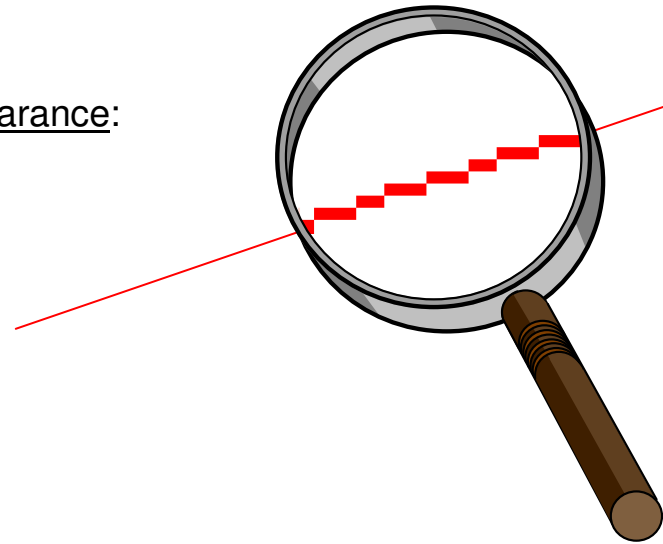
True location



Alias location

For this reason, lines may have a stair-step appearance:

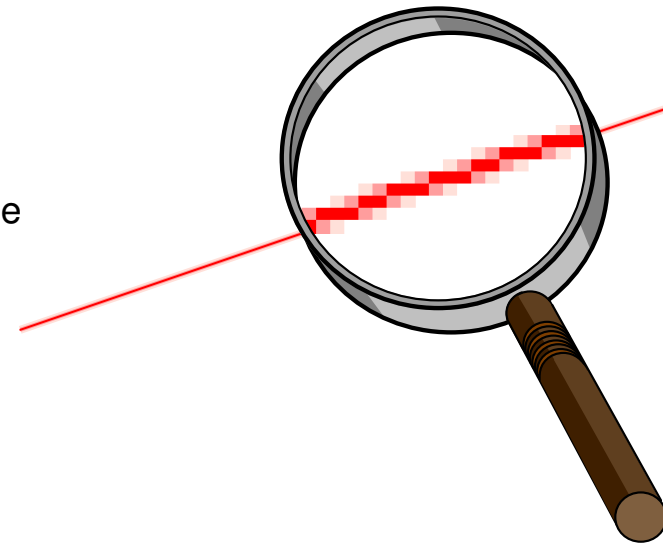
Efeito de Escada



Antialiasing technique:

Turning on more than one pixel in a column (see Bresenham or Midpoint Line algorithms) by using several intensity levels.

The proper value to be used will be chosen according to a function of the distance between the pixel location and the true location.



Application to TEXT CHARACTERS:

Normal sample in Times New Roman

CGI CGI *Zoom in*

... and with Antialias

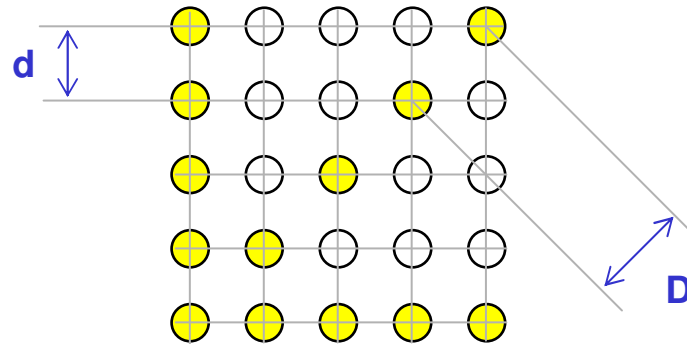
CGI CGI *Zoom in*

Unequal Intensity

Diagonal lines of pixels appear dimmer than vertical or horizontal lines.

Why?

For the same intensity of light sources, our perception of light also depends on their density.



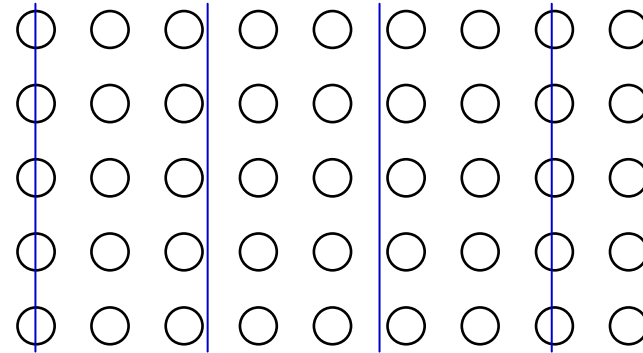
$$D = d \sqrt{2}$$

Picket Fence

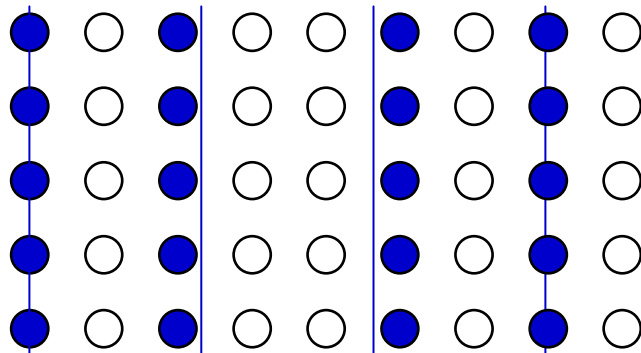
A decision problem about dimensions will occur if an object does not fit exactly into the raster.

Given a picket fence as the original object:

Vedação c/ estacas
(paliçada)

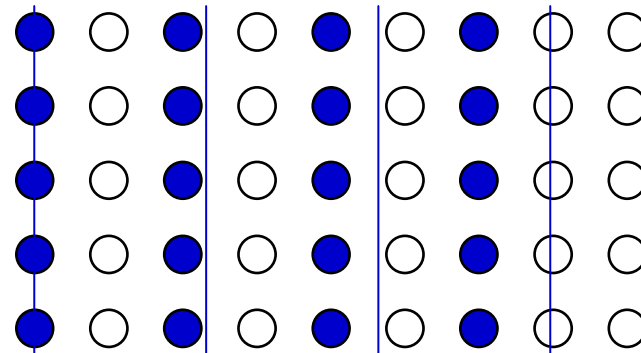


1st solution: **Local Aliasing**



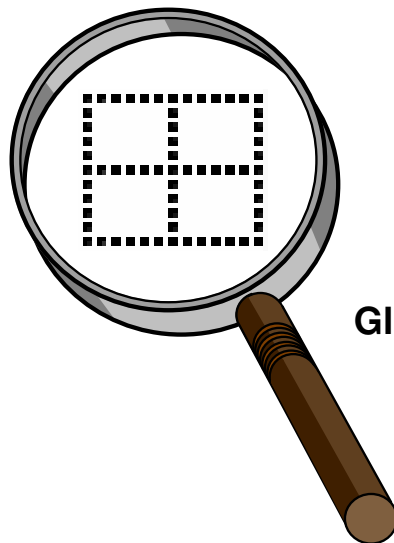
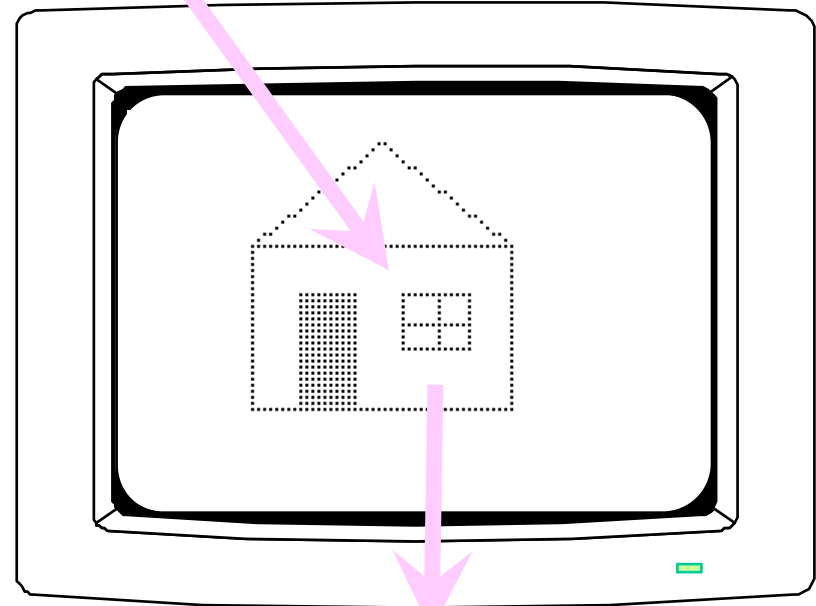
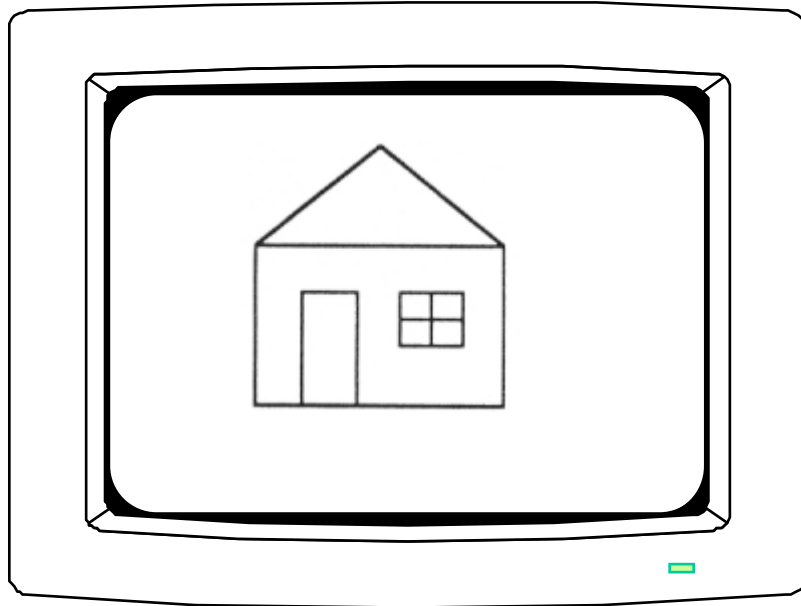
Equal overall length
but different distances between pickets.

2nd solution: **Global Aliasing**



Equal distances between pickets
but different overall length.

Example



Global Aliasing

Which one
do you prefer?



Local Aliasing

ANTIALIASING pelo Método da Filtragem

Aplica-se (por pós-processamento) a uma imagem já existente.

Método: o valor de cada pixel contribui, por soma ponderada, para os valores dos pixels vizinhos e na relação inversa da distância.

Este cálculo não é cumulativo, dando resultados bastante aceitáveis tanto para linhas como para polígonos.

Como a menor das vizinhanças é a de pixels adjacentes, o menor dos filtros possíveis é 3x3.

Por exemplo:

		$\frac{1}{36}$	$\frac{1}{9}$	$\frac{1}{36}$	
	y	$\frac{1}{9}$	$\frac{4}{9}$	$\frac{1}{9}$	
		$\frac{1}{36}$	$\frac{1}{9}$	$\frac{1}{36}$	
			x		

Os valores numéricos indicam as ponderações de intensidade de todos os nove pixels para o cálculo da intensidade no pixel (x,y) .

Exemplo de aplicação concreta do filtro 3x3 :

Imagem original:

(com o cálculo de três dos pixels a tratar)

Black=0
White=1

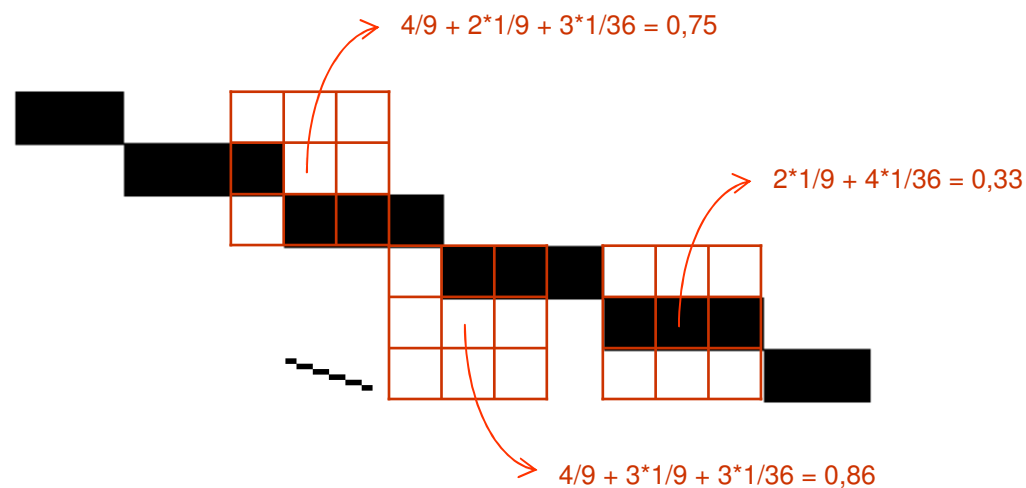


Imagem tratada:



(inconveniente: as linhas finas ficarão mais tênues)

MODELO de COR

(1) ORIENTADO PARA O EQUIPAMENTO

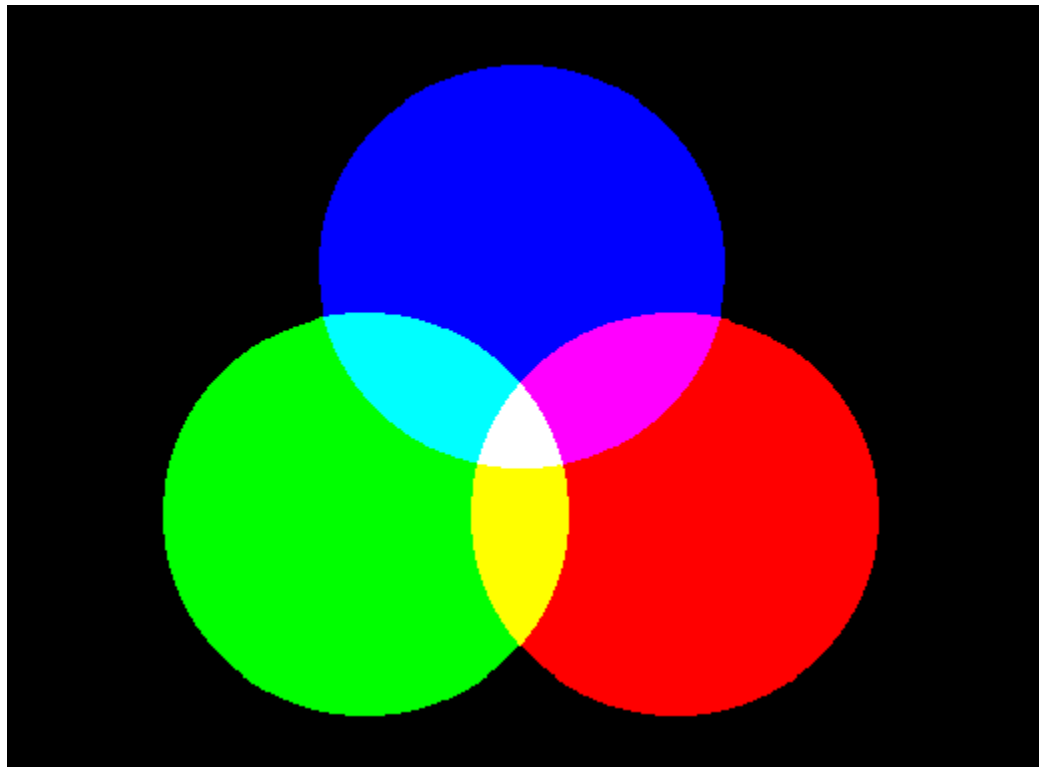
Finalidade? Especificação precisa das cores numa certa gama, para determinada classe de equipamentos.

O que é?

- Subespaço 3D
- Sistema de coordenadas 3D

Combinação das cores

- Sistema aditivo



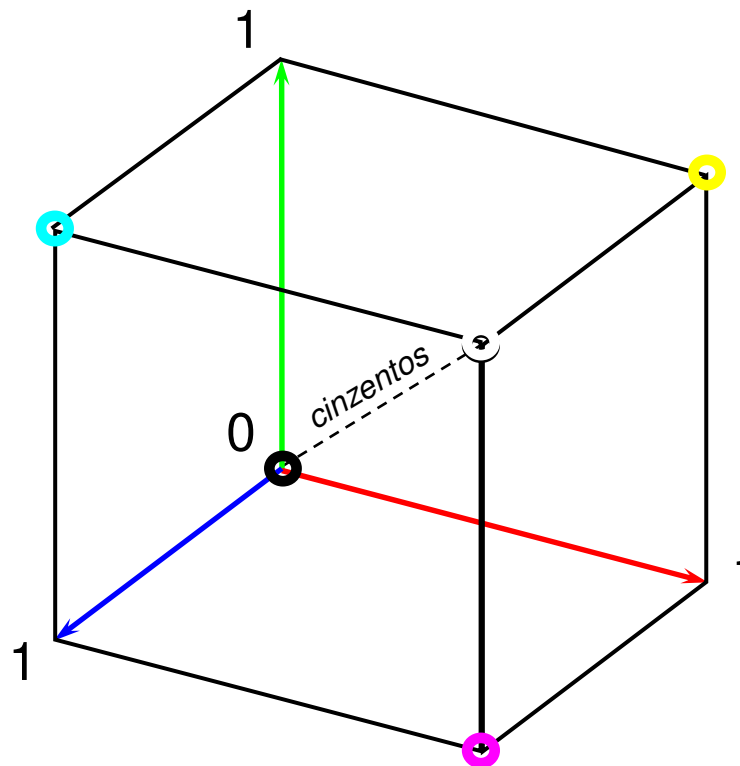
Luz emitida
por 3 focos
de cores
primárias

(implica a
ausência de
qualquer
outra fonte
de luz)

Modelo RGB

- Cubo
- Sistema de coordenadas cartesianas

Red
Green
Blue

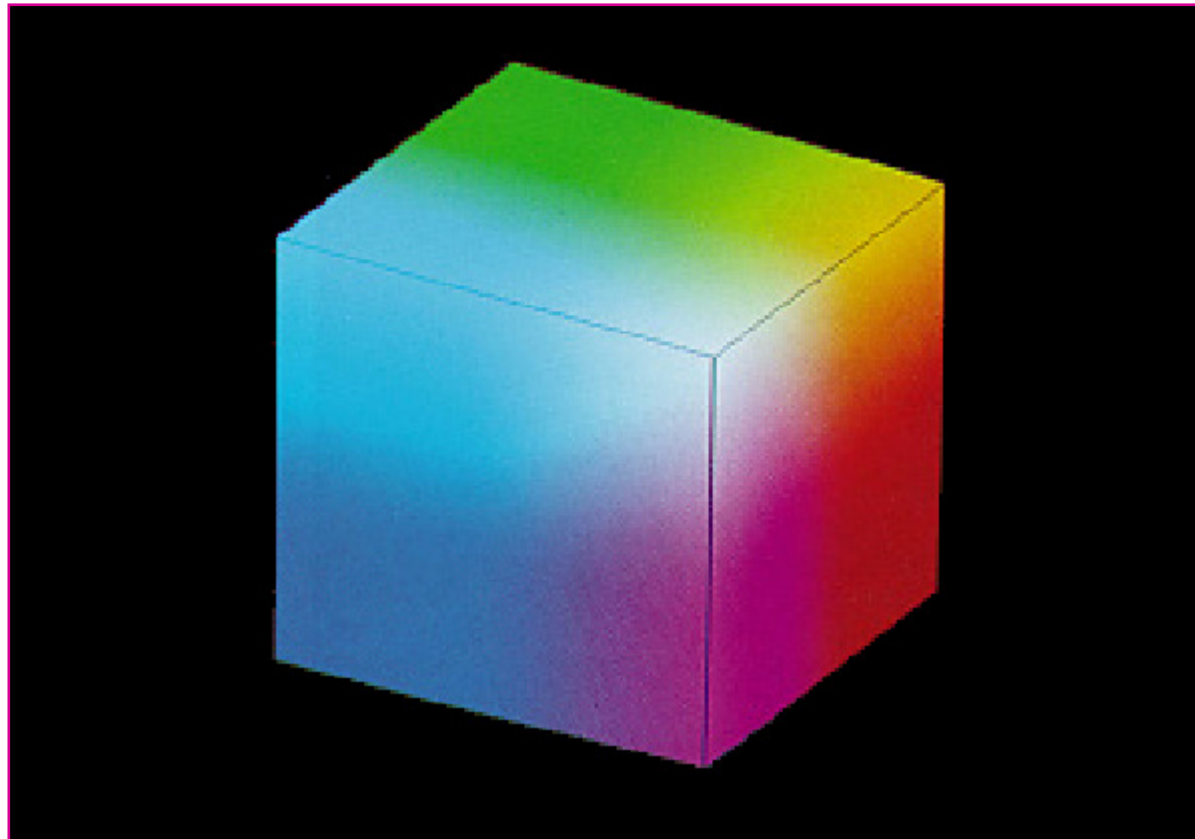


Codificação de 1 byte / cor primária:

$$\text{TOTAL} = 2^8 \times 2^8 \times 2^8 = 2^{24} = 16\,777\,216 \text{ cores possíveis}$$

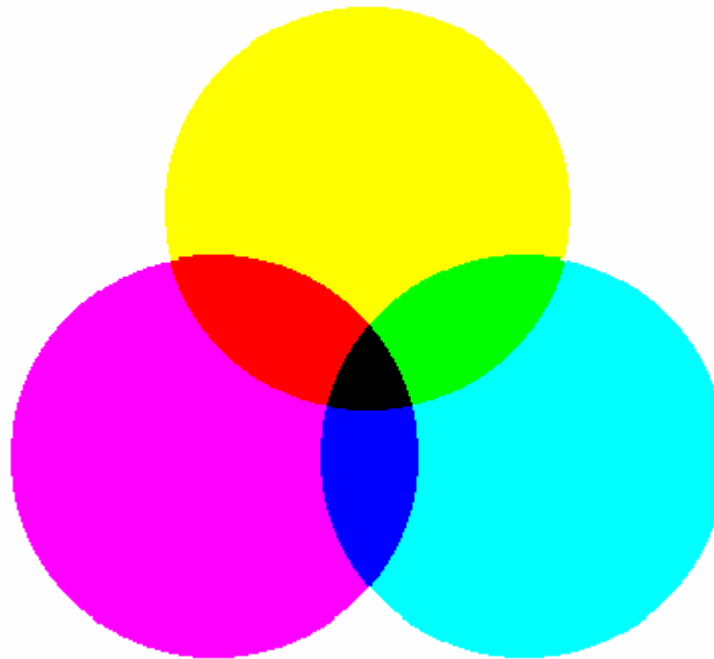
Espaço RGB

Red
Green
Blue



Combinação das cores

- Sistema subtrativo



Luz
refletida por
3 cores
primárias

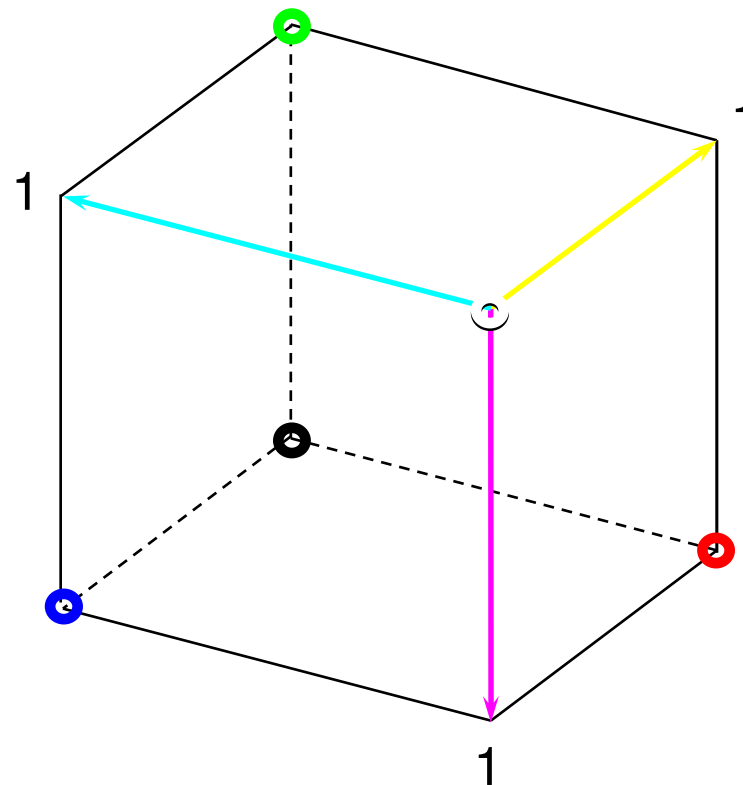
(implica a
existência
de luz
branca
incidente)

Modelo CMY

Relaciona-se com a mistura de pigmentos no papel.

Cyan não reflete R
Magenta não reflete G
Yellow não reflete B

Nota: Um objeto transparente que possua uma destas cores será um filtro que bloqueia a luz com a cor complementar (p.ex. o amarelo bloqueia o azul).



Exercício proposto:

Qual a característica comum aos tons de cinzento?

M. Próspero

Modelo CMYK

Para melhorar a qualidade, os sistemas de *hardcopy*, para além dos 3 primários, também usam pigmento BLACK.

Cyan
Magenta
Yellow
BlackK

Conversão entre sistemas:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 - R \\ 1 - G \\ 1 - B \end{bmatrix}$$

$$\begin{aligned} K &= \min(C, M, Y) \\ C &= (1 - R) - K \\ M &= (1 - G) - K \\ Y &= (1 - B) - K \end{aligned}$$

Atributos Psicológicos da Cor

- Tom, tonalidade, matiz (*Hue*)
 - Distinção das cores propriamente ditas
- Saturação
 - Da cor pura (dita saturada) ao cinzento (ou branco)
- Claridade, brilho ou intensidade
 - Distinção pela quantidade de luz associada à excitação da fonte de iluminação

MODELO de COR

(2) ORIENTADO PARA O SER HUMANO

Finalidade? Especificação precisa das cores e de acordo com os respectivos atributos psicológicos.

Por exemplo: Variar o valor da saturação para um mesmo tom e uma mesma intensidade.

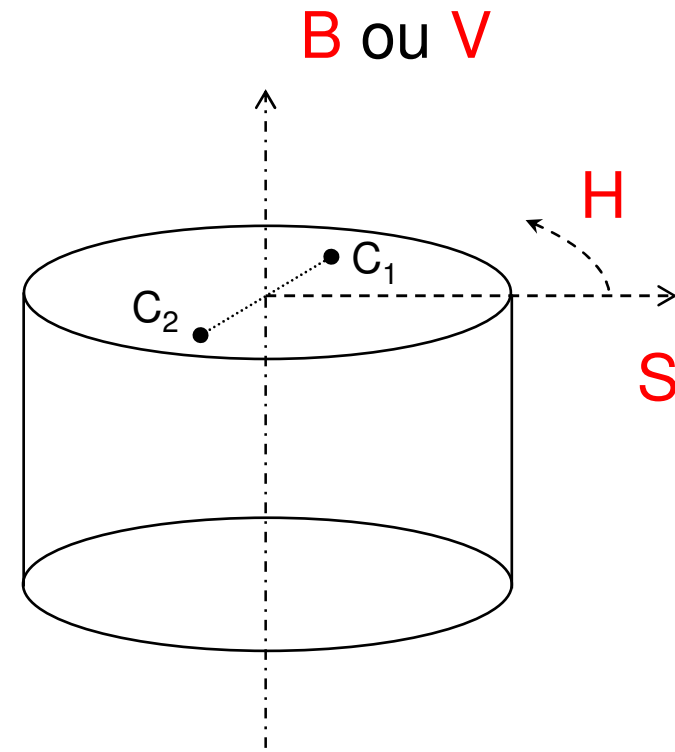
Modelo HSV (ou HSB)

- Sistema de coordenadas cilíndricas
- Cilindro (com as cores complementares em oposição, p.ex. C_1 e C_2)

Hue $[0^\circ, 360^\circ]$
Saturation $[0, 1]$
Brightness $[0, 1]$

ou

Hue $[0^\circ, 360^\circ]$
Saturation $[0, 1]$
Value $[0, 1]$



Modelo HSV

- ❖ Para $V=0$ (ou $B=0$) deveríamos ter apenas 1 ponto, pelo que, mais corretamente, o cilindro se pode substituir por um cone!
- ❖ O cálculo de V (ou B) é uma simplificação computacional grosseira do que deveria ser a intensidade.

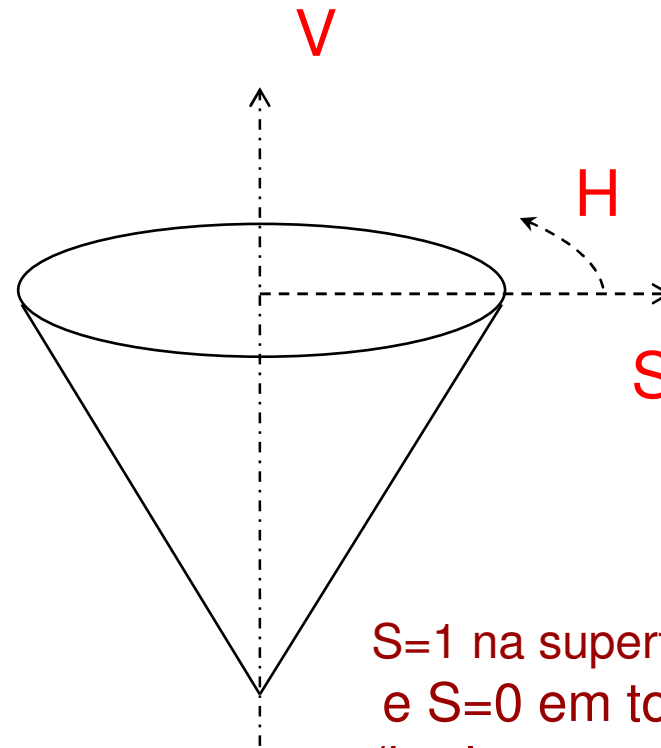
Conversão de RGB para HSV:

$$V = \max(R, G, B) ;$$

$$S = (V - \min(R, G, B)) / V ;$$

H (*v. algoritmo em Foley et al.*)

$S == 0 \longrightarrow H \text{ undefined}$



$S=1$ na superfície cônica
e $S=0$ em todo o eixo
(irrelevante no vértice)

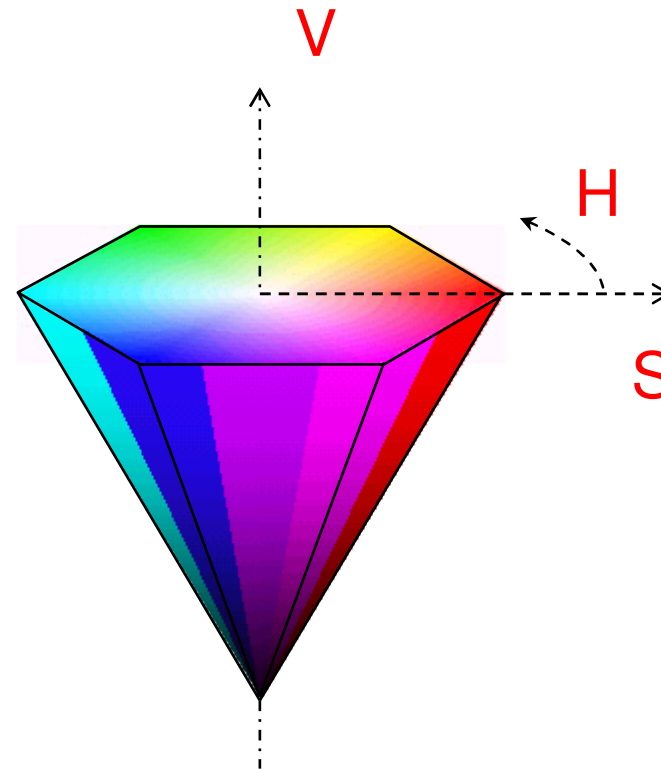
Modelo HSV

Representação alternativa com a substituição do cone pela pirâmide
(por analogia com o espaço RGB)

Hue
Saturation
Value

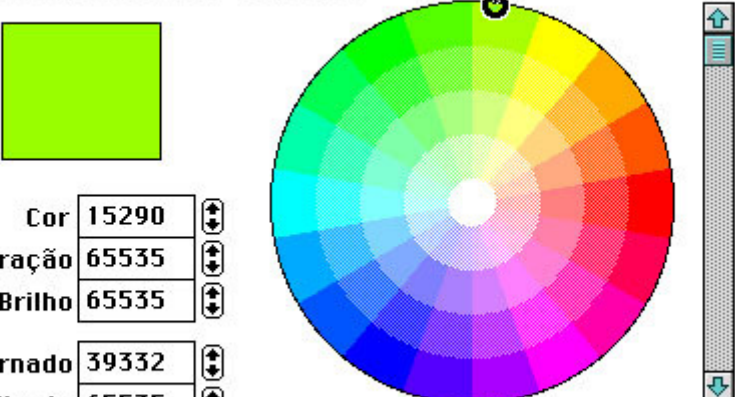


*Projeção ortogonal do cubo
segundo a direção (1,1,1)*



Modelo HSV: seleção de cores

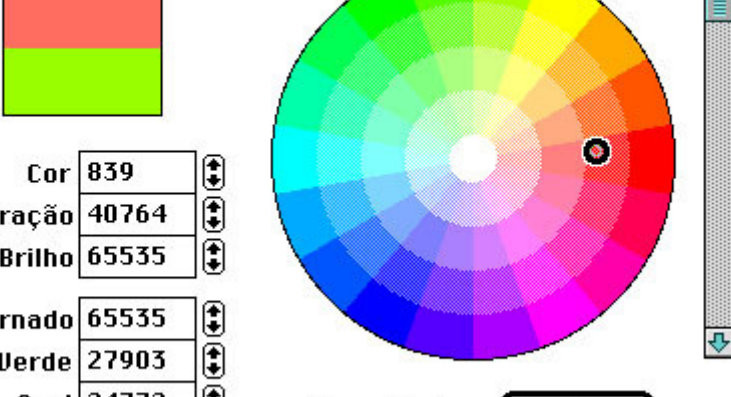
Selecione uma cor de destaque:



Cor	15290	↑↓
Saturação	65535	↑↓
Brilho	65535	↑↓
Encarnado	39332	↑↓
Verde	65535	↑↓
Azul	1	↑↓

Cancelar OK

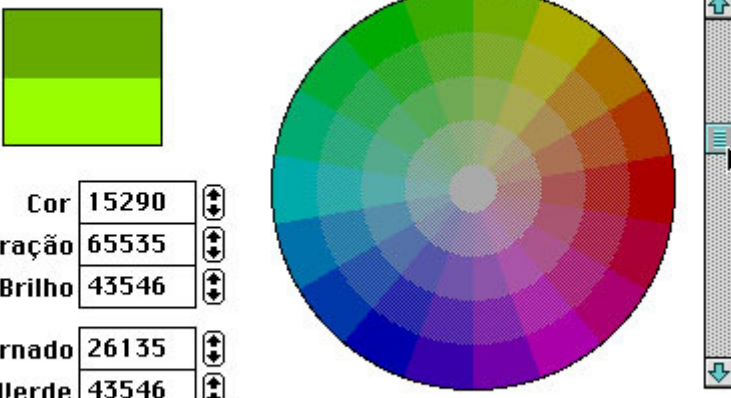
Selecione uma cor de destaque:



Cor	839	↑↓
Saturação	40764	↑↓
Brilho	65535	↑↓
Encarnado	65535	↑↓
Verde	27903	↑↓
Azul	24772	↑↓

Cancelar OK

Selecione uma cor de destaque:



Cor	15290	↑↓
Saturação	65535	↑↓
Brilho	43546	↑↓
Encarnado	26135	↑↓
Verde	43546	↑↓
Azul	1	↑↓

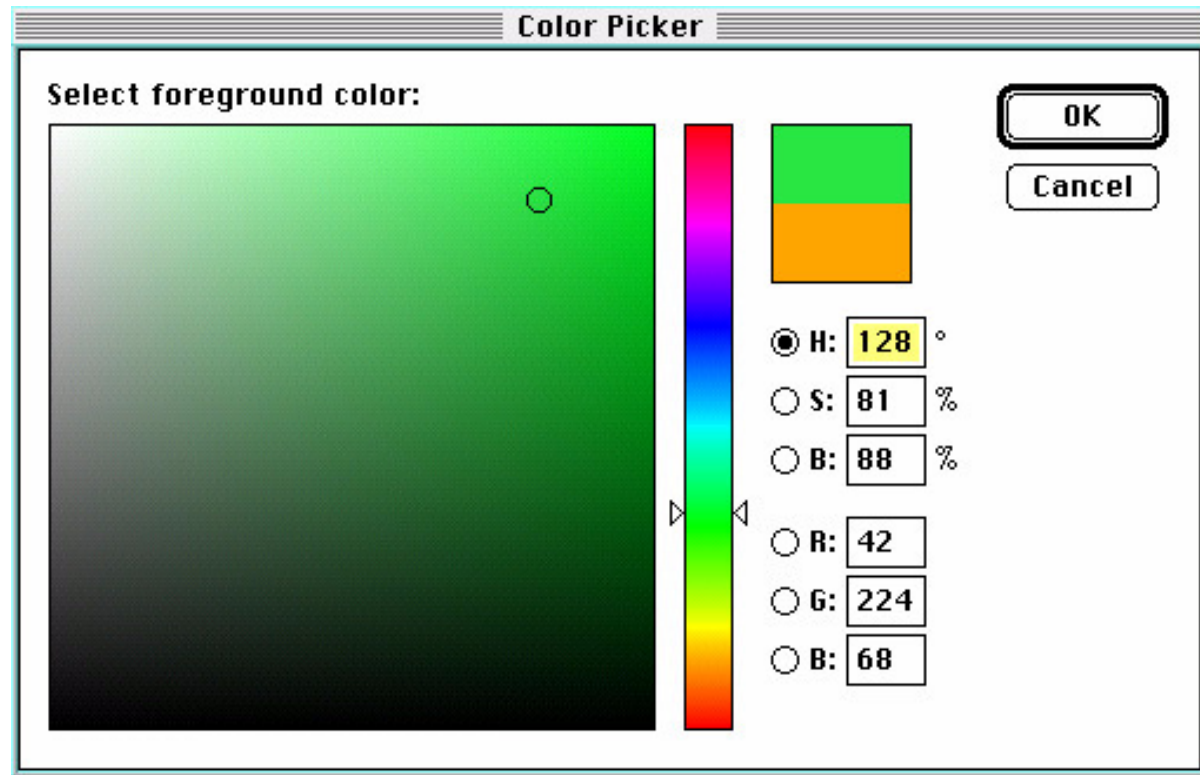
Cancelar OK

Hue
Saturation
Value

Modelo HSV: seleção de cores

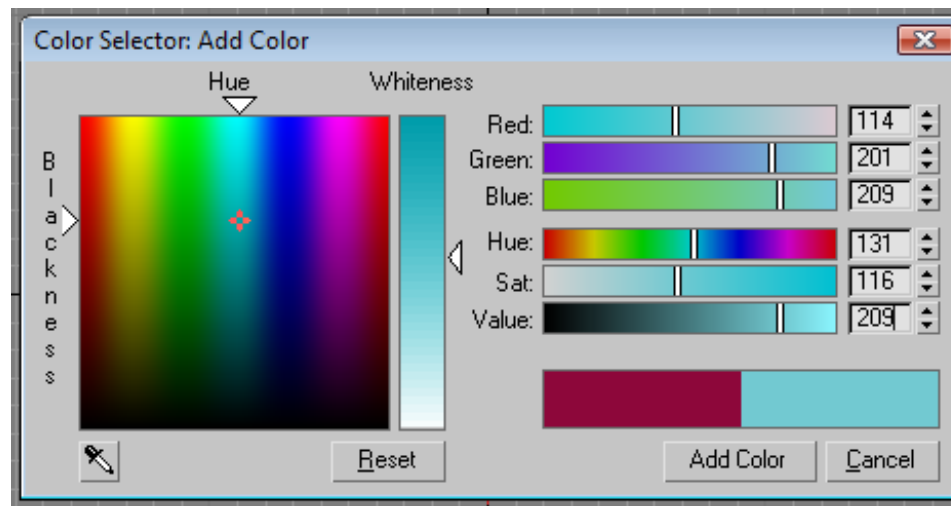
Hue
Saturation
Value

(Note-se, neste caso, o rigor tido na escolha das unidades em que se exprimem as componentes de HSB)



Modelo HSV: seleção de cores

Hue
Saturation
Value

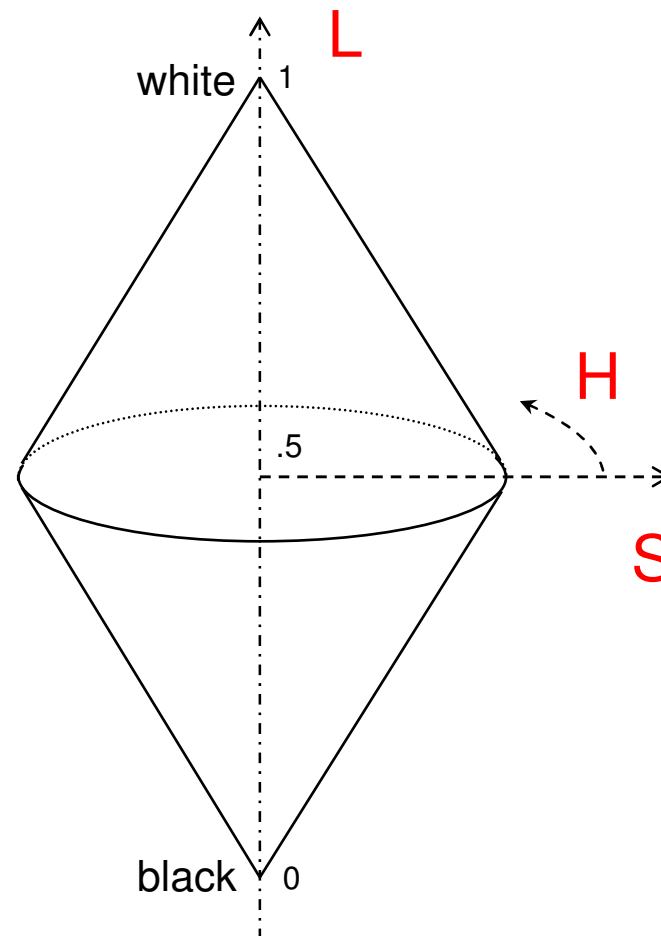


Modelo HLS

Hue
Lightness (Luminosidade)
Saturation

O branco, em HLS, é considerado a cor mais luminosa.
O espaço pode substituir-se por duas pirâmides hexagonais.

Curiosidade:
Desenvolvido pela Tektronix (1977), primitivamente atribuía **Blue** a $H=0^\circ$



Conversão do sistema RGB em HLS

$L = (\max(R,G,B) + \min(R,G,B)) / 2 ;$

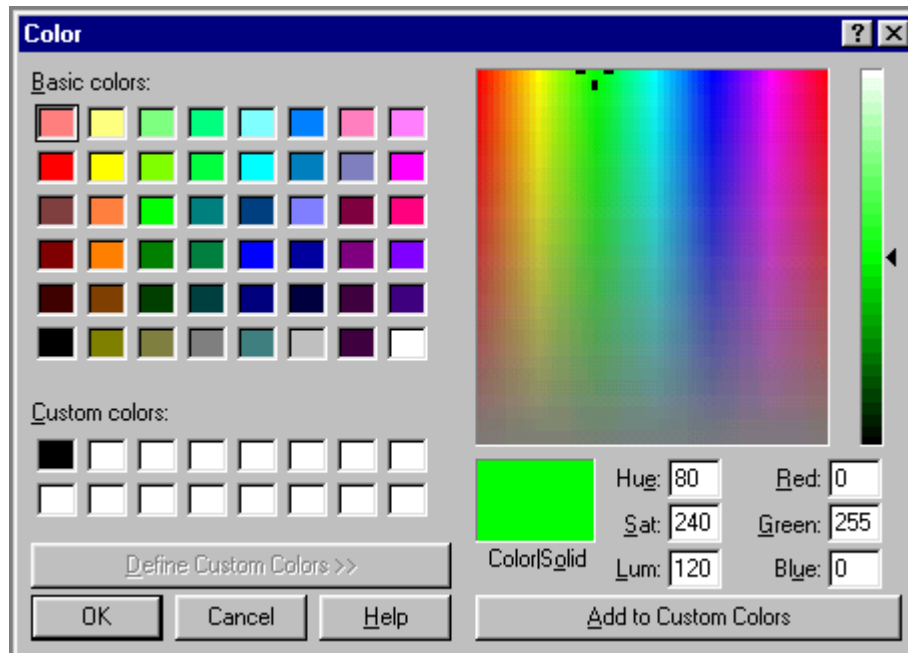
if ($\max(R,G,B) == \min(R,G,B)$) $S = 0 ;$ // H undefined

else if ($L \leq .5$) $S = (\max(R,G,B) - \min(R,G,B)) / (2 * L) ;$

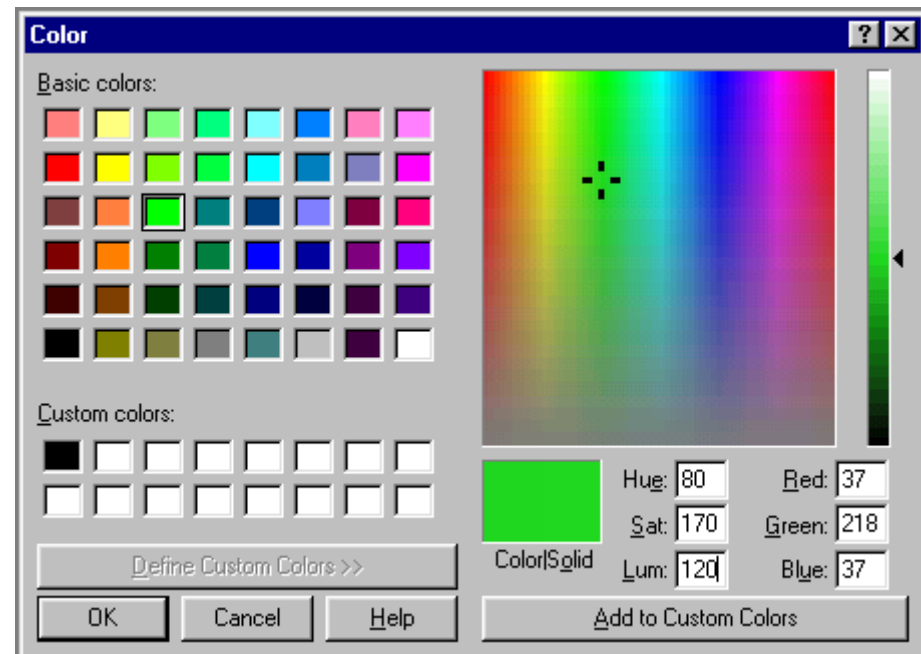
else $S = (\max(R,G,B) - \min(R,G,B)) / (2 - 2 * L) ;$

H (v. algoritmo no livro *Foley et al.*)

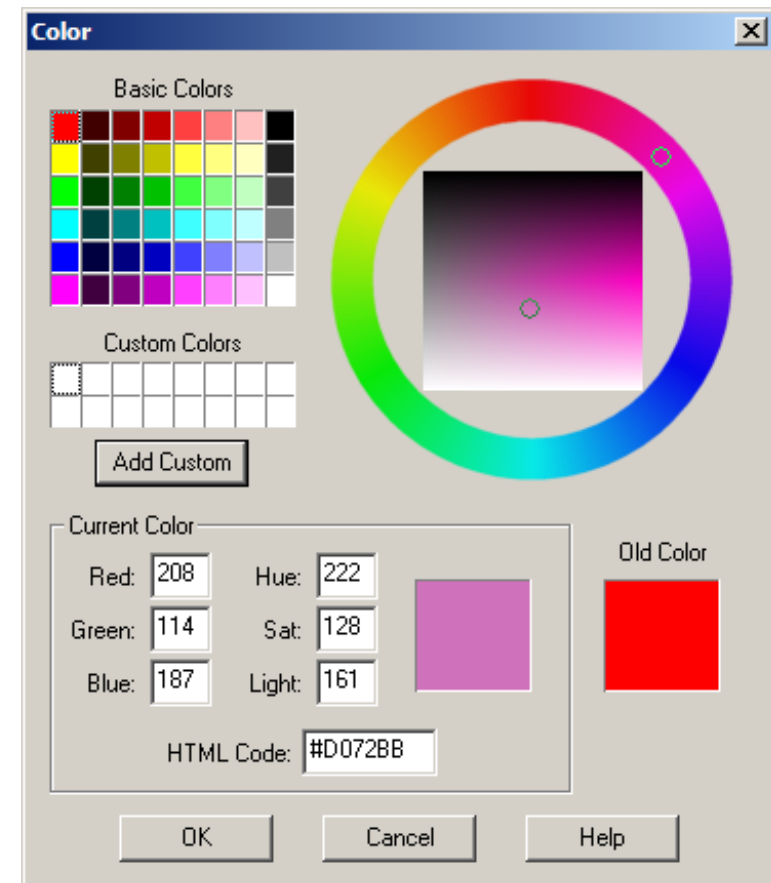
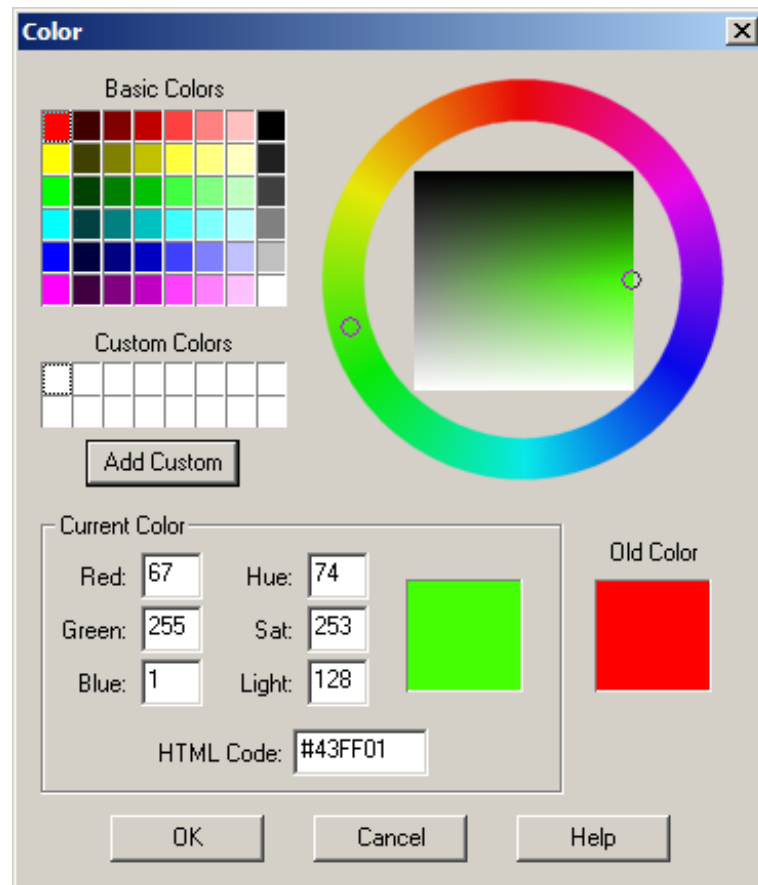
Modelo HLS: seleção de cores



Hue
Luminance
Saturation



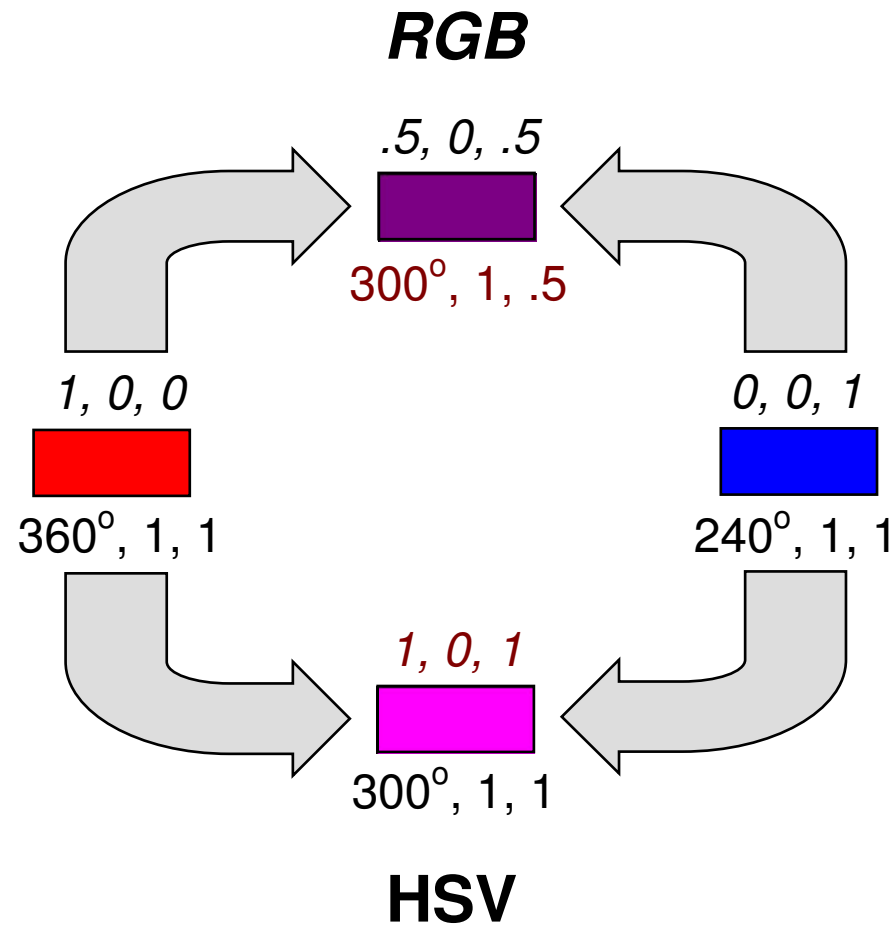
Modelo HLS: seleção de cores



Hue
Luminance
Saturation

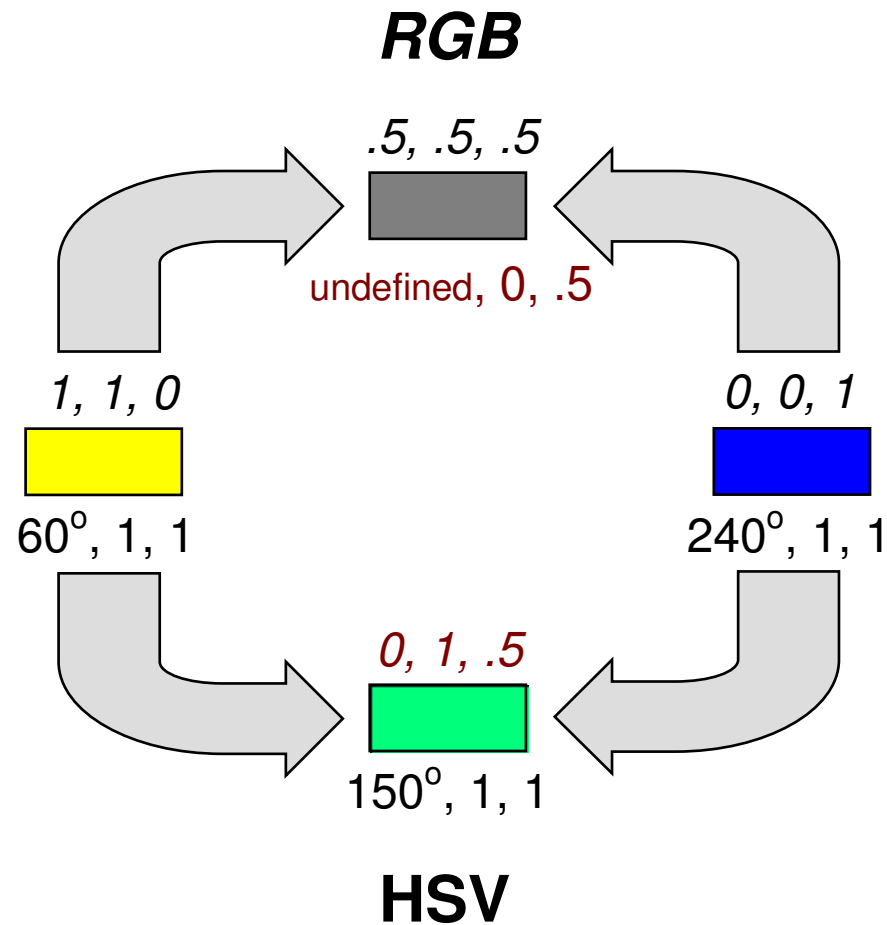
Interpolação linear de Cores (1)

O resultado
pode depender
do modelo de
cor utilizado



Interpolação linear de Cores (2)

O resultado
pode depender
do modelo de
cor utilizado



Uso e harmonia das cores (1)

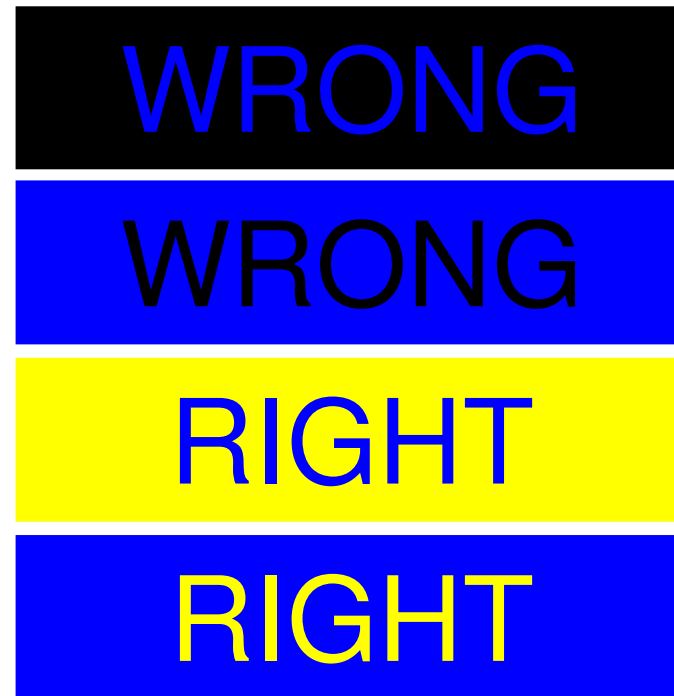
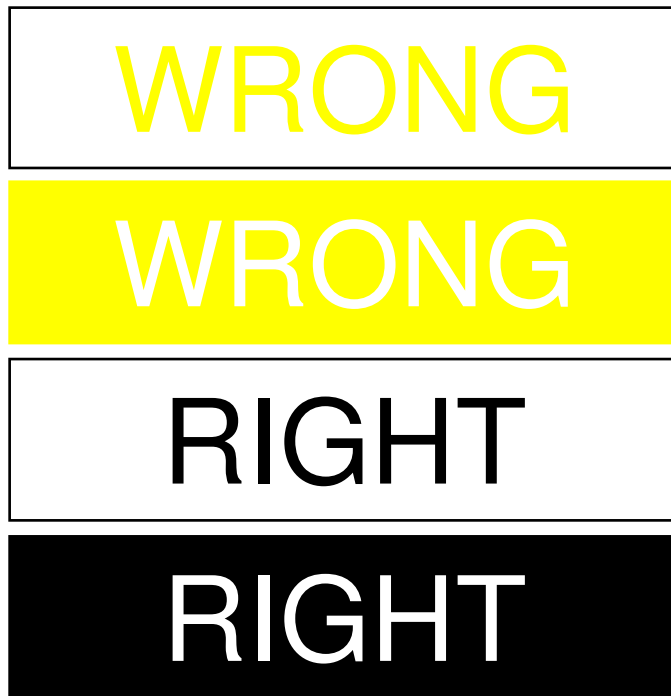
- Se cada cor tiver um significado especial que o utilizador da aplicação deva aprender, não usar mais do que 6 ou 7.
- Agrupar elementos através da utilização da mesma cor de fundo.
- Cores semelhantes conotam significados semelhantes.
 - Verde é mais semelhante a azul do que a vermelho.
- O grau da mudança da cor pode relacionar-se com a variação de grandezas próprias da aplicação.

Uso e harmonia das cores (2)

- Num gráfico com poucas cores, o complemento de uma delas deve ser usado para fundo.
- Num gráfico com muitas cores, o fundo deve ser um tom neutro.
- Se duas cores adjacentes não forem particularmente harmoniosas, dever-se-á separá-las por fina divisória em preto.

Uso e harmonia das cores (3)

- Evitar o uso de amarelo em fundo branco.
 - Evitar azul saturado sobre preto.



Uso e harmonia das cores (4)

- Cores opostas causam boa harmonia.
- Para obter cores harmoniosas selecioná-las segundo um determinado método.
 - Por exemplo, seguindo um determinado caminho no subespaço de um modelo de cores.
- Fornecer paletas com as cores ordenadas.
 - Por exemplo, na sequência de cores do arco-íris (*Hue*).

Uso e harmonia das cores (5)

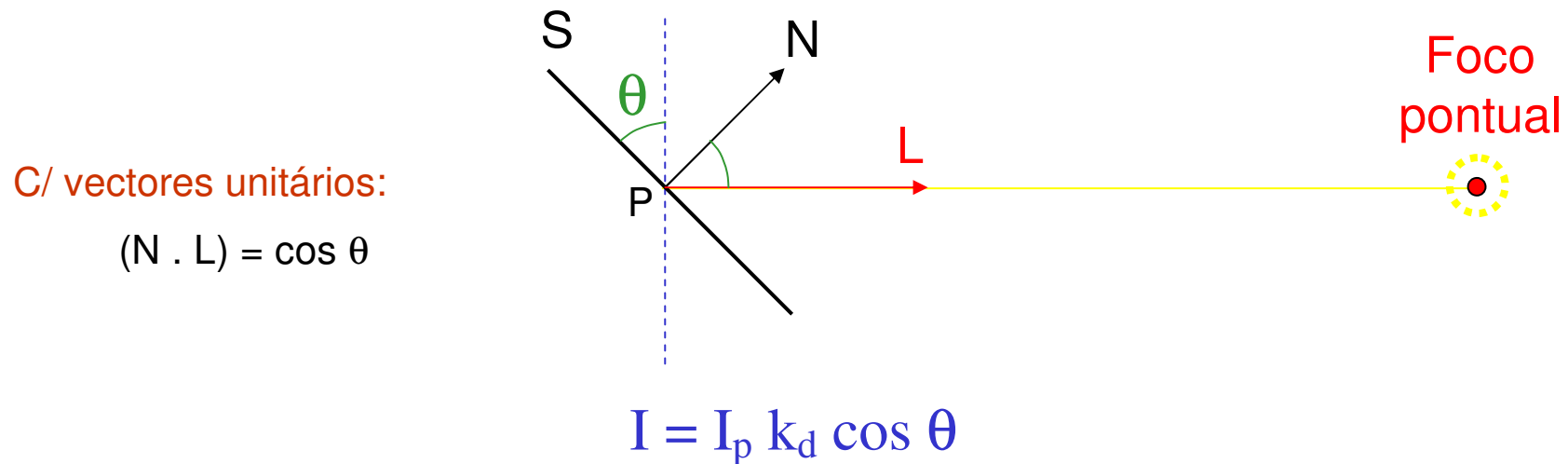
- Maior brilho e maior saturação chamam a atenção.
- A «temperatura da cor» deve indicar níveis de ação.
 - Cores frias para informação de *status* e cores quentes para pedidos de resposta.
- Usar as cores de modo consistente.
- Atender aos aspectos culturais relacionados com a cor.

Resposta emocional à Cor

Different colors evoke different reactions in viewers. Be aware that some of these reactions will be culturally specific. For example, in the U.S., brides often wear white as a symbol of purity and widows wear black as a symbol of mourning. However, in many asian cultures, brides wear black. The following is a guide as to the possible meaning of colors.

- Red - danger, stop, negative, excitement, hot
- Dark Blue - stable, calming, trustworthy, mature
- Light Blue - youthful, masculine, cool
- Green - growth, positive, organic, go, comforting
- White - pure, clean, honest
- Black - serious, heavy, death
- Gray - integrity, neutral, cool, mature
- Brown - wholesome, organic, unpretentious
- Yellow - emotional, positive, caution
- Gold - conservative, stable, elegant
- Orange - emotional, positive, organic
- Purple - youthful, contemporary, royal
- Pink - youthful, feminine, warm
- Pastels - youthful, soft, feminine, sensitive
- Metallic - elegant, lasting, wealthy

Modelo de Iluminação: Reflexão Difusa



I – intensidade luminosa num ponto P da superfície refletora S

I_p – intensidade do foco

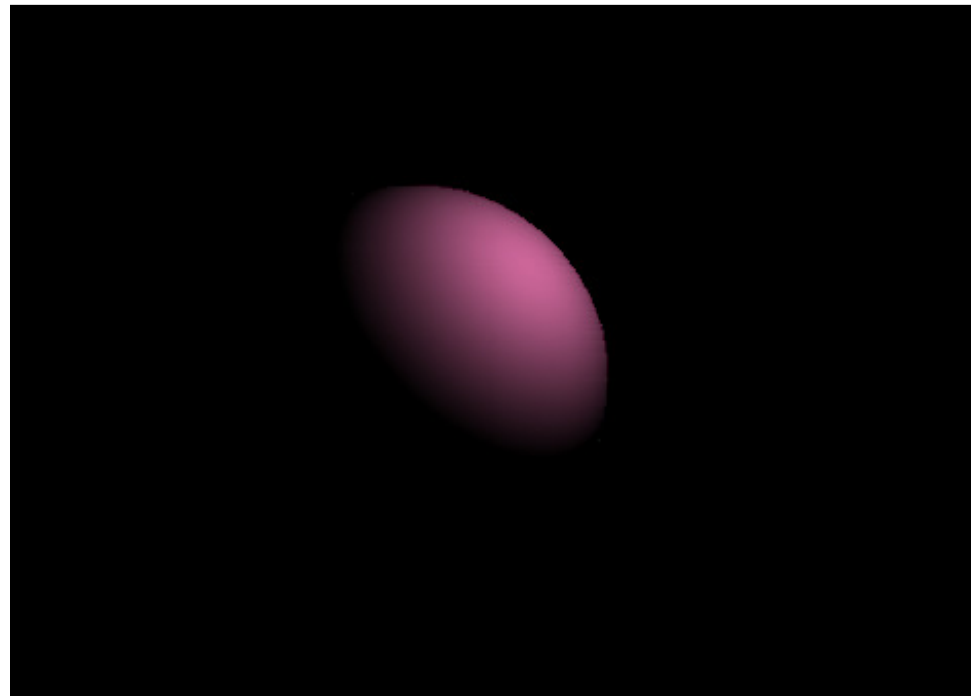
k_d – coeficiente de reflexão difusa $\in [0,1]$

θ – ângulo de incidência da luz

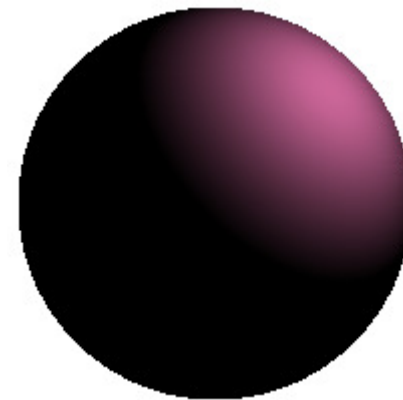
$(N \cdot L) = \max(\cos \theta, 0)$ —> ilumina só o lado considerado exterior da superfície

$(N \cdot L) = \text{abs}(\cos \theta)$ —> ilumina ambos os lados da superfície

Reflexão difusa: exemplo



Alteração da cor de fundo, para
melhor interpretação das formas



Reflexão difusa: composição em RGB

$$I_r = I_{pr} k_{dr} \cos \theta$$

$$I_g = I_{pg} k_{dg} \cos \theta$$

$$I_b = I_{pb} k_{db} \cos \theta$$

↓ | |
Componentes de cor do pixel

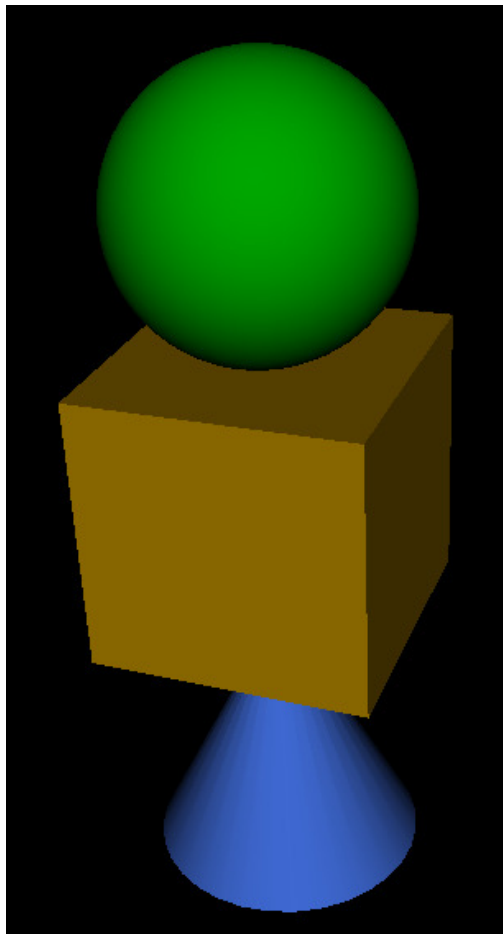
↓ |
Componentes da fonte de luz ($I_p=(1,1,1)$ para a luz branca,
 $I_p=(1,0,1)$ para luz magenta, etc.)

↓
Componentes do coeficiente de reflexão da superfície
($k_p=(0,1,0)$ para uma superfície verde)

Questão: Qual a cor percebida num objeto verde sob luz azul?

M. Próspero

Reflexão difusa em X3D



Headlight $\begin{cases} I_p=(1,1,1) \\ L=(0,0,1) \end{cases}$

k_{dr} k_{dg} k_{db}

<Material diffuseColor="0.0 0.8 0.0"/>

<Material diffuseColor="0.8 0.6 0.0"/>

<Material diffuseColor="0.3 0.5 1.0"/>

Dialogue Design

Logical Input Devices

Physical Devices are mapped onto Logical Input Devices,
which are divided into six different classes.

/ ISO/ANSI API standards: GKS, PHIGS, PHIGS+ **/**

Logical Input Classes

LOCATOR	Returns a position in World Coordinates.
STROKE	Returns a sequence of points in World Coordinates.
VALUATOR	Returns a real number.
CHOICE	Returns a selection (positive integer) from a set of alternatives.
PICK	Identifies a displayed and selected object (pick path).
STRING	Inputs a sequence of characters.

Crosshairs



re-creation in DVST
(old technology)

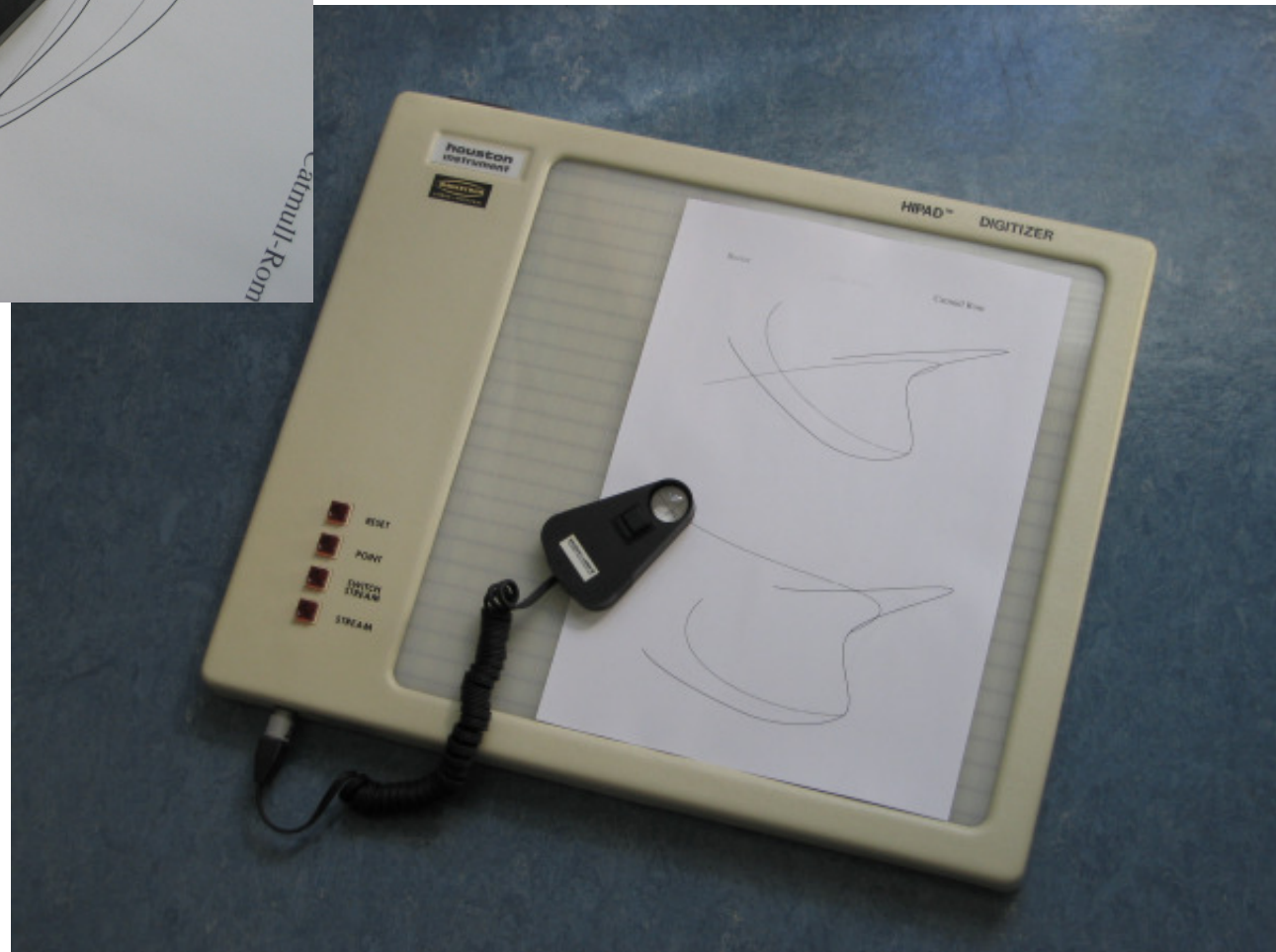


(1-button puck)

Physical Devices



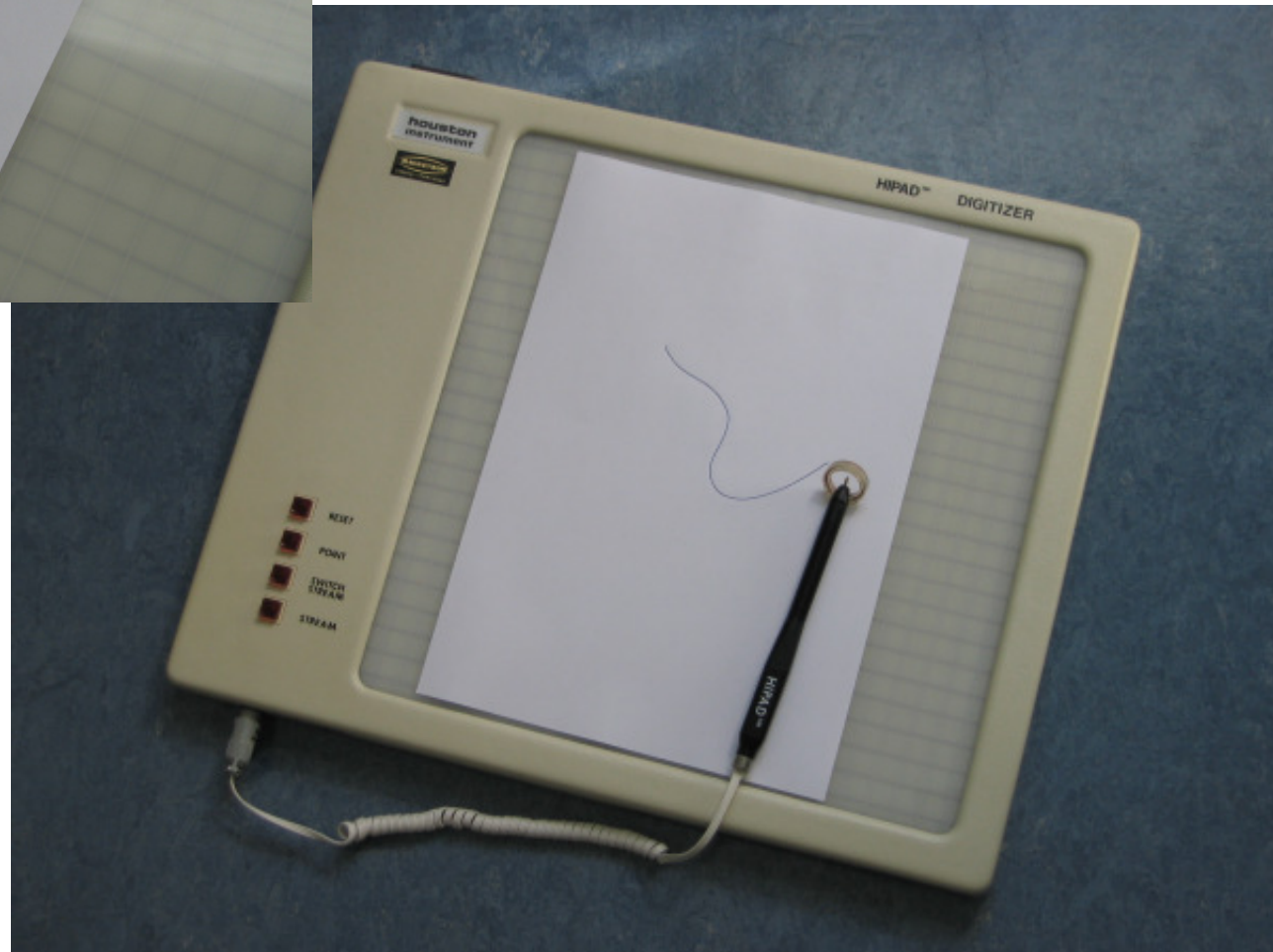
Tablet

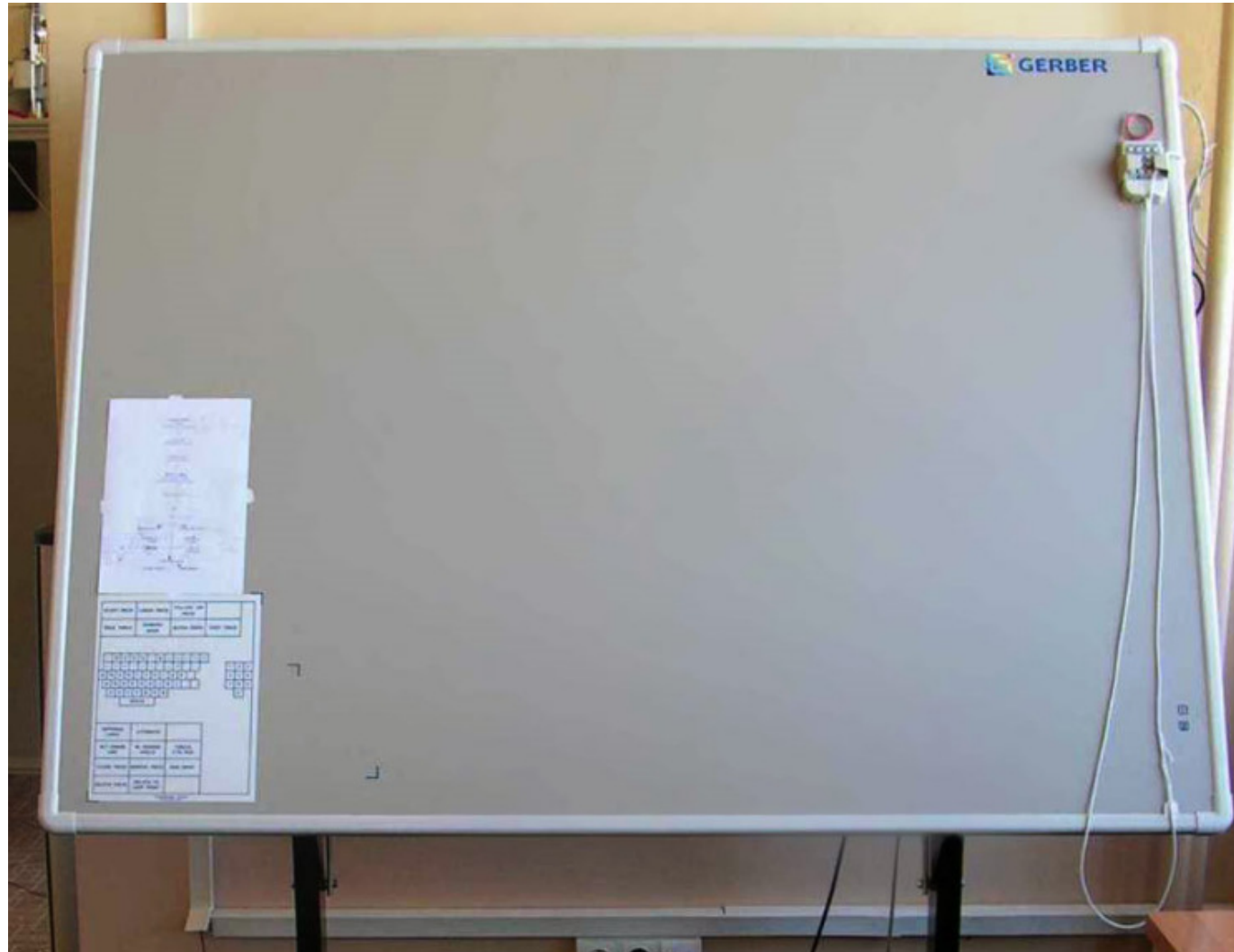


Physical Devices



Tablet





Tablet
(with
16-button
puck)

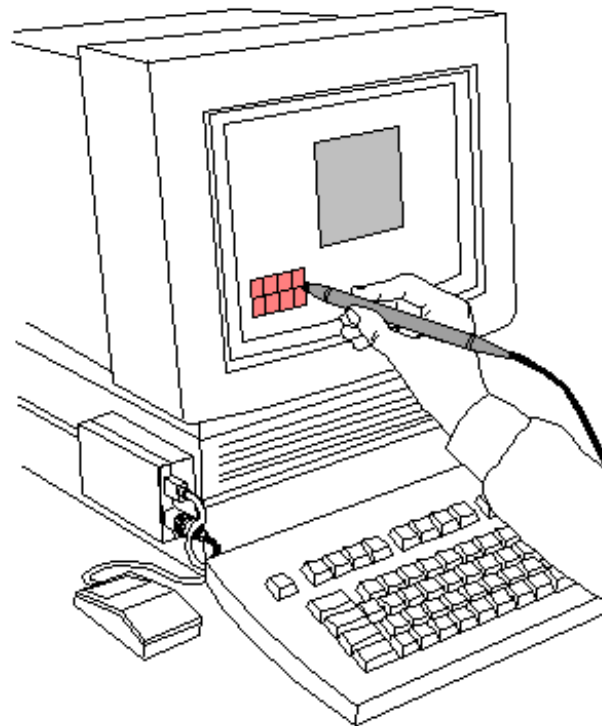
Dials
(Potentiometers)





Button Box

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co., Inc.



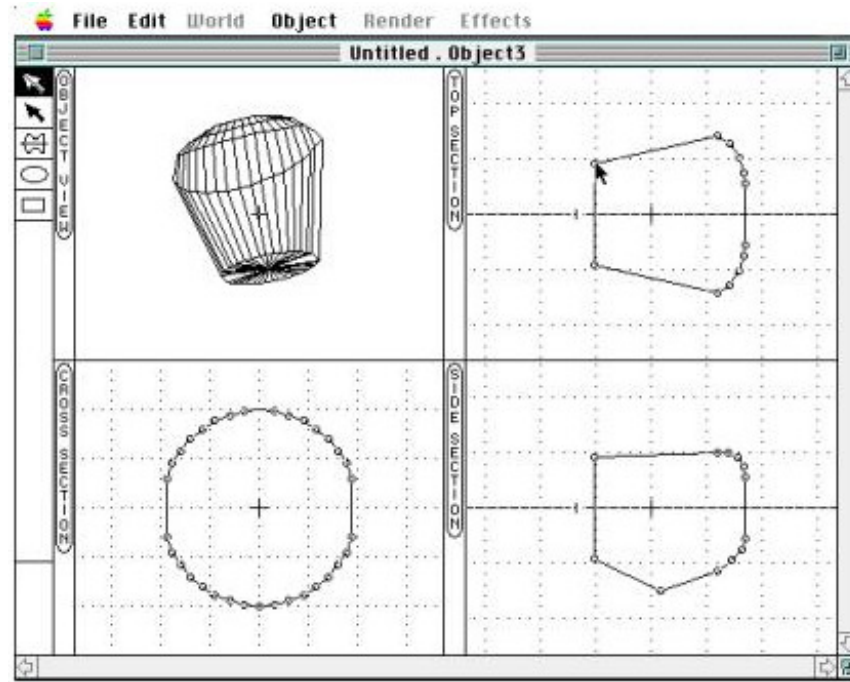
Light pen
(old technology)

The diagram illustrates the relationship between physical input devices and their corresponding abstractions. It features two large ovals: a light blue one on the left labeled "REAL DEVICES" and a light pink one on the right labeled "ABSTRACTIONS". A central white box with a purple border contains the title "Typical Physical Input Devices and mapping" and a table of mappings. Each row in the table consists of a device name, a series of yellow arrows pointing right, and an abstraction name.

Real Device	Abstraction
Crosshairs	LOCATOR
Tablet (Mesa Digitalizadora)	STROKE
Potenciometer	VALUATOR
Button Box	CHOICE
Light pen	PICK
Keyboard	STRING

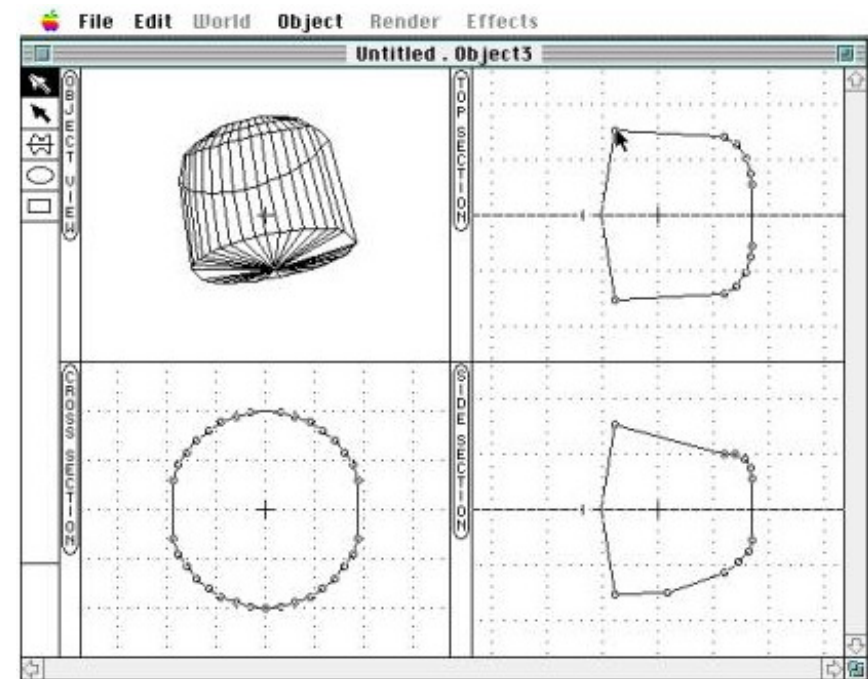
Note: the keyboard can emulate any input class !

Note: the keyboard can emulate any input class !



PICK

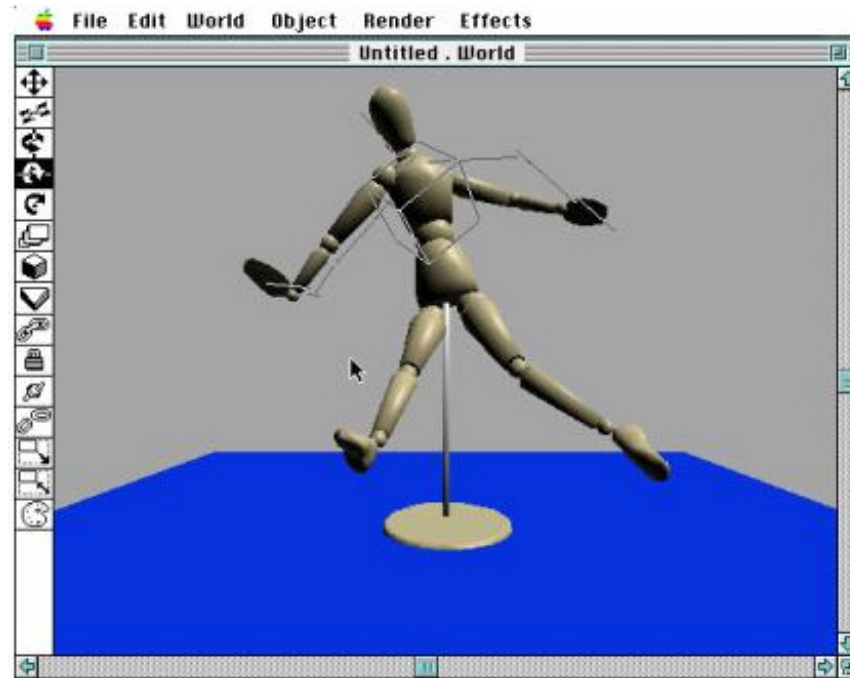
LOCATOR





PICK
(with feedback)

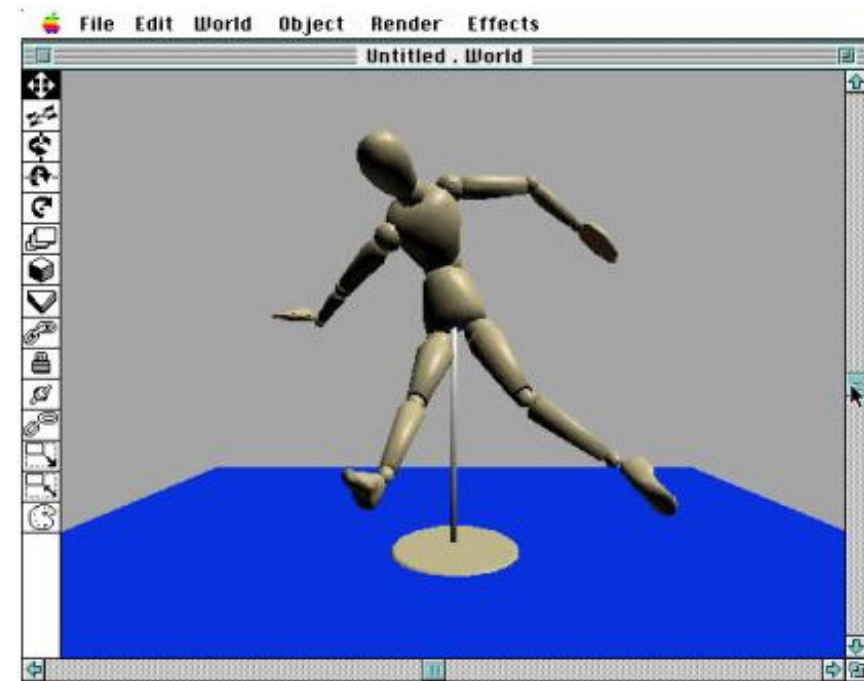


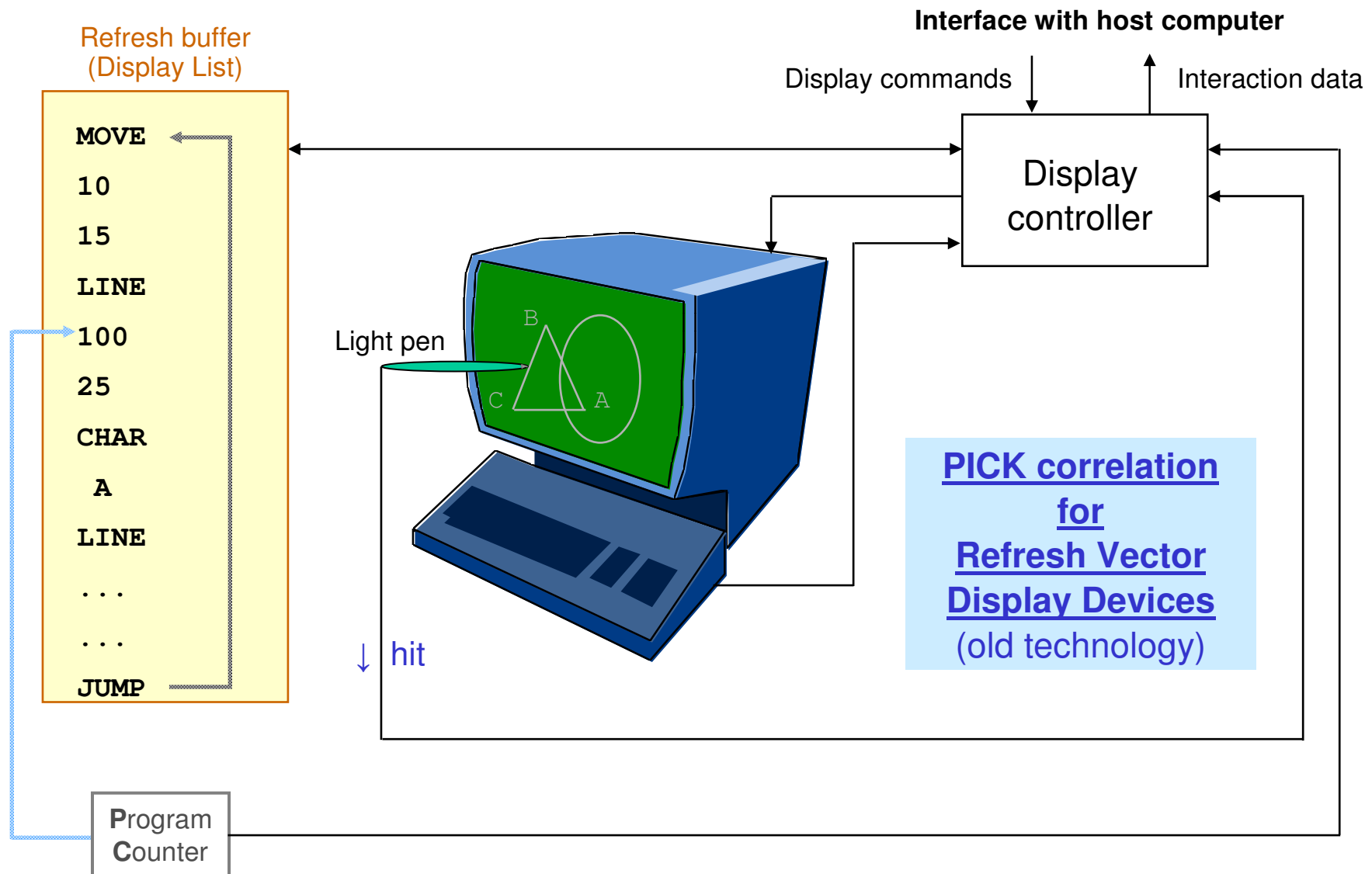


CHOICE

followed by

VALUATOR





When a hit is detected by the Display Controller, the PC indicates the location of the selected object in the code

PICK CORRELATION TRAVERSAL FOR RASTER GRAPHICS

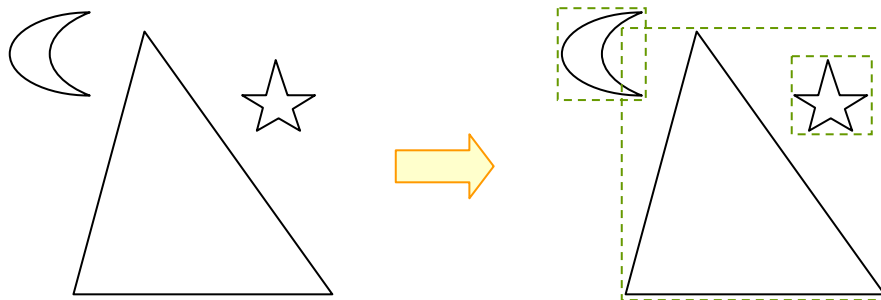
(1) ANALYTICAL HIT DETECTION

Algebraic equations are used to determine whether the DC (device coordinate) primitive lies sufficiently close to the 2D DC locator measure.

Algorithms are needed for:

- Computing the distance from the cursor position to each line segment.
- Determining if the cursor position lies inside a polygon [see the even-odd fill area algorithm].
- Comparing the locator position to the rectangular screen extent for nongeometric text.

Optimization/approximation can be done by using **Screen Extents**:



Problems with the selection of the *star* ?

A possible solution: priority lists based on the object size
(*star* → *moon* → *triangle*)

PICK CORRELATION TRAVERSAL FOR RASTER GRAPHICS

(2) HIT DETECTION VIA CLIPPING

Hardware clipping devices and/or optimized software clipping utilities allow the application to determine whether any part of a primitive's image lies inside a 2D integer clip rectangle (a small square surrounding the cursor position called **PICK WINDOW**) without having actually to draw the primitive on the screen for that purpose.

One of the following methods can be used :

- Drawing into an offscreen pixmap (buffer) and checking if any pixels are changed.
- Hit detection returned by the clipper.

A stack of names is used and the current name is read every time a hit is detected. Then we can easily identify the associated set of primitives responsible for the hit.

To be compared to the Light Pen technique (see applet example)

Próximos capítulos: 2.º Ciclo

Interação Pessoa-Máquina

&

