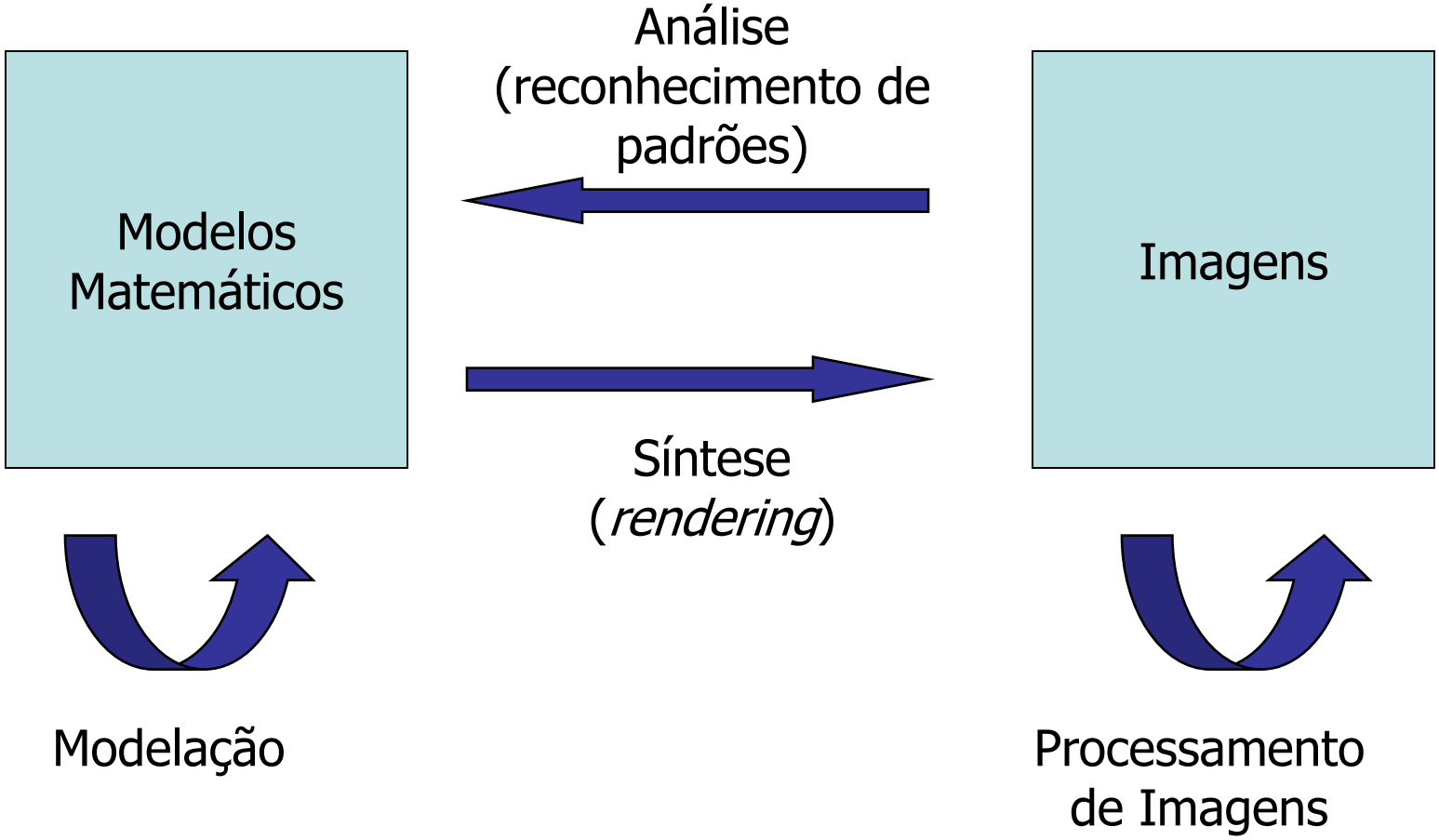
Introdução à Computação Gráfica
Preâmbulo

Adaptação: João Paulo Pereira
António Costa

Autoria: Claudio Esperança
Paulo Roma Cavalcanti

Computação Gráfica



Áreas relacionadas

- Computação
  - Algoritmos
  - Estruturas de Dados
  - Métodos Numéricos
- Matemática
  - Geometria
  - Álgebra Linear
- Física
  - Óptica
  - Mecânica
- Psicologia
  - Percepção
- Artes

Aplicações

- Desenho Assistido por Computador (CAD)
- Desenho Geométrico Assistido por Computador (CAGD)
- Sistemas de Informação Geográfica (SIG)
- Visualização Científica
- Visualização Médica
- Educação
- Entretenimento / Lazer

Representações Gráficas

- Gráficos "Vectoriais"
  - Representados por colecções de objectos geométricos
    - Pontos
    - Rectas
    - Curvas
    - Planos
    - Polígonos
- Gráficos "Matriciais"
  - Amostragem em grelhas rectangulares
  - Tipicamente imagens digitais
    - Matrizes de "pixels"
    - Cada pixel representa uma cor
  - Dados volumétricos
    - Imagens médicas
    - Cada pixel representa densidade ou intensidade de um campo físico

Representações Vectoriais

- Permitem uma série de operações (quase) sem perda de precisão
  - Transformações lineares / afim
  - Deformações
- Porquê "quase"? Estruturas de dados utilizam pontos e vectores cujas coordenadas são números reais
  - É necessário usar aproximações
    - Representação em vírgula-flutuante
    - Números racionais
- Complexidade de processamento = O (n.º vértices / vectores)
- Visualização
  - Dispositivos vectoriais
  - Dispositivos matriciais (requer amostragem, i.e., rasterização)

Representações Matriciais

- Representação flexível e muito comum
- Complexidade de processamento = *O* (n.º de pixels)
- Muitas operações implicam uma perda de precisão (reamostragem)
  - Ex.: rotação, escala
  - Técnicas para lidar com o problema
    - Ex.: técnicas anti-discretização (*anti-aliasing*)
- Visualização
  - Dispositivos matriciais
  - Dispositivos vectoriais (requer uso de técnicas de reconhecimento de padrões)

Conversão entre representações

Repr. Vectoriais

↓ ↑

Rasterização, *Reconhecimento*

"Scan conversion" *de padrões*

↓ ↑

Repr. Matriciais

Dispositivos Gráficos

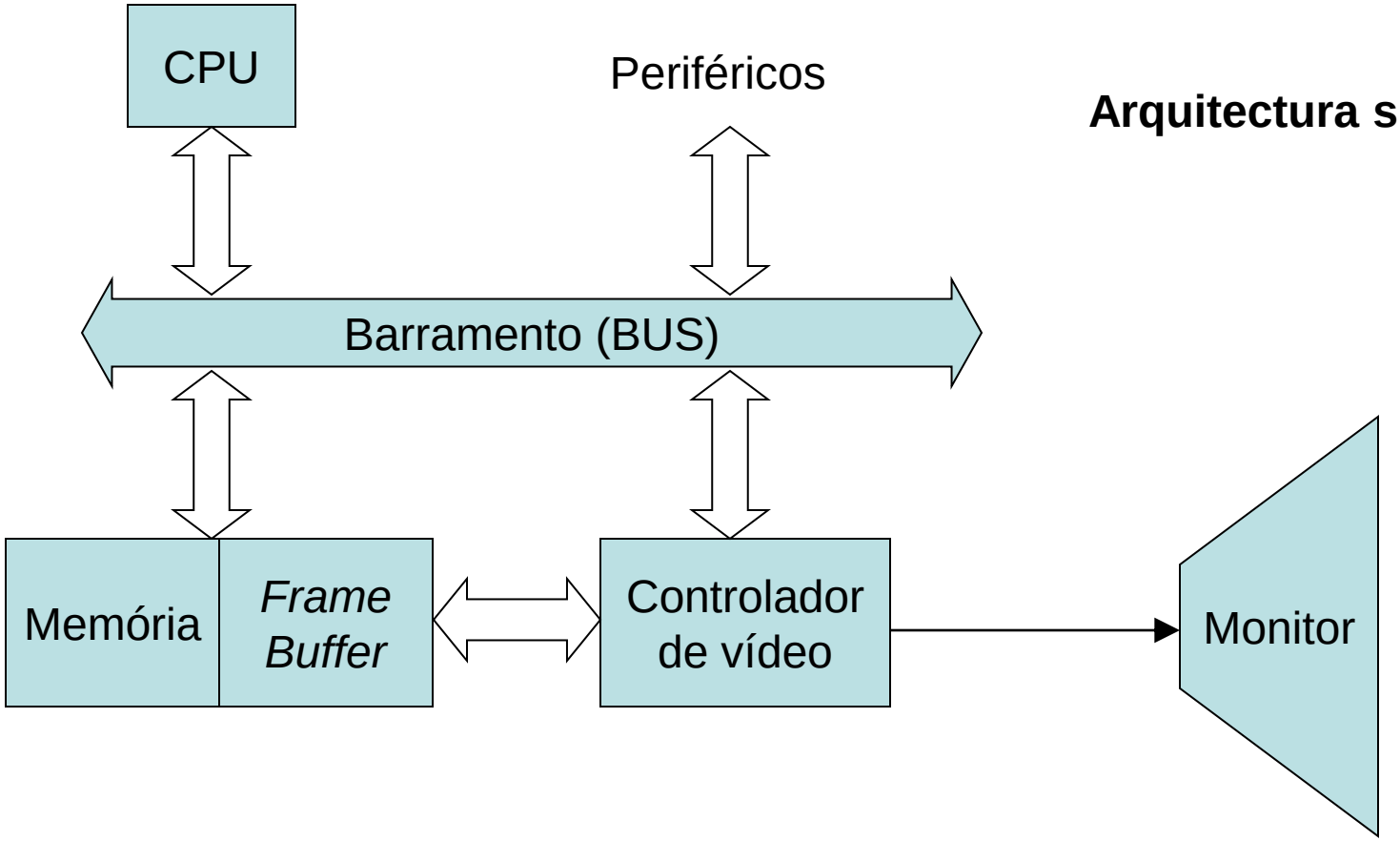
- Dispositivos Vectoriais
  - Terminais gráficos vectoriais (obsoletos)
  - Traçadores (*plotters*)
  - Dispositivos virtuais
    - Ex.: Linguagens de descrição de página (HPGL / Postscript)
    - Rasterização implícita
- Dispositivos Matriciais
  - Praticamente sinónimo de dispositivo gráfico
  - Impressoras, *displays*

Displays

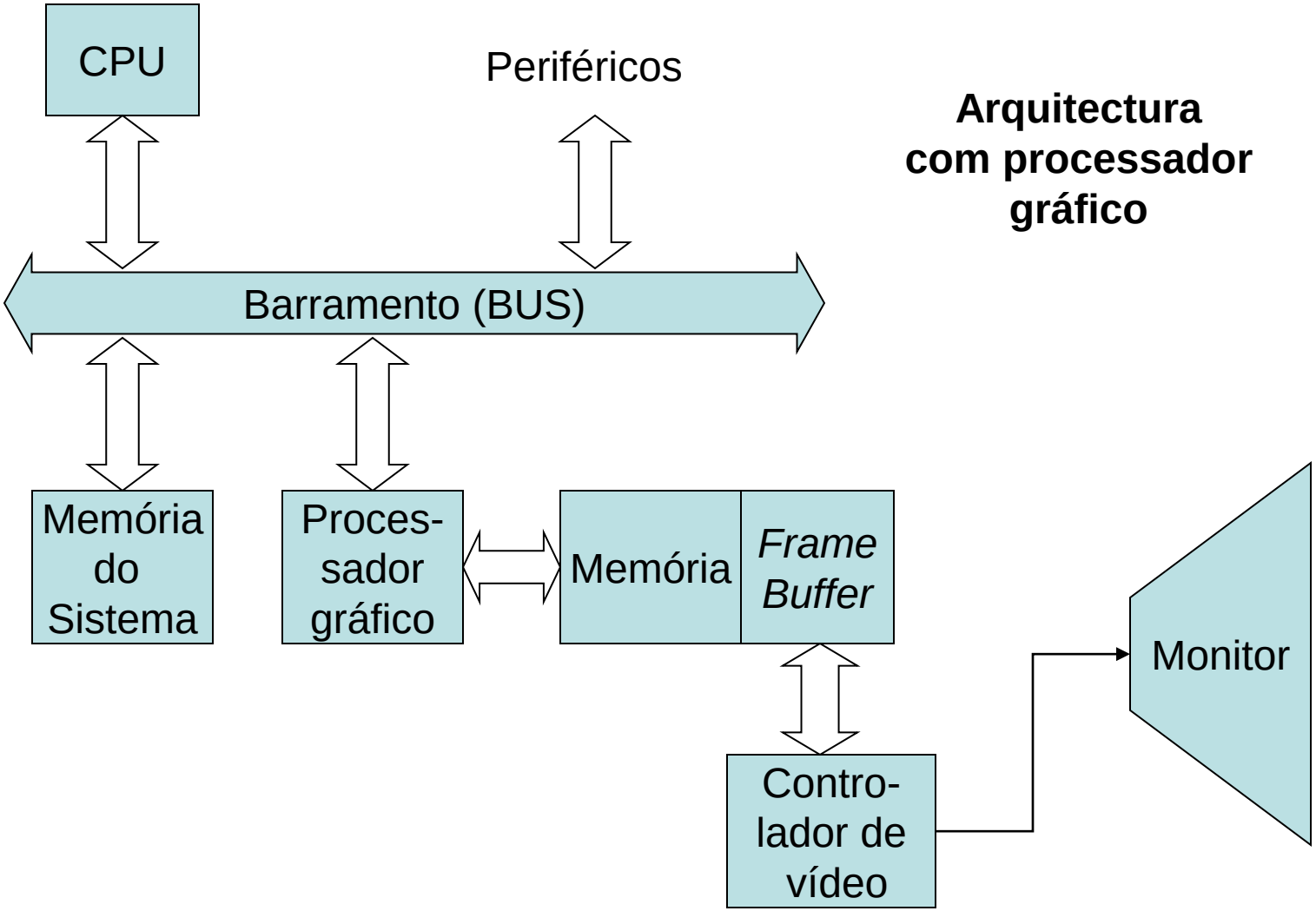
- Resolução espacial
  - Tipicamente de 640x480 até 1600x1200
  - Tendência para aumentar
- Resolução no espaço de cor
  - Monocromático (preto e branco)
    - Praticamente restrito a equipamentos de baixo custo
  - Tabela de cores
    - Cada pixel é representado por um número (tipicamente 8 bits – de 0 a 255) que indexa uma tabela de cores (tipicamente RGB 24 bits)
    - Poucas (ex.: 256) cores simultâneas mas cada cor pode ser escolhida de um universo grande (ex.: 2^{24})
    - Problema da quantização de cores
  - RGB
    - Cor é expressa por quantidades discretas de vermelho (*red*), verde (*green*) e azul (*blue*)
    - Tipicamente 24 bits (8 bits para cada componente)
    - Quando o número de bits não é divisível por 3, a resolução do azul costuma ser menor do que a das outras 2 componentes

Arquitectura de Sistemas Gráficos

Arquitectura simples



Arquitectura de Sistemas Gráficos



Arquitectura com processador gráfico

Processador (acelerador) gráfico

- Hardware especializado
- Uso de paralelismo para atingir alto desempenho
- Alivia o CPU do sistema de algumas tarefas, incluindo:
  - Transformações
    - Rotação, translação, escala, etc.
  - Recorte (clipping)
    - Supressão de elementos fora da janela de visualização
  - Projecção (3D $\rightarrow$2D)
  - Mapeamento de texturas
  - Rasterização
  - Amostragem de curvas e superfícies paramétricas
    - Geração de pontos a partir de formas polinomiais
- Normalmente usa memória separada da do sistema
  - Maior largura de banda no seu uso

Programação Gráfica

- À primeira vista: basta desenhar
  - Uma função para desenhar cada tipo de objecto
- Mas ...
  - Como fazer a interacção?
  - Como estruturar a cena?
  - Como controlar os atributos dos objectos?
  - Como resolver problemas de visibilidade?
  - Como suportar diversos dispositivos gráficos?
  - Como fazer programas independentes dos sistemas operativos?
- Ferramentas
  - API gráficas (ex.: OpenGL, WebGL, Three.js, Babylon.js)

Introdução à Computação Gráfica
Geometria

Adaptação: João Paulo Pereira
António Costa

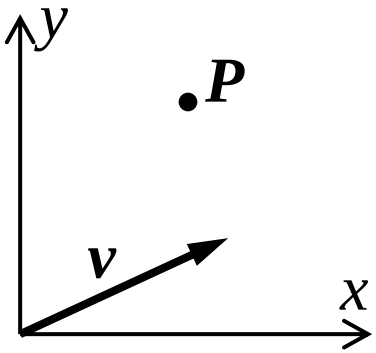
Autoria: Claudio Esperança
Paulo Roma Cavalcanti

Pontos e Vectores (2D)

- Ponto: Denota posição no plano
- Vector: Denota deslocamento, isto é, inclui a noção de direcção e magnitude
- Ambos são normalmente expressos por pares de coordenadas (em 2D) mas <u>não são a "mesma coisa"</u>

$$P = (x_P, y_P)$$

$$\vec{v} = (x_v, y_v)$$



Operações com Pontos e Vectores (2D)

- Soma de vectores

  $t = v + u$

- Multiplicação de vector por escalar

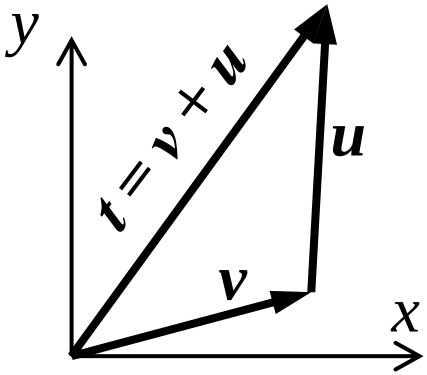
  $u = 2\,v$

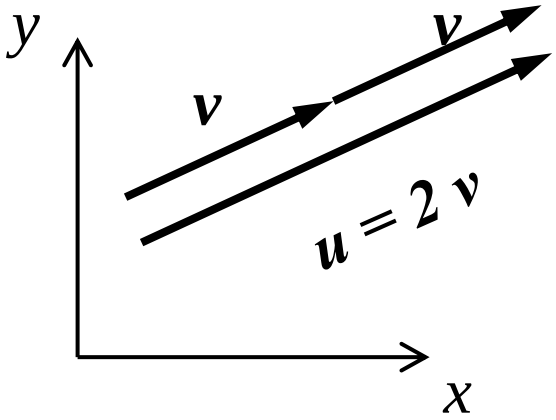
- Subtracção de pontos

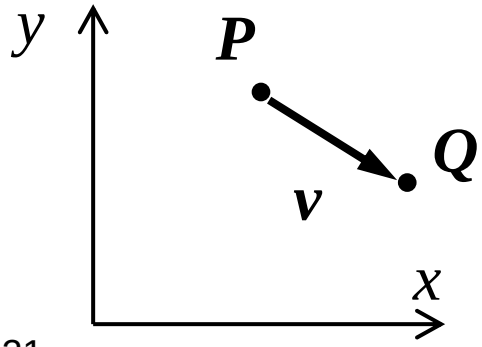
  $v = Q - P$

- Soma de ponto com vector

  $Q = P + v$







Transformações

- Transformação é uma função que faz corresponder pontos de um espaço Euclidiano a outros (ou possivelmente os mesmos) pontos do mesmo espaço
- Se uma transformação é <u>linear</u>, então
  - Se um conjunto de pontos pertence a uma recta, depois de transformados eles também pertencerão a uma recta
  - Se um ponto P guarda uma relação de distância com dois outros pontos Q e R, então essa relação de distância é mantida pela transformação
- Transformação faz corresponder a origem à origem?
  - Sim: Transformação *Linear*
  - Não: Transformação *Linear Afim*: Translações são permitidas

Transformações Lineares em 2D

- Uma transformação linear

$$\begin{cases} x' = ax + by \\ y' = cx + dy \end{cases}$$

- Uma transformação linear afim

$$\begin{cases} x' = ax + by + e \\ y' = cx + dy + f \end{cases}$$

Forma Matricial

- Mais conveniente para uso em computador. Sejam

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad D = \begin{bmatrix} e \\ f \end{bmatrix}$$

- Então uma transformação linear afim pode ser escrita $T(P) = P'$ em que

$$P' = A \times P + D$$

Transformação de Vectores

- Um vector não está aplicado a um ponto no espaço
- Uma transformação linear afim aplicada a um vector não inclui translação
- Demonstração: Seja V um vector e V' a sua imagem sob a transformação linear afim. Então:

$$V = Q - P \Leftrightarrow V' = Q' - P'$$

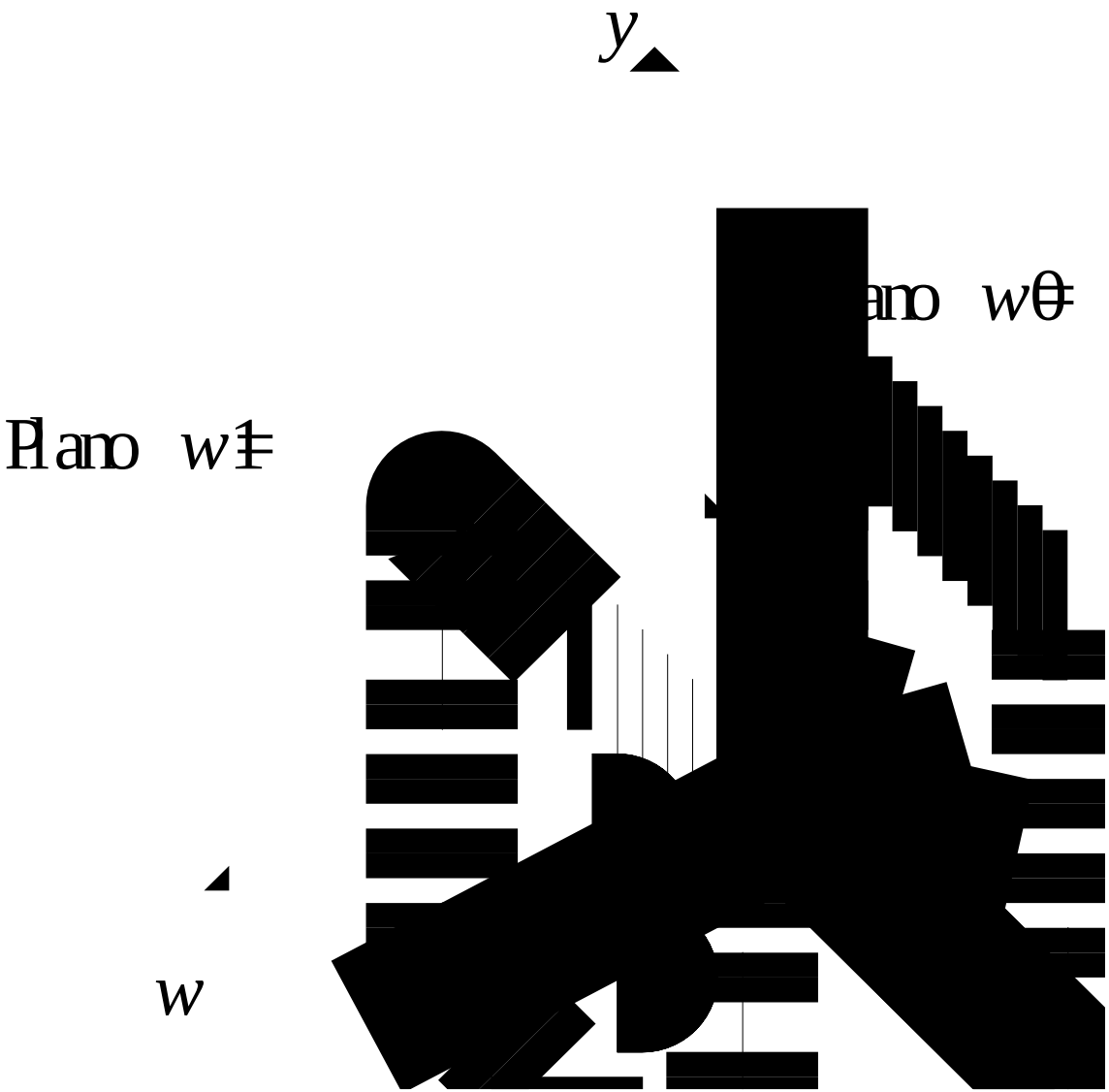
$$\begin{aligned} V' &= Q' - P' \\ &= (A \times Q + D) - (A \times P + D) \\ &= A \times (Q - P) \\ &= A \times V \end{aligned}$$

Coordenadas Homogéneas

- A transformação de vectores é operacionalmente diferente da de pontos
- Coordenadas homogéneas permitem unificar o tratamento
- Problema é levado para uma dimensão superior:
  - Coordenada extra $w = 0$ para vectores e $w = 1$ para pontos
  - Termos independentes formam uma coluna extra na matriz de transformação

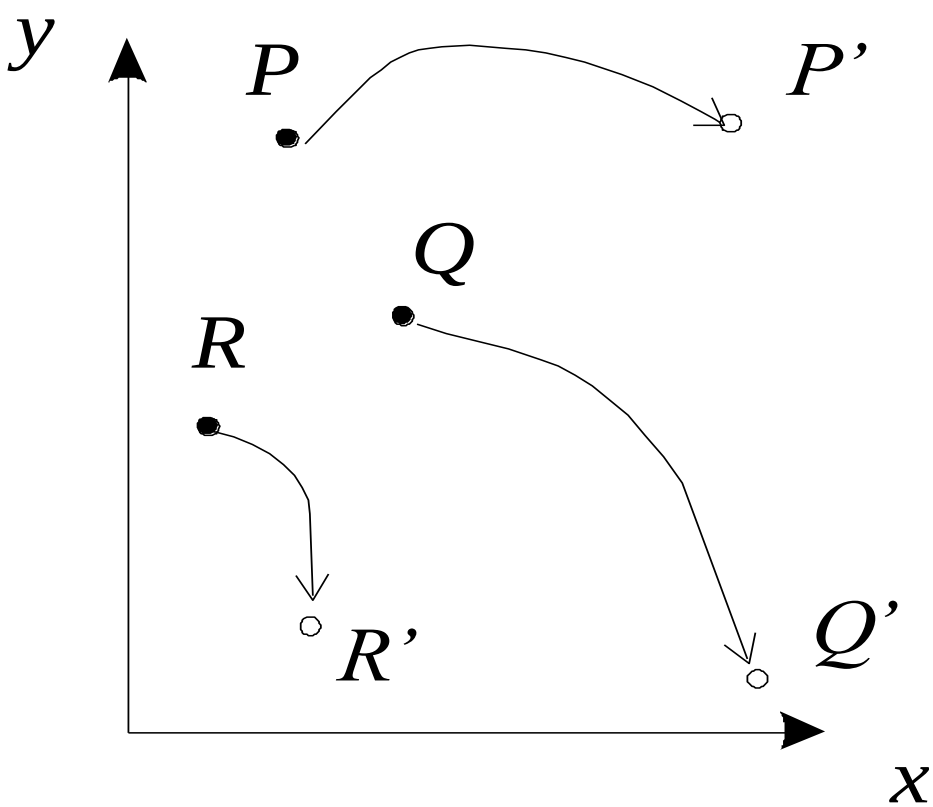
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \\ 0 \end{bmatrix} = \begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

Coordenadas Homogéneas - Interpretação



Modelação de Transformações

- Uma t.l.a. em 2D pode ser definida se dispusermos da imagem de 3 pontos do domínio



$$\begin{cases} x_{P'} = a.x_P + b.y_P + e \\ y_{P'} = c.x_P + d.y_P + f \\ x_{Q'} = a.x_Q + b.y_Q + e \\ y_{Q'} = c.x_Q + d.y_Q + f \\ x_{R'} = a.x_R + b.y_R + e \\ y_{R'} = c.x_R + d.y_R + f \end{cases}$$

Sistemas de coordenadas

- Um sistema de coordenadas para $\mathbf{R}^n$ é definido por um ponto (origem) e n vectores
- Ex.: Seja um sistema de coordenadas para $\mathbf{R}^2$ definido pelo ponto O e os vectores X e Y. Então
  - Um ponto P é dado por coordenadas x_P e y_P tais que

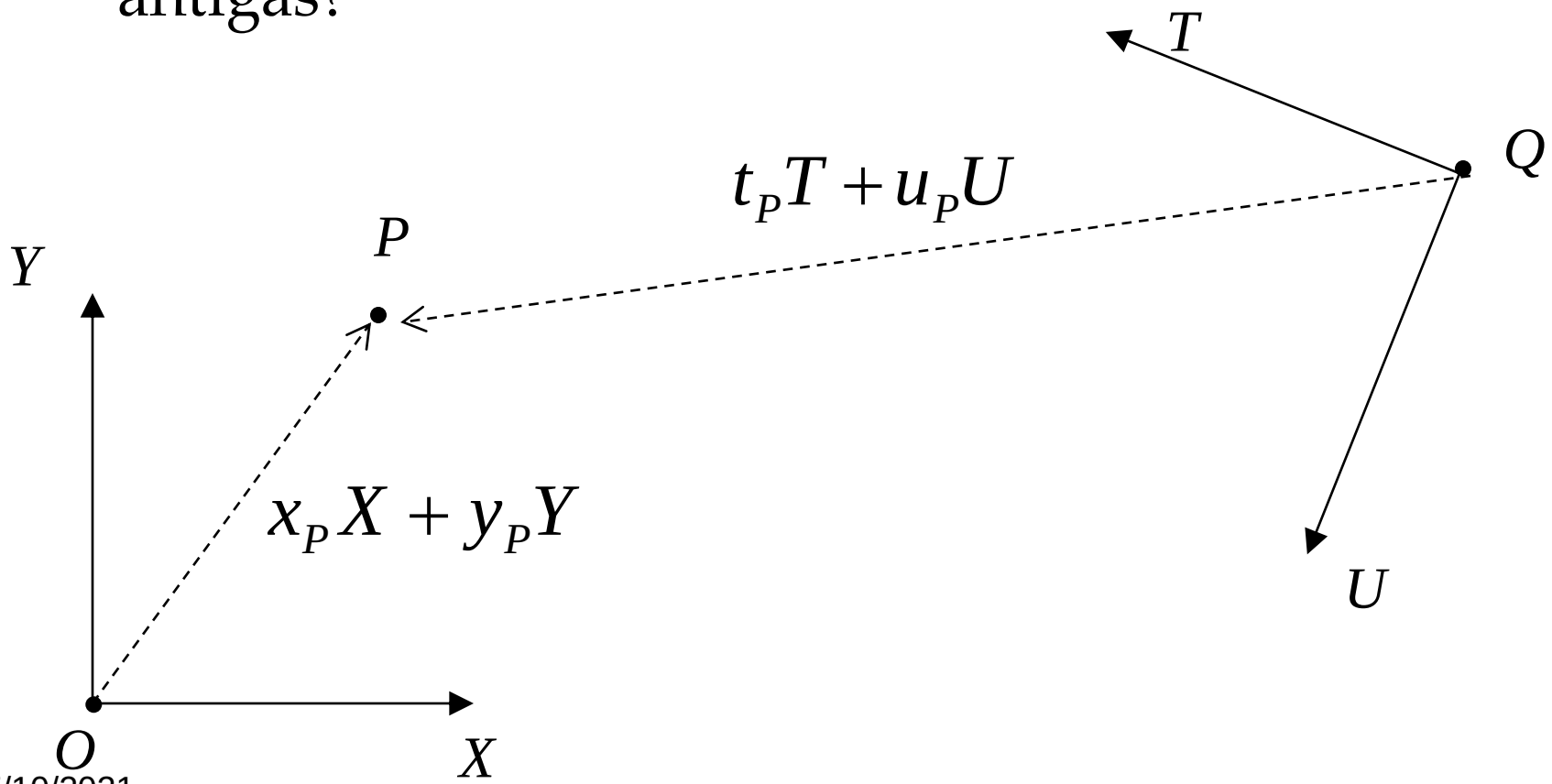
$$P = x_P.X + y_P.Y + O$$

  - Um vector V é dado por coordenadas x_V e y_V tais que

$$V = x_V.X + y_V.Y$$

Mudança de Sistema de Coordenadas

- Se estabelecemos um outro sistema (ex.: Q/T/U), como calcular as novas coordenadas em função das antigas?



Mudança de Sistema de Coordenadas

- Como calcular as coordenadas de um ponto P = (x_P, y_P) em O/X/Y dadas as coordenadas de P em Q/T/U, isto é, (t_P, u_P) ?

$$
\begin{aligned}
P &= t_P.T + u_P.U + Q \\
&= t_P.(x_T.X + y_T.Y) + u_P.(x_U.X + y_U.Y) + (x_Q.X + y_Q.Y + O) \\
&= (t_P.x_T + u_P.x_U + x_Q).X + (t_P.y_T + u_P.y_U + y_Q).Y + O
\end{aligned}
$$

Logo,

$$x_P = t_P.x_T + u_P.x_U + x_Q$$

$$y_P = t_P.y_T + u_P.y_U + y_Q$$

Mudança de Sistema de Coordenadas

- Matricialmente:

$$\begin{bmatrix} x_P \\ y_P \end{bmatrix} = \begin{bmatrix} x_T & x_U \\ y_T & y_U \end{bmatrix} \times \begin{bmatrix} t_P \\ u_P \end{bmatrix} + \begin{bmatrix} x_Q \\ y_Q \end{bmatrix}$$

- Usando coordenadas homogéneas:

$$\begin{bmatrix} x_P \\ y_P \\ 1 \end{bmatrix} = \begin{bmatrix} x_T & x_U & x_Q \\ y_T & y_U & y_Q \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} t_P \\ u_P \\ 1 \end{bmatrix}$$

- Para resolver o problema inverso:

$$\begin{bmatrix} t_P \\ u_P \\ 1 \end{bmatrix} = \begin{bmatrix} x_T & x_U & x_Q \\ y_T & y_U & y_Q \\ 0 & 0 & 1 \end{bmatrix}^{-1} \times \begin{bmatrix} x_P \\ y_P \\ 1 \end{bmatrix}$$

Transformações em 3D

- Vectores e pontos em 3D

$$\vec{V} = \begin{bmatrix} V_x \\ V_y \\ V_z \\ 0 \end{bmatrix} \qquad P = \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

- Transformação linear afim

$$T = \begin{bmatrix} a & b & c & j \\ d & e & f & k \\ g & h & i & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformações Rígidas

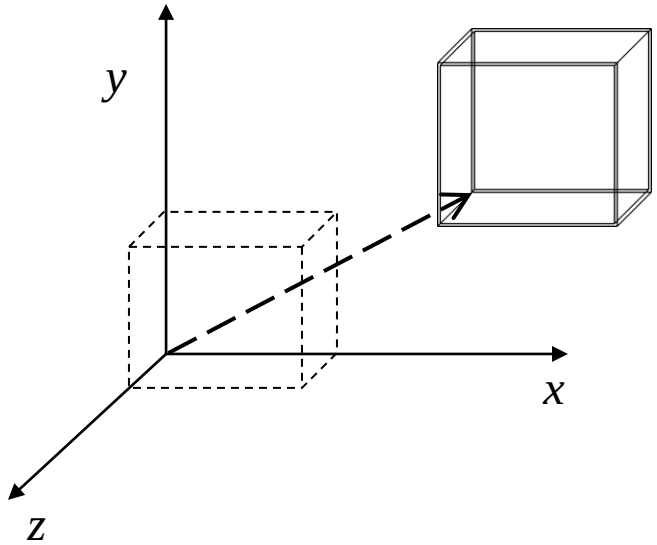
- Não modificam a forma (dimensões/ângulos) do objecto
- São compostas por uma rotação e uma translação

Submatriz de Rotação

Vector de Translação

$$T = \begin{bmatrix} a & b & c & j \\ d & e & f & k \\ g & h & i & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translação



$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} I_3 & t \\ 0 & 1 \end{bmatrix}$$

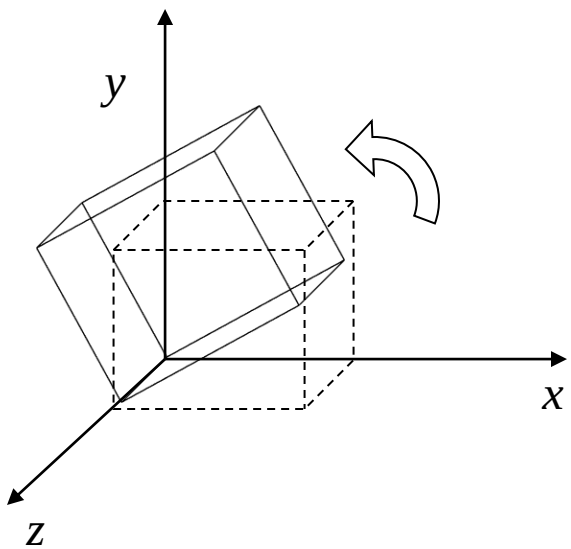
$$P' = T \times P = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} P_x + t_x \\ P_y + t_y \\ P_z + t_z \\ 1 \end{bmatrix} = P + t$$

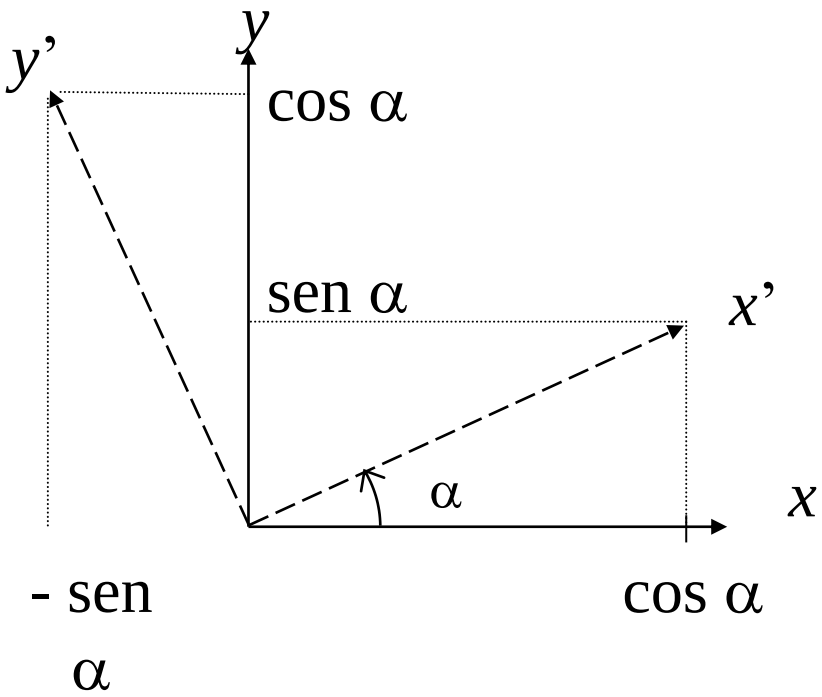
- As translações são comutativas:

  $P + t + v = P + v + t$

Rotação em torno do eixo *Z*

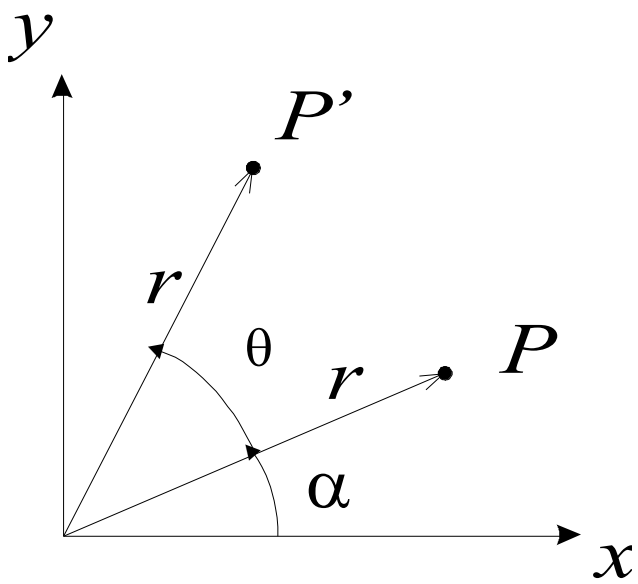
- Podemos ver que ao vector $(1,0,0)^T$ corresponde $(\cos\alpha, \text{sen}\,\alpha, 0)^T$ e que ao vector $(0,1,0)^T$ corresponde $(-\text{sen}\,\alpha, \cos\alpha, 0)^T$





Rotação em torno do eixo *Z*

- Outra maneira de ver:



Sabemos que

$$P_x = r\cos\alpha \qquad P'_x = r\cos(\alpha+\theta)$$

$$P_y = r\sin\alpha \qquad P'_y = r\sin(\alpha+\theta)$$

Então

$$P'_x = r\cos\alpha\cos\theta - r\sin\alpha\sin\theta$$

$$P'_y = r\cos\alpha\sin\theta + r\sin\alpha\cos\theta$$

Resultando que

$$P'_x = P_x\cos\theta - P_y\sin\theta$$

$$P'_y = P_x\sin\theta + P_y\cos\theta$$

Rotação em torno dos eixos coordenados

- Rotação em torno de Z é dada pela matriz

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Do mesmo modo, em torno dos eixos Y e X

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotações em geral

- Qualquer rotação pode ser definida por um eixo de rotação dado pelo vector unitário **u** = $(x, y, z)^{\mathrm{T}}$ e um ângulo de rotação α
- Seja S a matriz

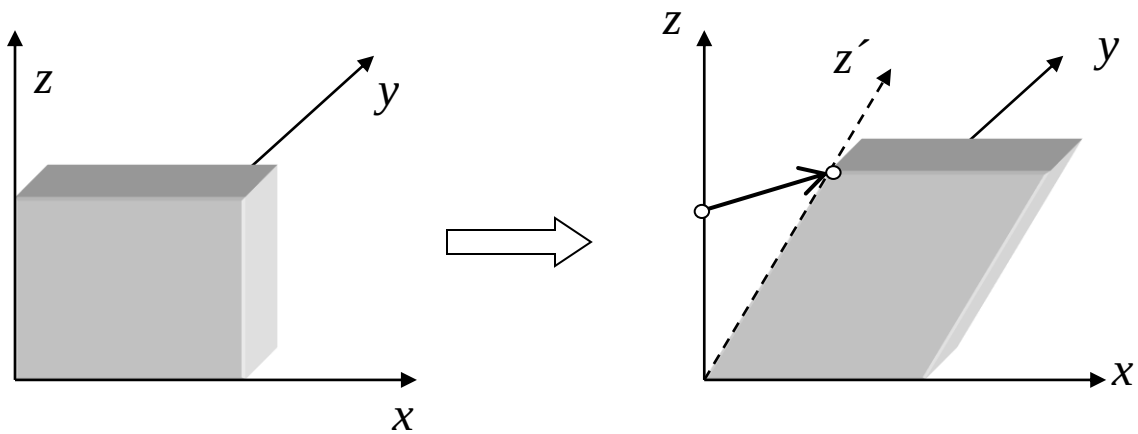
$$\mathrm{S} = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

- Então a submatriz de Rotação M é dada por

$$\mathrm{M} = \mathrm{u}\mathrm{u}^{\mathrm{T}} + (\cos\alpha)(\mathrm{I} - \mathrm{u}\mathrm{u}^{\mathrm{T}}) + (\sin\alpha)\mathrm{S}$$

Inclinação (“*shear*”)

- É uma transformação de deformação onde um eixo é “entortado” em relação aos restantes

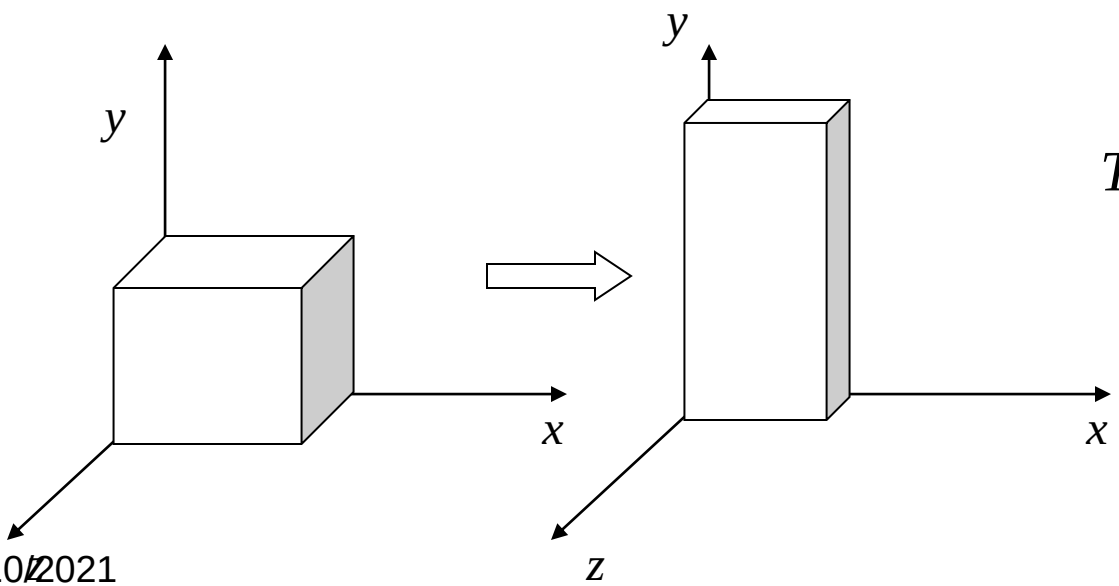


- Se o vector unitário do eixo z é transformado em $[Sh_x\ Sh_y\ 1\ 0]^T$, então a matriz de transformação é dada por

$$T_{inclinação} = \begin{bmatrix} 1 & 0 & Sh_x & 0 \\ 0 & 1 & Sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Escalamento

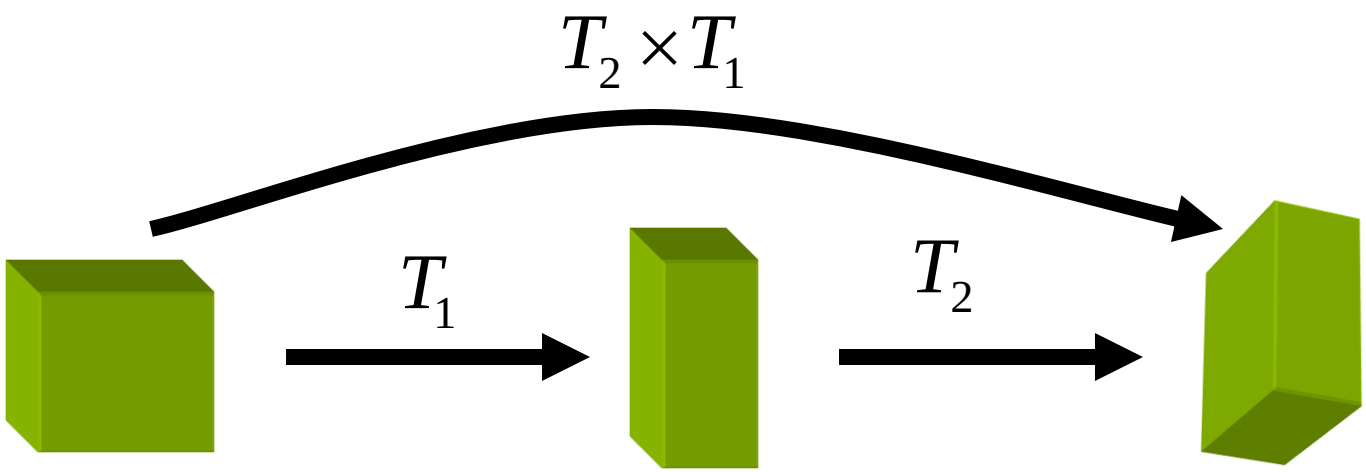
- Especificado por três factores (S_x, S_y , S_z) que multiplicam os vectores unitários x, y, z
- Escalamento não é uma transformação rígida
- Escalamento uniforme ($S_x = S_y = S_z$) é uma operação ortogonal ou homotética, isto é, preserva os ângulos
- Para obter reflexão em torno do plano y=0, usar os factores de escala (1, -1, 1)



$$T_{escala} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Composição de transformações em 3D

- Na nossa notação, usamos a pré-multiplicação:
  - $P' = T \times P$
- Para compor 2 transformações temos:
  - Se $P' = T_1 \times P$ e $P'' = T_2 \times P'$, então, $P'' = T_2 \times T_1 \times P$



Geometria Afim

- Composta dos elementos básicos
  - escalares
  - pontos - denotam posição
  - vectores - denotam deslocamento (direcção e magnitude)
- Operações
  - escalar · vector = vector
  - vector + vector ou vector – vector = vector
  - ponto – ponto = vector
  - ponto + vector ou ponto – vector = ponto

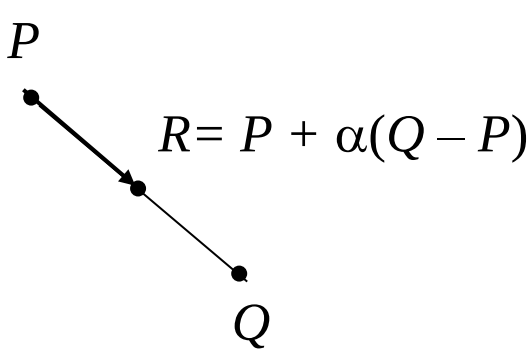
Combinações Afim

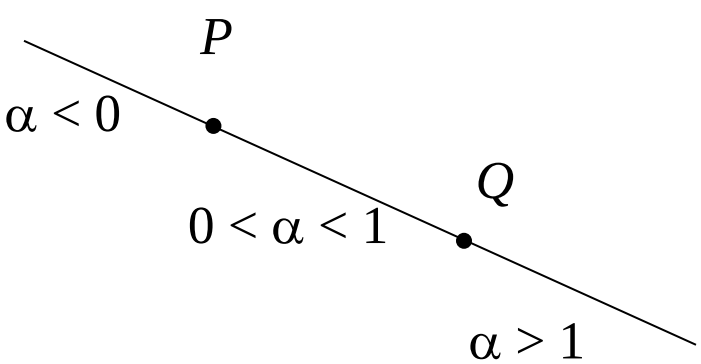
- Maneira especial de combinar pontos

$$\alpha_1 P_1 + \alpha_2 P_2 + \ldots + \alpha_n P_n$$

$$\text{onde } \sum_{i=1}^{n} \alpha_i = 1$$

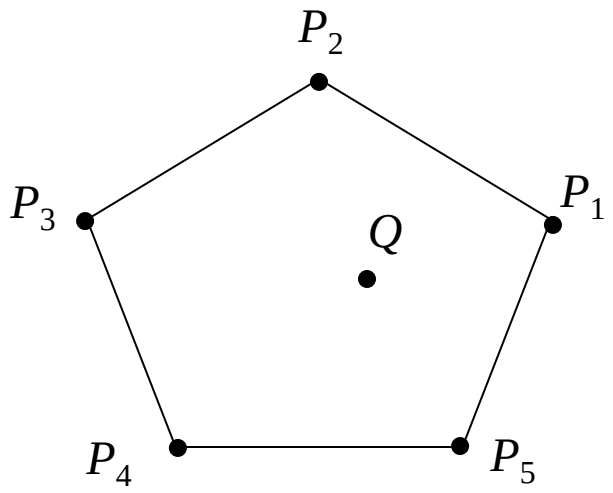
- Para 2 pontos P e Q poderíamos ter uma combinação afim $R = (1 - \alpha)P + \alpha Q = P + \alpha(Q - P)$





Combinações Convexas

- Combinações afim onde se garante que todos os coeficientes α_i são positivos (ou nulos)
- Usa-se esse nome porque qualquer ponto que é uma combinação convexa de *n* outros pontos pertence à <u>envolvente convexa</u> desses pontos



Geometria Euclidiana

- Extensão da geometria afim pela adição de um operador chamado *produto interno*
- Produto interno é um operador que transforma um par de vectores num escalar. Tem as seguintes propriedades:
  - Positividade : $(u,u) \geq 0$ e $(u,u) = 0$ sse $u=0$
  - Simetria: $(u,v) = (v,u)$
  - Bilinearidade: $(u,v+w)= (u,v)+ (u,w)$ e

$$(u,\alpha v)= \alpha(u,v)$$

Geometria Euclidiana

- Normalmente usamos o produto escalar como operador de produto interno:

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{d} u_i v_i$$

- Comprimento de um vector é definido como:

$$\left|\vec{v}\right| = \sqrt{\vec{v} \cdot \vec{v}}$$

- Vector unitário (normalizado):

$$\hat{v} = \frac{\vec{v}}{\left|\vec{v}\right|}$$

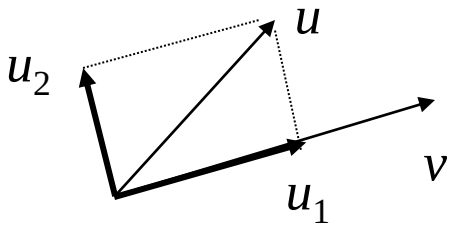
Geometria Euclidiana

- Distância entre dois pontos P e Q = $|Q - P|$
- O ângulo entre dois vectores pode ser determinado por

$$\hat{a}ngulo(\vec{u},\vec{v}) = \cos^{-1}\left(\frac{\vec{u}\cdot\vec{v}}{|\vec{u}||\vec{v}|}\right) = \cos^{-1}(\hat{u}\cdot\hat{v})$$

- Projecção ortogonal: dados dois vectores u e v, deseja-se decompor u na soma de dois vectores u_1 e u_2 tais que u_1 é paralelo a v e u_2 é perpendicular a v

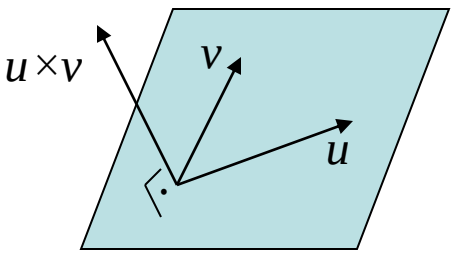
$$\vec{u}_1 = \frac{\vec{u}\cdot\vec{v}}{\vec{v}\cdot\vec{v}}\vec{v} \qquad \vec{u}_2 = \vec{u} - \vec{u}_1$$



Produto Vectorial (3D)

- Permite achar um vector perpendicular a outros dois vectores
- Útil na construção de sistemas de coordenadas

$$\vec{u} \times \vec{v} = \begin{bmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{bmatrix} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix}$$

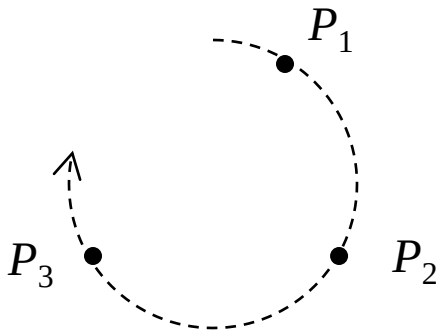


- Propriedades (assume-se *u*, *v* linearmente independentes):
  - Antisimetria: $u \times v = -v \times u$
  - Bilinearidade: $u \times (\alpha v) = \alpha\,(u \times v)$ e $u \times (v + w) = (u \times v) + (u \times w)$
  - $u \times v$ é perpendicular tanto a *u* quanto a *v*
  - O comprimento de $u \times v$ é igual a área do paralelogramo definido por *u* e *v*, isto é, $|u \times v| = |u|\,|v| \sin\theta$

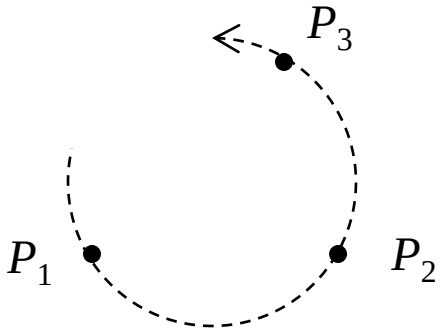
Orientação

- Orientação de 2 pontos em 1D
  - $P_1 < P_2$, $P_1 = P_2$ ou $P_1 > P_2$
- Orientação de 3 pontos em 2D
  - O percurso P_1 , P_2 , P_3 é feito no sentido dos ponteiros do relógio, no sentido contrário ou são colineares

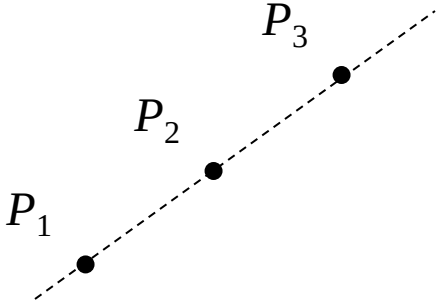
Or (P_1, P_2, P_3) = -1



Or (P_1, P_2, P_3) = +1



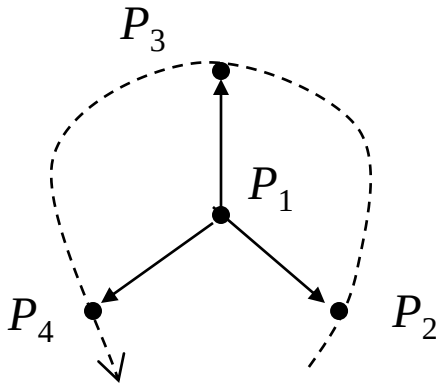
Or (P_1, P_2, P_3) = 0



Orientação

- Orientação de 4 pontos em 3D
  - O percurso P_1 , P_2 , P_3 , P_4 define um parafuso segundo a regra da mão direita, mão esquerda ou são coplanares

Or (P_1, P_2, P_3, P_4) = +1



- *O conceito pode ser estendido a qualquer número de dimensões ...*

Determinação da Orientação

- A orientação de n+1 pontos num espaço n-dimensional é dada pelo sinal do determinante da matriz cujas colunas são as coordenadas homogéneas dos pontos *com o 1 em primeiro lugar*

$$\mathrm{Or}_2(P_1, P_2, P_3) = \mathrm{sign}\left(\begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix}\right)$$

$$\mathrm{Or}_3(P_1, P_2, P_3, P_4) = \mathrm{sign}\left(\begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{vmatrix}\right)$$

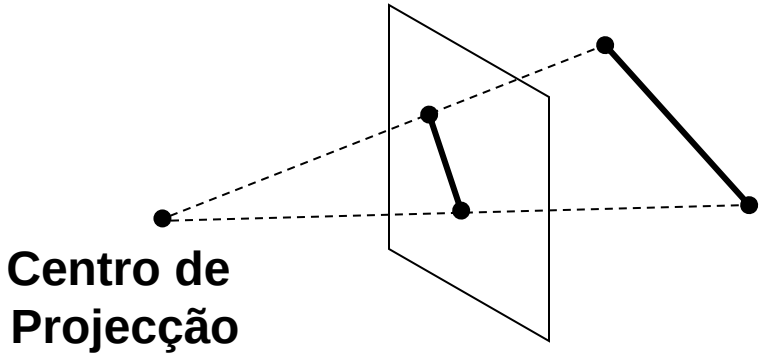
Introdução à Computação Gráfica
Projecções

Adaptação: João Paulo Pereira
António Costa

Autoria: Claudio Esperança
Paulo Roma Cavalcanti

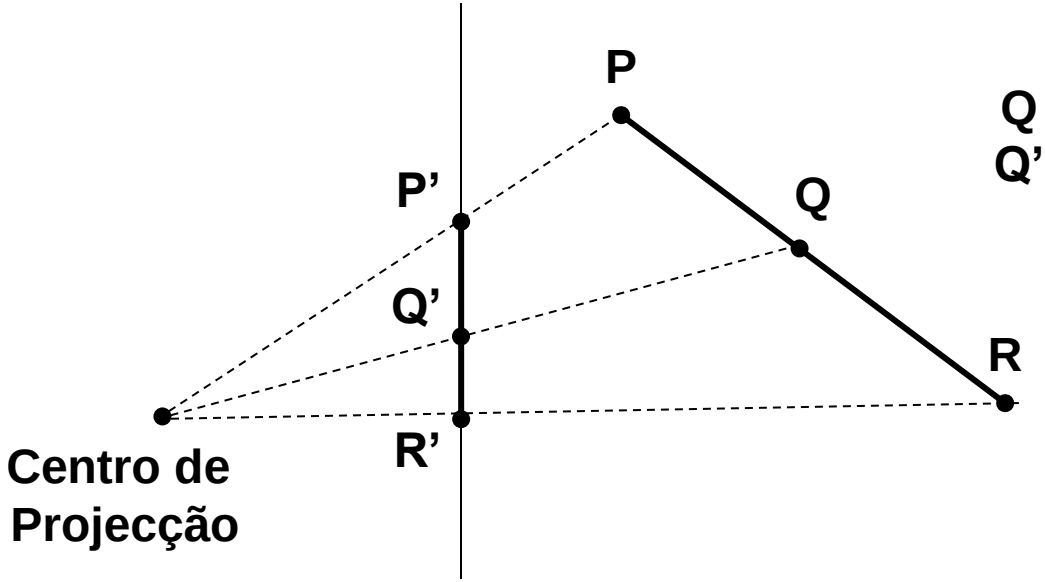
Perspectiva

- É o estudo de transformações projectivas
- O problema consiste em projectar pontos do espaço *d*-dimensional no plano *d*-1 dimensional usando um ponto especial designado por centro de projecção



Transformações Projectivas

- As transformações projectivas transformam rectas em rectas mas não preservam as combinações afim

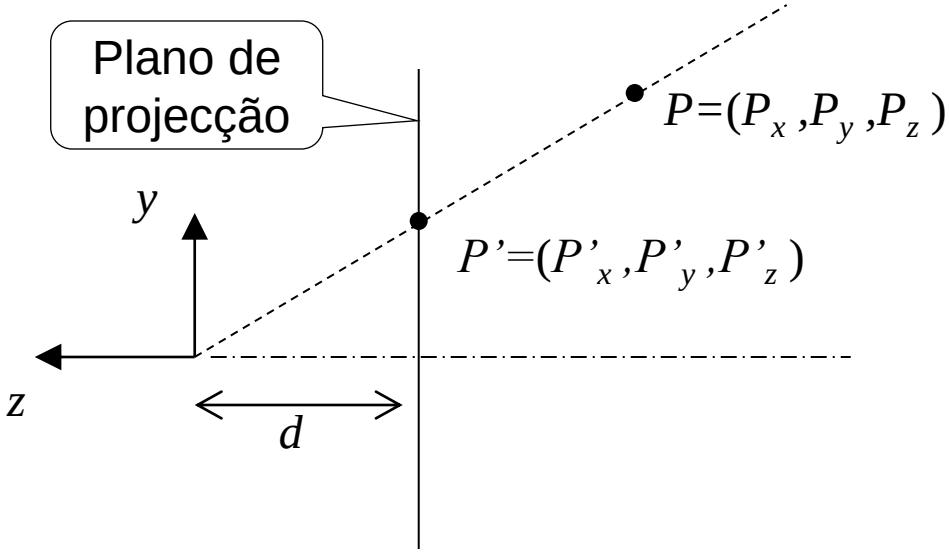


Q é o ponto médio de PR, mas Q' não é o ponto médio de P'R'

Coordenadas homogéneas em espaço projectivo

- Representamos apenas pontos (não vectores)
- Em 2D, um ponto (x,y) será representado em c.h. pela matriz-coluna $[x \cdot w \quad y \cdot w \quad w]^T$, para $w \neq 0$
  - Assim, o ponto (4,3) pode ser representado por $[8\ 6\ 2]^T$, $[12\ 9\ 3]^T$, $[-4\ -3\ -1]^T$, etc.
- Dado um ponto com coordenadas homogéneas $[x \ \ y \ \ w]^T$, a sua representação canónica é dada por $[x/w \ \ y/w \ \ 1]^T$
  Designamos essa operação por *divisão perspectiva*
- Considere os pontos da recta $x=y$: (1,1), (2,2), etc
  - Podemos representá-los em c.h. por $[1\ 1\ 1]^T$, $[1 \ \ 1 \ \ ½]^T$, etc.
  - Claramente, o ponto ideal dessa recta é dado por $[1\ 1\ 0]^T$

Transformações projectivas



- Se o plano de projecção é perpendicular ao eixo *z*, está a uma distância *d* do C.P. (que está na origem) e intercepta o semi-eixo z negativo, então a projecção de um ponto P é dada por

$$P' = \begin{bmatrix} \dfrac{P_x}{-P_z/d} & \dfrac{P_y}{-P_z/d} & -d & 1 \end{bmatrix}^{\mathrm{T}}$$

- Por semelhança de triângulos, vemos que $P_y/-P_z = P'_y/d$

Transformação perspectiva em coordenadas homogéneas

- Não existe matriz 4x4 capaz de realizar tal transformação em espaços euclidianos, mas se assumirmos que o ponto está no espaço projectivo, então

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \times \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z \\ -P_z/d \end{bmatrix} = \begin{bmatrix} \dfrac{P_x}{-P_z/d} \\ \dfrac{P_y}{-P_z/d} \\ -d \\ 1 \end{bmatrix}$$

Perspectiva - Sumário

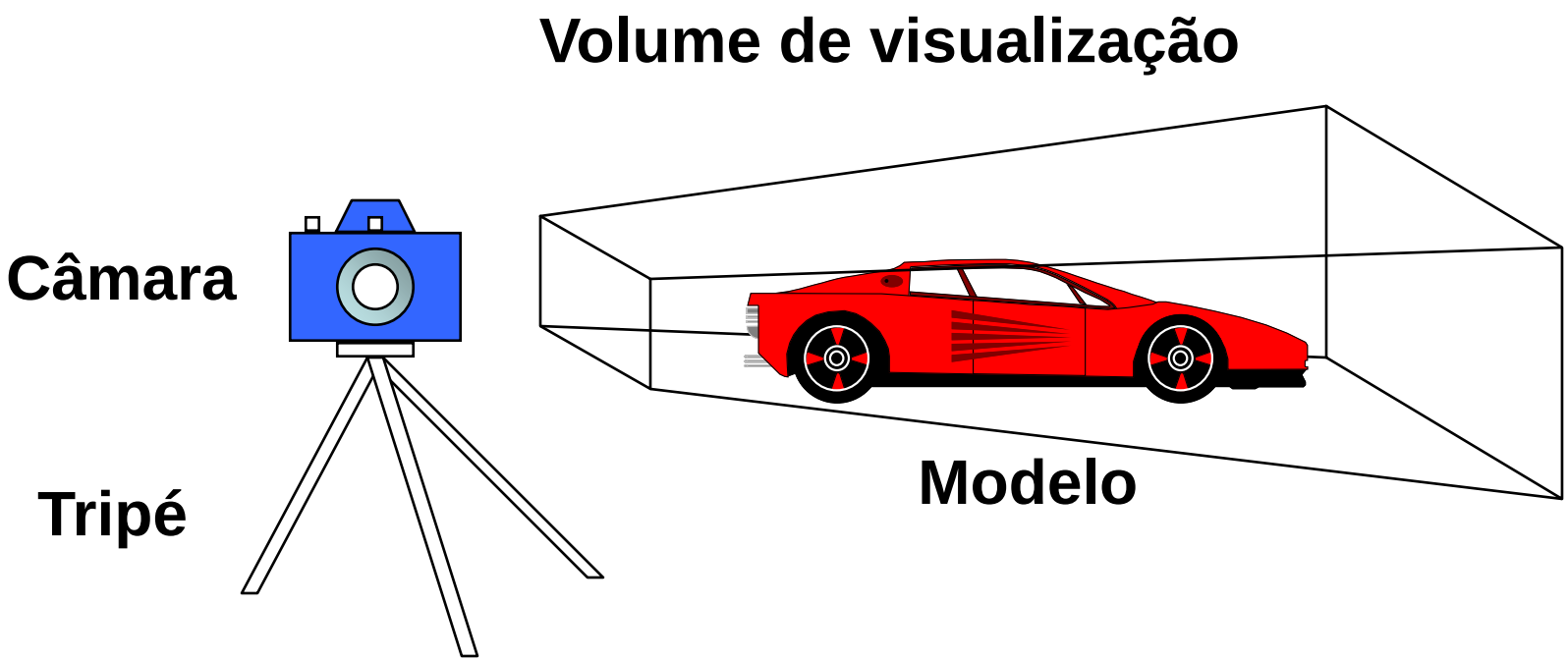
- Para fazer projecção perspectiva de um ponto *P*, seguem-se os seguintes passos
  - *P* é levado do espaço euclidiano para o projectivo
    - Trivial – mesmas coordenadas homogéneas
  - *P* é multiplicado pela matriz de transformação perspectiva resultando em *P′*
  - *P′* é levado novamente ao espaço euclidiano
    - Operação de divisão perspectiva

Projecção genérica

- E se não quisermos que o Centro de Projecção esteja na origem ou se a cena não estiver correctamente posicionada no semi-eixo z negativo?
  - Aplicam-se transformações afins para posicionar todos os elementos correctamente
  - As maneiras pelas quais essas transformações são feitas caracterizam um dado modelo de projecção

Modelo de câmara sintética

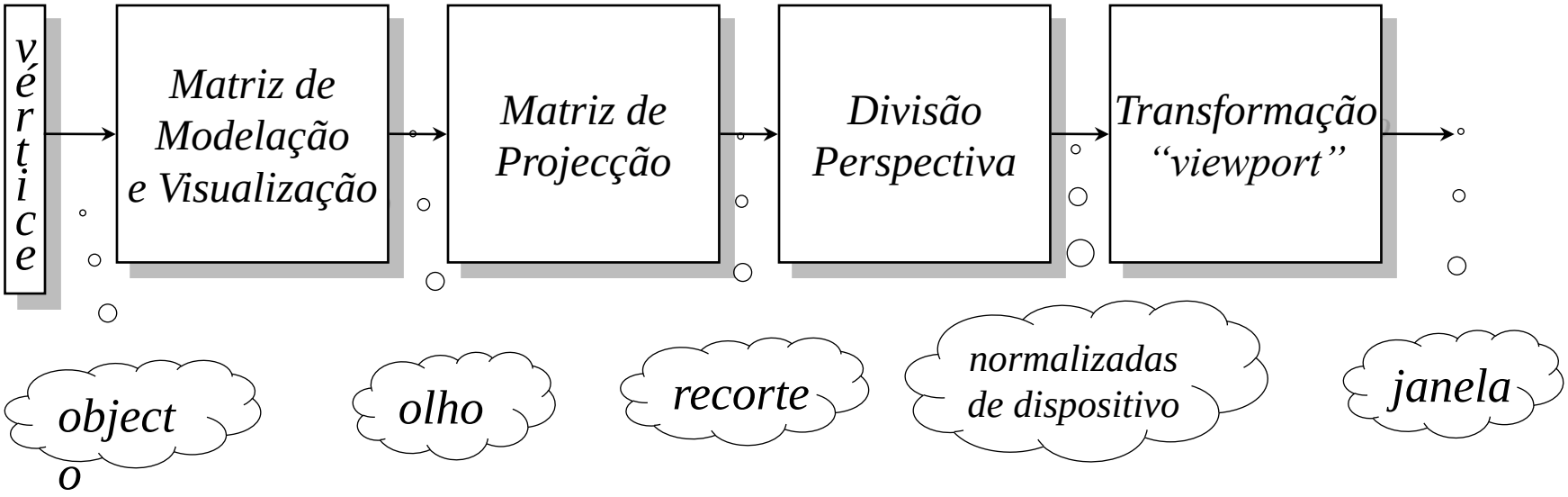
- API utiliza uma analogia comparando visualização 3D com tirar fotografias com uma câmara



Tipos de transformações

- Modelação
  - Mover / deformar os objectos
- Visualização
  - Mover e orientar a câmara
- Projecção
  - Ajuste da lente / objectiva da câmara
- "*Viewport*"
  - Aumentar ou reduzir a fotografia

Pipeline de Transformações



Coordenadas

Estado inicial do *pipeline*

- Inicialmente,
  - As matrizes "*modelview*" e "*projection*" são matrizes-identidade
    - Vértices não são transformados e a projecção é paralela ao plano *xy*
    - O mundo visível é restrito ao cubo $-1 \leq x,y,z \leq 1$
  - A transformação "*viewport*" transforma o quadrado $-1 \leq x,y \leq 1$ (em coordenadas normalizadas de dispositivo) na superfície total da janela

Especificação do *viewport*

- Para especificar a área da janela na qual será transformado o quadrado do plano de projecção, utiliza-se

```
renderer.setViewport (x: Integer, y: Integer, width:
  Integer, height: Integer)
renderer.setViewport (vector: Vector4)
```

  - (parâmetros em *pixels,* sendo que (0, 0) se refere ao canto inferior esquerdo da janela)
- Normalmente não é necessário modificar, mas é útil para
  - Manter a razão de aspecto da imagem
  - Fazer *zooming* e *panning* sobre a imagem

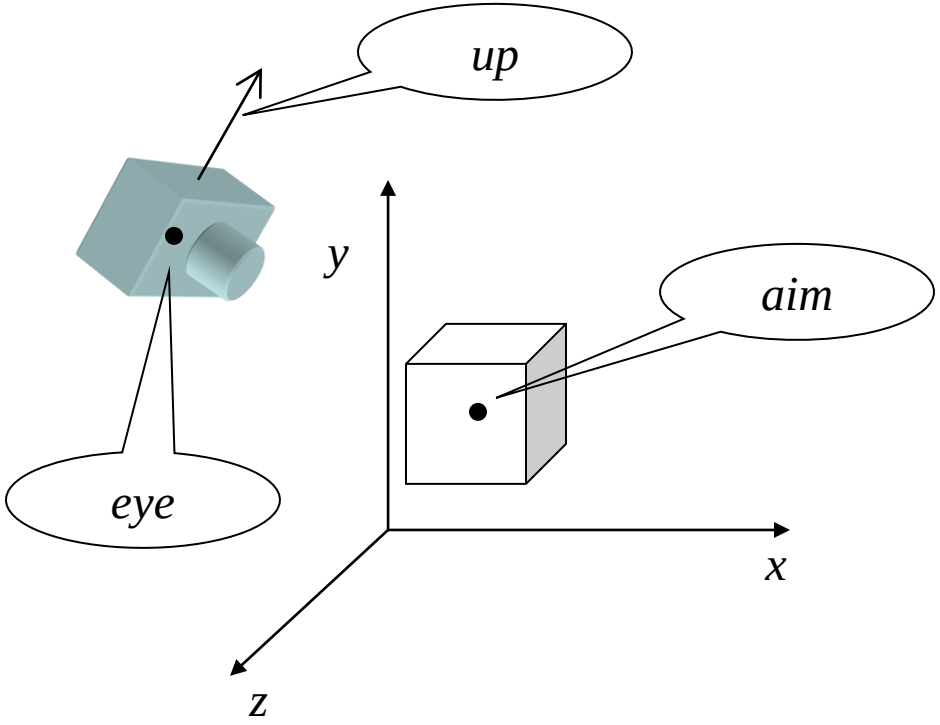
Transformação de objectos

- Usam-se funções para multiplicar o topo da pilha de matrizes por transformações especificadas por parâmetros
  - **`object.position: Vector3`**
  - **`object.translate*(distance: Float)`**
  - **`object.rotation: Euler`**
  - **`object.rotate* (rad: Float)`**
  - **`object.scale: Vector3`**

- Cuidado: a ordem é importante

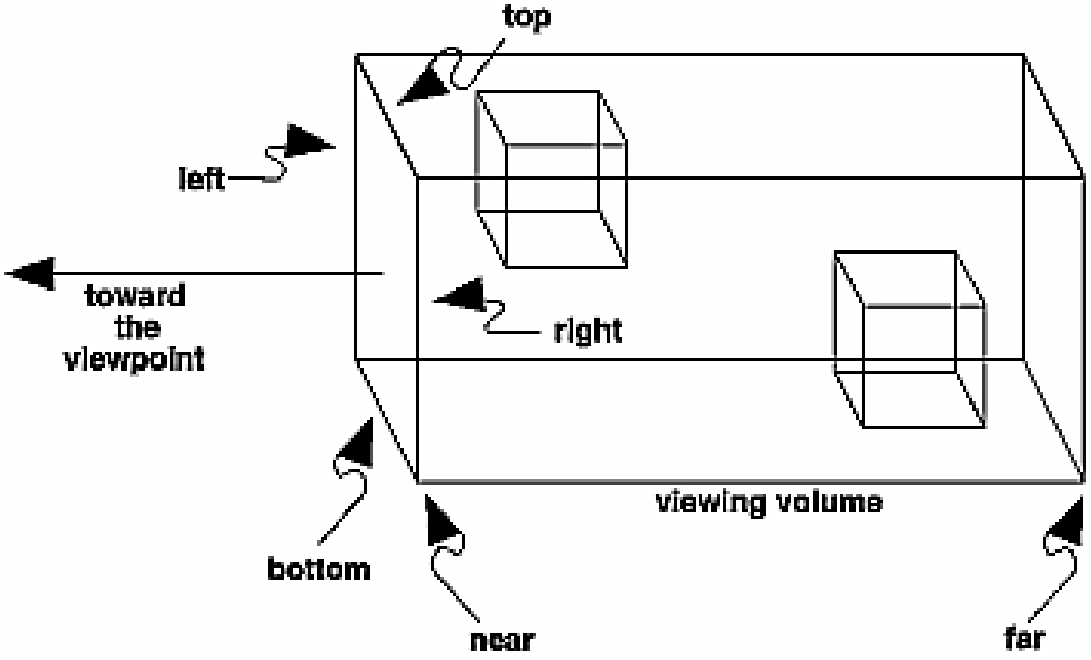
Transformações de visualização

- Duas interpretações:
  - Levam a câmara até a cena que se quer visualizar
  - Levam os objectos da cena até uma câmara estacionária
- `camera.position: Vector3`
- `camera.lookAt(vector: Vector3)`
- `camera.lookAt (x: Float, y: Float, z: Float);`
- `camera.up: Vector3`
  - eye = ponto onde a câmara será posicionada
  - aim = ponto para onde a câmara será apontada
  - up = vector que dá a direção "para cima" da câmara
  - *Cuidado com casos degenerados*



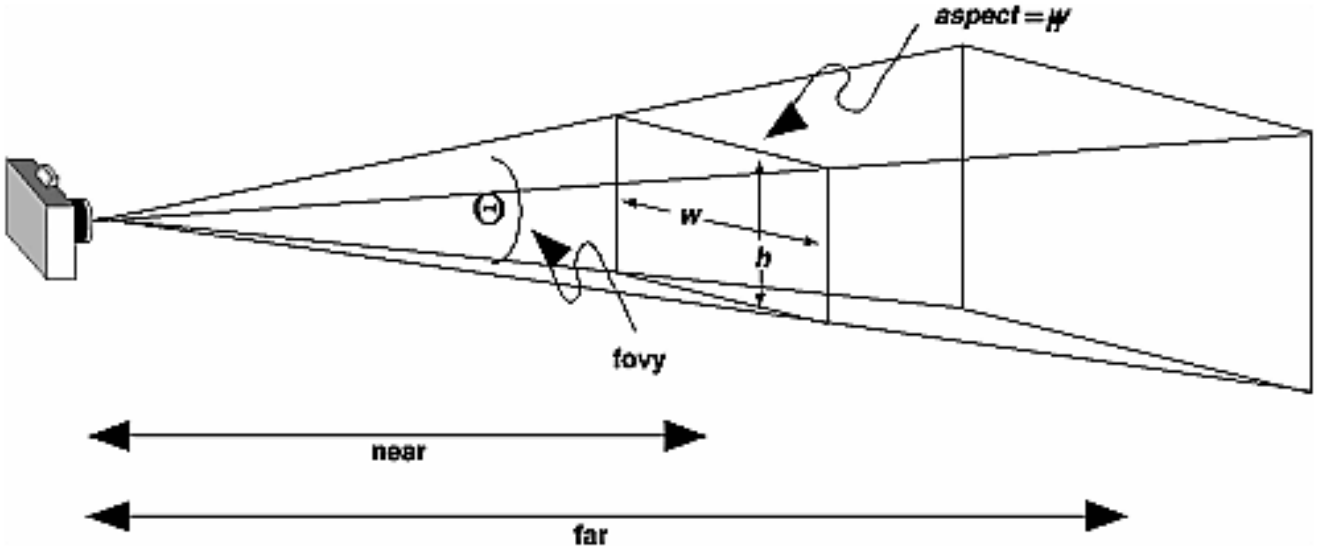
Projecção Paralela

- Volume de visualização especificado com
- `OrthographicCamera(left: Number, right: Number, top: Number, bottom: Number, near: Number, far: Number)`
  - Obs.: near >= 0; far > near



Projecção Perspectiva

- Volume de visualização especificado com `PerspectiveCamera(fov: Number, aspect: Number, near: Number, far: Number)`
  - Obs.: near > 0; far > near



Introdução à Computação Gráfica
Modelação

Adaptação: João Paulo Pereira
António Costa

Autoria: Claudio Esperança
Paulo Roma Cavalcanti

História

- Modelação por malha de arame (*wireframes*)
  - Representa os objectos por arestas e pontos sobre a sua superfície
  - Gera modelos ambíguos
- Modelação por superfícies (década de 60)
  - Fornece a descrição matemática das superfícies que delimitam o objecto
  - Poucos testes de integridade do modelo

História

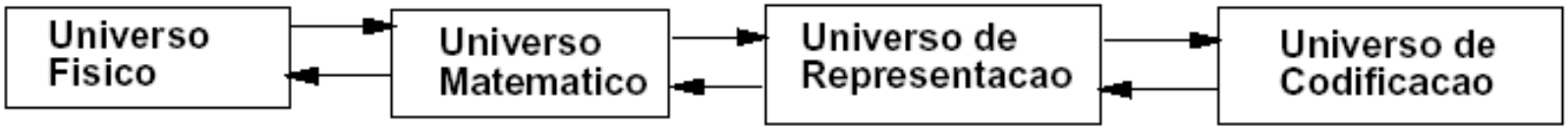
- Modelação de Sólidos (década de 70)
  - Implícita ou explicitamente contém informações sobre o fecho e conectividade dos objectos
  - Garante a realização física
  - Sistemas CAD-CAM utilizados pela indústria

Estado da Arte

- Modelação de dimensão mista ou *non-manifold* (década de 80)
  - Permite representar objectos com estruturas internas ou com elementos pendentes de dimensão inferior
  - Sólido delimitado por superfícies não necessariamente planas localmente
  - Ex.: ACIS (Spatial Technology) – *AutoCad*, etc.

Paradigmas de Abstracção

- A necessidade de paradigmas (Ari Requicha).



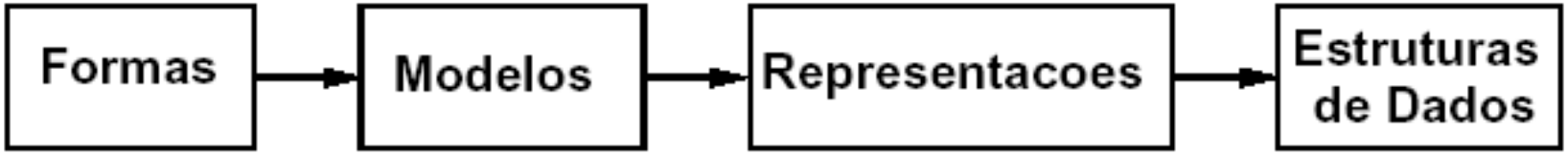
- Paradigma dos universos
  - Físico *F*
  - Matemático *M*
  - Representação *R*
  - Implementação *I*

Problemas da Área

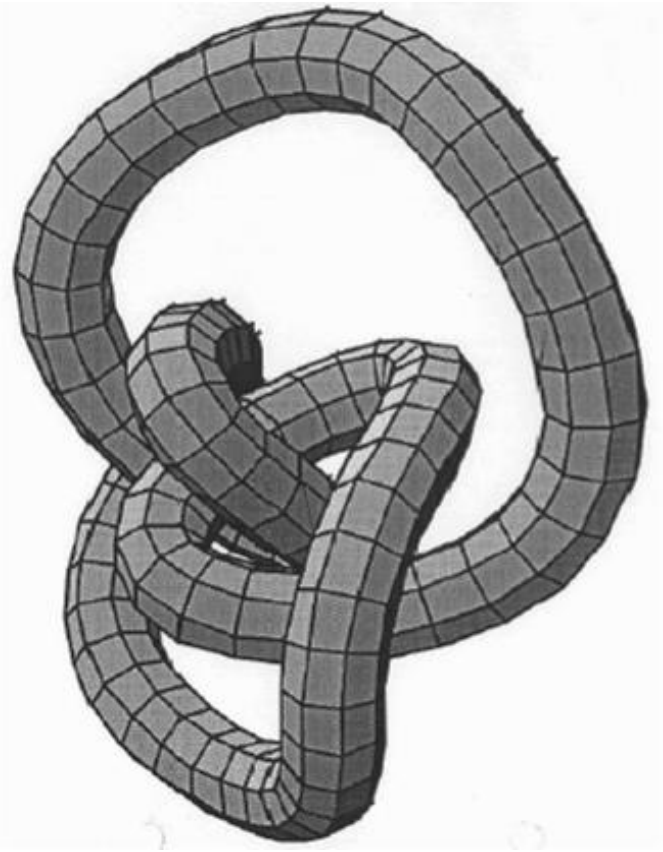
- Estudar fenómenos em F
- Definir os modelos
- Estudar as relações entre R e M
- Definir representações de modelos em M
- Estudar conversões entre representações
- Definir métodos de implementação
- Comparar estratégias em I

Esquemas de Representação

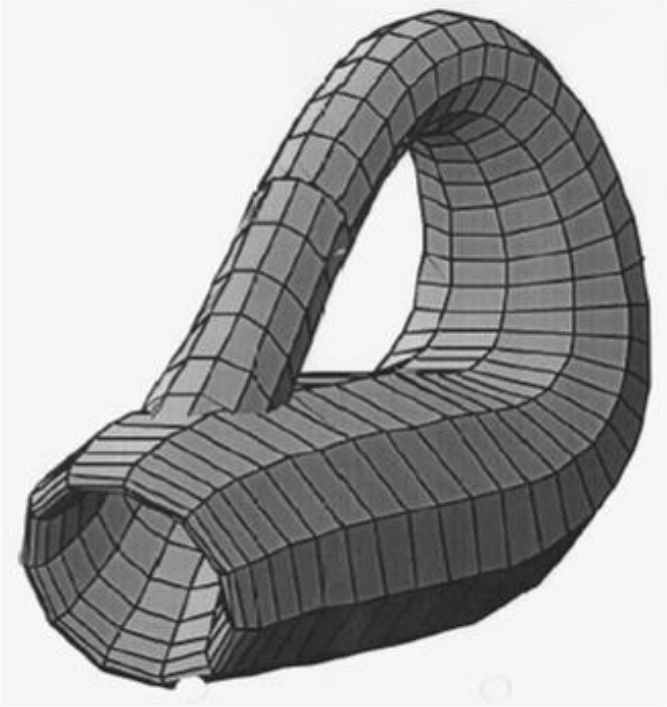
- Objectos do universo físico: “sólidos”
  - O que é um sólido?
- Objectos do universo matemático vêm da:
  - Geometria diferencial
  - Topologia diferencial



Geometria pode ser complicada



Nó



Garrafa de Klein
(não orientável)

Descrição de Sólidos

- Assumir que um sólido é um conjunto tridimensional de pontos
- Conjuntos de pontos podem ser descritos
  - Pelas suas fronteiras
  - Por campos escalares
    - Definidos por equações
    - Amostrados
- Originam três tipos de representação:
  - Por fronteira (B-rep – *Boundary Representation*)
  - Operações de conjuntos (CSG – *Constructive Solid Geometry*)
  - Por enumeração do espaço em células (BSP-trees, Octrees, etc.)

Representação por Fronteira

- Sólido definido indirectamente através da superfície que o delimita
  - compacta (fechada e limitada)
  - sem bordo
- Superfícies são descritas parametricamente por **parametrização**:

$$\varphi : U \subset \Re^2 \rightarrow \Re^3$$

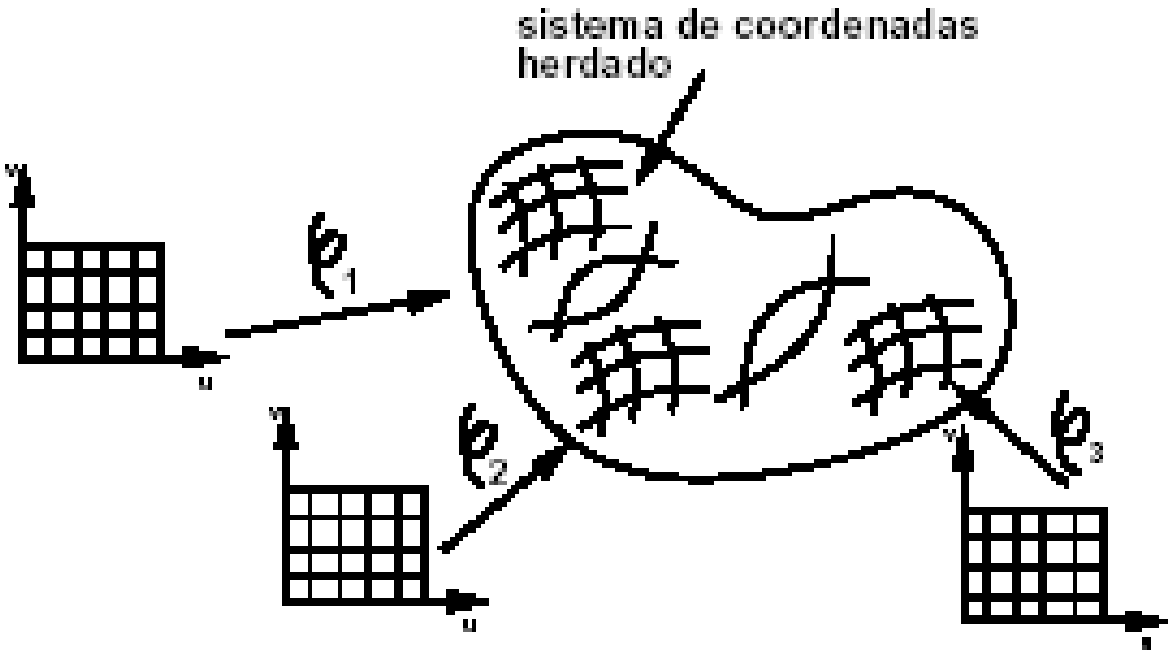
Parametrização

- Estabelece um sistema de coordenadas sobre a superfície <u>herdado</u> de um sistema de coordenadas no plano

$$\varphi(u,v) = \left(\varphi_x(u,v), \varphi_y(u,v), \varphi_z(u,v)\right)^T = (x, y, z)^T$$

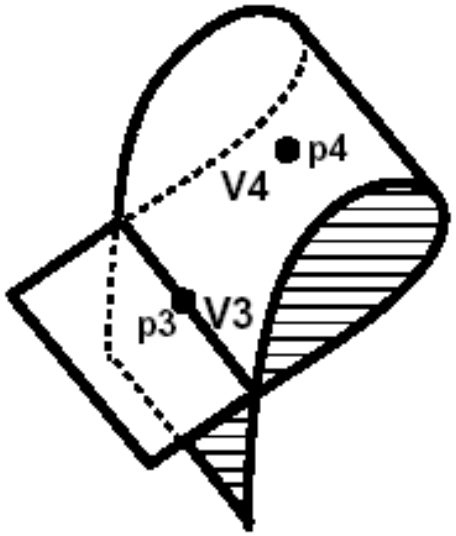
- Em geral, não é possível cobrir (descrever) toda a superfície com uma única parametrização
  - Usam-se várias parametrizações que formam um **Atlas**

Parametrização de uma Superfície



Parametrizações Válidas

- Sólido deve estar bem definido
  - Superfície sem auto-intersecção
  - Vector normal não se anula sobre a superfície
  - Normal é usada para determinar o interior e o exterior do sólido



$$N = \left(\frac{\partial \varphi}{\partial u} \times \frac{\partial \varphi}{\partial v} \right)$$

Exemplo

- Parametrização da esfera de raio unitário, centrada na origem

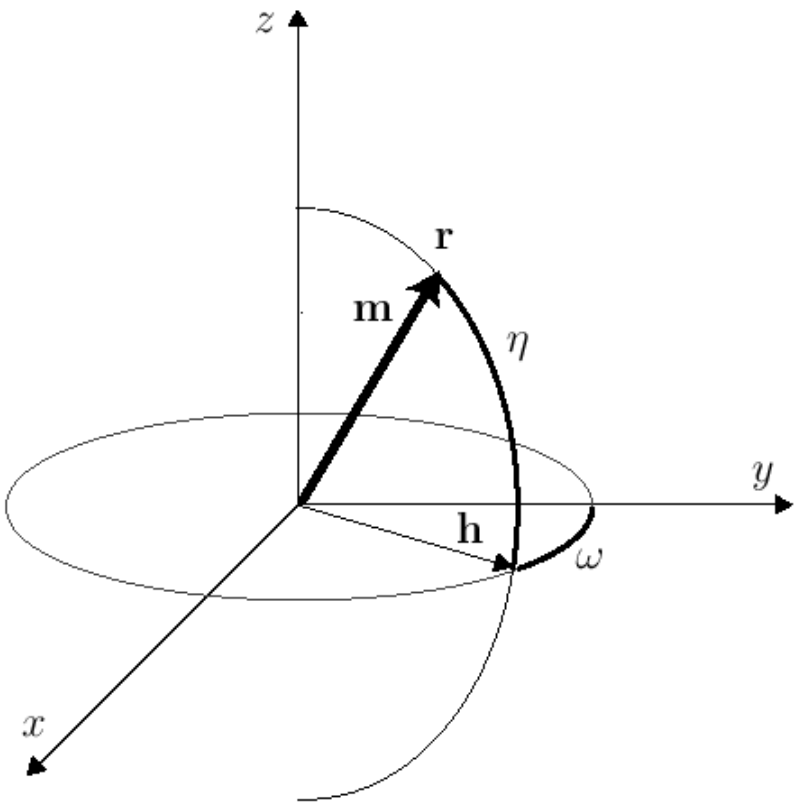
$$f(\omega,\eta) = \begin{bmatrix} \sin(\omega)\cos(\eta) \\ \cos(\omega)\cos(\eta) \\ \sin(\eta) \end{bmatrix}$$

$$N = \frac{\partial f}{\partial \omega} \times \frac{\partial f}{\partial \eta} = \begin{bmatrix} \cos(\omega)\cos(\eta) \\ -\sin(\omega)\cos(\eta) \\ 0 \end{bmatrix} \times \begin{bmatrix} -\sin(\omega)\sin(\eta) \\ -\cos(\omega)\sin(\eta) \\ \cos(\eta) \end{bmatrix}$$

- Se $\eta = -\pi/2$ ou $\eta = \pi/2$ a normal não está definida nos pólos por esta parametrização

Domínio do Exemplo Anterior

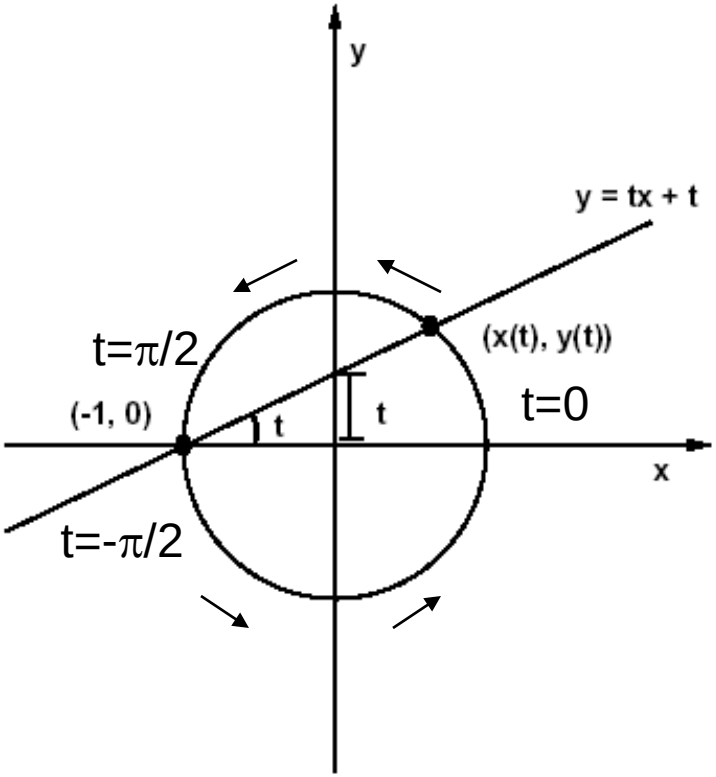
- Todas as parametrizações da esfera deixam pelo menos um ponto de fora
- É impossível parametrizar continuamente a esfera no plano sem retirar pelo menos um ponto



$$U = \left\{ (\omega, \eta) \in \Re^2 : 0 \le \omega < 2\pi; -\frac{\pi}{2} < \eta < \frac{\pi}{2} \right\}$$

Parametrização do Círculo

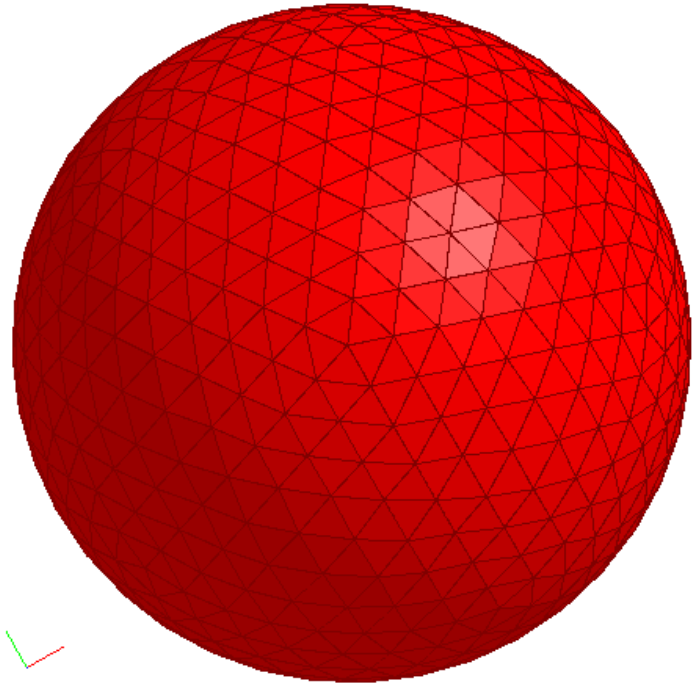
- Forma implícita
  - $y = tx + t$
  - $x^2 + y^2 = 1$
- Resolvendo esse sistema chega-se a uma parametrização alternativa do círculo



$$x(t) = \frac{1-t^2}{1+t^2}; y(t) = \frac{2t}{1+t^2}; t \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$$

Representação Linear por Partes

- Superfície parametrizada com geometria complexa pode ser <u>aproximada</u> por uma superfície linear por partes
- Pode-se particionar o domínio da parametrização por um conjunto de polígonos
  - Cada vértice no domínio poligonal é levado para a superfície pela parametrização
  - Em seguida é ligado aos vértices adjacentes mantendo as conectividades do domínio



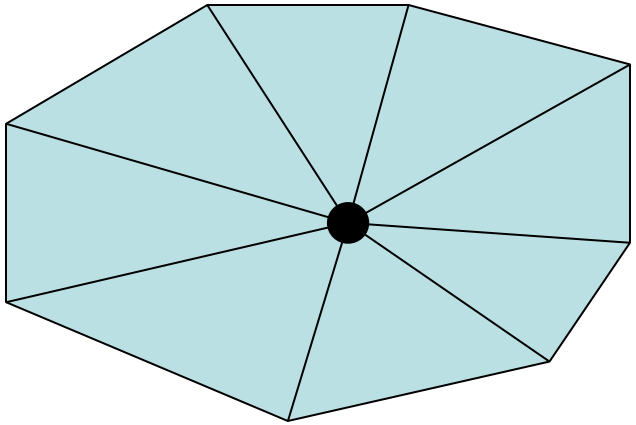
Propriedades

- Gera uma malha poligonal, definida por um conjunto de vértices, arestas e faces
  - Cada aresta é compartilhada, no máximo, por duas faces
  - A intersecção de duas faces é uma aresta, um vértice ou vazia
- Adjacência de vértices, arestas e faces é designada **topologia** da superfície

Decomposição Poligonal

Operações sobre Malhas Poligonais

- Achar todas as arestas que incidem num vértice
- Achar as faces que incidem numa aresta ou vértice
- Achar as arestas na fronteira de uma face
- Desenhar a malha



Codificação

- Explícita
- Apontadores para lista de vértices
- Apontadores para lista de arestas
- *Winged-Edge* (*Half-Edge, Face-Edge*)
- *Quad-Edge* (Guibas-Stolfi)
- *Radial-Edge*

Codificação Explícita

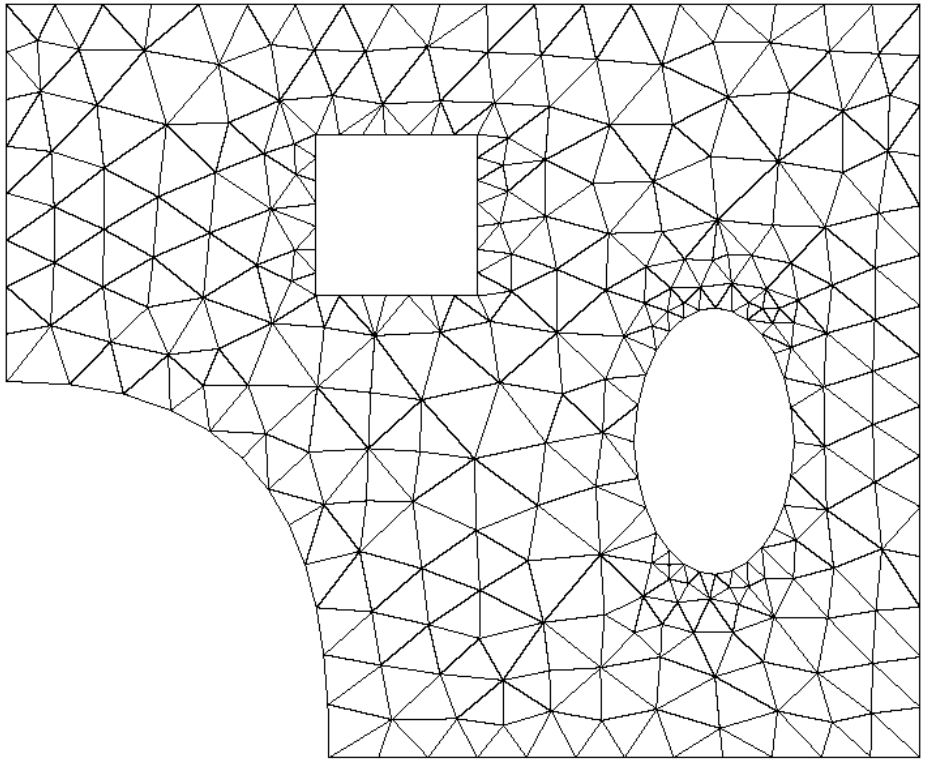
- A mais <u>simples</u>
- Cada face armazena explicitamente a lista ordenada das coordenadas dos seus vértices:

$$P = \{(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)\}$$

- Muita <u>redundância</u> de informação
- Consultas são complicadas
  - Obriga à execução de algoritmos geométricos para determinar adjacências

Desenho da Malha

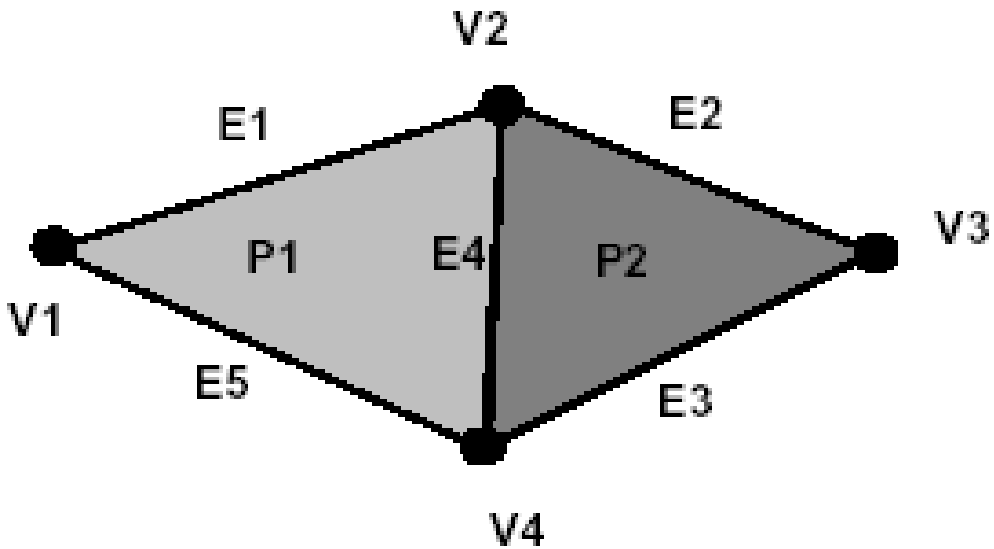
- Cada aresta é desenhada duas vezes - pelas duas faces que a compartilham
- Não é bom para plotters ou filmes



Apontadores para Lista de Vértices

- Vértices são armazenados separadamente
- Há uma lista de vértices
- Faces referenciam os seus vértices através de ponteiros
- Proporciona maior economia de memória
- Achar adjacências ainda é complicado
- Arestas ainda são desenhadas duas vezes

Exemplo



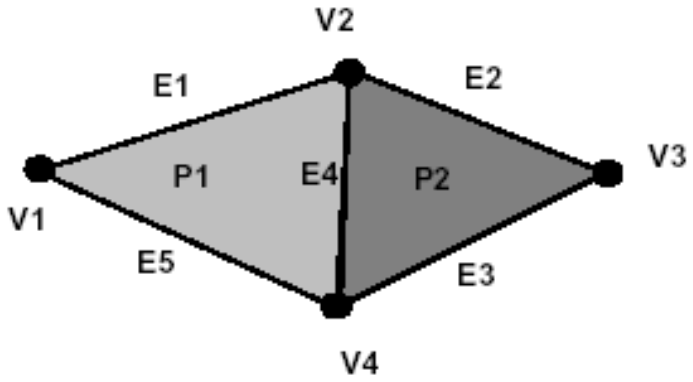
- $V = \{V_1 = (x_1, y_1, z_1), V_2 = (x_2, y_2, z_2), V_3 = (x_3, y_3, z_3), V_4 = (x_4, y_4, z_4)\}$;
- $P_1 = \{V_1, V_2, V_4\}$;
- $P_2 = \{V_4, V_2, V_3\}$.

Apontadores para Lista de Arestas

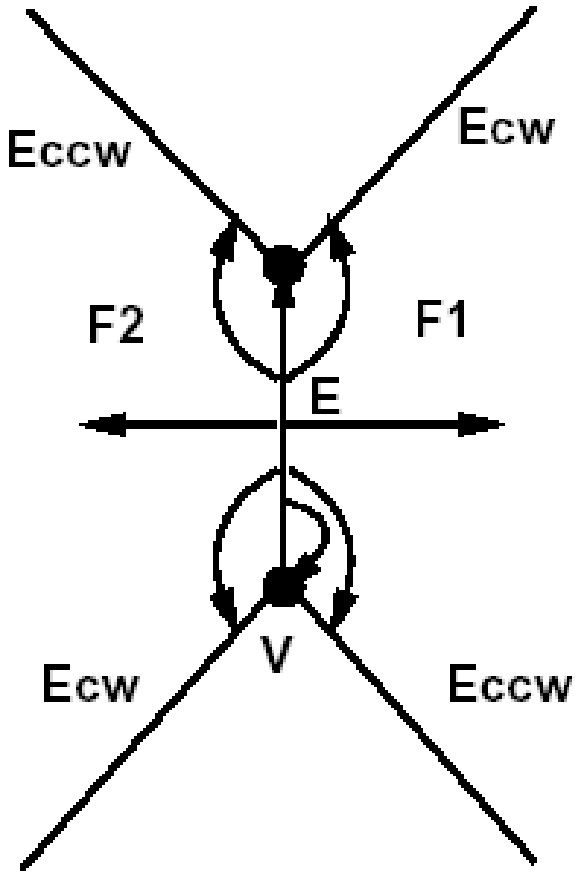
- Há também uma lista de arestas
- Faces referenciam as suas arestas através de ponteiros
- Arestas são desenhadas percorrendo-se a lista de arestas
- Introduzem-se referências para as duas faces que compartilham uma aresta
  - Facilita a determinação das duas faces incidentes na aresta

Exemplo

- $V = \{V_1 = (x_1, y_1, z_1), V_2 = (x_2, y_2, z_2), V_3 = (x_3, y_3, z_3), V_4 = (x_4, y_4, z_4)\}$;
- $E_1 = \{V_1, V_2, P_1, \lambda\}$;
- $E_2 = \{V_2, V_3, P_2, \lambda\}$;
- $E_3 = \{V_3, V_4, P_2, \lambda\}$;
- $E_4 = \{V_2, V_4, P_1, P_2\}$;
- $E_5 = \{V_4, V_1, P_1, \lambda\}$;
- $P_1 = \{E_1, E_4, E_5\}$;
- $P_2 = \{E_2, E_3, E_4\}$.



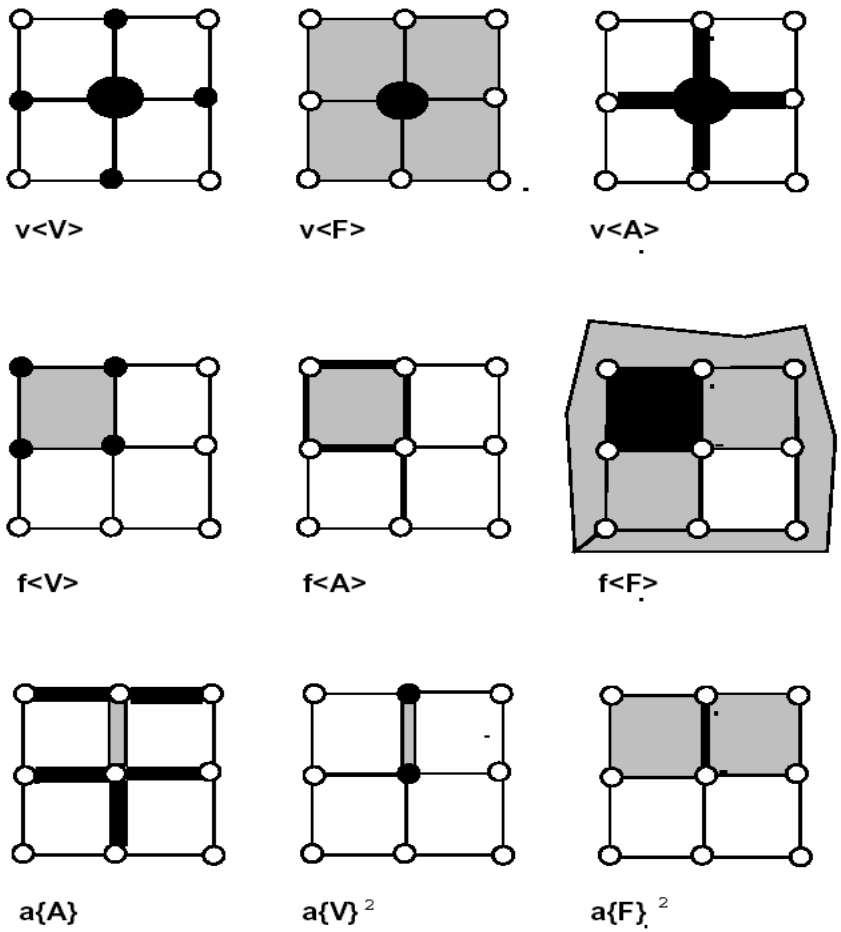
Winged-Edge



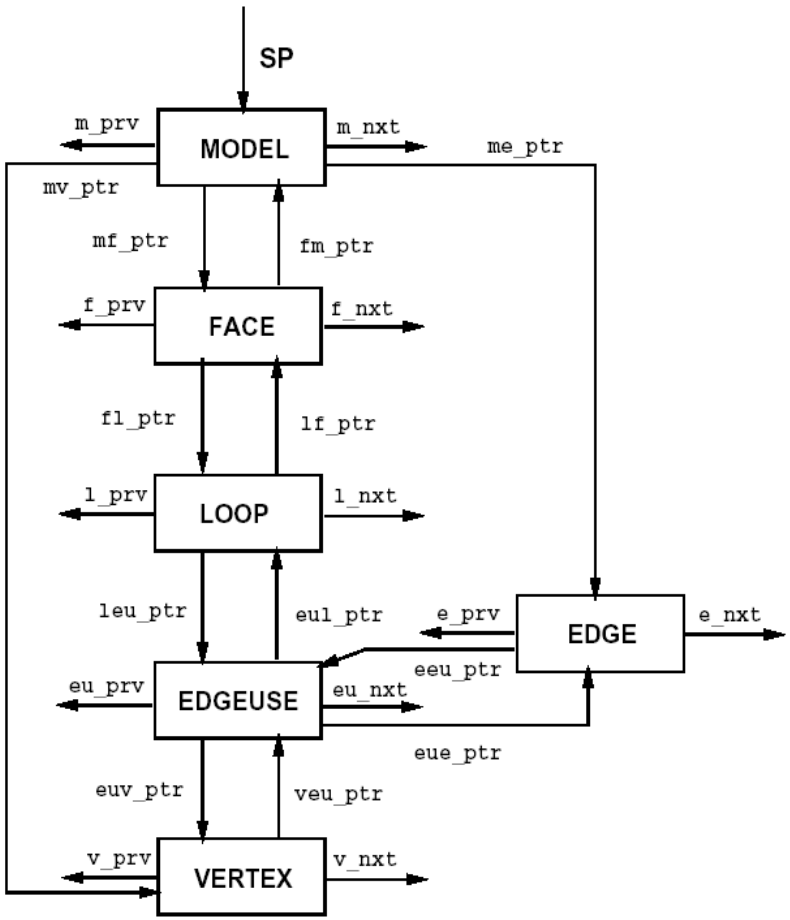
Winged-Edge

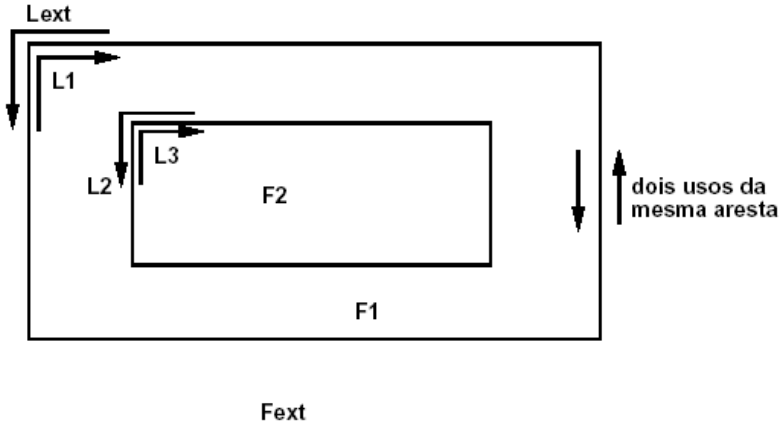
- Criada em 1974 por Baumgart
- Foi um marco na representação por fronteira
- Armazena informação na estrutura associada às arestas (número de campos é fixo)
- Todos os 9 tipos de adjacência entre vértices, arestas e faces são determinados em tempo constante
- Actualizada com o uso de operadores de Euler, que garantem: $V - A + F = 2$

9 tipos de Relacionamentos de Adjacência



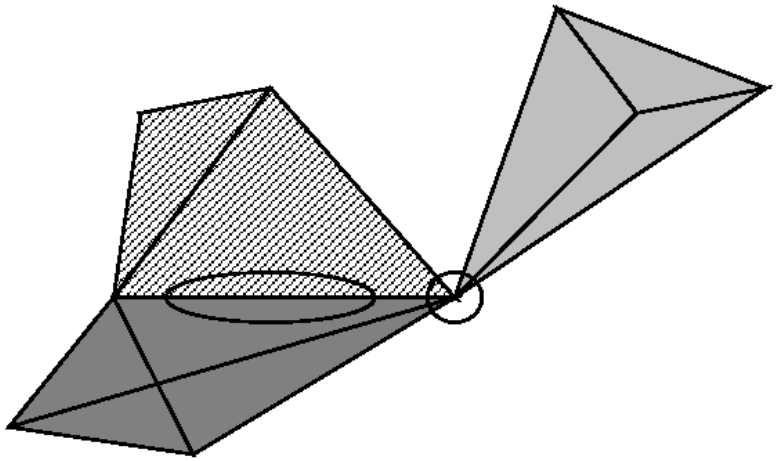
Face-Edge


SP
MODEL
m_prv
m_nxt
me_ptr
mv_ptr
mf_ptr
fm_ptr
FACE
f_prv
f_nxt
fl_ptr
lf_ptr
LOOP
l_prv
l_nxt
leu_ptr
eul_ptr
EDGE
e_prv
e_nxt
eeu_ptr
EDGEUSE
eu_prv
eu_nxt
eue_ptr
euv_ptr
veu_ptr
VERTEX
v_prv
v_nxt

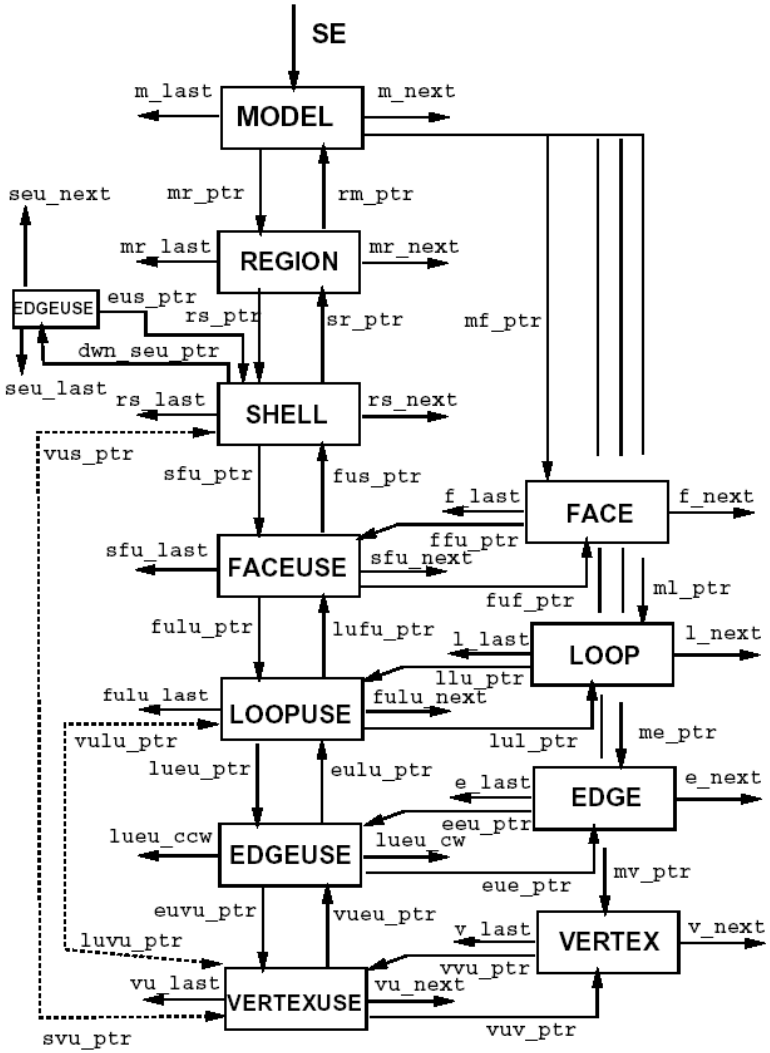

Lext
L1
L2
L3
F2
F1
Fext
dois usos da
mesma aresta

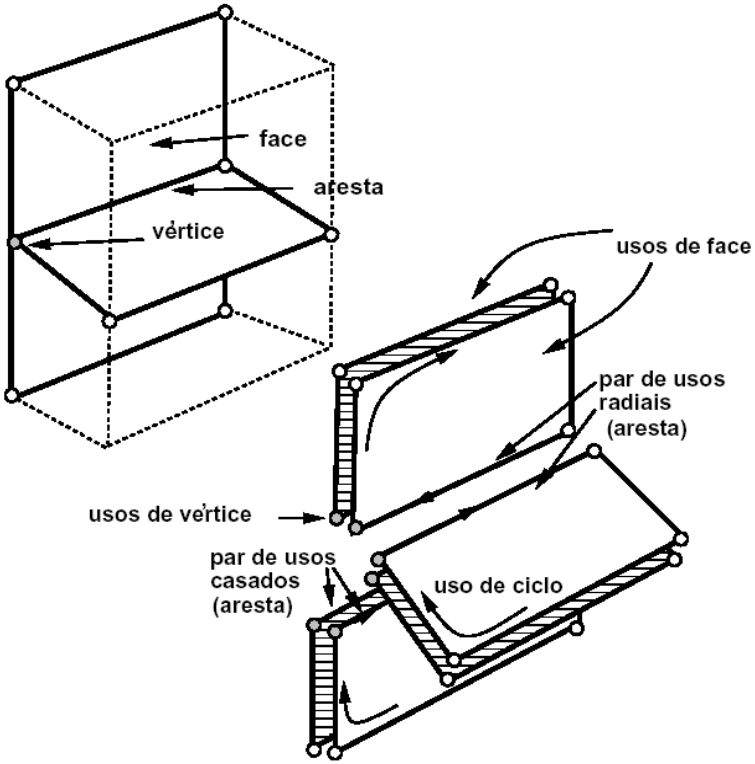
Radial-Edge

- Criada em 1986 por Weiler
- Representa objectos *non-manifold* (não variedades)
- Armazena a lista ordenada de faces incidentes em uma aresta
- Muito mais complicada que a *Winged-Edge*



Radial-Edge


SE
MODEL
m_last
m_next
mr_ptr
rm_ptr
seu_next
REGION
mr_last
mr_next
EDGEUSE
eus_ptr
rs_ptr
sr_ptr
mf_ptr
dwn_seu_ptr
seu_last
SHELL
rs_last
rs_next
vus_ptr
sfu_ptr
fus_ptr
FACE
f_last
f_next
ffu_ptr
sfu_last
FACEUSE
sfu_next
ml_ptr
fuf_ptr
fulu_ptr
lufu_ptr
LOOP
l_last
l_next
llu_ptr
fulu_last
LOOPUSE
fulu_next
me_ptr
vulu_ptr
lul_ptr
lueu_ptr
eulu_ptr
EDGE
e_last
e_next
eeu_ptr
lueu_ccw
EDGEUSE
lueu_cw
mv_ptr
eue_ptr
euvu_ptr
vueu_ptr
VERTEX
v_last
v_next
luvu_ptr
vvu_ptr
vu_last
VERTEXUSE
vu_next
vuv_ptr
svu_ptr


face
aresta
vértice
usos de face
par de usos
radiais
(aresta)
usos de vértice
par de usos
casados
(aresta)
uso de ciclo

Representação Implícita

- Sólido é definido por um conjunto de valores que caracterizam os seus pontos
- Descreve a superfície dos objectos, implicitamente, por uma equação:

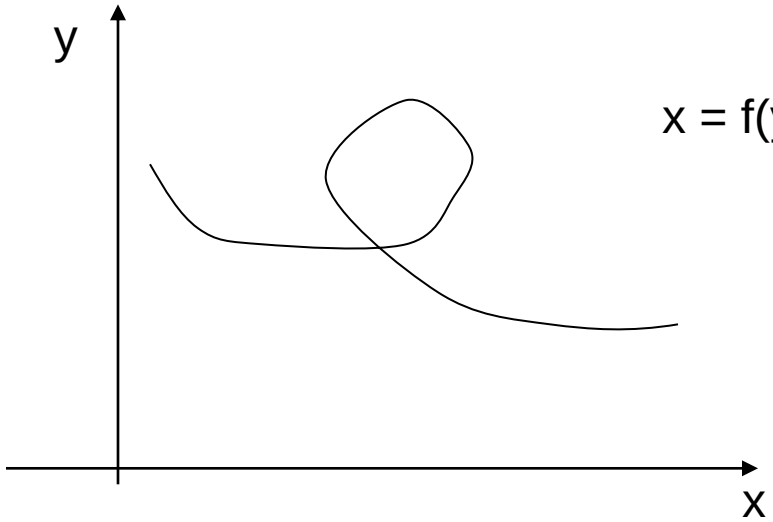
$$F(x) = c; X \in \Re^n, c \in \Re.$$

$$F : \Re^n \rightarrow \Re \text{ de classe } C^k.$$

- F é designada função implícita

Funções Implícitas

- Uma superfície definida de forma implícita pode apresentar <u>auto-intersecção</u>
- Pergunta: $F(x,y,z)$ define implicitamente $z = f(x,y)$ em algum domínio razoável?



x = f(y) ou y = f(x) ?

Teorema da Função Implícita

- Seja $F : \Re^n \rightarrow \Re$ definida num conjunto aberto U
- Se F possui derivadas parciais contínuas em U e $\nabla F \neq 0$ em U, então F é uma <u>subvariedade</u> de dimensão n - 1 do $\Re^n$
  - Superfície sem auto-intersecção

Valores Regulares

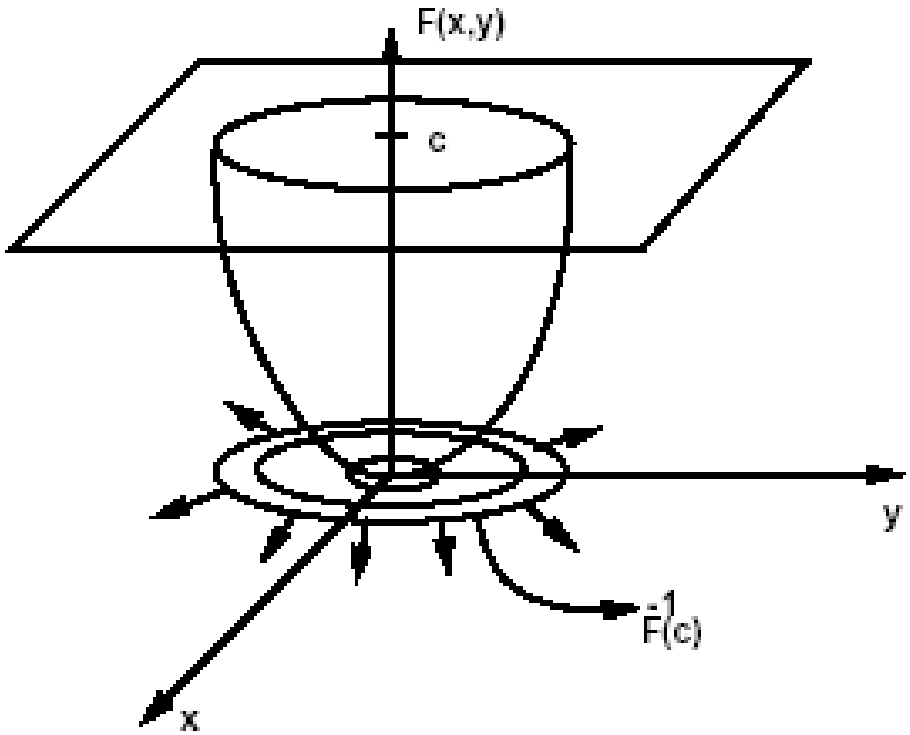
- Um valor c é dito **regular** se $F^{-1}(c)$ não contém pontos onde $\nabla F = 0$ (pontos singulares)

$$\forall p \in F^{-1}(c) \Rightarrow \nabla F_p = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z} \right)\Big|_p \neq 0.$$

- Neste caso interessam apenas os casos em que $n = 2$ ou 3 (curvas e superfícies implícitas)

Exemplo 1

- Seja $F(x,y) = x^2 + y^2$ que define um parabolóide no $\Re^3$
- Curvas de nível são círculos
- $\nabla F = (2x, 2y)$ anula-se na origem
- 0 não é valor regular de F Logo $F(x,y) = 0$ não define uma função implícita



Exemplo 2

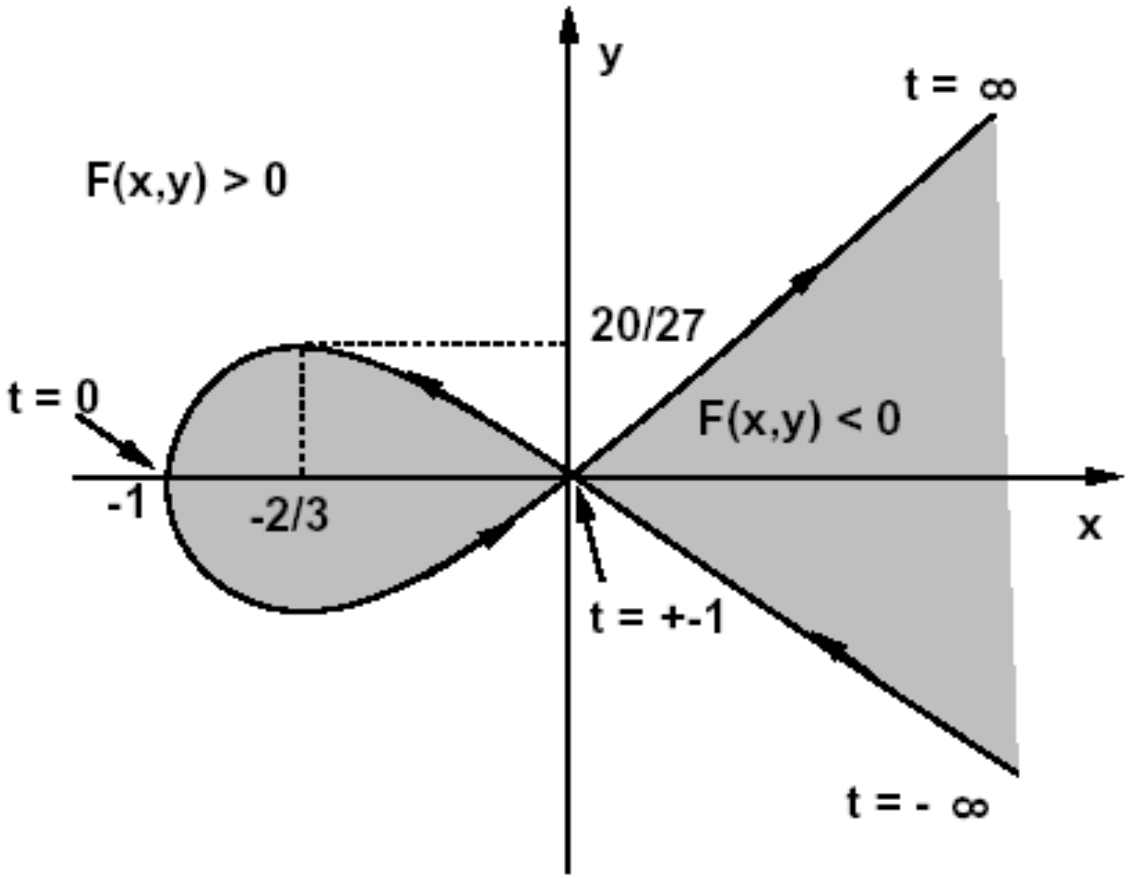
- Cascas esféricas: $F(x,y,z) = x^2 + y^2 + z^2$
- Para todo $k > 0$, $F^{-1}(k)$ representa a superfície de uma esfera em $\Re^3$
- 0 não é valor regular de F
  - $F^{-1}(0) = (0,0,0)$ e $\nabla F=(2x, 2y, 2z)$ anula-se na origem

Exemplo 3

- $F(x,y) = y^2 - x^2 - x^3$, $\nabla F = (2y, -3x^2 - 2x)$
- Na forma paramétrica:
  - $x(t) = t^2 - 1$ e $y(t) = t\,(t^2 - 1)$
- Curva de nível 0 é um laço, com uma singularidade na origem:

$$z = F(x,y) = y^2 - x^2 - x^3 = 0$$

Gráfico do Exemplo 3



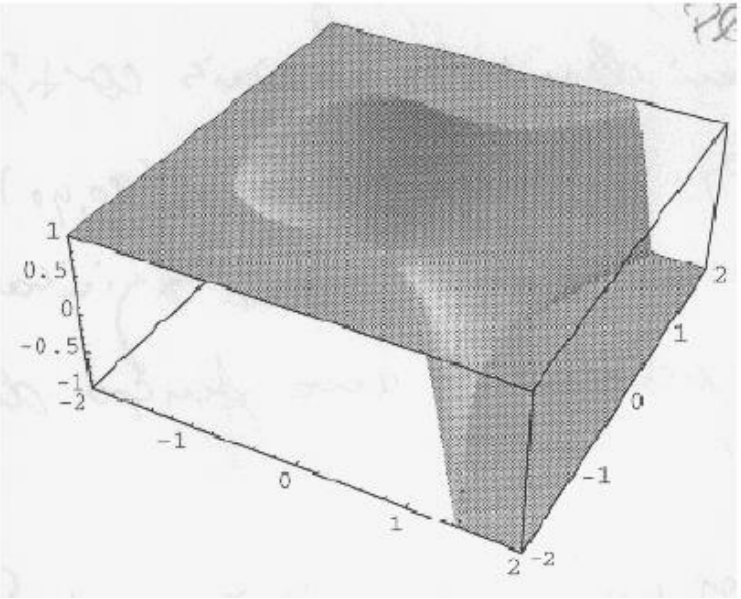
Observação

- Olhando $F(x,y)$ como superfície de nível 0 da função $H : \Re^3 \to \Re$,

  $H(x,y,z) = -z + y^2 - x^2 - x^3$,

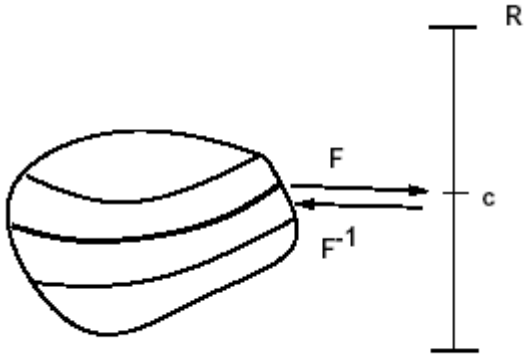
  $\nabla H = (-3x^2 - 2x, 2y, -1)$;

  $\nabla H(0,0,0) = (0,0,-1)$
- Todos os pontos são regulares
- Gráfico de F em $\Re^3$ é realmente o gráfico de uma função!



Objecto Implícito

- Um subconjunto $O \subset \Re^n$ é designado **objecto implícito** se existe $F : U \rightarrow \Re$, $O \subset U$, e existe um subconjunto $V \subset \Re$ / $O = F^{-1}(V)$ ou $O = \{p \in U, F(p) \in V\}$.
- Um objecto implícito é dito **regular** se F satisfaz a condição de regularidade
- Um objecto implícito é <u>válido</u> se define uma superfície em $\Re^n$

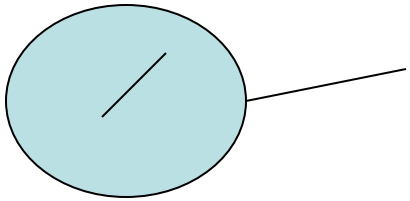


Interior x Exterior

- A função F faz a classificação dos pontos do espaço
- Permite decidir se o ponto está no interior, na fronteira ou no exterior
  - $F > 0 \Rightarrow p \in$ exterior de O
  - $F = 0 \Rightarrow p \in$ fronteira de O
  - $F < 0 \Rightarrow p \in$ interior de O

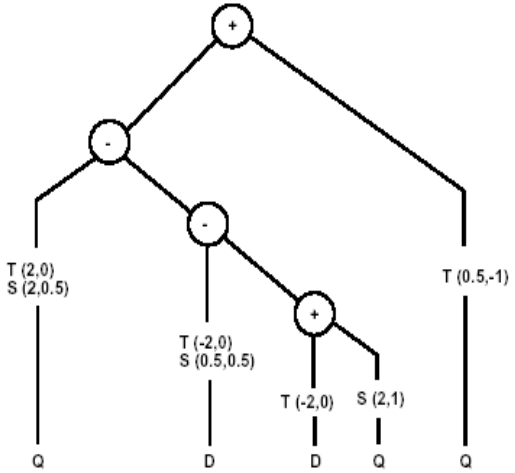
Esquema de Representação CSG

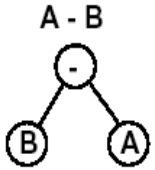
- Operações CSG definem objectos através de operações **regularizadas** de conjuntos de pontos
  - União, Intersecção e Diferença
- Um objecto é **regular** se o <u>fecho</u> do <u>interior</u> do seu conjunto de pontos é igual ao próprio conjunto de pontos



Árvore CSG

- Um modelo CSG é codificado por uma <u>árvore</u>
  - Os nós internos contêm operações de conjunto ou transformações lineares afins
  - Folhas contêm objectos primitivos (tipicamente, quádricas)





T = Translação + = União de Conjuntos

S = Escalamento - = Diferença de Conjuntos

CSG com Objectos Implícitos

- Primitivas CSG são definidas por $F_i(X) \leq 0$
- Operações *booleanas* são definidas nesse caso por:
  - $F_1 \cup F_2 = \min(F_1, F_2)$
  - $F_1 \cap F_2 = \max(F_1, F_2)$
  - $F_1 \setminus F_2 = F_1 \cap \overline{F_2} = \max(F_1, -F_2)$

Prós e Contras de Representações

- Representações por fronteira e por campos escalares apresentam vantagens e desvantagens
- Numa B-rep as intersecções estão representadas <u>explicitamente</u> e é mais fácil exibir um ponto sobre a superfície do objecto
- Porém é difícil determinar, dado um ponto, se ele está no interior, fronteira ou exterior do objecto
- Operações *booleanas* são complicadas

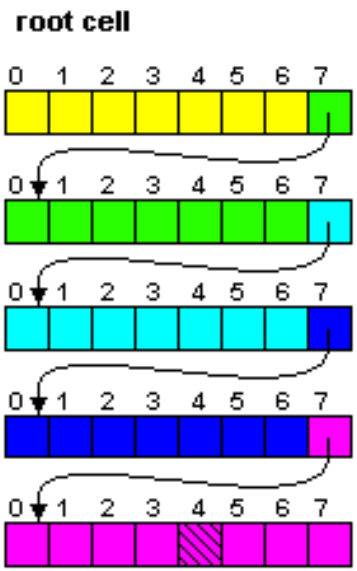
Representações por Campos Escalares

- Em tais representações a classificação de um ponto é imediata, bastando avaliar o sinal do valor do campo no ponto
- Exibir um ponto sobre a superfície do objecto requer a solução de uma equação, a qual pode ser complicada
- Operações *booleanas* são avaliadas facilmente

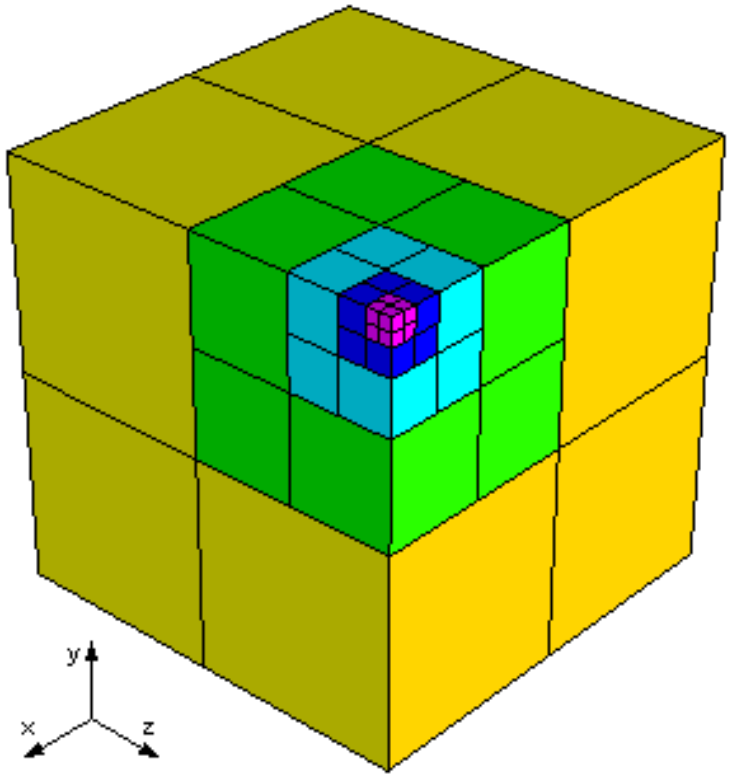
Representações por Células

- Dividem o espaço em sub-regiões convexas
  - Grelhas: Cubos de tamanho igual
  - Octrees: Cubos cujos lados são potências de 2
  - BSP-trees: Poliedros convexos
- Às células são atribuídas valores de um campo escalar $F(x, y, z)$
  - Campo é assumido constante dentro de cada célula
- Sólido é definido como o conjunto de pontos tais que $A < F(x, y, z) < B$ para valores A e B estipulados

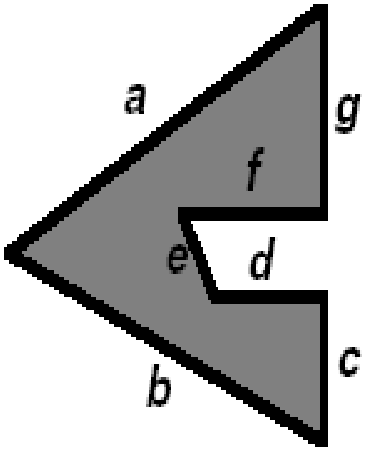
Octrees



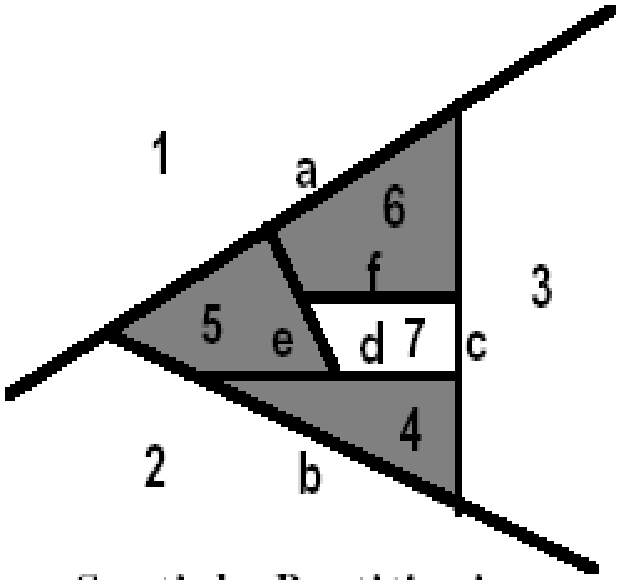
		0	1	2	3	4
Z	31	1	1	1	1	1
Y	30	1	1	1	1	0
X	30	1	1	1	1	0
		7	7	7	7	4



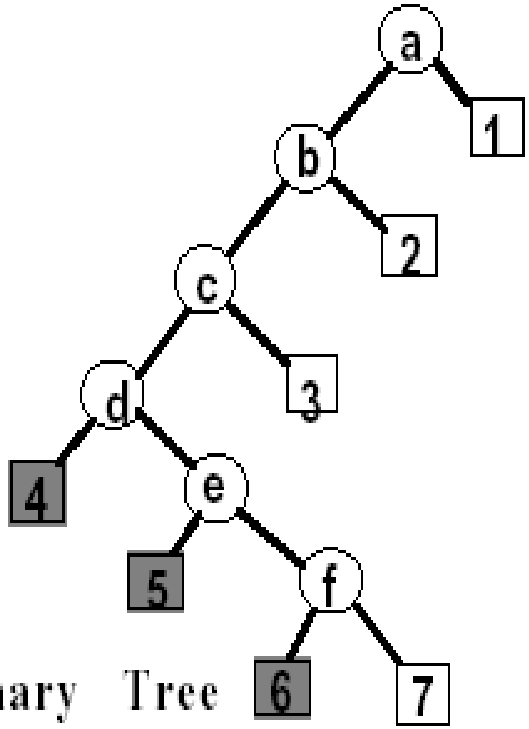
BSP-Trees



Orignal B-rep



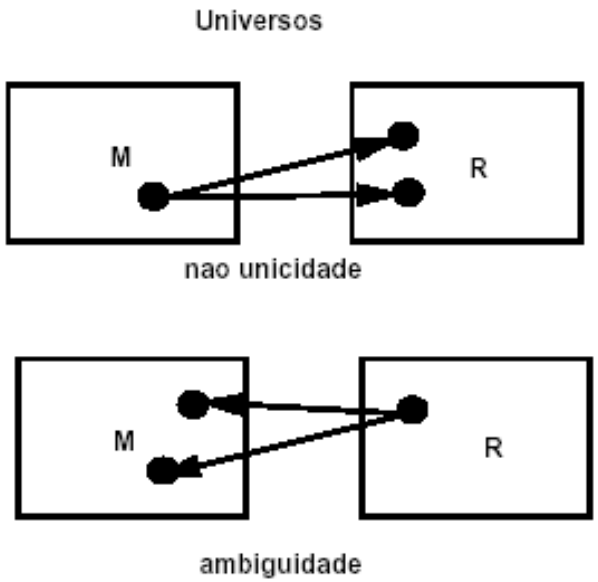
Spatial Partitioning



Binary Tree

Ambiguidade e Unicidade

- Uma representação é única quando o modelo associado possui uma única representação
- Uma representação é ambígua quando pode representar mais de um modelo
- Representação ambígua é catastrófica (*wireframe*)
  - Inviabiliza máquinas de controlo numérico



Conversão entre Representações

- Conversão CSG → B-rep é denominada **avaliação de fronteira**
- Conversão B-rep → CSG é muito mais complicada
- Conversão B-rep → Células é simples
- Conversão Células → B-rep é relativamente simples (*marching cubes*)
- Conversão CSG → Células é simples
- Conversão Células → CSG é complicado

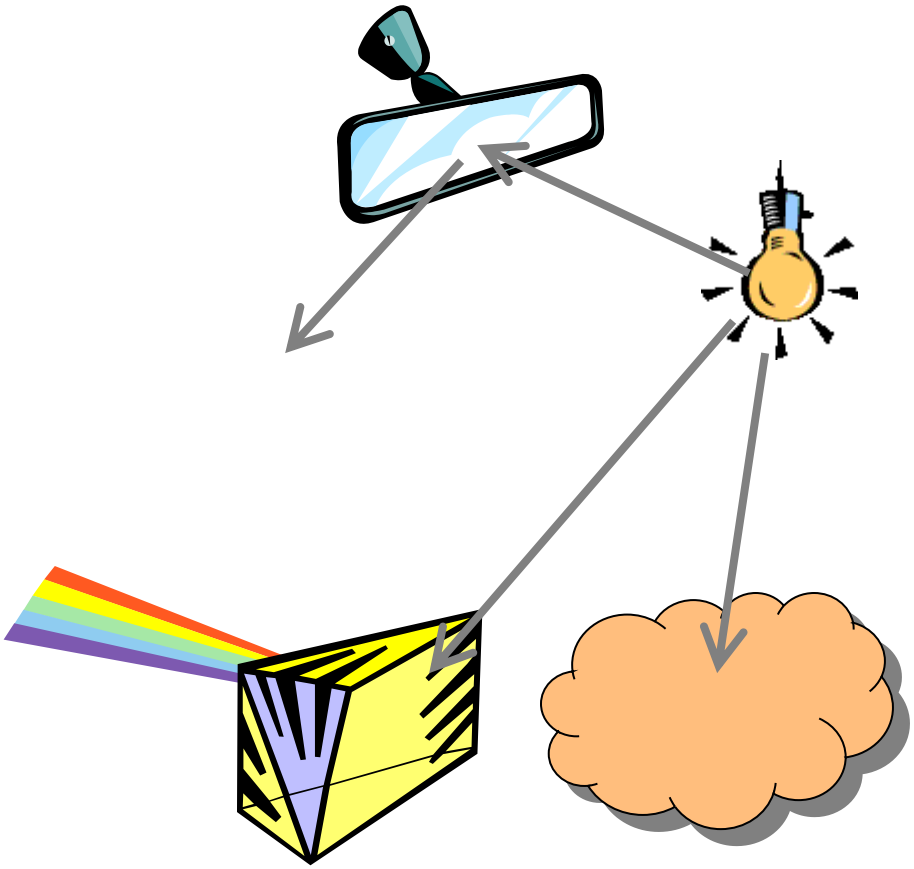
Introdução à Computação Gráfica
Iluminação

Adaptação: João Paulo Pereira
António Costa

Autoria: Claudio Esperança
Paulo Roma Cavalcanti

Iluminação

- Estudo de como a luz interage com os objectos de uma cena
  - Emissão
  - Transmissão
  - Absorção
  - Refracção
  - Reflexão



Iluminação

- Modelos físicos
  - Luz modelada como radiação electromagnética
  - Leva em conta todas as interacções (todos os caminhos da luz)
  - Intratável computacionalmente

Modelos de Iluminação em CG

- Tipicamente, a luz é amostrada num número discreto de primárias (cor)
- Modelos locais
  - Apenas caminhos do tipo *fonte luminosa* → *superfície* → *olho* são tratados
  - Simples
- Modelos globais
  - Muitos caminhos (*ray tracing,* radiosidade)
  - Complexos

Iluminação

- Fontes posicionais
  - Omnidireccionais
  - *Spot*
- Fontes direccionais
- Interacções de luz com superfícies modeladas em componentes (modelo de Phong):
  - Ambiente
  - Difusa
  - Especular
  - Emissão

Iluminação

- Suporte de efeitos atmosféricos como
  - *Fog*
  - Atenuação
- Reflexão difusa calculada apenas nos vértices das superfícies (*vertex shading*)
  - Cor dos restantes pixels é interpolada linearmente (sombreamento de *Gouraud*)
- Reflexão especular calculada nos píxeis (*pixel shading*)

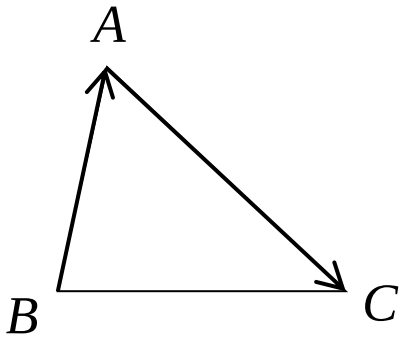
Geometria

- Além das propriedades da luz e do material, a geometria do objecto é também importante
  - A posição dos vértices em relação ao olho e à fonte luminosa contribui para o cálculo dos efeitos atmosféricos
  - A *normal* é fundamental
    - Não é calculada automaticamente

Cálculo do Vector Normal

- Triângulo
  - Dados três vértices,

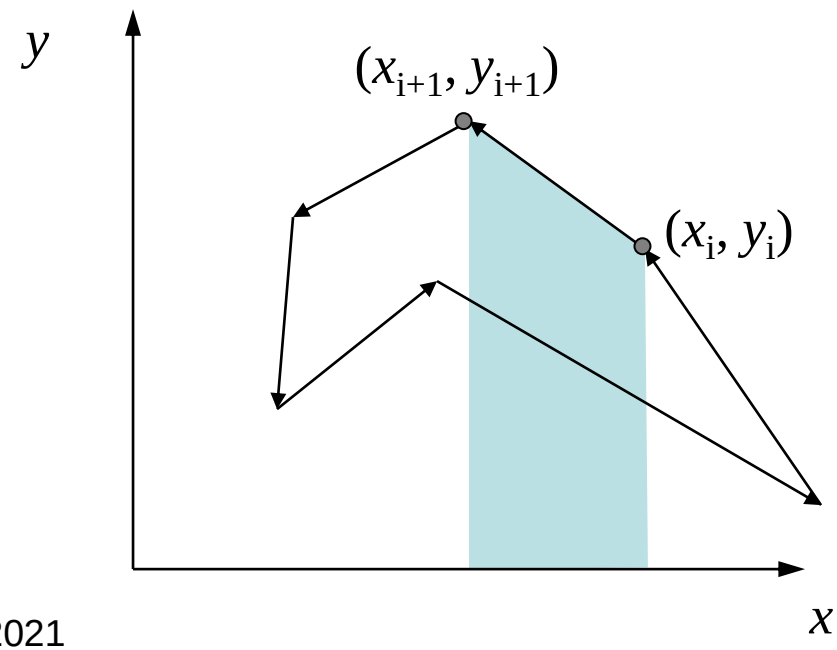
$$\vec{n} = \text{normalizar}((A - B) \times (C - A))$$



- Polígono planar
  - Uma opção é usar a fórmula do triângulo para quaisquer 3 vértices
    - Sujeito a erros (vectores pequenos ou quase colineares)
  - Outra opção é determinar a equação do plano
    - $ax + by + cz + d = 0$
    - Normal tem coordenadas (a, b, c)

Cálculo do Vector Normal

- Polígono planar (cont.)
  - Coeficientes *a, b, c* da equação do plano são proporcionais às áreas do polígono projectado nos planos *yz*, *zx* e x*y*



$$AreaXY_i = \frac{1}{2}(y_i + y_{i+1})(x_i - x_{i+1})$$

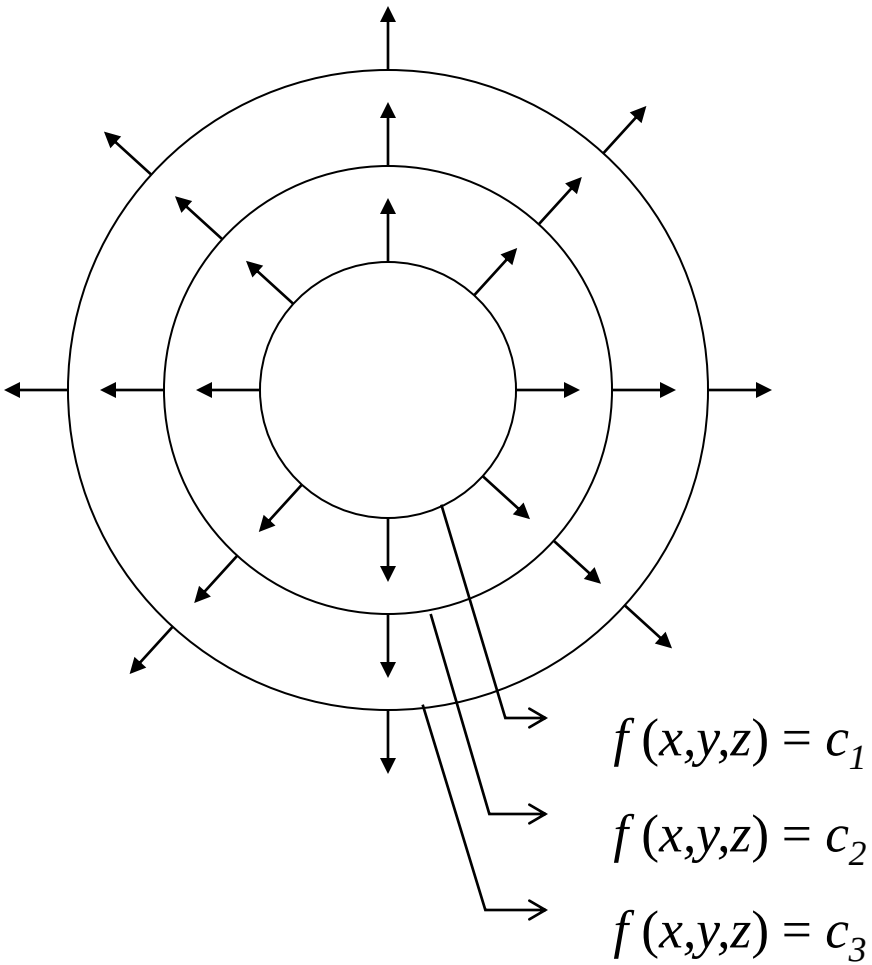
$$c = \sum AreaXY_i$$

Cálculo do Vector Normal de Superfícies Implícitas

- Normal é dada pelo vector gradiente

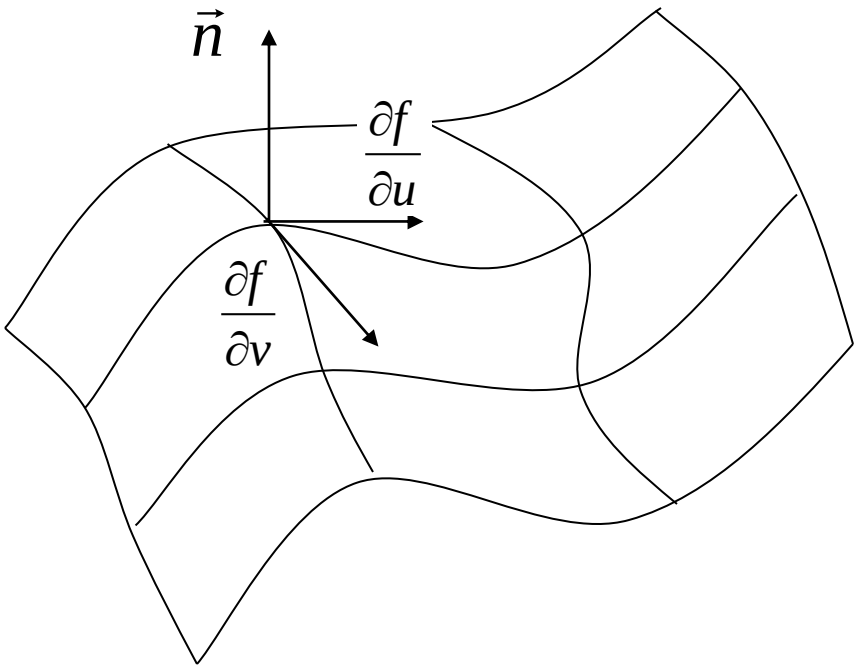
$$f(x, y, z) = 0$$

$$\vec{n} = \begin{pmatrix} \partial f / \partial x \\ \partial f / \partial y \\ \partial f / \partial z \end{pmatrix}$$



Cálculo do Vector Normal de Superfícies Paramétricas

- Normal é dada pelo produto vectorial dos gradientes em relação aos parâmetros u e v



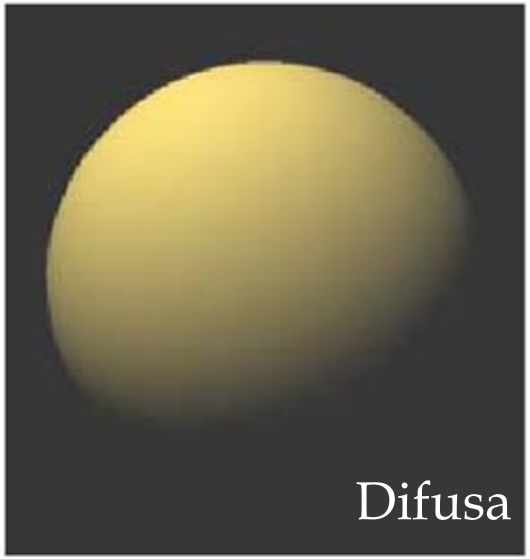
$$P = \begin{pmatrix} f_x(u,v) \\ f_y(u,v) \\ f_z(u,v) \end{pmatrix}$$

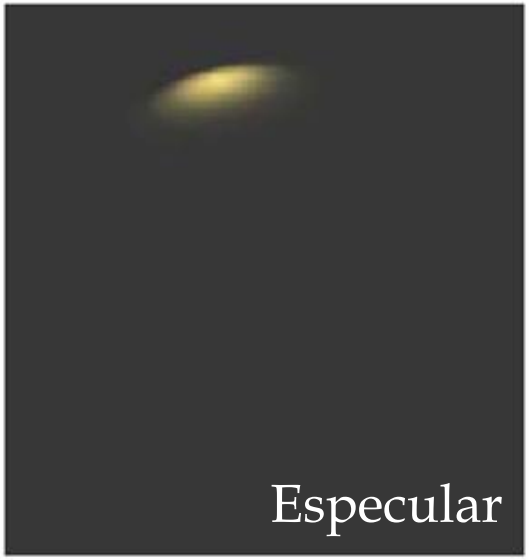
$$\vec{n} = \frac{\partial f}{\partial u} \times \frac{\partial f}{\partial v} = \begin{pmatrix} \partial f_x / \partial u \\ \partial f_y / \partial u \\ \partial f_z / \partial u \end{pmatrix} \times \begin{pmatrix} \partial f_x / \partial v \\ \partial f_y / \partial v \\ \partial f_z / \partial v \end{pmatrix}$$

Componentes do Modelo de Phong

- Ambiente: contribuição que não depende da geometria
- Difusa: contribuição correspondente ao espalhamento da reflexão *lambertiana* (independente da posição do observador)
- Especular: contribuição referente ao comportamento de superfícies polidas
- Emissão: contribuição que não depende de fontes de luz (fluorescência)

Componentes do Modelo de Phong





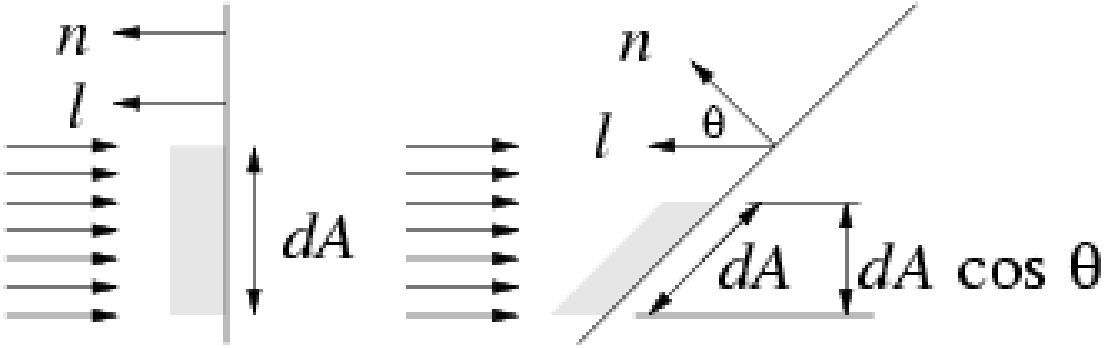
Iluminação Ambiente

- Componente que modela como uma constante o efeito da reflexão de outros objectos do ambiente
- Contribuição é dada por

$$A = I_A M_A$$

Iluminação Difusa

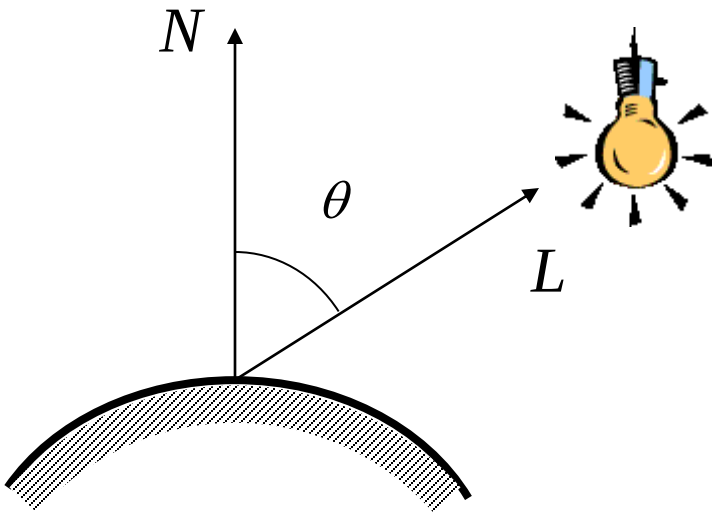
- Iluminação recebida por uma superfície e que é reflectida uniformemente em todas as direcções
- Característica de materiais baços ou foscos
- Esse tipo de reflexão é também designada por reflexão *lambertiana*
- A luminosidade aparente da superfície não depende do observador, mas apenas do co-seno do ângulo de incidência da luz



Iluminação Difusa

- Contribuição é dada por

$$D = I_D M_D \cos\theta = I_D M_D \left(\frac{L \cdot N}{|L||N|}\right)$$



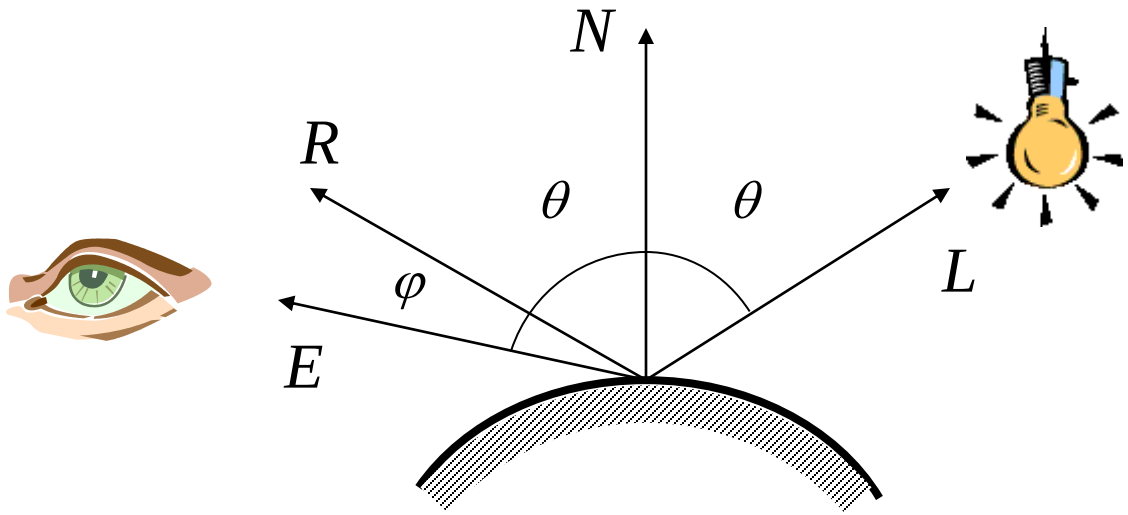
Iluminação Especular

- Simula a reflexão à maneira de um espelho (objectos altamente polidos)
- Depende da posição do observador, objecto e fonte de luz
- Num espelho perfeito, a reflexão dá-se em ângulos iguais
  - Observador só veria a reflexão de uma fonte pontual se estivesse na direção certa
- No modelo de Phong simulam-se reflectores imperfeitos, assumindo que luz é reflectida segundo um *cone* cujo eixo passa pelo observador

Iluminação Especular

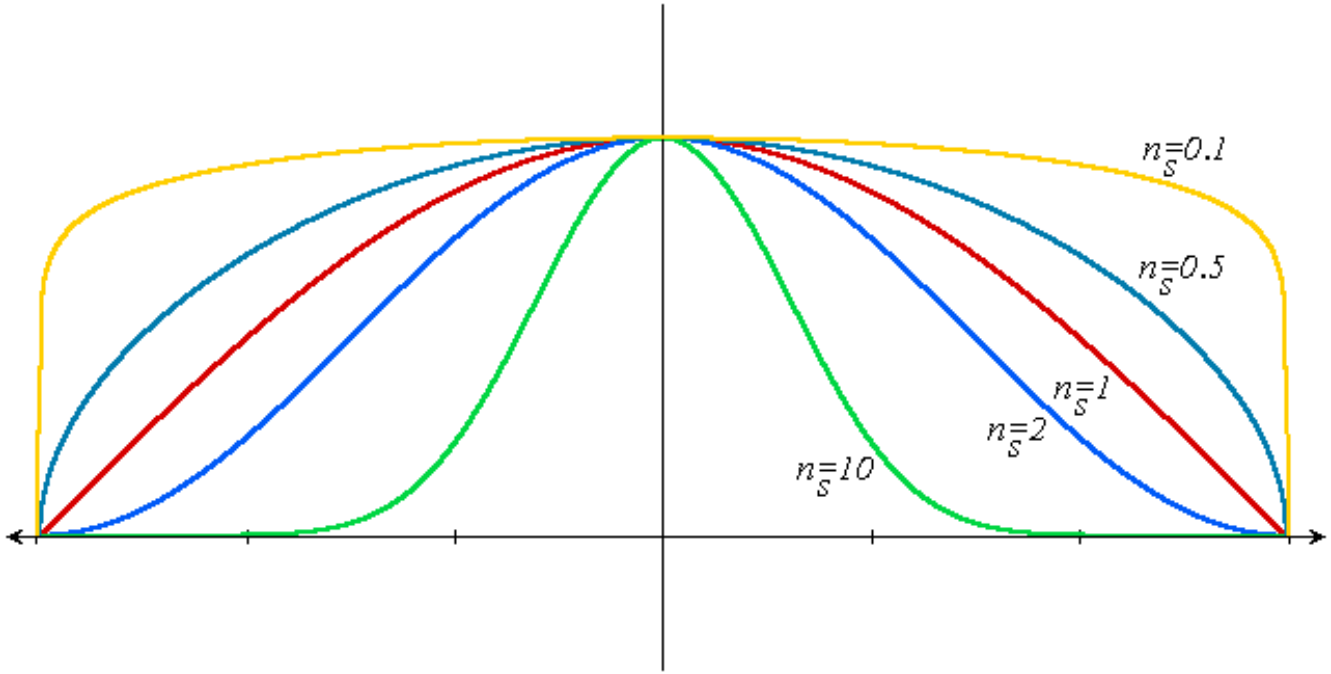
- Contribuição é dada por

$$S = I_S M_S \cos^n \varphi = I_S M_S (R \cdot E)^n$$



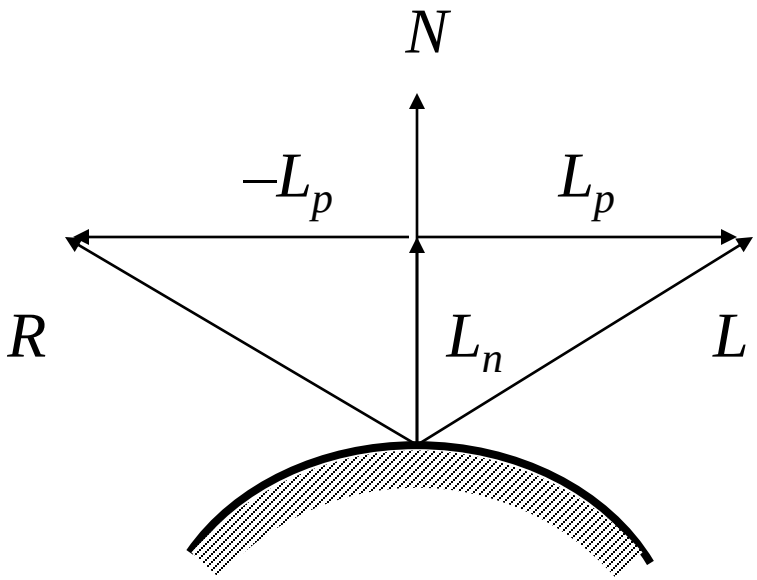
Coeficiente de Especularidade

- Indica quão polida é a superfície
  - Espelho ideal tem coeficiente de especularidade infinito
  - Na prática, usam-se valores entre 5 e 100



Cálculo do vector da luz reflectida

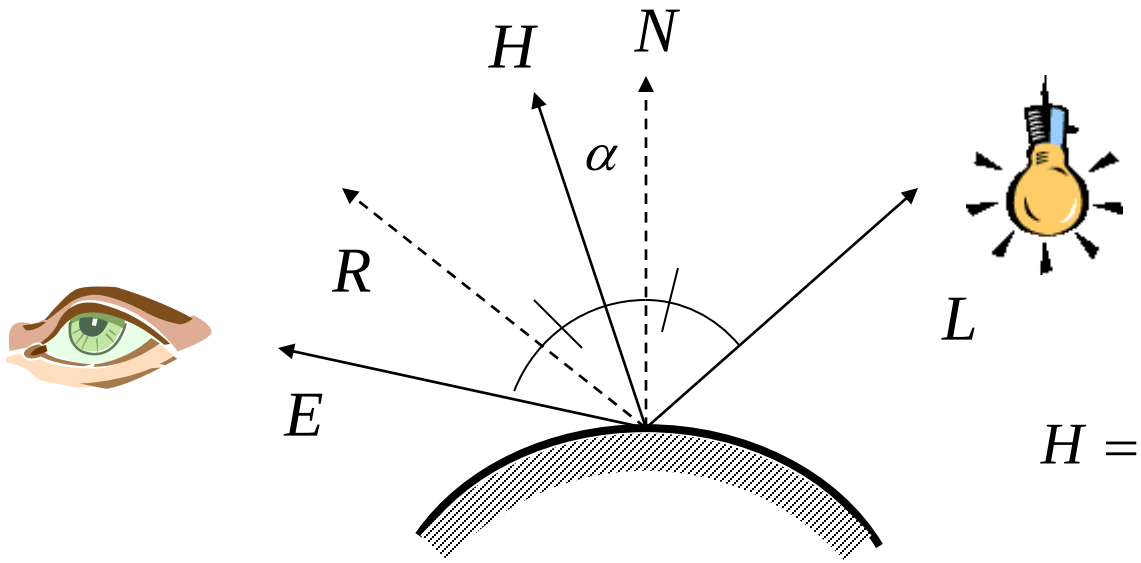
$$R = L_n - L_p = (N \cdot L)N - (L - L_n) = 2(N \cdot L)N - L$$



Componente Especular (Modelo de Blinn-Phong)

- Utiliza o ângulo entre a normal e o vector *halfway*

$$S = I_S M_S \cos^n \alpha = I_S M_S (H \cdot N)^n$$



$$H = \text{normalizar}\left(\frac{E+L}{2}\right)$$
$$= \text{normalizar}(E+L)$$

Atenuação

- Para fontes de luz posicionais (w = 1), é possível definir um factor de atenuação que leva em conta a distância d entre a fonte de luz e o objecto iluminado
- Não há atenuação quando c_0=1, c_1=c_2=0

$$aten = \frac{1}{c_0 + c_1 d + c_2 d^2}$$

Juntando tudo

- A atenuação só é aplicada sobre as componentes difusa e especular
- A fórmula que calcula a cor devida a uma fonte luminosa i é dada por

$$C_i = aten\,(D_i + S_i)$$

- No final, a cor é dada pela contribuição da iluminação ambiente (parcela não associada com fontes de luz) somada à luz emitida e às contribuições C_i

$$C = A + E + \sum aten\,(D_i + S_i)$$

Introdução à Computação Gráfica
Texturas

Adaptação: João Paulo Pereira
António Costa

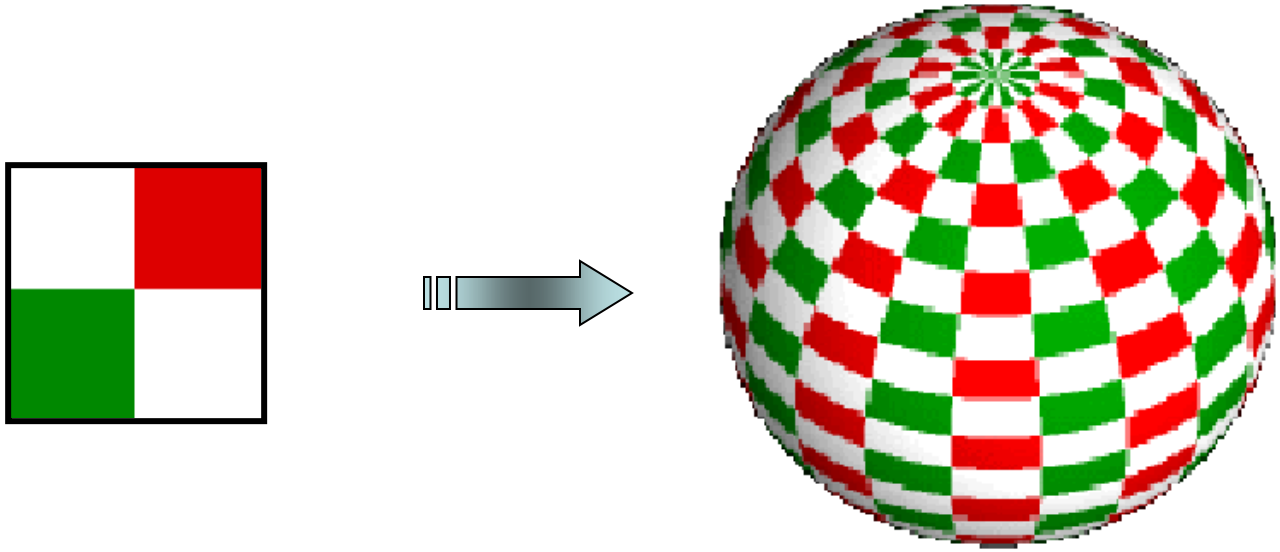
Autoria: Claudio Esperança
Paulo Roma Cavalcanti

Detalhes de Superfícies

- Modelos de iluminação não são apropriados para descrever todas as diferenças de cor observáveis numa superfície
  - Superfícies pintadas com padrões ou imagens
    - A capa ou uma página de um livro
  - Superfícies com padrões de rugosidade
    - Tecidos ou uma parede de tijolos
- Em princípio é possível modelar esses detalhes com geometria e usando materiais de propriedades ópticas distintas
- Na prática, esses efeitos são modelados usando uma técnica chamada *mapeamento de textura*

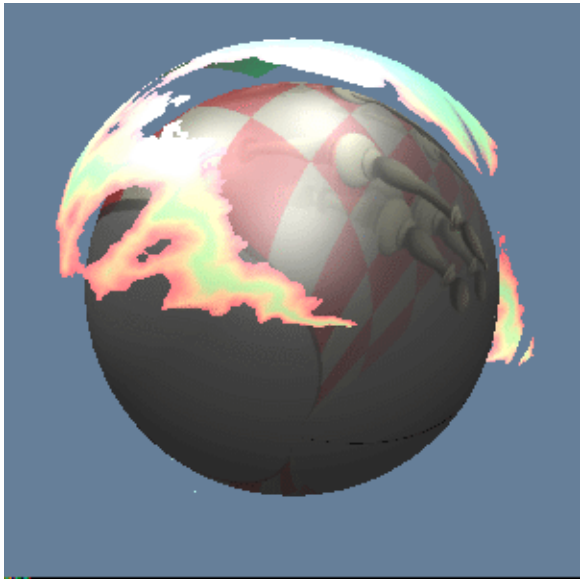
Mapeamento de Textura

- A ideia é reproduzir sobre a superfície de algum objecto da cena as propriedades de alguma função – ou mapa - bidimensional (cor, por exemplo)



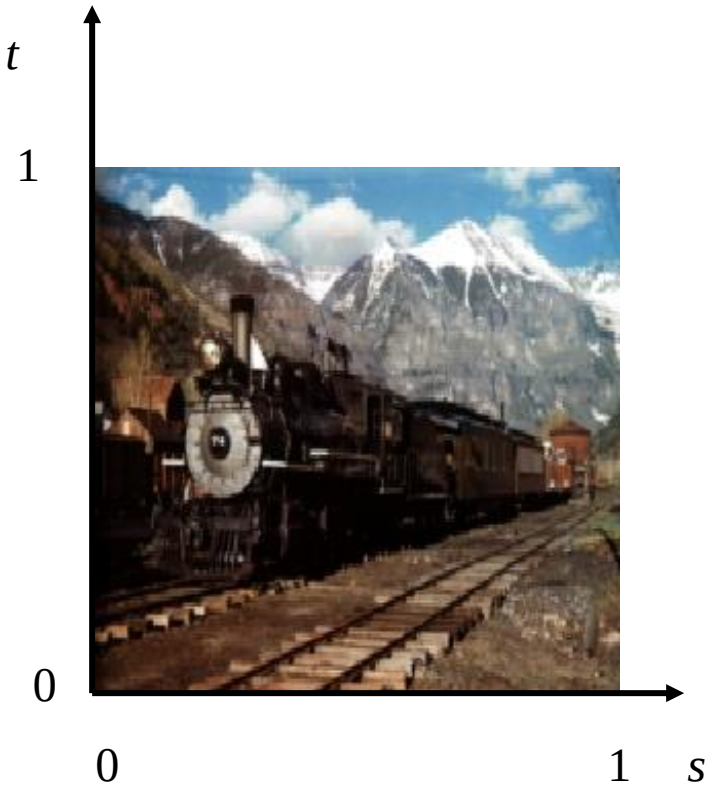
Propriedades Mapeáveis

- Que parâmetros ou propriedades se podem reproduzir a partir de mapas:
  - Cor (coeficientes de reflexão difusa)
  - Coeficientes de reflexão especular e difusa
    - "Environment Mapping"
  - Perturbação do vector normal
    - "Bump Mapping"
  - Perturbação da superfície na direcção da normal
    - "Displacement Mapping"
  - Transparência / opacidade



Espaço de Textura

- Texturas 2D são funções $T(s, t)$ cujo domínio é um espaço bidimensional e o contradomínio pode ser cor, opacidade, etc.
- É comum ajustar a escala da imagem de tal forma que a imagem toda se enquadre no intervalo $0 \leq s, t \leq 1$
- Normalmente a função em si é derivada de alguma imagem capturada
  - Se a imagem está armazenada numa matriz $Im\,[0..N-1\,,\,0..M-1]$
  - Então $T(s, t) = Im\,[\lfloor(1 - t)\,N\rfloor, \lfloor s\,M\rfloor]$



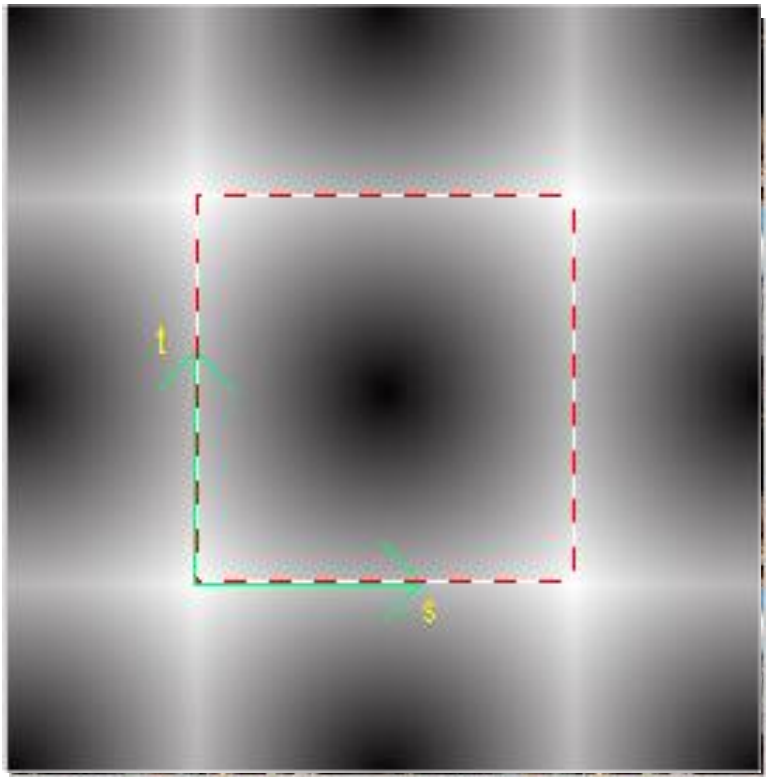
Espaço de Textura

- Pode ser vantajoso assumir que o padrão da imagem se repete fora desse intervalo

  $T(s, t) =$
  $Im\,[\,\lfloor(1 - t)\,N\rfloor \bmod N,$
  $\lfloor s\,M\rfloor \bmod M\,]$

- A função de textura pode ser também definida algebricamente:

$$T(s,t) = \sqrt{(s-0.5)^2 + (t-0.5)^2}$$



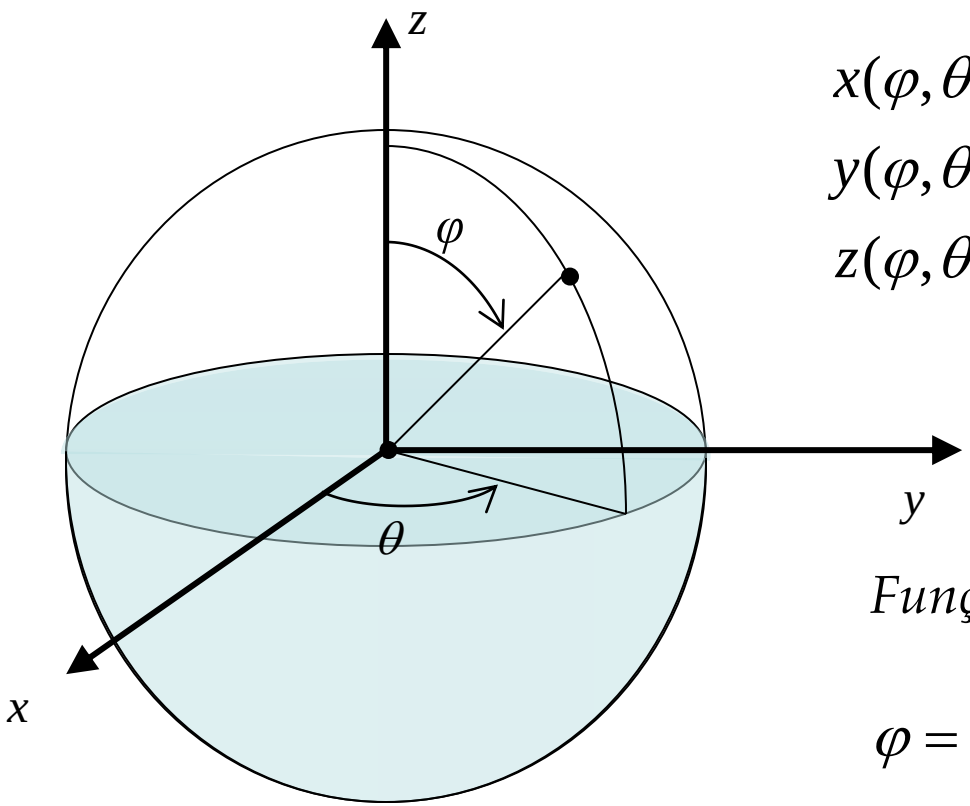
Função de Mapeamento

- Devolve o ponto do objecto correspondente a cada ponto do espaço de textura

  $(x, y, z) = F(s, t)$

- Corresponde à forma com que a textura é usada para "embrulhar" (*wrap*) o objecto
  - Na verdade, na maioria dos casos precisamos de uma função que nos permita "desembrulhar" (*unwrap*) a textura do objecto, isto é, a inversa da função de mapeamento
- Se a superfície do objecto puder ser descrita na forma paramétrica, esta pode servir como base para a função de mapeamento

Parametrização da Esfera



Função de mapeamento

$$x(\varphi,\theta) = \sin\varphi\cos\theta$$

$$y(\varphi,\theta) = \sin\varphi\sin\theta$$

$$z(\varphi,\theta) = \cos\varphi$$

$$\varphi = \pi \cdot t$$

$$\theta = 2\pi \cdot s$$

Função de mapeamento inversa

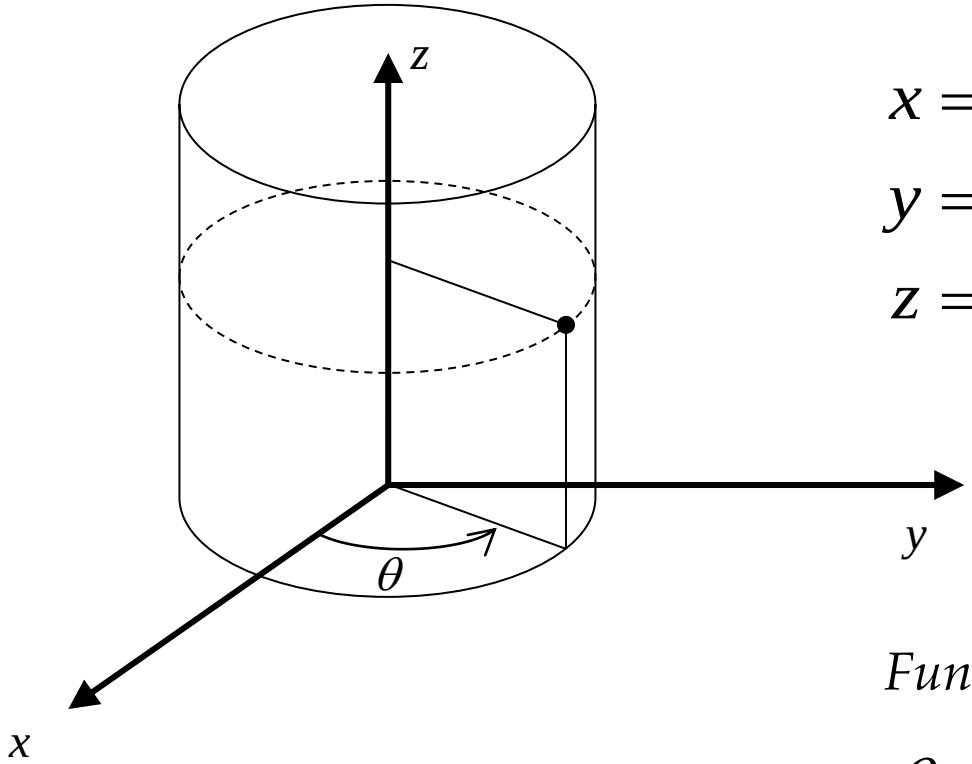
$$\varphi = \arccos z$$

$$\theta = \arctan\frac{y}{x}$$

$$t = \frac{\arccos z}{\pi}$$

$$s = \frac{\arctan\frac{y}{x}}{2\pi}$$

Parametrização do Cilindro



Função de mapeamento

$$x = \cos\theta \qquad \theta = 2\pi \cdot s$$

$$y = \sin\theta \qquad z = t$$

$$z = z$$

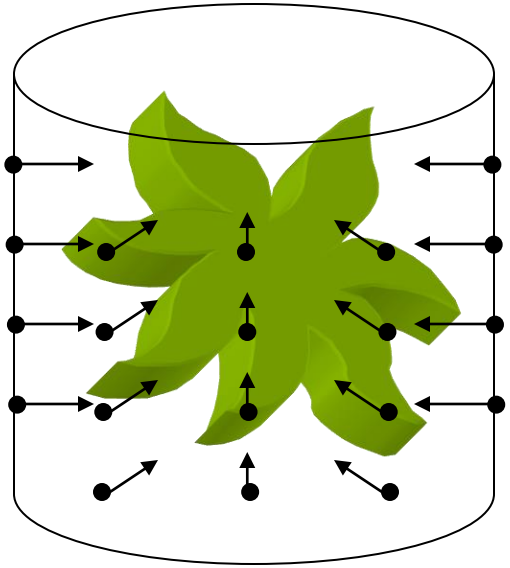
Função de mapeamento inversa

$$\theta = \arctan\frac{y}{x} \qquad s = \frac{\theta}{2\pi}$$

$$z = z \qquad t = z$$

Parametrização de Objectos Genéricos

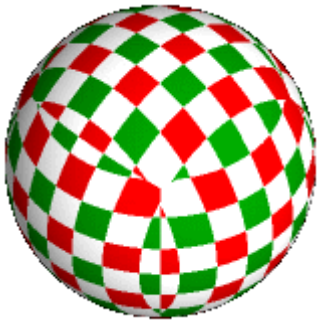
- O que fazer quando o objecto não comporta uma parametrização natural?
- Uma sugestão é usar um mapeamento em 2 estágios [Bier e Sloan]:
  - Mapear textura sobre uma superfície simples como cilindro, esfera, etc. Que englobe aproximadamente o objecto
  - Mapear superfície simples sobre a superfície do objecto. Pode ser feito de diversas maneiras
    - Raios passando pelo centróide do objecto
    - Raios normais à superfície do objecto
    - Raios normais à superfície simples
    - Raios reflectidos (*environment mapping*)



Exemplos

Parametrização cúbica

Projectada numa esfera



Projectada num cilindro

Exemplos

Parametrização cilíndrica

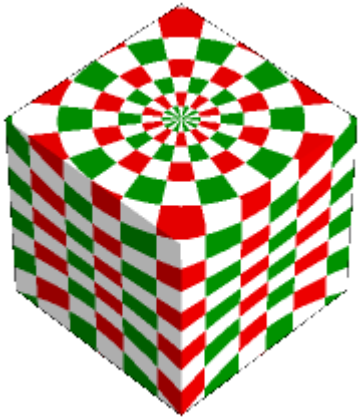
Projectada numa esfera

Projectada num cubo

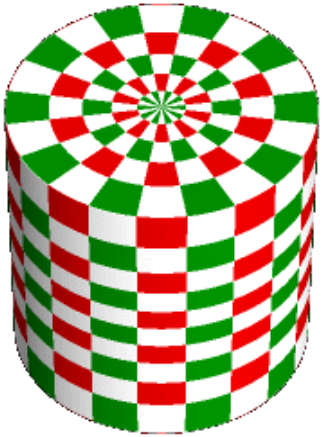
Exemplos

Parametrização esférica

Projectada num cubo

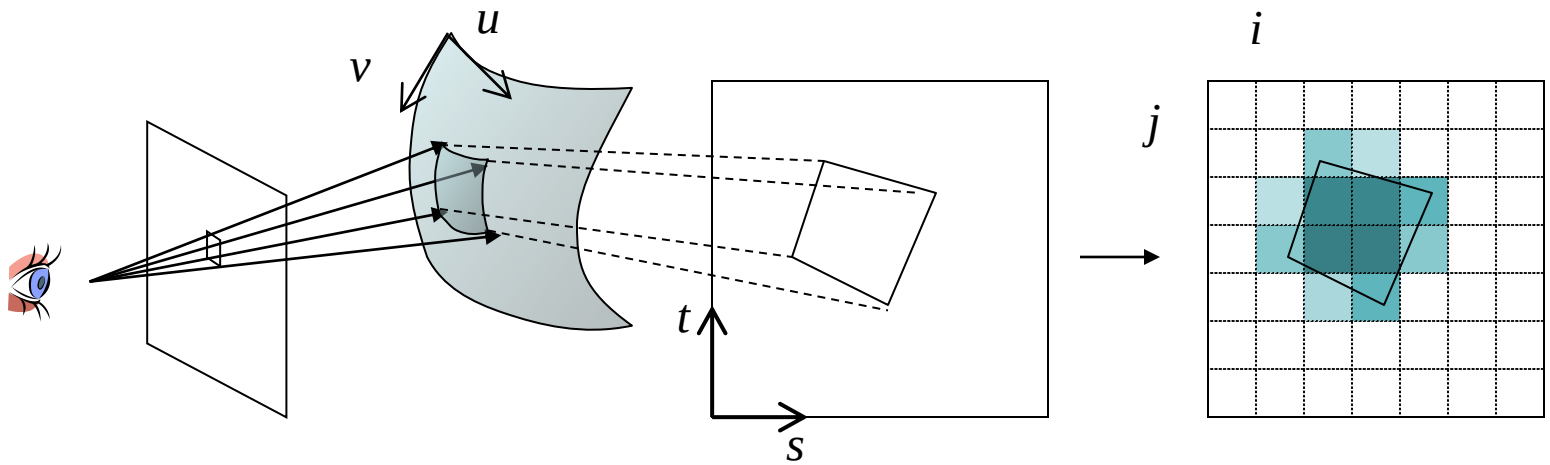


Projectada num cilindro



Processo de Mapeamento de Texturas

- Projecção do pixel sobre a superfície
  - Pontos da superfície correspondentes aos vértices do pixel
- Parametrização
  - Coordenadas paramétricas dos vértices do pixel projectados
- Mapeamento inverso
  - Coordenadas dos vértices no espaço de textura
- Média
  - Cor média dos 'Texels' proporcional à área coberta pelo quadrilátero



Mapeamento de Texturas em Polígonos

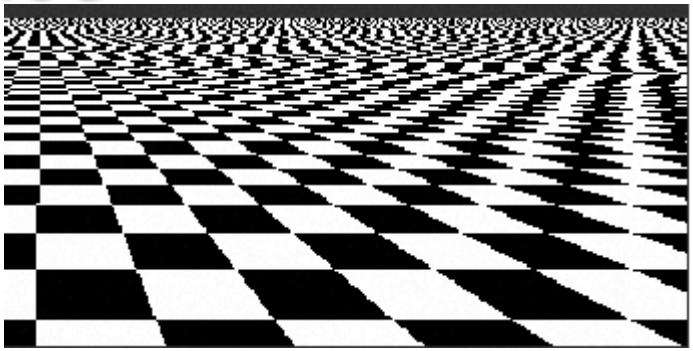
- A maneira mais simples e rápida:
  - Projectar os vértices do polígono na imagem
  - A cada vértice projectado P_i corresponde um ponto Q_i no espaço de textura
  - Um pixel P do polígono na imagem é dado por uma combinação afim. Ex.:

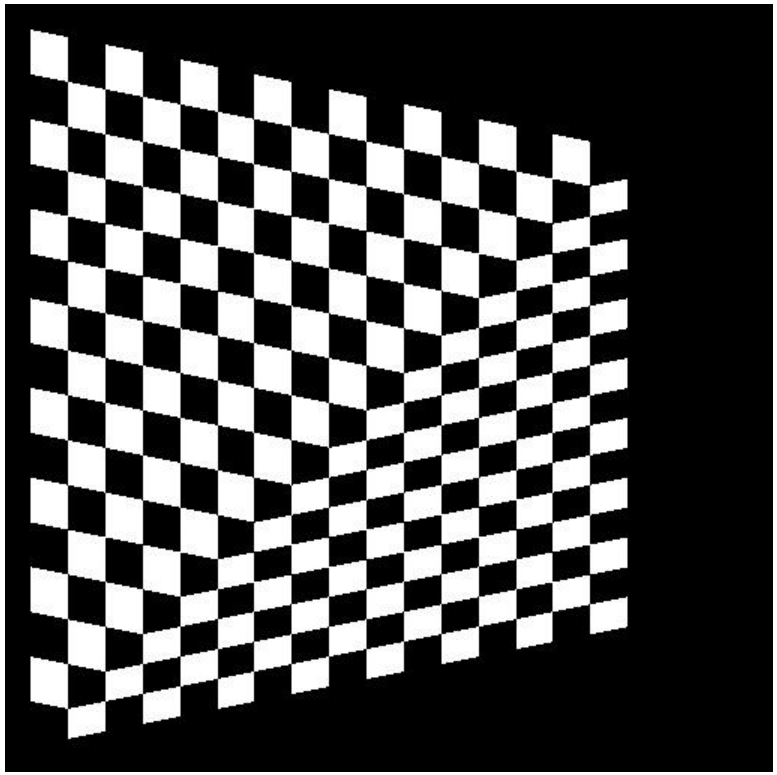
    $P = \alpha_1 P_1 + \alpha_2 P_2 + \alpha_3 P_3$
  - Pixel P é pintado com a cor do texel obtido com a mesma combinação afim. Ex.:

    $Q = \alpha_1 Q_1 + \alpha_2 Q_2 + \alpha_3 Q_3$

Mapeamento de Texturas em Polígonos

- Problemas da abordagem simples:
  - Aliasing
    - Pixel <≠> Texel
    - Soluções:
      - Interpolação
      - *Mip-mapping*
  - Deformação
    - Combinações afim não são preservadas pelas projecções perspectivas
    - Soluções:
      - Mais vértices
      - Coordenadas homogéneas

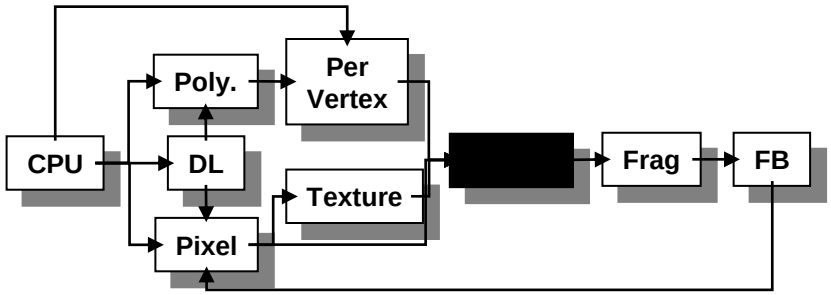




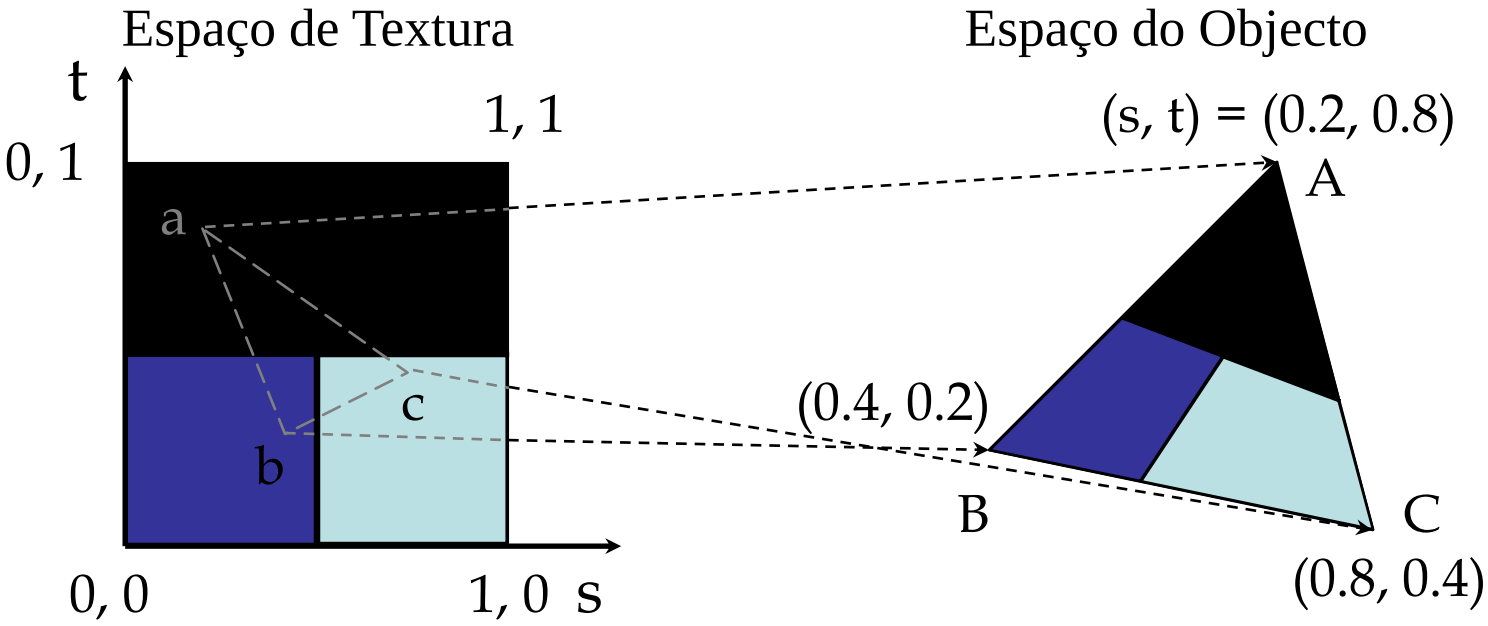
Mapeamento de Texturas

3. Configurar diversos parâmetros
   - Modos de filtragem
     - Magnificação ou minificação
     - Filtros mipmap de minificação
   - Modos de repetição de padrões
     - Cortar ou repetir
   - Funções de aplicação de textura
     - Como misturar a cor do objecto com a da textura
       - Misturar, modular ou substituir texels
4. Especificar coordenadas de textura
   - Por vértice
   - Coordenadas calculadas automaticamente

Mapeamento da Textura

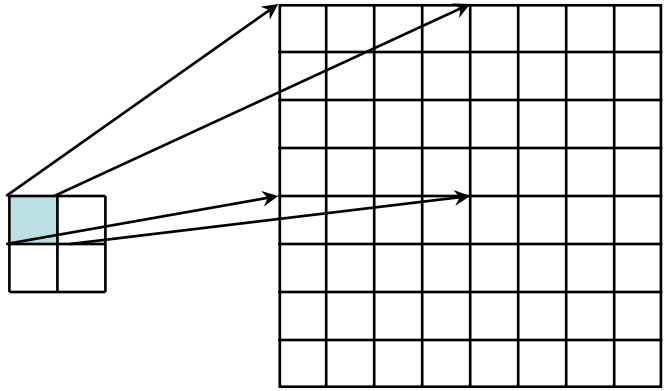


- Baseado em coordenadas paramétricas de textura



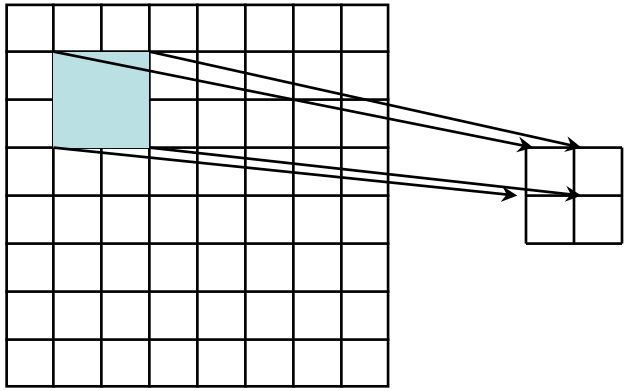
Filtragem

NEAREST
LINEAR
NEAREST_MIPMAP_NEAREST
NEAREST_MIPMAP_LINEAR
LINEAR_MIPMAP_NEAREST
LINEAR_MIPMAP_LINEAR



Textura Polígono

Magnificação

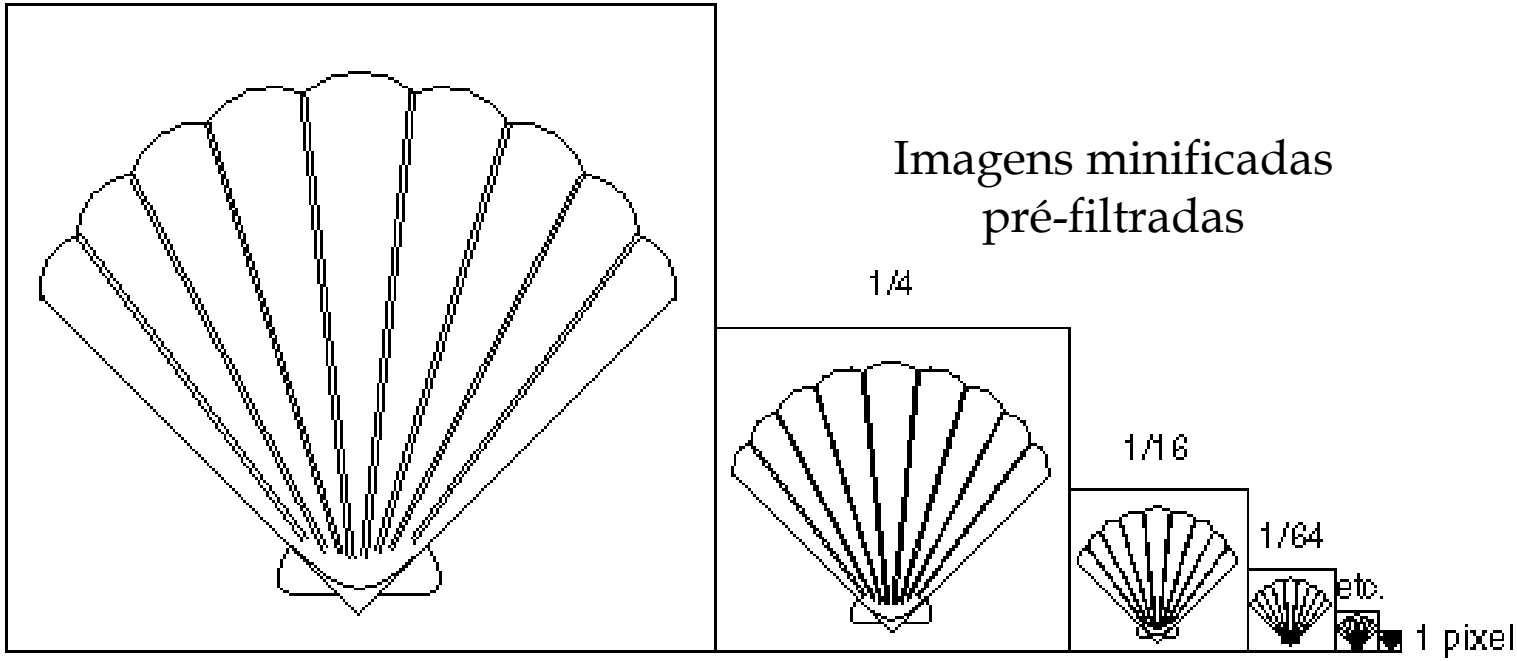


Textura Polígono

Minificação

Texturas Mipmap

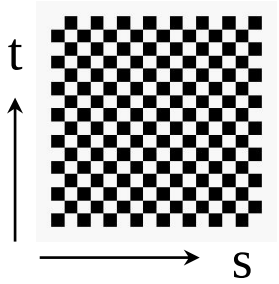
Textura original



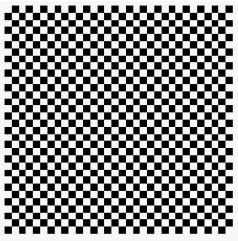
Texturas Mipmap

- Permite que texturas de diferentes níveis de resolução sejam aplicadas de forma adaptativa
- Reduz discretização devido a problemas de interpolação
- O nível da textura na hierarquia mipmap pode ser especificado durante a definição da textura
- Construção automática de texturas mipmap com filtragem adequada

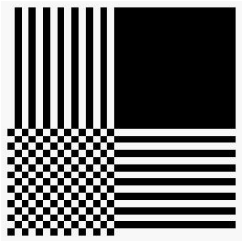
Modos de Repetição



textura



REPEAT



CLAMP

Outras Facilidades

- Objectos de Textura (Texture Objects)
  - Permite mudar rapidamente de texturas durante a renderização de diversos objectos
- Controlo de espaço na memória de texturas
  - Texturas residentes na placa são mais rápidas
- Multitexturas
  - Mais de uma textura mapeada no mesmo objecto
  - Permite uma série de efeitos interessantes
    - *Shadow mapping*
    - *Bump mapping*