

Resumos CGRA

Transformações Geométricas 2D	3
Translação	3
Escalamento	3
Rotação	4
Composição/Concatenação de Transformações	4
Transformações relativas a um ponto arbitrário (pivot)	4
Rotação	4
Escalamento	5
Outras transformações	5
Transformações Inversas	5
Transformações Geométricas 3D	5
Translação	5
Escalamento	6
Rotação	6
Modelos de Iluminação	7
Modelo Elementar	7
Iluminação ambiente	7
Reflexão Difusa	8
Reflexão especular	8
Atenuação com a distância	9
Atenuação Atmosférica	9
Dicas para resolução de exercícios	10
Projeção de Sombras	11
Atherton & Weiller	11
Ray Casting	11
Volume de Sombras (BSP)	11
Z-Buffer	11
Sombreamento (Shading)	12
Sombreamento constante	12
Smooth Shading	12
Método de Gourard	12
Método de Phong	13
Texturas	15
Mapeamento de Texturas	15
Bump Mapping	15
Texturas 3D	16

Cor	16
Visibilidade	25
Back Face Culling	25
Algoritmos no espaço objeto	26
Algoritmo de Roberts	27
Algoritmo de Appel, Loutrel, Galimberti e Montanari:	27
Algoritmo de Atherthon Weiller	27
Algoritmos no espaço imagem	27
Algoritmo de Warnock	28
Algoritmo de Linha de Varrimento	28
Algoritmo Z-Buffer	28
Algoritmos tipo lista de prioridades	29
Algoritmo Newel, Newel & Sancha(Depth-sort algorithm):	29
Iluminação Global	30
Ray Tracing	31
Diminuição do número de raios a processar	32
Diminuição do número de interseções a testar	32
Radiosity	33
Curvas e Superfícies	37
Modelação de Sólidos	49
Rasterização de Linhas	55
Algoritmo Midpoint	56
Algoritmo Midpoint para desenho de circunferências	59
Rasterização de Regiões	61
Algoritmos de Preenchimento de Regiões	61
Limitado ao interior de região	62
Preenchimento segundo contorno existente	62
Preenchimento por varrimento segundo descrição de contorno	64
Perguntas Teóricas	65

Transformações Geométricas 2D

Translação

$$\begin{bmatrix} x_T \\ y_T \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$
$$\begin{bmatrix} x_T \\ y_T \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Matriz de Translação

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

Escalamento

$$\begin{bmatrix} x_T \\ y_T \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Matriz de Escalamento

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = S(s_x, s_y)$$

Factor de escala:

- >1 aumenta o objecto.
- <1 diminui o objecto.
- $s_x=s_y$ factor de escala uniforme -> não distorce o objeto.

Rotação

$$\begin{bmatrix} x_R \\ y_R \end{bmatrix} = \begin{bmatrix} \cos(b) & -\sin(b) \\ \sin(b) & \cos(b) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Matriz de Rotação

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

b positivo no sentido contrário ao movimento dos ponteiros do relógio.

Composição/Concatenação de Transformações

A escrita das transformações é feita pela ordem inversa

Em coordenadas homogêneas um objecto de n dimensões é representado num espaço a n+1 dimensões.

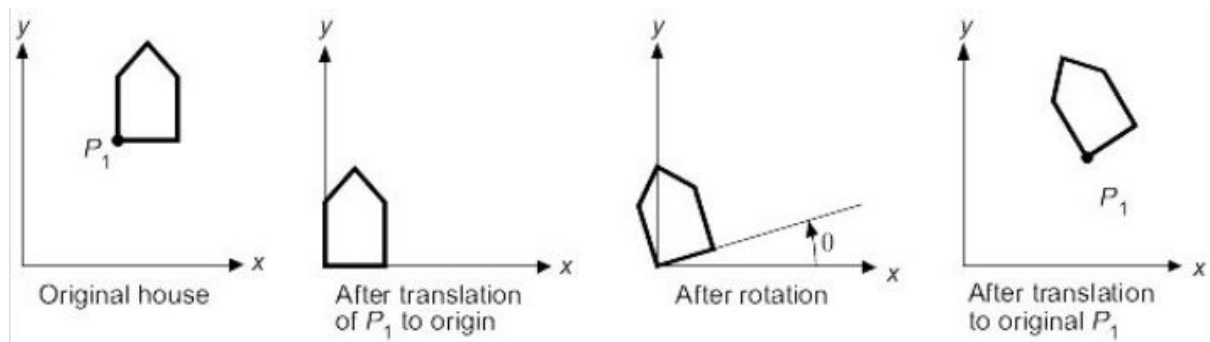
Coordenadas homogêneas ajudam a combinação/multiplicação de matrizes, visto que têm todas a mesma dimensão. Em geral, produto de matrizes é não comutativo, por exemplo:

$$T(T_x, T_y) \cdot R(a) \neq R(a) T(T_x, T_y)$$

Transformações relativas a um ponto arbitrário (pivot)

Rotação

- Fazer a translação do objeto de modo que o ponto pivot coincida com a origem
- Rodar o objecto em torno da origem
- Fazer a translação do objeto de modo que o ponto pivot volte à posição inicial (inversa da primeira).



Escalamento

- Fazer a translação do objecto de modo que o ponto pivot coincida com a origem.
- Escalar o objecto.
- Fazer a translação o objecto de modo que o ponto pivot volte à posição inicial (inversa da primeira).

Outras transformações

Reflexão

Em relação ao eixo dos X: Equivale a um escalamento $S(1,-1)$.

Em relação ao eixo dos Y: Equivale a um escalamento $S(-1,1)$.

Em relação à linha $y = x$: $R(45).S(1, -1).R(-45)$.

Em relação à linha $y = -x$: $R(45).S(-1,1).R(-45)$.

Transformações Inversas

Se uma transformação, possivelmente composta, é dada por uma matriz M de dimensões 3×3 , então a transformação inversa que coloca o objecto na sua posição inicial (ou seja sem transformação) é dada por M^{-1} .

Transformações Geométricas 3D

Translação

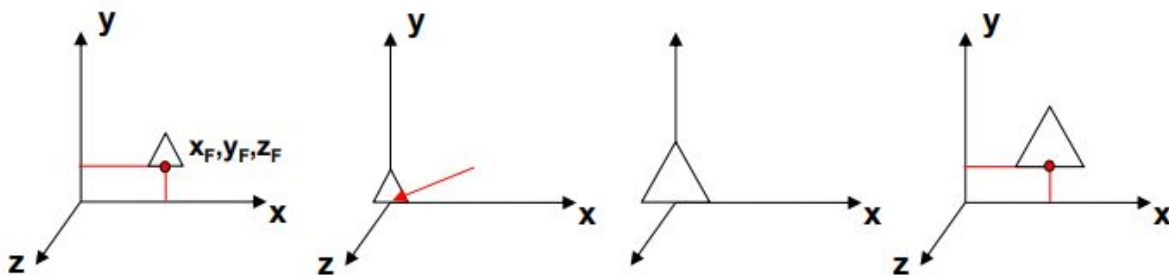
A Translação de um Objecto é efectuada aplicando a operação a cada um dos seus vértices.

$$\begin{bmatrix} x_T \\ y_T \\ z_T \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Escalamento

$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Em relação a um ponto arbitrário



$$T(x_F, y_F, z_F) \cdot S(s_x, s_y, s_z) \cdot T(-x_F, -y_F, -z_F)$$

Rotação

Eixo dos X

$$\begin{bmatrix} x_{Rx} \\ y_{Rx} \\ z_{Rx} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) & 0 \\ 0 & \sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Eixo dos Y

$$\begin{bmatrix} x_{Ry} \\ y_{Ry} \\ z_{Ry} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(a) & 0 & \sin(a) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(a) & 0 & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Eixo dos Z

$$\begin{bmatrix} x_{Rz} \\ y_{Rz} \\ z_{Rz} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(a) & -\sin(a) & 0 & 0 \\ \sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotação em torno de um eixo colocado arbitrariamente no espaço 3D

1. Aplicar a translação que coloque o eixo de rotação a passar pela origem do sistema de coordenadas.
2. Rodar o objecto de modo a que o eixo de rotação coincida com um dos eixos de coordenadas.
3. Aplicar a rotação pretendida sobre esse eixo.
4. Aplicar a rotação inversa do ponto 2.
5. Aplicar a translação inversa de 1.

Modelos de Iluminação

Modelo Elementar

Iluminação ambiente

$$I = k_a \cdot I_a$$

k_a : coeficiente de reflexão ambiente (difusa) da face

I: intensidade observada

Reflexão Difusa

$$I = \frac{k_d \cdot I_p}{d + d_0} \cos(\theta)$$

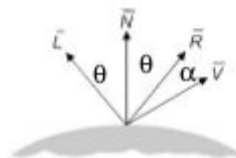


N: normal à superfície no ponto de reflexão

L: vetor na direção da fonte de luz

d: distância à fonte de luz

Reflexão especular



R: direção de reflexão máxima

alfa: ângulo entre R e a direção do observador, V.

$$I_s = \frac{k_s \cdot I_p}{d + d_0} \cos^n(\alpha)$$

$$I_s = \frac{k_s \cdot I_p}{d + d_0} (\vec{V} \cdot \vec{R})^n$$

Iluminação total = Ambiente + Difusa + Especular

Normalmente, quando nos exercícios falam de ignorar qualquer tipo de atenuação, isso significa ignorar o denominador nas fórmulas ($d + d_0$). Senão, com atenuação linear, usar d , e com atenuação quadrática, usar d^2 .

k_a , k_d e k_s estão sempre entre 0 e 1.

Atenuação com a distância

Fator de atenuação, provoca a diminuição da iluminação com a distância da fonte de luz ao ponto iluminado. k são constantes definidas pelo utilizador.

$$f_{att} = \min \left(1, \frac{1}{k_c + k_l d + k_q d^2} \right)$$

$$I = k_a I_a + f_{att} \cdot [k_d (N \cdot L_{ls}) + k_s \cdot (V \cdot R_{ls})^n] \cdot I_{ls}$$

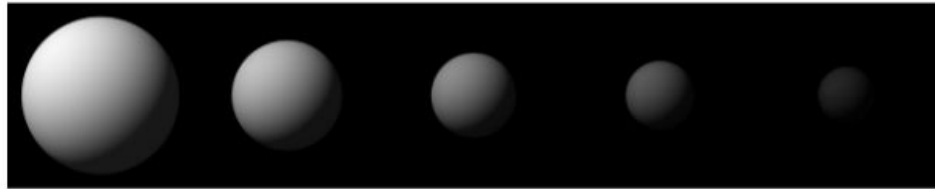
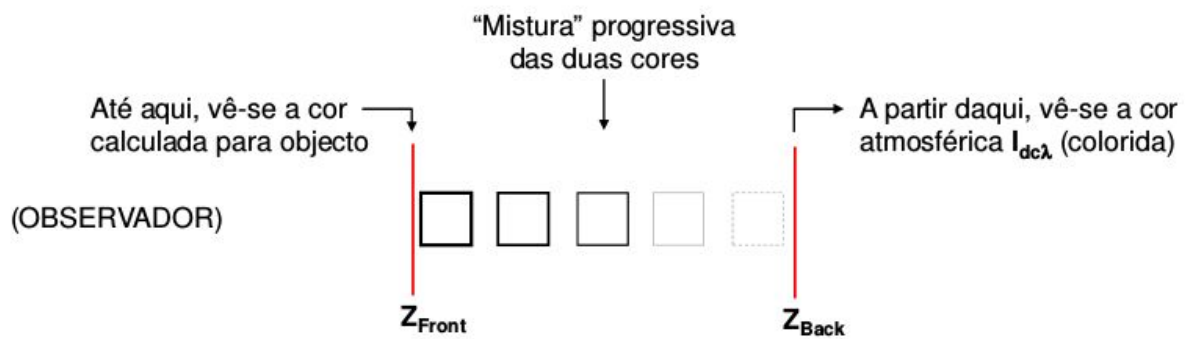
O factor $\frac{1}{d^2}$ (quadrático) não funcionaria bem. Para fontes de luz muito distantes este factor não varia suficientemente. Se a fonte estiver próxima, resultam variações muito acentuadas entre objectos semelhantes.

Atenuação Atmosférica

Com o aumentar da distância do objeto ao observador, a cor vista altera-se. Na coordenada Z mais próxima, a cor é a calculada para o objeto. À medida que se afasta, acontece uma mistura progressiva com a cor atmosférica (resultando maioritariamente num tom mais negro).

Em teoria os objetos deviam escurecer com o quadrado da distância, no entanto quando aplicamos isso no virtual a cena fica muito escura, pois o mundo virtual não tem em conta as imensas reflexões que acontecem entre os objetos.

Assim, muitas vezes acaba-se por usar uma variação linear que faz com que os objetos não escureçam tão rapidamente, contrariando o facto de não haver reflexões, e cria assim uma aparência mais “natural” na cena.



$$I'_{\lambda} = s_0 I_{\lambda} + (1 - s_0) I_{dc\lambda}$$

$$s_0 = s_b + \frac{(z_0 - z_b)(s_f - s_b)}{z_f - z_b}$$

Dicas para resolução de exercícios

Anotar as constantes fornecidas (fazem parte das fórmulas de iluminação)

Fazer esboço da cena dada, colocando:

- as normais aos pontos
- o vetor em direção à fonte de luz
- o vetor em direção ao observador
- o vetor de reflexão máxima

Normalmente uma das perguntas pede para ignorar projeção de sombras e/ou atenuação com distância, portanto é preferível seguir as fórmulas do que propriamente a lógica.

Técnica para achar hipotenusa de triângulo com catetos iguais

Seja o valor dos catetos c , a hipotenusa é $c\sqrt{2}$

Projeção de Sombras

Cálculo de Visibilidade e projeção de sombras são semelhantes, tanto que alguns algoritmos criados para resolver o problema da visibilidade são usados para o segundo (por exemplo, Atherton & Weiller). Comparando:

No primeiro caso, temos a posição do observador, os locais (ou parte deles) que ele consegue observar, e aqueles que estão 'invisíveis'. Se quisermos adaptar isso para a projeção de sombras, basta substituir o observador por uma fonte de luz, e admitir que as zonas não atingidas pela luz, se encontram em sombra.

Atherton & Weiller

Considerar posição da fonte de luz e determinar as partes iluminadas. Depois, classificar as partes em iluminadas, ou em sombra.

Determinar as partes visíveis em relação ao observador. As partes visíveis e iluminadas são desenhadas iluminadas e as restantes como sombra.

É possível utilizar algoritmo numa cena com várias fontes de luz, aplicando-o para cada uma delas, sequencialmente.

Ray Casting

Emite-se um raio luminoso a partir do ponto de observação, através do centro de um pixel para 'dentro' da cena. O ponto de intercepção entre o raio e o objecto mais próximo define o objecto visível nesse pixel.

sombra: emitir novo raio a partir do ponto de intercepção para a fonte de luz. Se interceptar algum objecto então esse ponto está na sombra.

Volume de Sombras (BSP)

Partindo de uma fonte de luz, cria-se uma pirâmide de sombra por cada polígono encontrado em cena. Posteriormente, qualquer objecto ou parte dele que ali se encontre é declarado como estando em sombra.

A modelação BSP-Binary Space Partition é especialmente adequada à representação dos volumes de sombra (limitação por planos).

Z-Buffer

O algoritmo de cálculo de visibilidade Z-Buffer pode ser utilizado, a dois passos, para o cálculo de projecção de sombras.

Sombreamento (*Shading*)

Sombreamento constante

A cor é calculada apenas para um ponto do polígono e replicada em todos os pontos restantes do mesmo polígono. Ou seja, a totalidade do polígono terá a mesma cor.

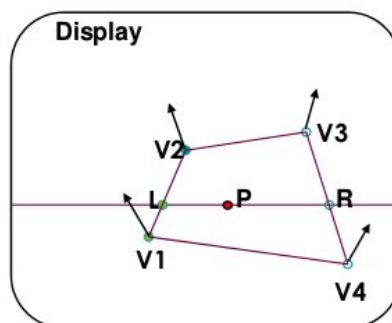
Nota-se a malha poligonal devido ao Efeito de *Mach Band*, com descontinuidade da própria função de iluminação

Smooth Shading

Método de Gourard

Calcular a cor de cada vértice através o modelo de iluminação pretendido

Calcular a cor dos restantes pontos do polígono por interpolação bi-linear - primeiro vertical, depois horizontal.

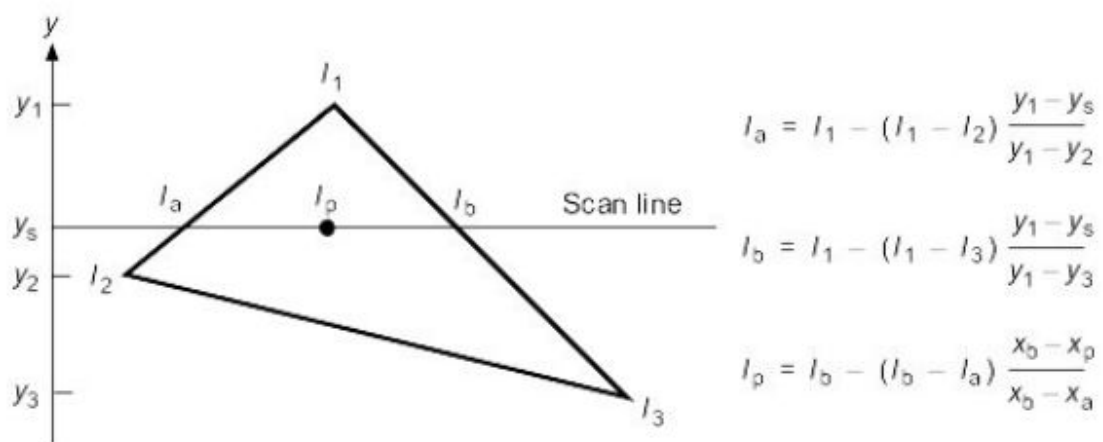


1. Cor do ponto L é obtida por interpolação da cor em V1 e V2
2. R = interpolação de V3 e V4
3. P = interpolação de L e R

Nota-se a localização das arestas, mesmo em polígonos que deviam ser curvos, devido ao efeito de Mach Band, com descontinuidade da derivada da função de iluminação



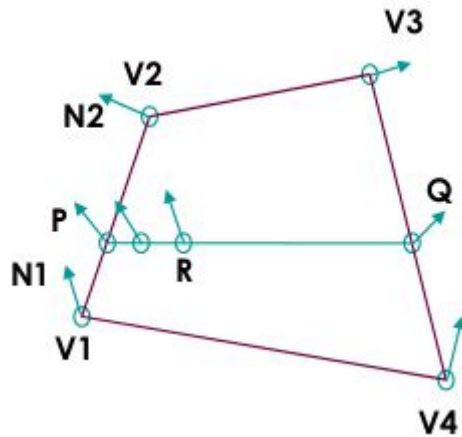
Calculo dos valores interpolados



Método de Phong

Calcular vetores normais aos vértices da malha poligonal.

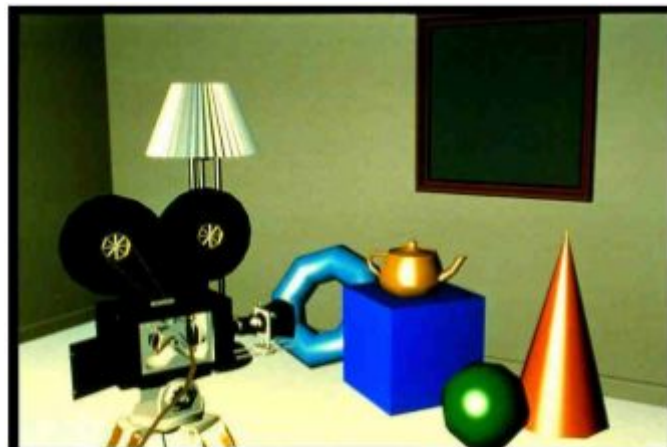
As normais nas arestas são calculadas por interpolação linear das normais nos vértices. As normais ao longo dos pontos de uma linha de varrimento obtêm-se por interpolação linear das normais nas arestas.



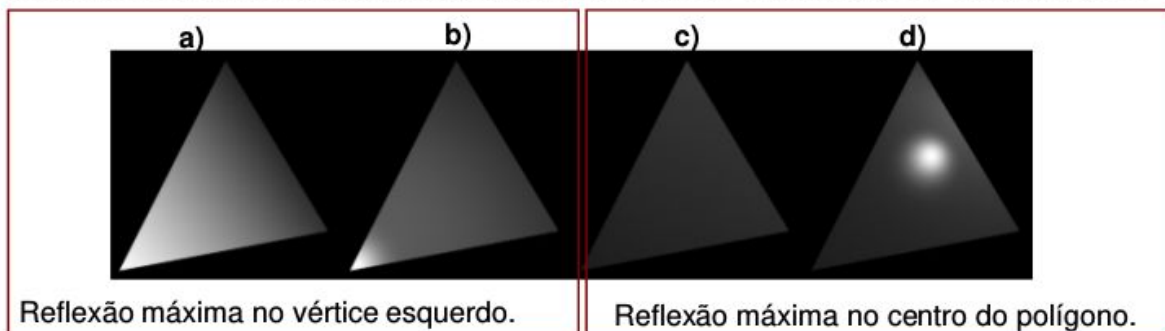
1. Normal em P obtida por interpolação das normais em V1 e em V2.
2. Normal em Q = interpolação de V3 e V4
3. Normal em R = interpolação de normais em P e Q

High-lights bem colocados

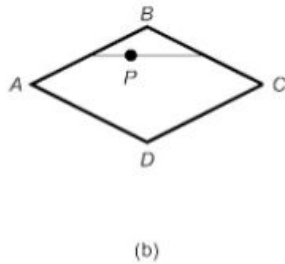
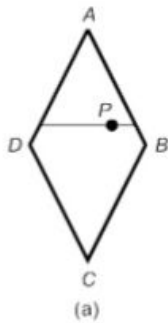
Efeito de Mach Band, não é notório



Reflexão especular com sombreamento pelo modelo de Gouraud a) e c) e Phong b) e d)

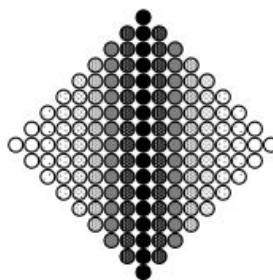
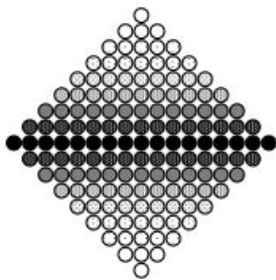


Problema do Sombreamento Interpolado

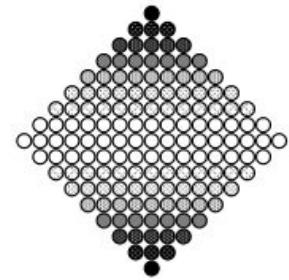


O resultado depende da orientação do polígono:

- Em (a) o cálculo de P usa as cores dos vértices A,D,B.
- Em (b) o cálculo de P usa as cores dos vértices A,B,C.



Rotação de 90°



Resultado

Texturas

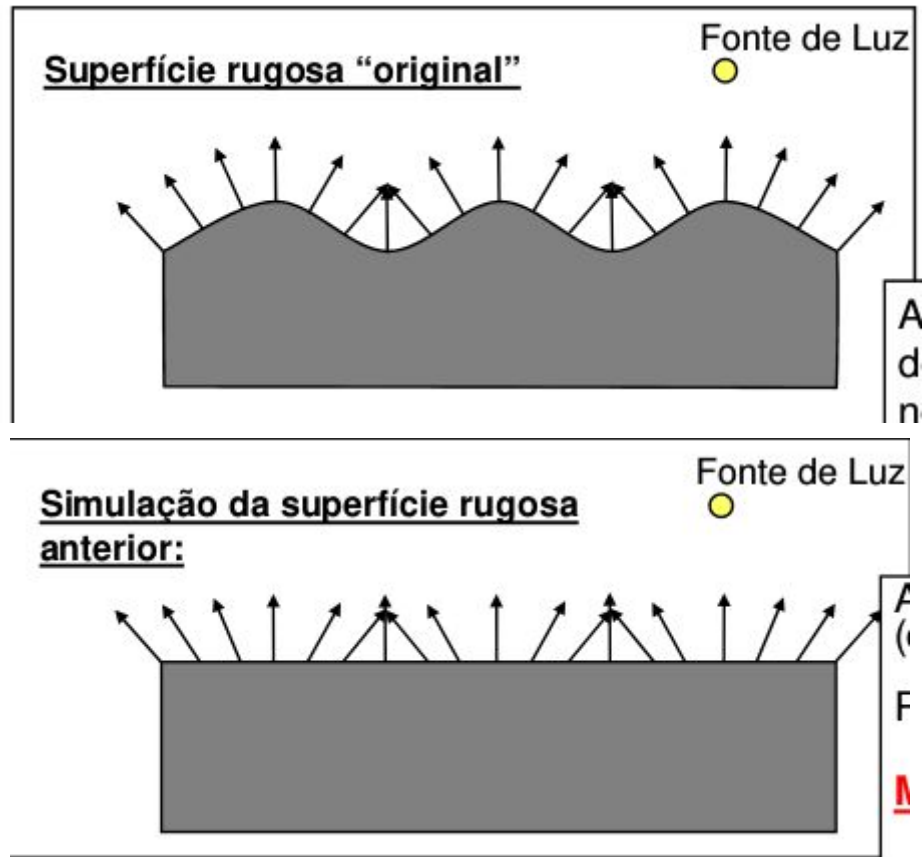
Mapeamento de Texturas

Uma textura é uma imagem com coordenadas normalizadas $(u,v) \in [0,1]$. Podem ser 'coladas' nos polígonos nos objetos, de várias formas (repetição, entre outras).

Bump Mapping

Além de mapear uma imagem, simular rugosidade mas sem alterar geometria dos objetos.

Para isso, estudar a geometria de uma superfície rugosa que se quer simular. Depois, afetar a direção das normais na nossa superfície para obter um resultado semelhante. Como as normais são diferentes, então o cálculo da iluminação é diferente.



Texturas 3D

A textura evolui continuamente dentro do objeto, ou seja: se pegarmos num objeto com uma textura 3D e o cortarmos ao meio, ficamos com dois objetos que aparentam ser completamente normais, sem quaisquer falhas de textura.

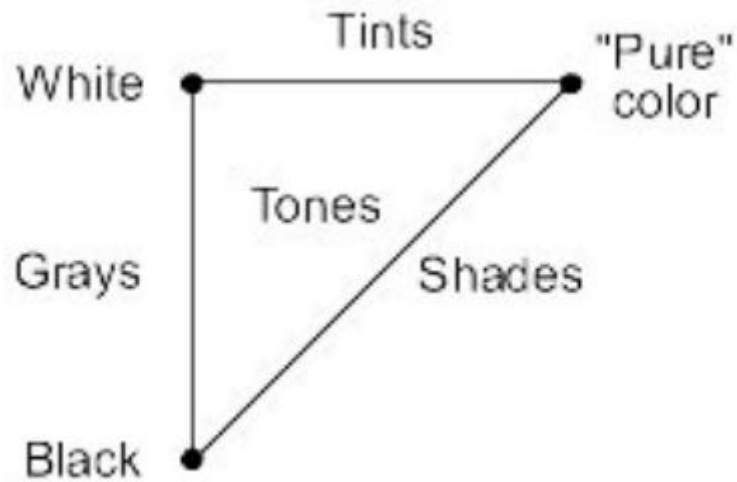
Texturas 3D e *bump mapping* têm propriedades muito diferentes pelo que não se podem substituir umas às outras.

Cor

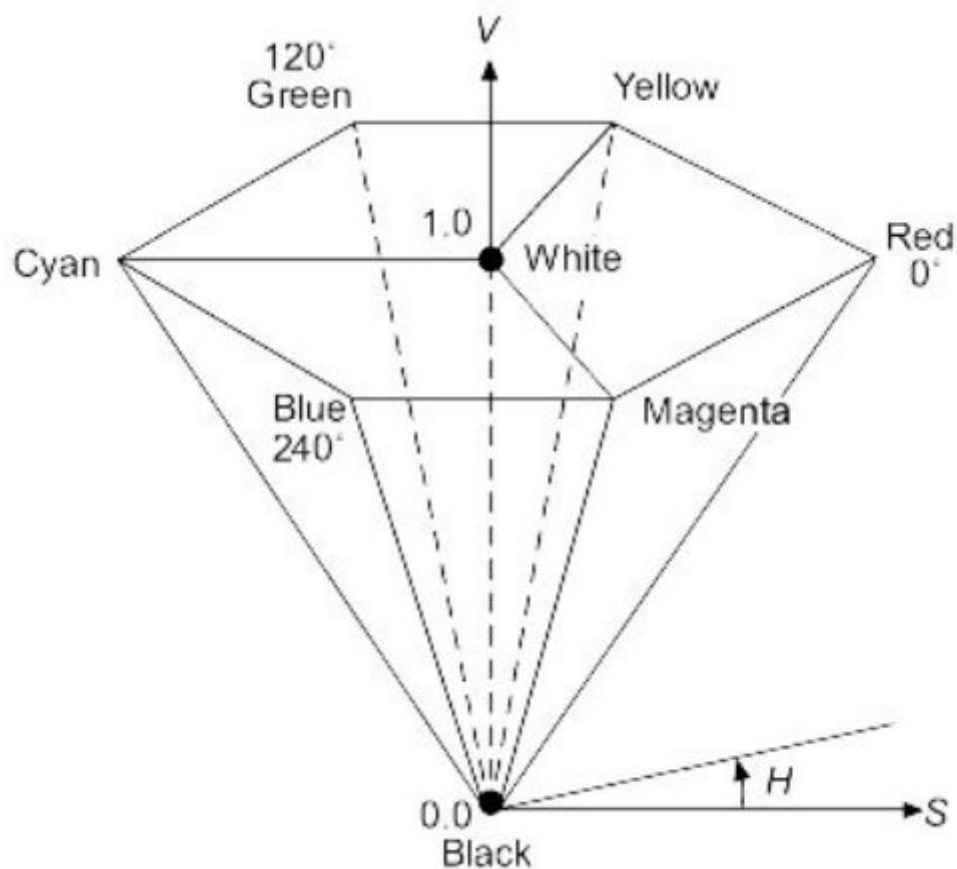
Luz Cromática

Características:

- Matiz (distinção entre verde, vermelho, amarelo, etc.)
- Saturação (distância da cor ao cinzento de igual intensidade – a saturação diminui com a adição de branco)
- Intensidade (reflexão)
- Brilho (emissão – válido para fontes de luz)



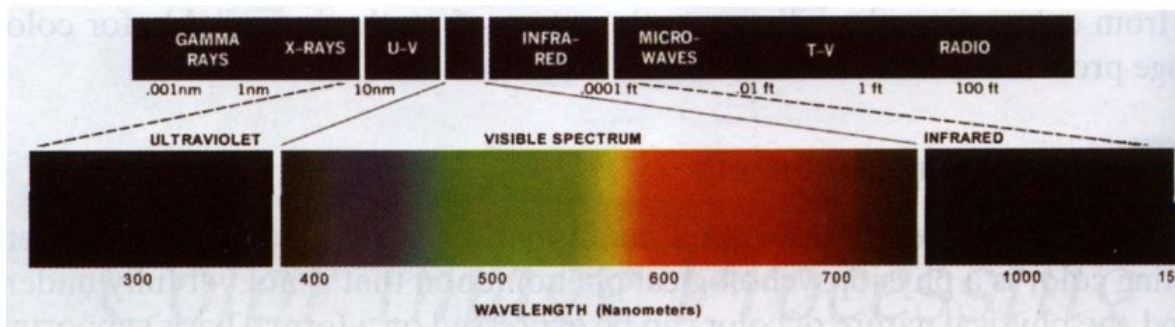
Para a mesma Matiz (Hue), podemos obter diferentes tons com a adição de branco (Tintos), preto (Sombreados) ou branco e preto (tons). Estas cores têm a mesma Matiz, mas diferentes intensidades e saturações.



Modelo HSV: generalização do modelo anterior para várias Matizes

Todas estas definições são subjetivas. Objetivamente, a cor pode ser quantificada através da sua frequência (espectro eletromagnético)

- Frequência visível mais baixa: Vermelho com $\lambda = 700 \text{ nm}$
- Frequência visível mais alta: Violeta com $\lambda = 400 \text{ nm}$
- Frequência e comprimento de onda: $c = \lambda \cdot f$
onde c é a velocidade da luz ($3 \cdot 10^{10} \text{ cm/s}$)

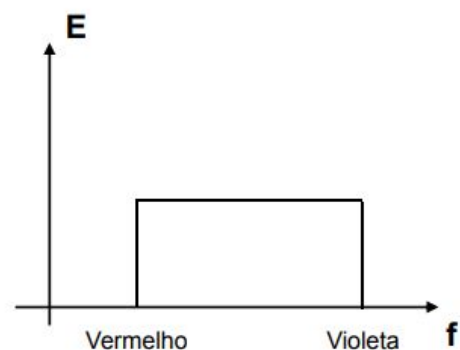


Colorimetria: ramo da física que estuda a cor

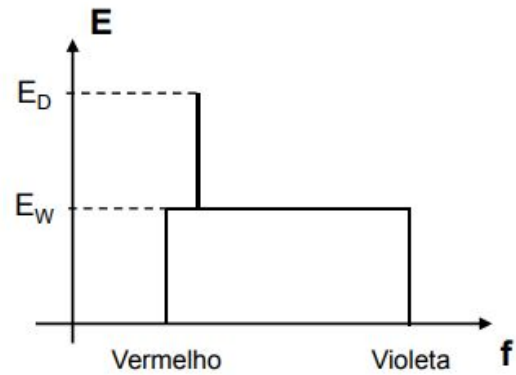
- Comprimento de onda dominante: cor dominante, Matiz (Hue)
- Luminância: intensidade da luz (Brightness)
- Excitation purity: Saturação

Distribuição de Energia de uma fonte de luz branca:

Todas as frequências estão igualmente presentes.



Distribuição de Energia de uma fonte de luz com comprimento de onda dominante perto do vermelho:



E_D – Energia dominante

E_W – Energia para o branco

Quanto maior $E_D - E_W$, tanto mais pura será a cor emitida.

$E_W = 0$, pureza 100%

$E_W = E_D$, pureza 0% (branco)

$$Sat = \frac{E_D - E_W}{E_D}$$

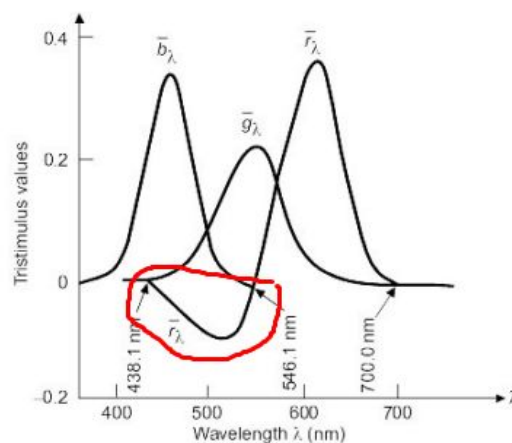
Luminância = área abaixo da curva da energia total emitida

Olho Humano:

- Bastonetes (rods): são sensíveis a baixos níveis de iluminação e não distinguem cor
- Cones: localizados principalmente na fóvea interpretam a cor; são pouco sensíveis com pouca luz.

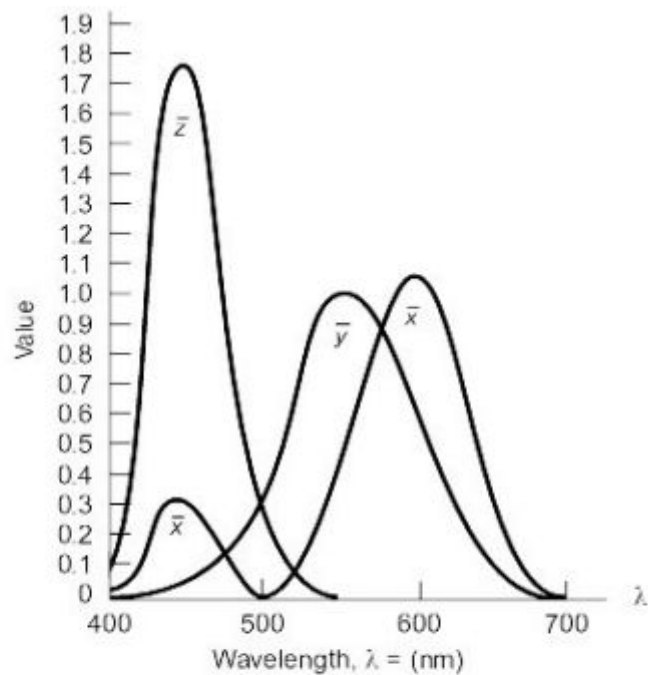
Teoria do tri-estímulo

- Três tipos de cones, cada qual sensível a um tipo diferente de luz (Azul, Verde e Vermelho)
- O olho humano responde menos a estímulos Azuis
- Nem todas as cores podem ser representadas como combinação de R, G, B



Rodeado na imagem: cores que necessitam de quantidades negativas de uma das cores primárias para serem formadas – impossível só com RGB

Resposta ao problema: Modelo CIE – Definição de três cores primárias imaginárias X, Y e Z para substituir RGB.



$$C = X\bar{X} + Y\bar{Y} + Z\bar{Z}$$

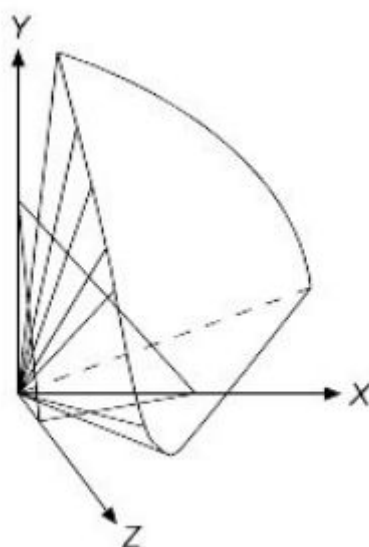
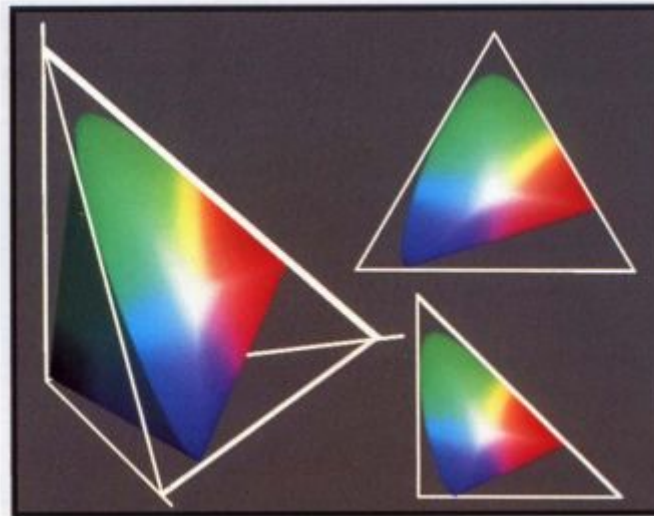
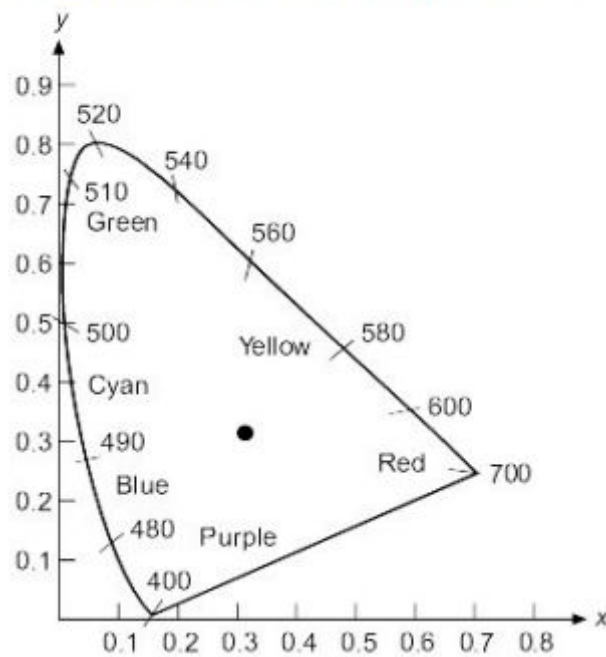


Diagrama das cores visíveis



Projeção do plano $X+Y+Z=1$ – Diagrama de Cromatância CIE

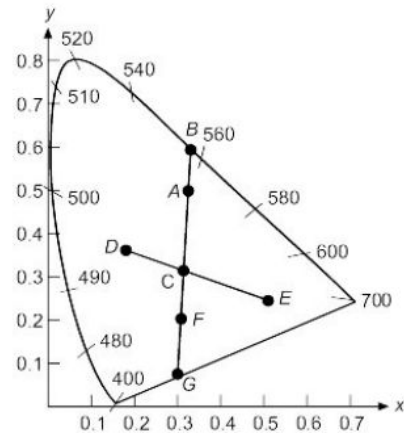
Diagrama de cromatância CIE



Cores puras estão nas fronteiras
Branco está no centro

Determinação do comprimento de onda dominante:

- Somando duas cores, a cor resultante encontra-se sobre o segmento que une as duas cores adicionadas.
- A cor **A** pode ser vista como **C + B**, logo **B** é o comprimento de onda dominante.



Determinação da pureza da cor:

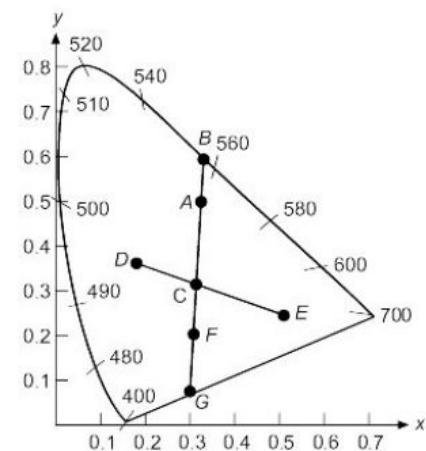
- AC / BC expressa a pureza (em percentagem) da cor **A**.
- Quanto mais perto estiver **A** de **C**, mais luz branca estará incluída em **A** e menos pura (menos saturada) será esta cor.

Cores complementares:

- São duas cores que somadas originam a cor branca
- Ex: Cores **D** e **E**.

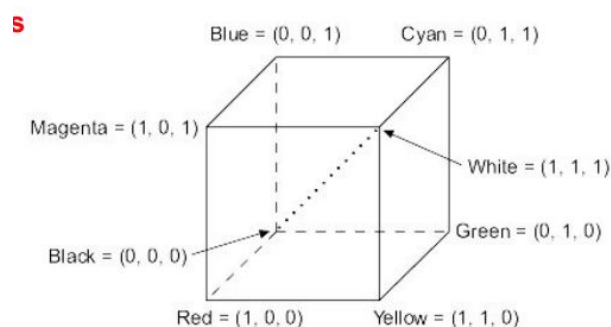
Cores não espectrais:

- Cores que não podem ser definidas por um comprimento de onda dominante. Ex: cor **F**.
- Neste caso, o comprimento de onda dominante é definido como o complementar do comprimento de onda onde a recta que passa por **F** e **C** intersecta em **B** (no exemplo 555 nm).
- A pureza /saturação da cor é definida por FC / GC .
- As cores não espectrais são as púrpuras e magentas



Outros Modelos:

Modelo RGB



(a diagonal representa os cinzentos)

Modelo CYM

Ciano, Magenta, Amarelo como cores primárias (cores complementares de RGB)
Cubo semelhante ao RGB, mas com branco na origem

Modelo HSV (imagem acima):

- $V=1$ – cores mais brilhantes
- Ângulo H corresponde às Matizes (0° = vermelho, 120° = verde, etc.)
- Saturação: 0 (centro) a 1 (periferia)

Conversões

Transformação RGB para CMY:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Transformação CMY para RGB:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

$$\text{Saturação} = (\text{max} - \text{min}) / \text{max}$$

RGB to HSV conversion formula

The R, G, B values are divided by 255 to change the range from 0..255 to 0..1:

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

$$C_{max} = \max(R', G', B')$$

$$C_{min} = \min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

Hue calculation:

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G' - B'}{\Delta} \bmod 6 \right) & , C_{max} = R' \\ 60^\circ \times \left(\frac{B' - R'}{\Delta} + 2 \right) & , C_{max} = G' \\ 60^\circ \times \left(\frac{R' - G'}{\Delta} + 4 \right) & , C_{max} = B' \end{cases}$$

Saturation calculation:

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

Value calculation:

$$V = C_{max}$$

HSV to RGB conversion formula

When $0 \leq H < 360$, $0 \leq S \leq 1$ and $0 \leq V \leq 1$:

$$C = V \times S$$

$$X = C \times (1 - |(H / 60^\circ) \bmod 2 - 1|)$$

$$m = V - C$$

$$(R', G', B') = \begin{cases} (C, X, 0) & , 0^\circ \leq H < 60^\circ \\ (X, C, 0) & , 60^\circ \leq H < 120^\circ \\ (0, C, X) & , 120^\circ \leq H < 180^\circ \\ (0, X, C) & , 180^\circ \leq H < 240^\circ \\ (X, 0, C) & , 240^\circ \leq H < 300^\circ \\ (C, 0, X) & , 300^\circ \leq H < 360^\circ \end{cases}$$

$$(R, G, B) = ((R' + m) \times 255, (G' + m) \times 255, (B' + m) \times 255)$$

Visibilidade

Back Face Culling

Permite reduzir o número de polígonos a processar em termos de cálculo de iluminação.

Remove (aprox.) metade dos polígonos.

Duas formas de determinar se face é posterior ou anterior:

Sinal da área do triângulo

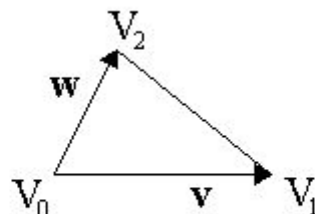


Figure 2: $\vec{v} = V_1 - V_0$, $\vec{w} = V_2 - V_0$

$$Area_{\triangle} = \frac{1}{2} \|\vec{v} \times \vec{w}\|.$$

Área é positiva se se V_0 , V_1 , e V_2 estão orientados no sentido contrário aos ponteiros do relógio, caso contrário é negativa. Neste último caso, significa que é um backface e não precisa de ser desenhado no ecrã.

Normais

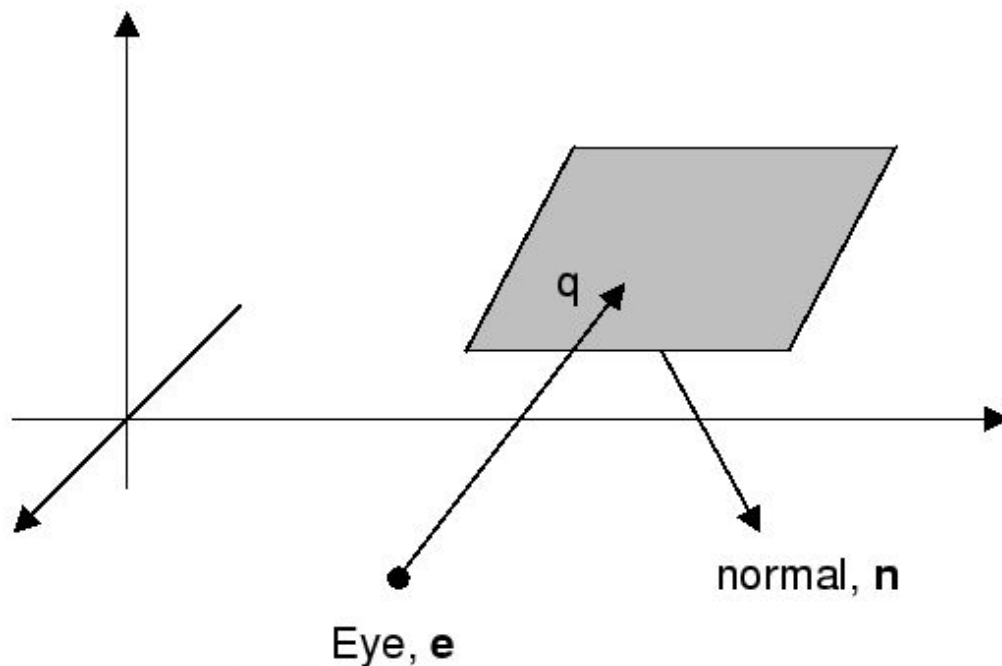


Figure 4: *Determining a Backface*

Se o ângulo entre EQ e N for maior que 90° , então a face é uma backface. Para fazer este teste, basta fazer:

$$\vec{n} \cdot (\vec{e} - \vec{q}) < 0$$

Se verdade, é backface.

Algoritmos no espaço objeto

Todas as arestas são testadas para produzir uma lista com os segmentos visíveis de todas as arestas.

Ao Volume (Algoritmo de Roberts): supõe-se que uma aresta pode ser oculta pelo volume de um objeto.

À Aresta (Algoritmo de Appel, Loutrel, Galimberti e Montanari): o teste é efetuado aresta contra aresta observando que a visibilidade de uma aresta goza de coerência, o que permite determinar a invisibilidade de uma aresta a partir da invisibilidade de outra aresta que possua com ela um vértice em comum.

Coerência de uma aresta: uma aresta só altera a sua visibilidade onde se cruza por trás de uma aresta visível.

Algoritmo de Roberts

Requisito: cada aresta deve pertencer a uma face de um **poliedro** convexo. Poliedros côncavos devem ser partidos em vários convexos para poder aplicar o algoritmo.

1. Backface Culling (Remover todas as faces posteriores dos objectos)
2. Comparar as arestas restantes do passo 1 contra cada volume (poliedro) da cena e deste teste podem ocorrer 4 situações:
 1. Aresta completamente oculta pelo volume.
 2. Aresta não oculta.
 3. Parte da aresta não é oculta.
 4. Duas partes da aresta não são ocultas.

Algoritmo de Appel, Loutrel, Galimberti e Montanari:

1. Backface Culling
2. Calcular QI (Quantitative Invisibility) de um vértice para cada objeto (QI de um ponto é o número de polígonos entre o observador e o próprio ponto), sendo este alterado quando a aresta passa por trás de uma contour line.

Nota : Uma Contour line é uma aresta partilhada por um polígono back-facing com outro front-facing.

Algoritmo de Atherthon Weiller

Requisito: exige a aplicação de um algoritmo sofisticado de clipping de polígonos, capaz de efetuar clipping de um polígono côncavo com buracos contra um outro qualquer.

Procedimento:

1. Ordenar os polígonos pela coordenada Z; o polígono mais próximo do observador é selecionado como janela de corte.
2. Cria 2 listas, uma com polígonos visíveis (lista exterior) e outra com invisíveis (lista interior).
3. Polígonos da lista interior marcados como invisíveis.
4. Passa ao polígono seguinte da lista de exteriores e regressa ao ponto 2.

Algoritmos no espaço imagem

Orientado à área: Algoritmo de Warnock

Orientado à linha: Linha de Varrimento, Watkins

Orientado ao pixel: Z-Buffer, Ray Casting

Algoritmo de Warnock

1. Divide sucessivamente a imagem projetada em áreas retangulares
2. Se uma área é coerente, desenhada com os polígonos que contém

ELSE aplica o procedimento recursivamente e a área é dividida em áreas menores.

Notas:

- Quanto menores foram as áreas menor é o número de polígonos sobrepostos nessa área, sendo assim mais fácil decidir qual o polígono a desenhar.
- O algoritmo utiliza o conceito de coerência de área: um grupo de pixels adjacentes é habitualmente coberto pela mesma face visível.

Algoritmo de Linha de Varrimento

A imagem é criada linha a linha.

Conceitos explorados:

- Coerência vertical: o conjunto de objetos visíveis determinados para uma linha de varrimento, difere pouco do conjunto correspondente da linha anterior.
- Coerência de aresta: uma aresta só altera a sua visibilidade quando se cruza com outra aresta visível ou quando penetra uma face.

Algoritmo Z-Buffer

Requisitos:

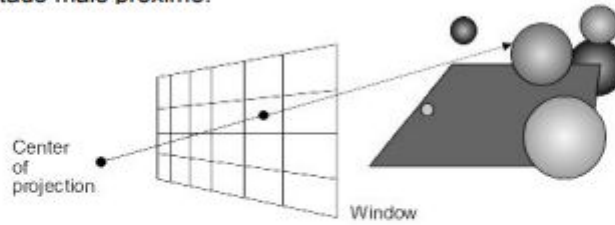
Frame Buffer: contém a imagem final, pixel a pixel

Depth Buffer / Z-Buffer: contém os valores Z, pixel a pixel.

Procedimento:

1. Preencher com zeros o Z-Buffer e o frame buffer com a cor de fundo (background). O maior valor de Z em Z-Buffer será o correspondente ao plano frontal de clipping.
2. Percorrer cada polígono (Scan-convert), por qualquer ordem.
3. Se o ponto (x,y) do polígono corrente estiver mais próximo do observador do que o ponto actual do Z-Buffer, então o corrente substitui o anterior.

A superfície visível em cada pixel da imagem é determinada traçando um raio de luz imaginário a partir do centro de projeção (observador), passando pelo centro do pixel para a cena 3D. A cor em cada pixel é definido pela cor, no ponto de interseção, do objeto intersetado mais próximo.



```

Definir Centro de Projecção e window no plano de visualização
for(cada linha da imagem)
    for(cada pixel da linha)
        {
            Definir o raio que a partir do centro de projecção passa no pixel
            for (cada objeto na cena)
            {
                if ((objeto intersectado) e (mais próximo do que registo anterior))
                    registar interceção e referência do objeto
            }
            atribuir ao pixel a cor da interceção mais próxima
        }
    
```

Algoritmos tipo lista de prioridades

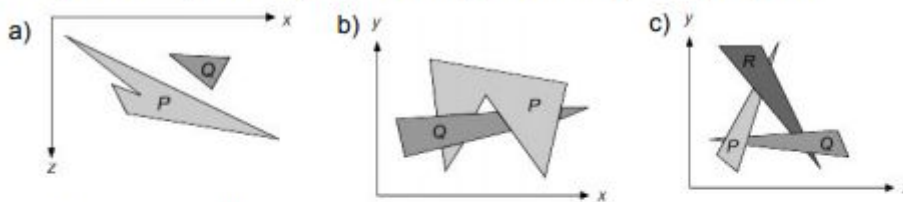
Objetivo:

- determinar a ordem de visibilidade para os objectos (polígonos), assegurando assim que a imagem será correctamente criada se os objectos forem desenhados por certa ordem:
 - Pintar as faces mais afastadas do observador em primeiro lugar
 - À medida que outras, mais próximas, vão sendo pintadas, ocultam as anteriores (Algoritmo do “Pintor”, wink wink).

Algoritmo Newel, Newel & Sancha(*Depth-sort* algorithm):

1. Ordenar os polígonos por ordem crescente de Z
2. Resolver qualquer ambiguidade na ordenação, nomeadamente se houver sobreposição de polígonos na coordenada Z. Poderá ser necessário dividir polígonos.
3. Pintar os polígonos por ordem do mais afastado para o mais próximo.

Tipo de ambiguidades que podem surgir na ordenação dos polígonos:



Pré-processamento:

Ordenar os polígonos pela coordenada **Z do vértice mais afastado**

Processamento:

Para o último polígono **P** da lista, verificar se existe algum polígono **Q** cujo maior **Z** seja mais afastado do que o menor **Z** de **P**, e que esteja a ser obstruído por **P**. Se não estiver, então **P** pode ser desenhado.

Iluminação Global

Objetivo: calcular a cor de cada ponto a partir da iluminação direta de uma fonte de luz (= **iluminação local**), mais a soma de todas as reflexões das superfícies próximas.

A iluminação Global respeita a Equação de Rendering:

$$I(x, x') = g(x, x') \cdot \left[\varepsilon(x, x') + \int_S \rho(x, x', x'') \cdot I(x', x'') \cdot dx'' \right]$$

$I(x, x')$

Iluminação de x' sobre x

$g(x, x')$

Termo geométrico:

$=0$, se x e x' não se vêem mutuamente

$=1/r^2$, se x e x' se vêem (r : dist. entre ambos)

$\varepsilon(x, x')$

Emissão de luz de x' para x

$\rho(x, x', x'')$

Perc. de iluminação oriunda de x'' e que é refletida em x' na direção de x

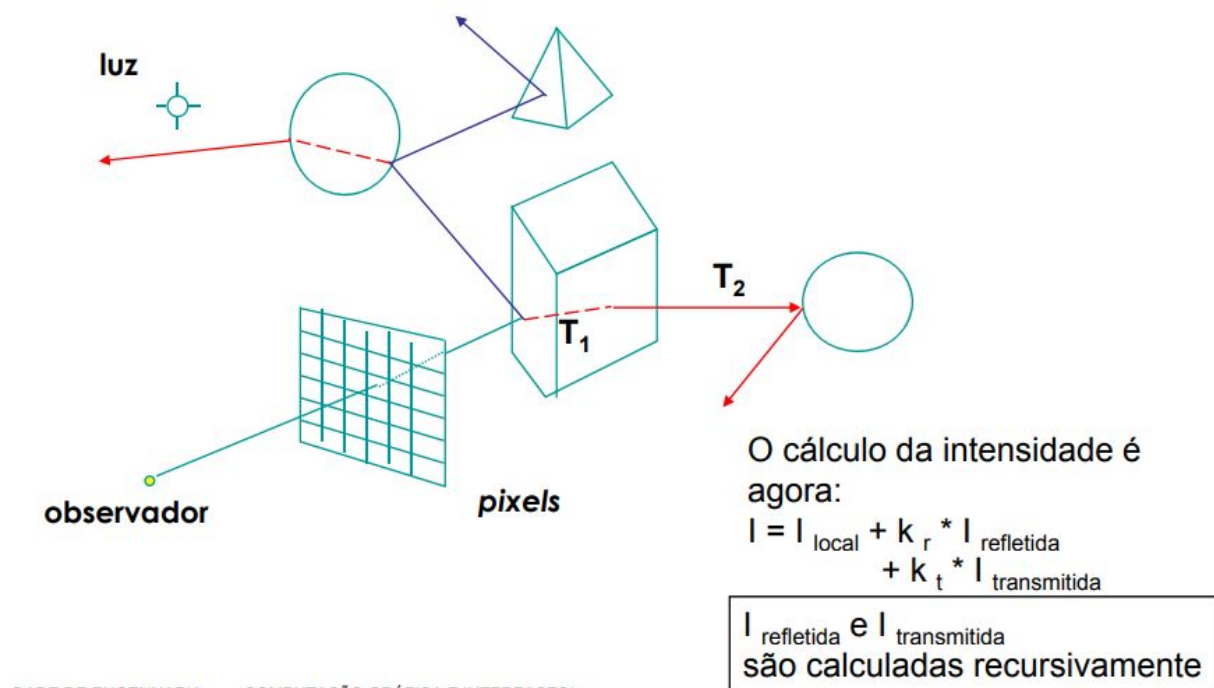
Algoritmos de Iluminação Global:

- Ray Tracing
- Radiosity

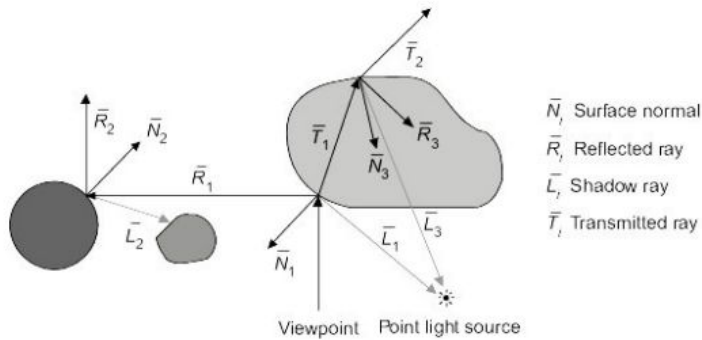
Ray Tracing

Depende da posição do observador.

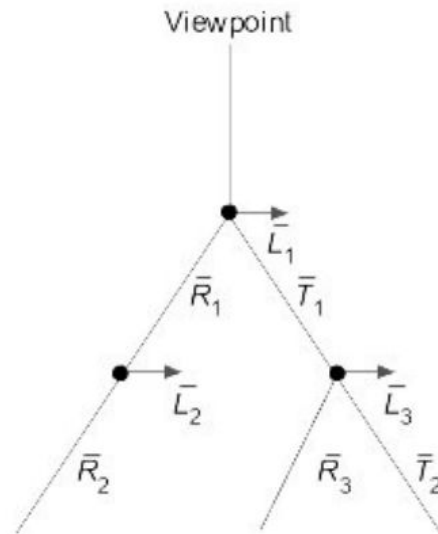
O plano de visualização é discretizado em pontos de amostragem (ex. pixels). Faz-se passar, por cada um destes pontos, um raio luminoso que parte do observador em direção ao interior da cena. O rasto de cada raio vai permitir somar as contribuições de reflexões entre faces mais próximas. A interseção do raio refletido com os restantes objetos é registada para obter as contribuições destes na iluminação do ponto. A atenuação devido à distância da face pode ser considerada. O processo é recursivo. Se os objetos forem transparentes ou semitransparentes é necessário considerar os raios transmitidos para o interior do objeto (ou exterior). Por exemplo, os raios T1 e T2.



Para cada pixel constrói-se uma árvore de interseções. A cor final do pixel determina-se percorrendo a árvore das folhas para a raiz e calculando as contribuições de cada ramo de acordo com o modelo de reflexão.



Nos objetos opacos não existe o raio transmitido. O ramo da árvore termina quando o raio atinge um objeto não refletor ou o ramo atinge uma determinada profundidade pré-estabelecida



O algoritmo de Ray Tracing é vantajoso porque:

- sombras, reflexões e refrações são facilmente incorporadas
- simula razoavelmente bem os efeitos especulares

O algoritmo de Ray Tracing tem custos computacionais elevados porque:

- o custo de cálculo das interseções é elevado
- não simula bem os efeitos de iluminação difusa

A otimização faz-se em duas áreas:

1. Diminuição do número de raios a processar
2. Diminuição do número de interseções a testar

Diminuição do número de raios a processar

- **Item Buffers** – determinam-se quais as áreas do ecrã onde se situam os objetos, raios que não intersetem essas áreas não são necessários
- **Adaptive Tree-Depth Control** - não é necessário levar todos os ramos da árvore de shading à sua profundidade máxima: usa a importância de um raio luminoso sobre o pixel a que pertence; esta importância diminui a cada reflexão ou transmissão
- **Light-Buffers** – a cada fonte de luz associam-se listas com os objetos que a rodeiam; os *shadow feelers*, uma vez definida a sua direção, são primariamente testados os objetos que se encontram na lista respetiva.

Diminuição do número de interseções a testar

- **Volumes Envolventes** – antes de efetuar o teste de interseção de um raio com um objeto, tenta-se a sua interseção com um volume simples envolvente

do objeto. Este teste prévio é muito rápido e exclui imediatamente muito teste de interseção mais complexos

- **Organização Hierárquica dos Volumes Envolventes** – a utilização de volumes envolventes de outros volumes envolventes permite economizar muito testes de interseção: se um raio não intersecciona um volume, então também não intersecciona os volumes nele contidos
- **Divisão Espacial em Grelhas Tridimensionais** – cada célula resultante desta divisão conhece os objetos que contém, total ou parcialmente. De acordo com a posição e a direção do raio em questão, só determinadas células são visitadas e, deste modo, só os objetos nelas contidos são testados. Dado que a ordem de progressão nas células é definida pelo sentido do raio, a primeira célula onde se detete uma interseção termina o processo de visita de raios às células

Radiosity

O algoritmo é independente do ponto de observação. O algoritmo só efetua, realmente, o cálculo de iluminação; trabalha no espaço objeto. É complementado por um algoritmo de cálculo de visibilidade para a produção da imagem final.

Fases de processamento:

1. Modela as interações de luz entre objetos e fontes de luz, sem considerar a posição do observador
2. Cria a imagem considerando o observador, efetua cálculo de visibilidade

Os métodos de radiossidade consideram que todas as superfícies podem produzir luz. Assim, as fontes de luz são modeladas como superfícies normais, com uma dada área.

O método assume que os processos de emissão e reflexão são difusos ideais. Necessita das faces discretizadas em **patches** de forma a garantir que a radiossidade se mantém constante na área correspondente a um patch.

A radiosidade (B_i) é definida como a energia expelida, por unidade de tempo e de área, de um **patch**, sendo composta por duas partes:

$$B_i A_i = E_i A_i + \rho_i \sum_j (F_{j-i} B_j A_j)$$

energia expelida
energia emitida (produzida)
energia refletida

Por unidade de área:

$$B_i = E_i + \rho_i \sum_j (F_{j-i} B_j A_j / A_i)$$

B_i - radiosidade, energia expelida do **patch** em Watt/m²

E_i - emissão de luz (auto-emitida) pelo **patch** i

ρ_i - refletividade, percentagem da energia incidente que é refletida pelo **patch** i

F_{j-i} - fator de forma, percentagem de energia que abandona o patch j e atinge i

Em ambientes difusos, existe a seguinte relação de reciprocidade entre fatores de forma:

$$A_i \cdot F_{i-j} = A_j \cdot F_{j-i}$$

Que aplicada na expressão anterior da radiosidade resulta em:

$$B_i = E_i + \rho_i \sum_j B_j F_{i-j}$$

Ou:

$$B_i - \rho_i \sum_j B_j F_{i-j} = E_i$$

Assim, a interação de luz entre **patches** pode ser representada por um sistema de equações lineares:

$$\begin{bmatrix} 1 - \rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 - \rho_2 F_{22} & \dots & -\rho_2 F_{2n} \\ \dots & \dots & \dots & \dots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \dots & 1 - \rho_n F_{nn} \end{bmatrix} \times \begin{bmatrix} B_1 \\ B_2 \\ \dots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \dots \\ E_n \end{bmatrix}$$

Criação da imagem:

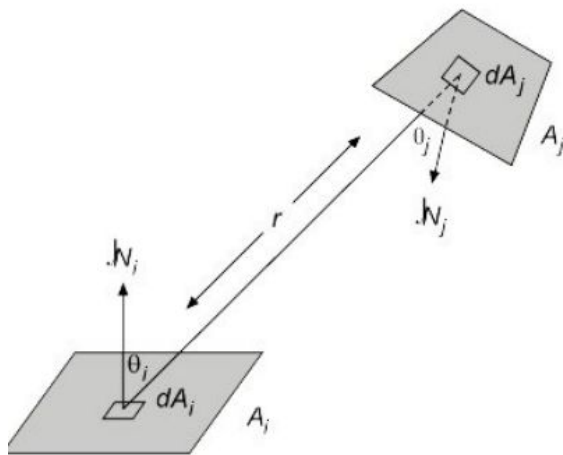
1. Resolvendo o sistema de equações, por eliminação Gaussiana, obtém-se a radiosidade para cada **patch**
2. Definir a posição do observador
3. Aplicar um algoritmo de visibilidade
4. Calcular a radiosidade dos vértices de cada polígono
5. Aplicar a interpolação de cor

A mesma solução do sistema é usada para qualquer posição do observador. É necessário resolver novamente o sistema de equações se houver alteração de coeficientes de reflexão ρ ou de valores de emissão E . É necessário recalcular os fatores de forma se a geometria da cena for alterada.

A complexidade do método está no cálculo dos **fatores de forma**.

Fatores de Forma

O fator de forma **F_{ij}** representa a fração (em percentagem) da energia total expelida pelo **patch "i"** que atinge o **patch "j"**, tomando em consideração a forma, orientação relativa e distância entre ambos os patches, bem como os obstáculos que obstruam o caminho.



O fator de forma da área diferencial dA_i para a área diferencial dA_j é dada por:

$$dF_{di-dj} = \frac{\cos \theta_i \cos \theta_j}{\pi r^2} H_{ij} dA_j$$

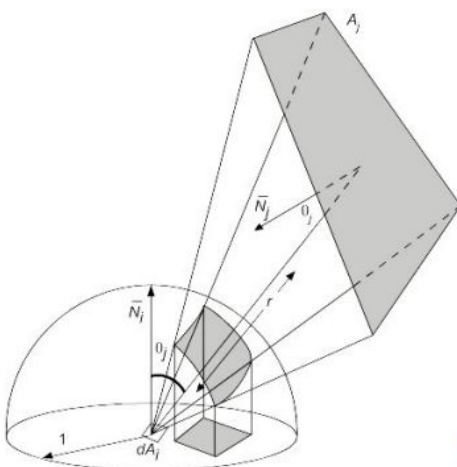
H_{ij} é 1 ou 0, dependendo de dA_j ser visível ou não a partir de dA_i .

Para determinar **F_{di-j}** , o fator de forma da área diferencial dA_i para a área finita **A_j** , integramos a área da **patch j**:

$$F_{di-j} = \int_{A_j} \frac{\cos \theta_i \cos \theta_j}{\pi r^2} H_{ij} dA_j$$

Finalmente o fator de forma da área **A_i** para a área **A_j** é dado por:

$$F_{i-j} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos \theta_i \cos \theta_j}{\pi r^2} H_{ij} dA_j dA_i$$

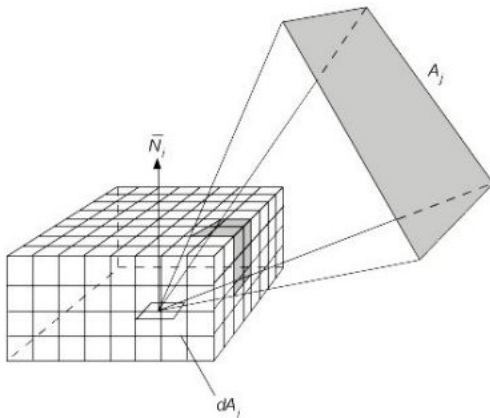


Verifica-se que o cálculo do Fator de Forma **F_{di-j}** corresponde a projetar as partes de **A_j** visíveis de **dA_i** num hemisfério centrado em **dA_i** , projetando depois esta projeção de forma ortográfica na base do hemisfério e dividindo pela área do círculo. (Analogia de Nusselt)

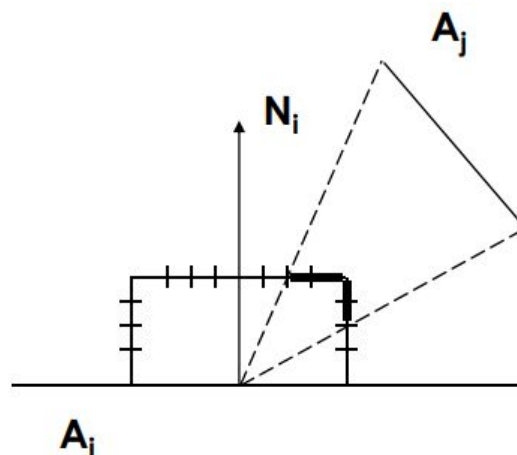
O cálculo é complexo.

Simplificação de Cohen e Greenberg: método do hemicubo

Em vez de usar a projeção num hemisfério, projeta na parte superior de um cubo centrado em dA_i , sendo a parte superior do cubo paralela com a superfície.

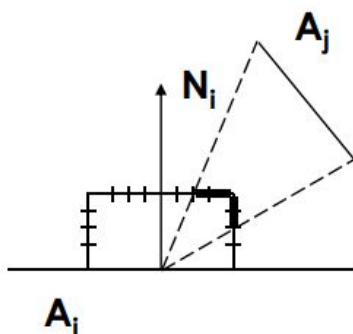


Cada face do hemicubo é dividida num conjunto de células quadradas de igual dimensão (ex: 50 por 50)



Project A_j no hemicubo, registrando os quadrados (mini-patch) que são cobertos. Para cada quadrado registrar quais as **patches visíveis** A_j e a sua distância.

Guardar apenas a patch mais próxima, uma vez que as outras serão invisíveis: usar algoritmo de visibilidade no espaço imagem, eventualmente (normalmente...) o Z-Buffer!



São calculados fatores de forma elementares F_q para cada quadrado/célula do hemicubo q .

O fator de forma F_{i-j} é então obtido somando todas as contribuições das células cobertas pelo patch j .

$$F_{i-j} = \sum F_q$$

Problemas do algoritmo de radiosidade:

- Algoritmo computacionalmente pesado em processamento e utilização de memória
- Para obter precisão é necessária a divisão de objetos em *patches* de pequena dimensão. Implica N^2 fatores de forma para calcular.

Progressive Refinement Radiosity

- Resolução do sistema de equações lineares
 - Métodos iterativos com convergência para a solução final
 - Aproveitamento dos resultados intermédios como sendo provisórios
- Imagem é apresentada desde o início dos cálculos
 - Usando os resultados intermédios
 - Qualidade dos resultados vai melhorando com o tempo de processamento

Junção Ray-Tracing + Radiosity

- Exploração do que cada um processa melhor
- Ray-Tracing: reflexão especular
 - Radiosity: reflexão difusa

Curvas e Superfícies

Representação de superfícies: permitem descrever objectos através das suas faces. As três representações mais comuns são:

- **Malha poligonal**
- **Superfícies paramétricas bicúbicas**
- **Superfícies quadráticas**

Representação paramétricas de curvas: importantes na computação gráfica 2D e pelo facto das superfícies paramétricas serem uma generalização destas curvas.

– **Malha poligonal:** é uma colecção de arestas, vértices e polígonos interligados de modo que cada aresta é apenas ligada por um ou dois polígonos.

Características da malha poligonal:

- Uma aresta liga 2 vértices.
- Um polígono é definido por uma sequência fechada de arestas.
- Uma aresta é ligada a um ou dois polígonos (adjacentes).
- Um vértice é partilhado pelo menos por 2 arestas.
- Todas as arestas fazem parte de algum polígono.

A estrutura de dados para **representar a malha poligonal** pode ter várias configurações, que são avaliadas pelo **espaço de memória** e **tempo de processamento** necessário para obter resposta, por exemplo, a:

- Obter todas as arestas que se unem num dado vértice.

- Determinar os polígonos que partilham uma aresta ou um vértice.
- Determinar os vértices ligados a uma aresta.
- Determinar as arestas de um polígono.
- Representar graficamente a malha.
- Identificar erros na representação, como falta de uma aresta, vértice ou polígono.

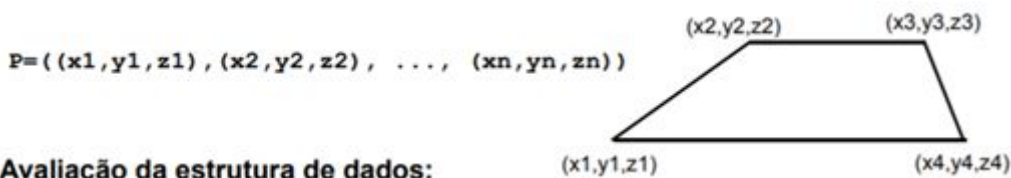
1. Representação Explícita: cada polígono é representado por uma lista de coordenadas dos vértices que o constituem. Uma aresta é definida por dois vértices consecutivos, fechando o polígono.

Consumo de memória (vértices repetidos).

Não há uma representação explícita das arestas e vértices partilhados.

Na representação gráfica a mesma aresta é usada (desenhada) mais do que uma vez.

Ao arrastar um vértice é necessário conhecer todas as arestas que partilham aquele vértice.



2. Representação por Apontadores para Lista de Vértices: cada polígono é representado por uma lista de índices (ou apontadores) para uma lista de vértices.

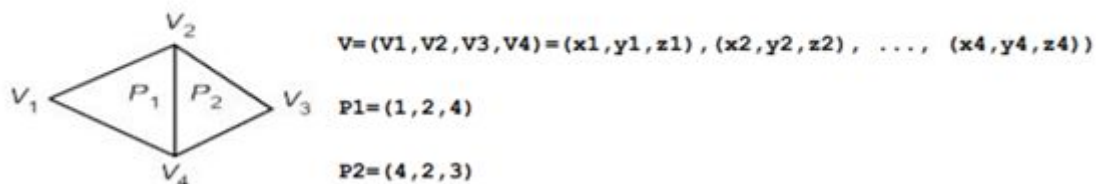
Vantagens:

- Cada vértice da malha poligonal é guardado uma única vez na memória.
- A coordenada de um vértice é facilmente alterada.

Desvantagens:

- Difícil obter os polígonos que partilham uma dada aresta.
- As arestas continuam a ser usadas (desenhadas) mais do que uma vez.

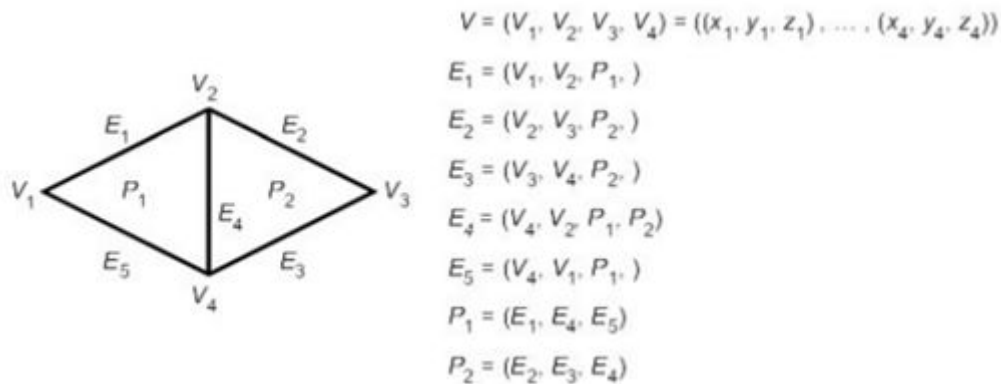
Lista de Vértices $V = ((x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_n, y_n, z_n))$



3. Representação por Apontadores para Lista de Arestas: cada polígono é representado por uma lista de apontadores para uma lista de arestas, na qual

cada aresta aparece uma única vez. Por sua vez, cada aresta aponta para os dois vértices que a definem e guarda também quais os polígonos a que pertence.

Um polígono é representado por $P=(E_1,E_2,...,E_n)$ e uma aresta como $E=(V_1,V_2,P_1,P_2)$. Se a aresta pertence apenas a um polígono então P_2 é null.



Vantagens:

- O desenho gráfico é facilmente obtido percorrendo a lista de arestas. Não ocorre a repetição de utilização (desenho) de arestas.
- Para o preenchimento (colorir) dos polígonos trabalha-se com a lista de polígonos. Fácil efectuar a operação de clipping sobre os polígonos.

Desvantagens:

- Continua a não ser imediato determinar quais as arestas que incidem sobre o mesmo vértice.

Solução de Baumgart

- Cada vértice tem um apontador para uma das arestas (aleatório) que incide nesse vértice.
- Cada aresta apresenta um apontador para a próxima aresta que incide nesse vértice.

Curvas Cúbicas

Motivação: representar curvas suaves do mundo real.

A representação por **malha poligonal** é uma aproximação de primeira ordem:

- A curva é aproximada por uma sequência de segmentos lineares.
- Grande quantidade de dados (vértices) para obter a curva com precisão.
- Difícil manipulação para mudar a forma da curva, i.e. necessário posicionar vários pontos com precisão.

Geralmente utilizam-se polinómios de grau 3 (Curvas Cúbicas), sendo a curva completa formada por um conjunto de curvas cúbicas.

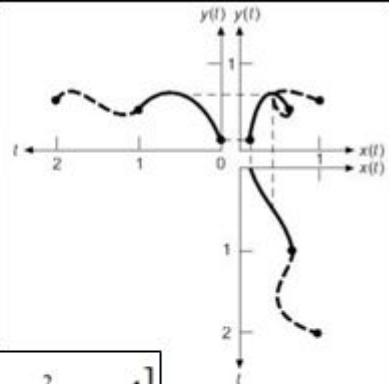
– grau < 3 oferecem pequena flexibilidade no controlo da forma das curvas e não permitem uma interpolação entre dois pontos com a definição da derivada nos pontos extremos. Um polinómio de grau 2 é especificado por 3 pontos que definem o plano onde a curva toma lugar.

– grau > 3 podem introduzir oscilações indesejáveis e exigir maior cálculo computacional.

A representação das curvas é feita na forma PARAMÉTRICA:
 $x = f_x(t), y = f_y(t)$ ex: $x=3t^3 + t^2$ $y=2t^3+t$

A forma Explícita: $y=f(x)$ ex: $y=x^3+2x^2$
 1. Não podemos ter vários valores de y para o mesmo x
 2. Não podemos descrever curvas com tangentes verticais

A forma Implícita: $f(x,y)=0$ ex: $x^2+y^2-r^2=0$
 1. Necessita de restrições para poder modelar apenas uma parte da curva
 2. Difícil juntar duas curvas de forma suave



Forma geral de representação da curva:

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \\ z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z \end{aligned} \quad 0 \leq t \leq 1$$

Sendo: $T = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$

$$C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}$$

$$Q(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix} = T \cdot C$$

-> A representação anterior é usada para representar uma única curva. **Como juntar os vários segmentos de curva?**

- Pretendemos a junção num ponto à continuidade geométrica.
- Que tenham o mesmo declive na junção à suavidade (continuidade da derivada).

A garantia de continuidade e suavidade na junção é garantida fazendo coincidir as derivadas (tangentes) das curvas no ponto de junção. Para isso calcula-se:

$$\frac{\partial Q(t)}{\partial t} = \begin{pmatrix} \frac{\partial x(t)}{\partial t} & \frac{\partial y(t)}{\partial t} & \frac{\partial z(t)}{\partial t} \end{pmatrix} = \frac{\partial (CT)}{\partial t} = C \frac{\partial T}{\partial t}$$

Com: $\frac{\partial T}{\partial t} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix}$

Tipos de Continuidade:

G0 – continuidade geométrica zero -> as curvas só se juntam num ponto.

G1 – continuidade geométrica um -> a direcção dos vectores tangentes é igual.

C1 – continuidade paramétrica 1 -> as tangentes no ponto de junção têm a mesma direcção e amplitude (primeira derivada igual).

Cn – continuidade paramétrica n -> as curvas têm no ponto de junção todas as derivadas iguais até à ordem n.

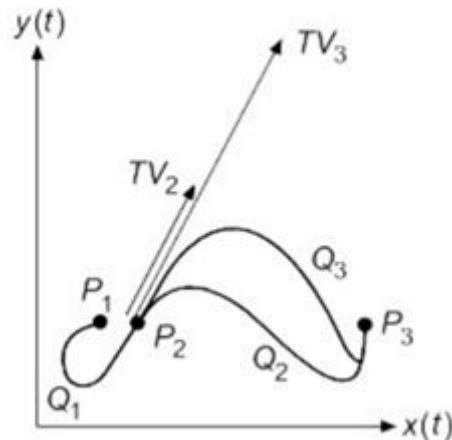
A continuidade paramétrica é mais restritiva que a continuidade geométrica:

Por exemplo: **C1 implica G1**

- No ponto de junção P_2 temos:

Q2 e Q3 são G1 com Q1

Só Q2 é C1 com Q1 ($TV_1=TV_2$)



Tipos de Curvas

1. Curvas de Hermite

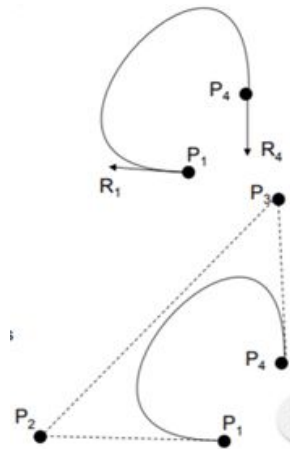
- Continuidade **G1** nos pontos de junção
- Vetor geométrico: • 2 pontos extremos e • Os vectores tangentes nesses pontos

2. Curvas de Bézier

- Continuidade **G1** nos pontos de junção
- Vetor geométrico: • 2 pontos extremos e • 2 pontos que controlam os vectores tangentes nesses extremos.

3. Curvas Splines

- Família de curvas muito alargada
- Maior controlo da continuidade nos pontos de junção (Continuidade C1 e C2)



Matriz de Base: Caracteriza o tipo de curva (Hermite, Bezier, etc).

Vetor Geométrico: Caracteriza a geometria de uma dada curva.

$$Q(t) = [x(t) \ y(t) \ z(t)] = T.C$$

$$Q(t) = T.M.G$$

$$\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$$

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

$$\begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}$$

Matriz T

Matriz de Base

Vetor Geométrico

Conclusão:

1. $Q(t)$ é uma soma pesada dos elementos do vetor geométrico.
2. Os pesos são polinomiais cúbicas em t -> **Funções de mistura**

$$Q(t) = T.C = T.M.G = B.G$$

$$Q(t) = T \cdot M_H \cdot G_H = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} M_H \cdot G_H = B_H \cdot G_H$$

$$Q'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} M_H \cdot G_H$$

Vector geométrico:

$$G_H = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}$$

$$\begin{cases} Q(0) = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} M_H \cdot G_H = P_1 \\ Q(1) = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} M_H \cdot G_H = P_4 \\ Q'(0) = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} M_H \cdot G_H = R_1 \\ Q'(1) = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} M_H \cdot G_H = R_4 \end{cases}$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot M_H \cdot G_H = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} = G_H \quad \Rightarrow \quad M_H = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Funções de Mistura (Blending functions)

$$Q(t) = T \cdot M_H \cdot G_H = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} M_H \cdot G_H = B_H \cdot G_H \quad Q(t) = (2t^3 - 3t^2 + 1)P_1 +$$

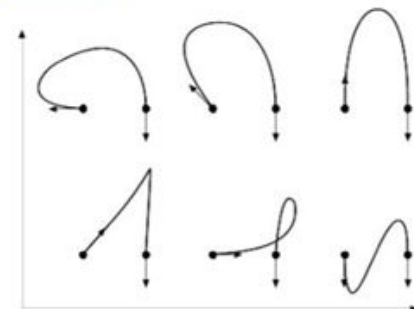
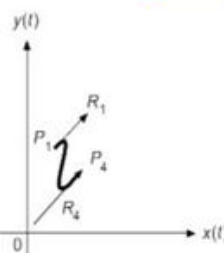
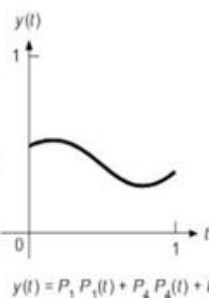
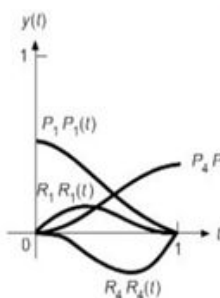
$$(-2t^3 + 3t^2)P_4 +$$

$$(t^3 - 2t^2 + t)R_1 +$$

$$(t^3 - t^2)R_4$$

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

$$G_H = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}$$



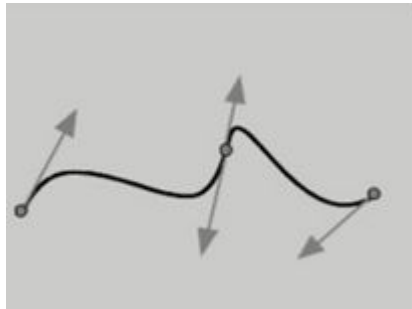
Esquerda: Funções de mistura

Centro: $y(t)$ = soma das quatro funções da esquerda

Direita: Curva de Hermite

P1 e P4 fixos, R4 fixo, R1 varia em direção

- Os pontos extremos podem ser reposicionados
- Os vectores tangentes podem ser alterados puxando as setas
- Os **vectores tangentes** são forçados a serem colineares (**continuidade G1**) e R4 é visualizado em sentido contrário (maior visibilidade)
- É comum dispor de comandos para forçar continuidade G0, G1 ou C1.



Continuidade na junção:

$$\begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} \rightarrow \begin{bmatrix} P_4 \\ P_7 \\ K \cdot R_4 \\ R_7 \end{bmatrix}$$

- $K > 0 \rightarrow G^1$
- $K = 1 \rightarrow C^1$

Para a continuidade ser C1...

Em C1 -> R4 (2,0) pois $R4_{C1} = R4_{C2}$

Em C2 -> P1 (3,3) pois $P1_{C2} = P2_{C1}$

-> P2 (6,6) pois $P1_{C3} = P2_{C2}$

A continuidade entre C2 e C3 é G1, mas não C1, pois os vetores são colineares, mas não de $K=1$ e sim de $K=2$

Em C4 -> R1 (0,-1) para $R2_{C3} = R1_{C4}$, Mas C4 e C3 não tem continuidade C1 pois os pontos $P2_{C3}$ e $P1_{C4}$ não são iguais.

$$C1 = \begin{bmatrix} 0,0 \\ 3,3 \\ 0,2 \\ ?,? \end{bmatrix}; \quad C2 = \begin{bmatrix} ?,? \\ ?,? \\ 2,0 \\ 0,2 \end{bmatrix}; \quad C3 = \begin{bmatrix} 6,6 \\ 3,6 \\ 0,1 \\ 0,-1 \end{bmatrix}; \quad C4 = \begin{bmatrix} 3,3 \\ 6,3 \\ ?,? \\ 2,0 \end{bmatrix}$$

Curvas de Bézier

comparando com G_H :

$$R_1 = Q'(0) = 3 \cdot (P_2 - P_1)$$

$$R_4 = Q'(1) = 3 \cdot (P_4 - P_3)$$

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

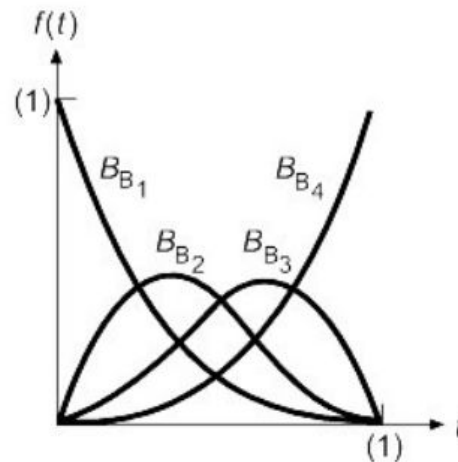
$$G_H = M_{HB} \cdot G_B$$

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

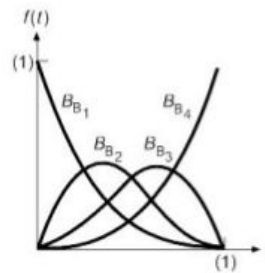
- A mesma curva em representação Bézier: $Q(t) = T.M_B.G_B$

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

$$Q(t) = \begin{aligned} &(1-t)^3 P_1 + \\ &3t(1-t)^2 P_2 + \\ &3t^2(1-t) P_3 + \\ &t^3 P_4 \end{aligned}$$

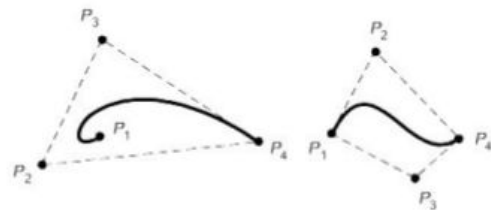


$$Q(t) = \begin{aligned} &(1-t)^3 P_1 + \\ &3t(1-t)^2 P_2 + \\ &3t^2(1-t) P_3 + \\ &t^3 P_4 \end{aligned}$$



Observações sobre as funções de Mistura:

- Para $t=0$ $Q(t)=P_1$, para $t=1$ $Q(t)=P_4 \rightarrow$ A curva passa em P_1 e P_4
- A soma em qualquer ponto é 1.
- Verifica-se que $Q(t)$ é uma **média pesada dos 4 pontos de controlo**, logo a **curva está contida no interior do polígono** definido por esses pontos, designado de "convex hull".



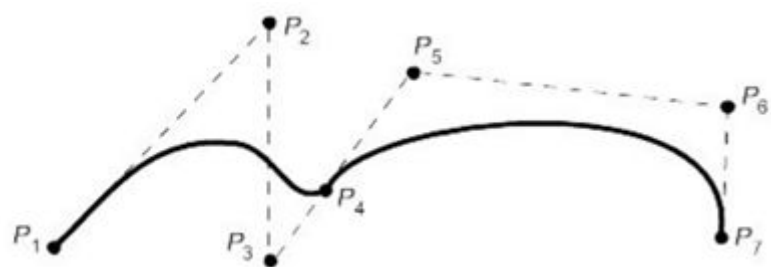
Junção de curvas de Bézier

Continuidade G^1 :

$P_4 - P_3 = K.(P_5 - P_4)$ com $K > 0$, i.e. P_3 , P_4 e P_5 devem ser colineares

Continuidade C^1 :

$P_4 - P_3 = K.(P_5 - P_4)$ com $K = 1$



Determinar matriz base

Matriz Base (M_N)

Exemplo: Exercício Recurso 2016

$$G_N = \begin{bmatrix} V_1 \\ P_1 \\ V_3 \\ P_3 \end{bmatrix} = \begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{bmatrix} \quad \begin{cases} V_1 = P_2 - P_1 \\ V_3 = P_3 - P_4 \\ P_1, P_2, P_3, P_4 \rightarrow \text{Curva Bezier} \end{cases}$$

Em cima renomeou-se a matriz para facilitar a interpretação.

1º Passo: Colocar P_1, P_2, P_3 e P_4 de Bezier em função de N_1, N_2, N_3 e N_4

$$P_1 = N_2$$

$$P_2 = N_1 + N_2 \quad (V_1 + P_1)$$

$$P_3 = N_4$$

$$P_4 = N_1 - N_3 \quad (P_3 - V_3)$$

~~Matriz Base~~

$$G_B = M_{NB} \cdot G_N$$

Neste exercício apenas de apenas se faz o procedimento para as curvas de Bezier, para as de Hermite seria semelhante, apenas diferente nas matrizes utilizáveis mais à frente e neste 1º passo (com Hermite temos 2 pontos e 2 valores em vez de 4 pontos)

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{bmatrix}$$

Para calcular M_N :

$M_N = M_B \circ M_{NB}$ ← No caso das curvas de Bezier

$M_N = M_H \circ M_{NH}$ ← No caso das curvas de Hermite

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

Neste caso utilizamos a fórmula baseada em Bezier

M_B $\begin{bmatrix} 0 & 1 & 0 & 0 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$ ← M_{NB}

$\left. \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 3 & 2 & -1 & 2 \\ -6 & 3 & 0 & 3 \\ 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \right\} \begin{array}{l} \text{Matriz} \\ \text{Base} \end{array}$

M_N

Desenho de Curvas Cúbicas

Dois algoritmos:

1. Avaliação de $x(t)$, $y(t)$ e $z(t)$ para valores incrementais de t entre 0 e 1.
2. Subdivisão da curva: **Algoritmo de Casteljau**

1. Avaliação de $x(t)$, $y(t)$ e $z(t)$

É possível reduzir o número de operações de 11 multiplicações e 10 adições para 9 e 10, respectivamente. $f(t) = at^3 + bt^2 + ct + d = ((at + b).t + c).t + d$

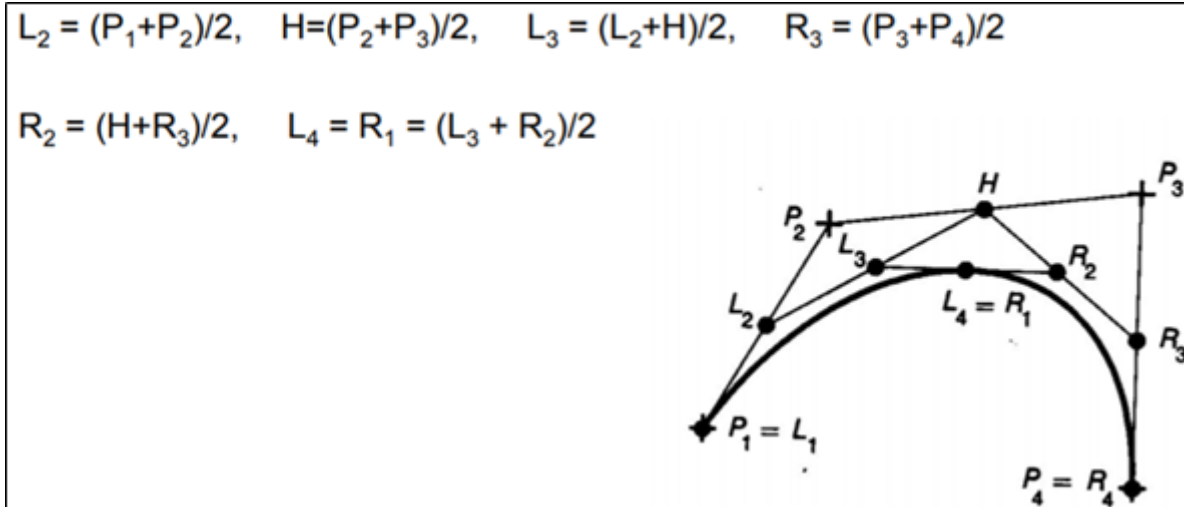
2. Algoritmo de Casteljau

Efectua a subdivisão recursiva da curva, parando apenas quando a curva em questão é suficientemente “plana” para poder ser aproximada por um segmento de recta.

Algoritmo eficiente: requer apenas 6 shifts e 6 adições em cada divisão.

Algoritmo de Casteljau

NOTAR: Casteljau != Casteljou



Critérios possíveis de paragem:

- A curva em questão é suficientemente “plana” para poder ser aproximada por um segmento de recta.
- Os 4 pontos de controlo estão no mesmo pixel

Algoritmo de Calteljau

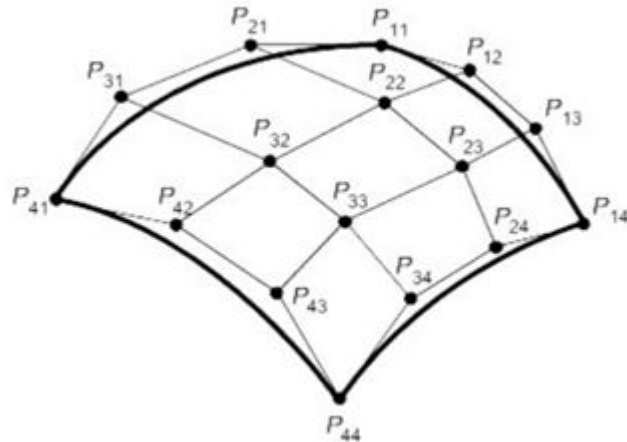
```
void DrawCurveRecSub (curve, ε)
{
    if (Straight (curve, ε))
        DrawLine (curve);
    else {
        SubdivideCurve (curve, leftCurve, rightCurve);
        DrawCurveRecSub (leftCurve, ε);
        DrawCurveRecSub (rightCurve, ε);
    }
}
```

As equações para a superfície de Bézier podem ser obtidas da mesma forma que as de Hermite, resultando:

$$x(s,t) = S.M_B.G_{Bx}.M_B^T.T^T$$

$$y(s,t) = S.M_B.G_{By}.M_B^T.T^T$$

$$z(s,t) = S.M_B.G_{Bz}.M_B^T.T^T$$



A matriz geométrica tem 16 pontos de controlo.

Continuidade C^0 e G^0 é obtida fazendo coincidir os quatro pontos de controlo de fronteira: P_{14} , P_{24} , P_{34} , P_{44}

Para obter G^1 devem ser colineares:

P_{13} , P_{14} e P_{15}

P_{23} , P_{24} e P_{25}

P_{33} , P_{34} e P_{35}

P_{43} , P_{44} e P_{45}

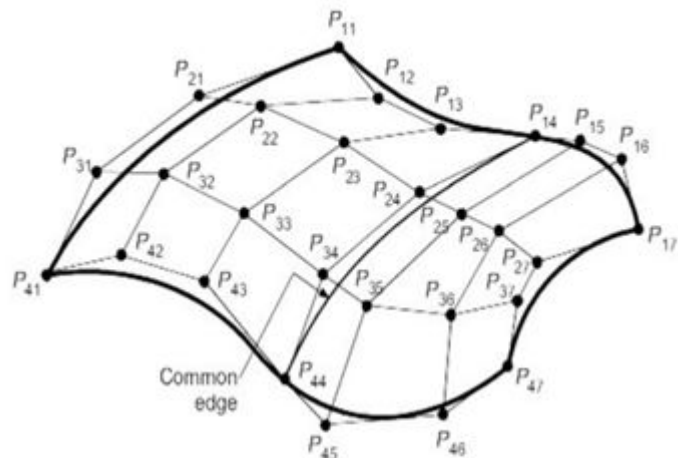
e

$$(P_{14} - P_{13}) / (P_{15} - P_{14}) = K$$

$$(P_{24} - P_{23}) / (P_{25} - P_{24}) = K$$

$$(P_{34} - P_{33}) / (P_{35} - P_{34}) = K$$

$$(P_{44} - P_{43}) / (P_{45} - P_{44}) = K$$



Modelação de Sólidos

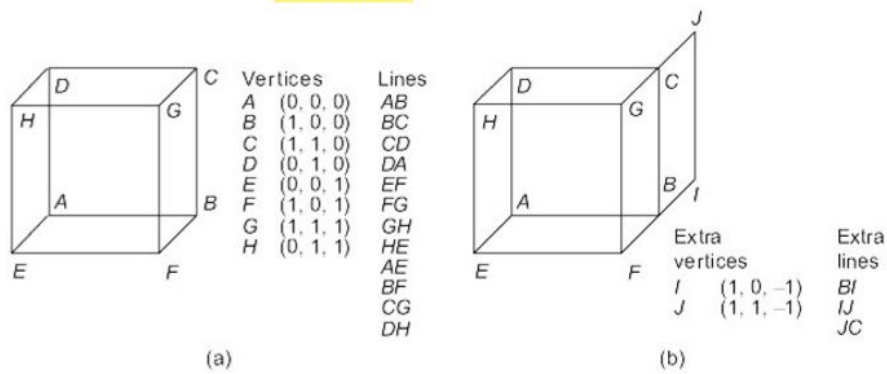
Em 2D um conjunto de segmentos de reta ou curvas não formam necessariamente uma área fechada.

Em 3D uma coleção de superfícies não envolve necessariamente um volume fechado.

Caraterísticas:

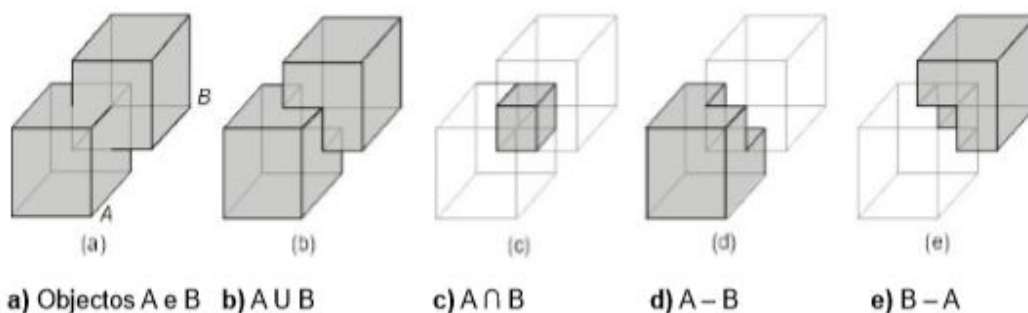
- Deve abranger um domínio de representação suficientemente abrangente para incorporar todo o tipo de objetos.
- A representação deve ser não ambígua (uma dada representação deve corresponder a um único sólido) e única (cada objeto deve ter apenas uma representação possível).
- Preciso / Exato.
- Impossibilidade de criar objetos inválidos.
- Representação fechada.
- Representação compacta para otimizar a utilização de memória.

Exemplo de objectos **não válidos** como sólido.



- A representação de **a)** não identifica claramente as faces do cubo, apenas indica arestas.
- Podemos considerar que uma sequência de 4 segmentos formam uma face? Mas o sólido **b)** seria erradamente considerado sólido.

Operações Booleanas: A combinação de objetos por operações booleanas permite definir novos objetos, independentemente da representação usada. As operações são união, diferença e interseção.



Representação por Instanciação de Primitivas:

O sistema de modelação tem pré-definido um conjunto de sólidos 3D úteis para a modelação pretendida. Não prevê a combinação de objetos como por exemplo por operações Booleanas.

Representação por Varrimento:

O deslocamento de um objeto segundo uma trajetória define um outro objeto:

- Translação
- Rotação

A utilização deste método sem restrições de trajetória pode resultar numa modelação ineficiente do objeto.

Em geral as ferramentas de software convertem os objectos criados por varrimento em uma outra forma de representação.

Representação pela Fronteira (b-rep):

Os sólidos são descritos pela sua superfície de fronteira. Vão ser considerados apenas os sólidos com fronteira 2-manifolds (cada aresta é partilhada por duas faces).

Poliedro: Sólido delimitado por um conjunto de polígonos cujas arestas pertencem a dois polígonos

Fórmula de Euler

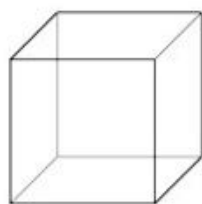
Um poliedro simples, sem buracos, obedece à **fórmula de Euler**:

$$V - E + F = 2$$

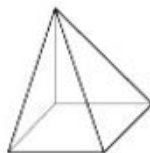
V – Vértices

E – Arestas (edges)

F – Faces



$$\begin{aligned} V &= 8 \\ E &= 12 \\ F &= 6 \end{aligned}$$



$$\begin{aligned} V &= 5 \\ E &= 8 \\ F &= 5 \end{aligned}$$



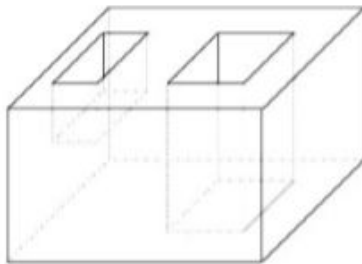
$$\begin{aligned} V &= 6 \\ E &= 12 \\ F &= 8 \end{aligned}$$

Para garantir que um objeto seja um poliedro simples / sólido válido:

- Cumprir a fórmula de Euler
- Cada aresta liga 2 vértices e é partilhada por 2 faces.
- Pelo menos 3 arestas encontram-se no mesmo vértice

**Generalização da Fórmula de Euler
para poliedros com buracos:**

$$V - E + F - H = 2(C - G)$$



$$V - E + F - H = 2(C - G)$$

24 36 15 3 1 1

V – Vértices

E – Arestas (edges)

F – Faces

H – Número de buracos nas faces

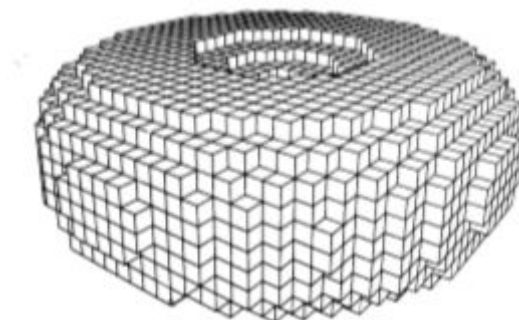
G – Número de buracos que
atravessam o objecto

C – número de partes do objecto

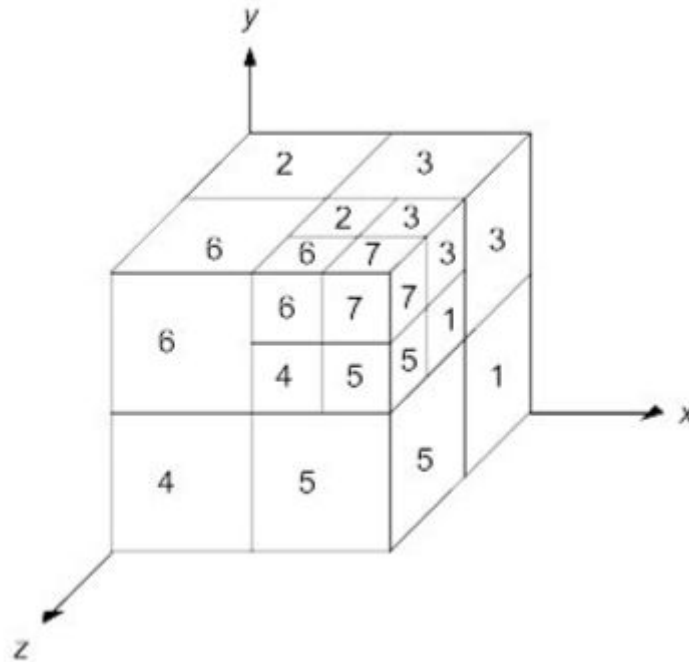
Número de partes do objeto: geralmente é 1, valor incrementa se objeto tiver objetos contidos sem ligação com o exterior

Representação por Decomposição Espacial

- Decomposição Celular: Trata-se de uma união de células que não se intersectam.
- Enumeração da Ocupação Espacial: Sólido formado por *voxels* (células) idênticas de igual dimensão colocadas numa grelha regular. Costumam ser cubos.



- Octrees: A necessidade de divisão do espaço só ocorre na superfície.

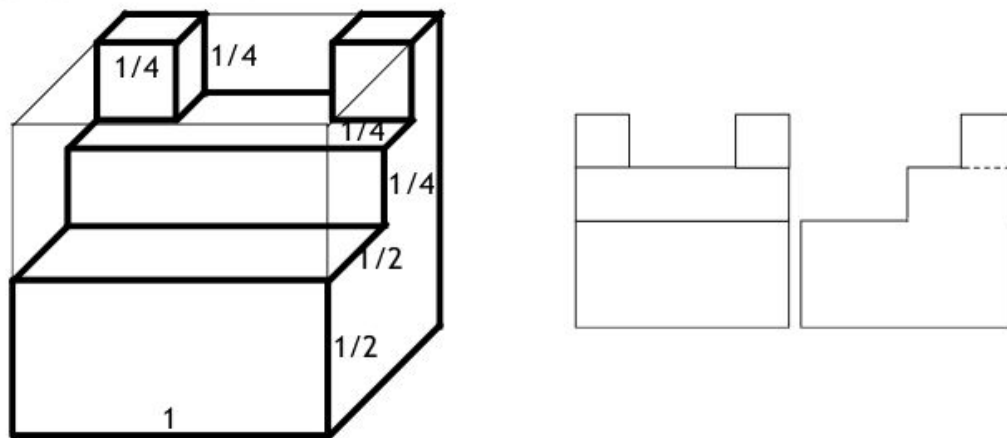


Representar um sólido em octree consiste em dividi-lo em octantes:

- Enquanto existir uma divisão que não esteja catalogada como completamente cheia (F) ou completamente vazia (E), dividi-la em oito partes iguais, numeradas a partir de 0 como mostrado na figura em cima.

Exemplo do exame recurso 2016

8) [1.5] Represente o objeto seguinte usando *octrees*.



Dividir sólido inicial

0	1	2	3	4	5	6	7
F	F	X	X	F	F	E	E

Dividir octante 2

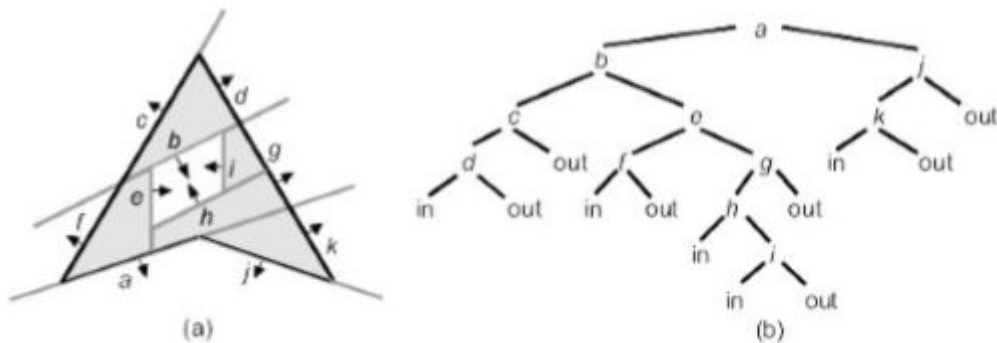
0	1	2	3	4	5	6	7
F	F	F	E	F	F	E	E

Divisão octante 3

0	1	2	3	4	5	6	7
F	F	E	F	F	F	E	E

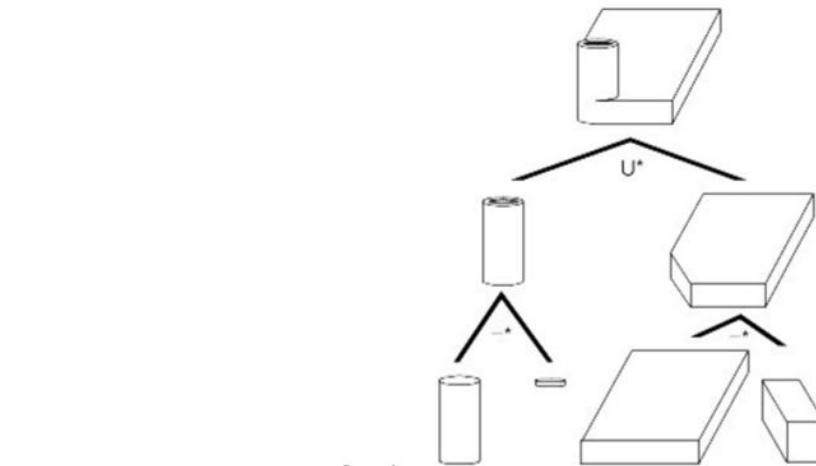
Já não há mais divisões sem E ou F, parar.

- Árvores binárias de Partição do Espaço (BSP):
 - Em cada passo, o espaço é dividido por um plano de posição e orientação arbitrárias
 - A cada nó interno da árvore está associado um plano e 2 apontadores (um para o lado de dentro do polígono e outro para o lado de fora).
 - Se um sub-espço é homogêneo (totalmente interior ou exterior), deixa de ser dividido.



Geometria Sólida Construtiva (Constructive Solid Geometry – CSG)

- O objecto é obtido pela combinação de primitivas simples através de operadores booleanos.
- O objecto é guardado como uma árvore, em que os nós interiores são operadores e as folhas são primitivas simples
- Os nós representam operações booleanas e transformações geométricas.



Rasterização de Linhas

Algoritmo para desenho de Segmentos de Reta

- Devem obter coordenadas inteiras.
- Devem ser algoritmos eficientes.
- Devem parecer retas, terminar com precisão e apresentar brilho constante.

DDA – Digital Differential Analyser

```
void DDA(int X1, int Y1, int X2, int Y2)
{
    //considerando -1 <=m <=1 e X1<X2
    int x;
    float dy, dx, y, m;

    dy = Y2 - Y1;
    dx = X2 - X1;
    m = dy/dx;
    y = Y1;
    for (x=X1; x<=X2; x++) {
        WritePixel(x, (int)(y + 0.5));
        y += m;
    }
}
```

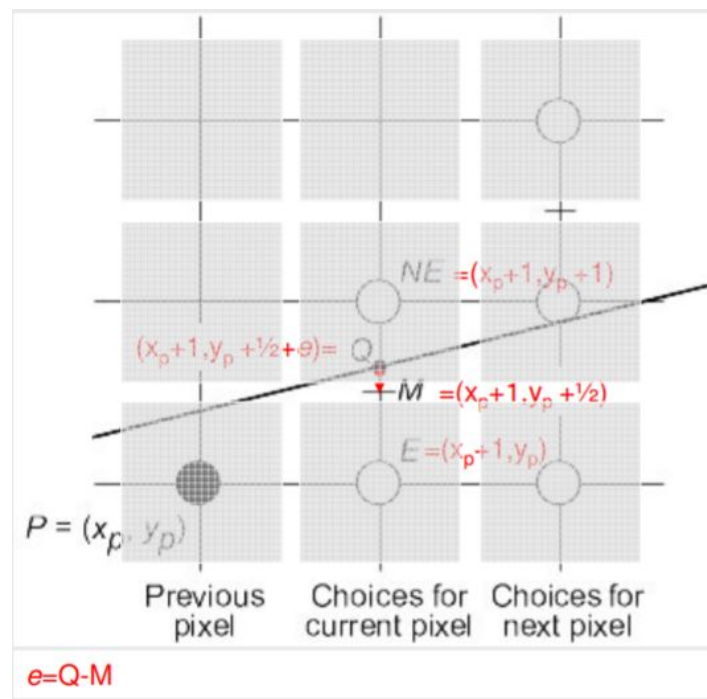
Problemas do algoritmo:

1. Operações em vírgula flutuante implicam menor eficiência do que com inteiros
2. O valor de y evolui pelo incremento sucessivo de m (valor real); variáveis reais têm precisão limitada -> soma acumulada de um valor inexato pode originar um desvio do valor real pretendido round(yi).

Algoritmo Midpoint

Se uma reta passar na origem: $y = (b/a).x + 0 \rightarrow f(x,y) = bx - ay = 0$ também é uma equação da reta.

"e" é o valor entre o ponto médio e o valor da reta em $(x+1)$. Se "e" for positivo, então escolhe-se NE, senão escolhe-se E.



A variável de decisão pode então ser:

$$\begin{aligned}
 f(x_p + 1, y_p + 1/2 + e) &= 0 \quad \Leftrightarrow \quad (\text{ponto pertence à reta}) \\
 b(x_p + 1) - a(y_p + 1/2 + e) &= 0 \quad \Leftrightarrow \\
 b(x_p + 1) - a(y_p + 1/2) - a.e &= 0 \quad \Leftrightarrow \\
 f(x_p + 1, y_p + 1/2) - a.e &= 0 \quad \Leftrightarrow \\
 f(x_p + 1, y_p + 1/2) &= a.e
 \end{aligned}$$

Designemos uma **variável de decisão** d_p como:

$$d_p = f(x_p + 1, y_p + 1/2)$$

Calcular d_p é ineficiente (duas adições, uma subtração e duas multiplicações). Pode-se otimizar usando a d_p anterior

Genericamente,

$$d_{i+1} = f(x_{i+1} + 1, y_{i+1} + 1/2)$$

Sendo que:

para $d_i \geq 0$ (movimento para **NE**),

$$x_{i+1} = x_i + 1 \text{ e } y_{i+1} = y_i + 1$$

para $d_i < 0$ (movimento para **E**),

$$x_{i+1} = x_i + 1 \text{ e } y_{i+1} = y_i$$

```
MidPoint(int X1, int Y1, int X2, int Y2)
{
    int a, b, d, incl, inc2, x, y;
    a = X2 - X1;
    b = Y2 - Y1;
    inc2 = 2*b;
    d = inc2 - a;          // d = 2*b - a;
    incl = d - a;          // incl = 2*(b-a);
    x = X1; y=Y1;
    for(i=0; i<a; i++)
    {
        plot(x,y);
        x = x+1;
        if (d >= 0)
        {
            y=y+1; d=d+incl; }
        else{d=d+inc2; }
    }
}
// Para retas no primeiro octante e 0<=m<=1
```

Vantagens:

- Apenas aritmética inteira (+ e *2).
- Permite o cálculo incremental dos pontos, i.e. obter (x_{i+1}, y_{i+1}) a partir de (x_i, y_i) .

O algoritmo pode ser então composto da seguinte forma:

// Calcular d_0 diretamente.

Para cada $i \geq 0$:

if $d_i \geq 0$ then

Plot $(x_i + 1, y_i + 1)$

// Escolhe **NE** como próximo ponto

$$d_{i+1} = f(x_{i+1} + 1, y_{i+1} + 1/2) = f(x_i + 1 + 1, y_i + 1 + 1/2)$$

$$= b(x_i + 1 + 1) - a(y_i + 1 + 1/2) = f(x_i + 1, y_i + 1/2) + b - a$$

$$= d_i + b - a$$

else

Plot $(x_i + 1, y_i)$

// Escolhe **E** como próximo ponto

$$d_{i+1} = f(x_{i+1} + 1, y_{i+1} + 1/2) = f(x_i + 1 + 1, y_i + 1/2)$$

$$= b(x_i + 1 + 1) - a(y_i + 1/2) = f(x_i + 1, y_i + 1/2) + b$$

$$= d_i + b$$

Conclusão: Sabendo d_i , apenas temos de somar um valor constante para saber d_{i+1} ; a soma pode ser $(d_i + b - a)$ ou $(d_i + b)$, dependendo de se ter avançado para **NE** ou para **E**.

O valor d_0 pode ser obtido por:

$$d_0 = f(x_0 + 1, y_0 + 1/2) = f(0 + 1, 0 + 1/2) = b \cdot 1 - a \cdot 1/2 = b - a/2$$

Quando a é um número ímpar d_0 assume valores não inteiros. Uma vez que só nos interessa conhecer o sinal de d_i em cada etapa, podemos multiplicar toda a equação por 2 que não alteramos em nada o funcionamento do algoritmo:

Inicialização de d :

$$D_0 = 2 \cdot (b - a/2) = 2b - a$$

Actualização de D quando movimento é para NE:

$$D_{i+1} = D_i + 2 \cdot (b - a)$$

Actualização de D quando movimento é para E:

$$D_{i+1} = D_i + 2 \cdot b$$

Algoritmo Midpoint para desenho de circunferências

3. Se centro em (0,0) $\rightarrow f(x,y)=x^2+y^2-r^2$
 $f(x,y) < 0$ então (x,y) está **dentro** da circunferência
 $= 0$ então (x,y) está **sobre** a circunferência
 > 0 então (x,y) está **fora** da circunferência

Da mesma forma que foi feito para a reta define-se a variável de decisão **d**:

$$d_p = f(x_p + 1, y_p - 1/2) =$$

$$(x_p + 1)^2 + (y_p - 1/2)^2 - r^2$$

Subtração

Algoritmo:

// Calcular d_0 diretamente.

Para cada $i \geq 0$:

if $d_i \geq 0$ then

Plot $(x_i + 1, y_i - 1)$ // Escolhe **SE** como próximo ponto

$$\begin{aligned} d_{i+1} &= f(x_{i+1} + 1, y_{i+1} - 1/2) = f(x_i + 1 + 1, y_i - 1 - 1/2) \\ &= (x_i + 2)^2 + (y_i - 3/2)^2 - r^2 \\ &= d_i + (2x_i - 2y_i + 5) \end{aligned}$$

else

Plot $(x_i + 1, y_i)$ // Escolhe **E** as next point

$$\begin{aligned} d_{i+1} &= f(x_{i+1} + 1, y_{i+1} - 1/2) = f(x_i + 1 + 1, y_i - 1/2) \\ &= (x_i + 2)^2 + (y_i - 1/2)^2 - r^2 \\ &= d_i + (2x_i + 3) \end{aligned}$$

Conclusão, podemos obter $d(i+1)$ a partir de $d(i)$ mas é necessário calcular o incremento em cada etapa.

O valor d_0 pode ser obtido considerando o primeiro ponto (0,R):

$$d_0 = f(0 + 1, R - 1/2) = 1 + (R^2 - R + 1/4) - R^2 = 5/4 - R$$

```
MidPointCircle(int R)
{   int x, y;
    float d;

    x=0; y=R;
    d = 5.0/4.0 - (float)R;
    plot(x,y);
    while(y > x)
    {   if (d >= 0)
        {   d=d+(x-y)*2+5;
            x++; y--; }
        else
        {   d=d+2.0*x+3;
            x++; }
        plot(x,y);
    }
}
```

Observações:

- Utiliza aritmética em vírgula flutuante.
- Minimiza as operações efectuadas em vírgula flutuante

Algoritmo otimizado:

```
MidPointCircle(int R)
{   int x, y, p, inc_E, inc_SE;

    x=0; y=R;
    p=1-R;
    inc_E=3; inc_SE=5-2*R;
    plot(x,y);

    while(y > x)
    {   if (p<0)
        {   p=p+inc_E;
            inc_E=inc_E+2;
            inc_SE=inc_SE+2;
            x++;
        }
        else
        {   p=p+inc_SE;
            inc_E=inc_E+2;
            inc_SE=inc_SE+4;
            x++; y--;
        }
        plot(x,y);
    }
}
```

Observações:

Utiliza somente aritmética de inteiros

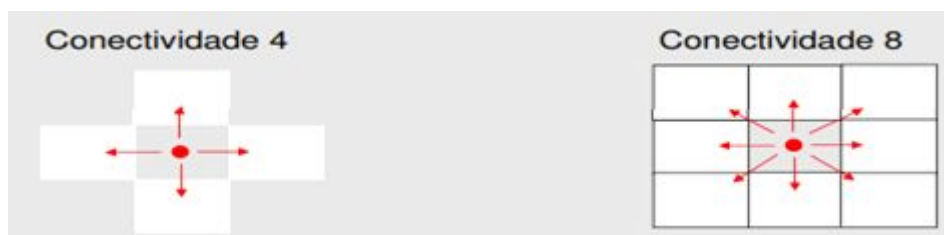
Faz uso de incrementos de segunda ordem

Rasterização de Regiões

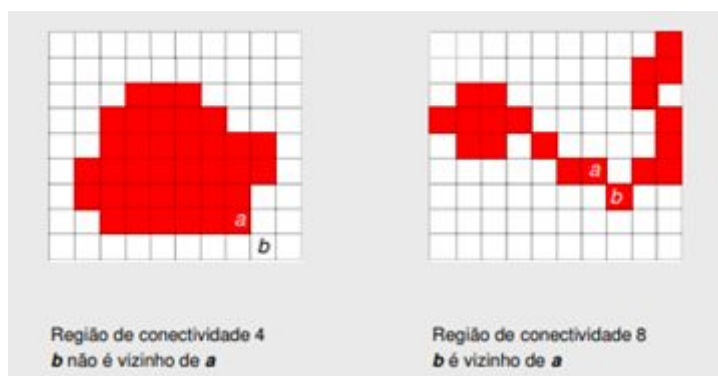
Algoritmos de Preenchimento de Regiões

Conetividade 4 – colora uma casa na vertical e horizontal.

Conetividade 8 – colora também na diagonal.



Dois pontos são vizinhos se se conseguir colorir o outro, diretamente a partir do ponto.



Preenchimento segundo contorno existente [flood-fill]

Limitado pelo contorno:

- Começa num ponto interior e “espalha-se” como se fosse líquido. Funciona em regiões com buracos.
- Algoritmo:

```
Algoritmo para região de conectividade 4:  
(contorno pode ser de conectividade 4 ou 8)  
  
void floodfill(int x, int y)  
{ if (pointColor(x,y) <> ContourColor && (pointColor(x,y) <> FillColor))  
  { ChangeColor(x,y, FillColor);  
    // apelo recursivo aos 4 vizinhos  
    floodfill(x+1,y);  
    floodfill(x-1,y);  
    floodfill(x,y+1);  
    floodfill(x,y-1);  
  }  
}
```

Problema: consumo de stack (pilha)
Soluções:
- Evitar declarar variáveis locais
- Não passar a cor de preenchimento como parâmetro...

Para região de conectividade 8: chama recursivamente a função floodfill() para os oito vizinhos. Para além dos indicados temos:
(x+1, y+1), (x-1, y+1), (x-1, y-1), (x+1, y-1)

Algoritmos de Preenchimento de Regiões

Limitado ao interior de região

- Se a fronteira não estiver completamente fechada, pode originar erro durante a execução.
- Evitam-se os erros se a leitura pointColor(x,y) fornecer o valor correspondente a ContourColor no caso do ponto se encontrar fora do ecrã.

· **Algoritmo:**

```
void floodfill(int x, int y)
{ if (pointColor(x,y) == RegionColor)
  { ChangeColor(x,y, FillColor);
    // apelo recursivo aos 4 vizinhos
    floodfill(x+1,y);
    floodfill(x-1,y);
    floodfill(x,y+1);
    floodfill(x,y-1);
  }
}
```

Preenchimento segundo contorno existente

Por análise do contorno [boundary algorithm]:

Algoritmo:

1. Parte de um ponto inicial, situado no interior, que começa por ser colocado na pilha.
2. Se pilha vazia, então termina, senão retira um ponto da pilha.
3. A partir desse ponto preenche na horizontal, para a direita e, em seguida, para a esquerda até encontrar o contorno. Toma nota das extremidades Xleft e Xright.
4. Na linha imediatamente abaixo procura, entre Xleft e Xright, os novos pontos de partida. Estes pontos são colocados na pilha.
5. Idem 4, para a linha imediatamente acima.
6. Salta para 2.

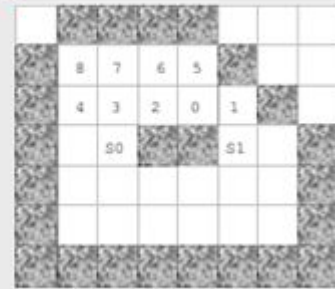
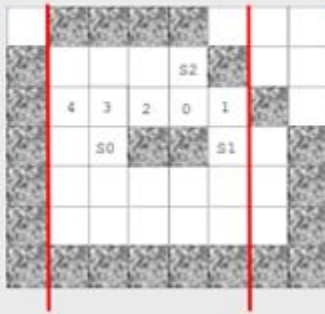
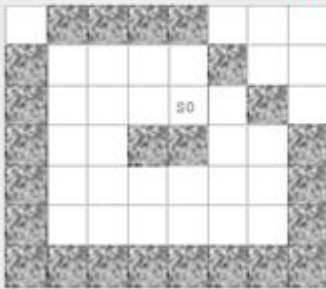
Os novos pontos de partida vão ser:

- Pixel de região que possui à sua direita um pixel de contorno.
- Pixel de região na coordenada XRight.

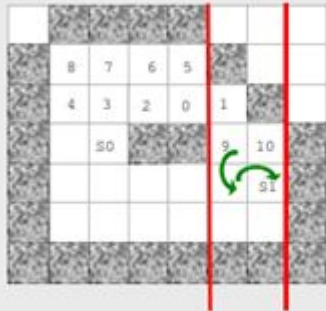
Exemplo:

Processa s0

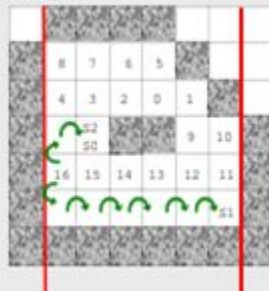
Processa s2



Processa s1



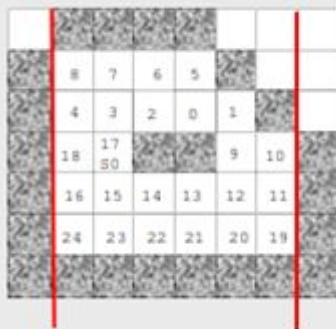
Processa s1



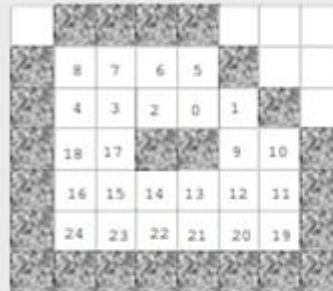
Processa s2



Processa s1



Processa s0



Pilha vazia - Fim

Preenchimento por varrimento segundo descrição de contorno

Algoritmo da lista de pontos de fronteira ordenados:

Algoritmos de Preenchimento de Regiões

Algoritmo:

1. Determinação dos Pontos de Fronteira

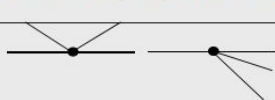
Intersecções das arestas com as linhas de varrimento do ecrã (utilizando, por exemplo, o algoritmo MidPoint modificado, de tal forma que produza um só ponto por horizontal)

2. Ordenação dos Pontos de Fronteira. Primeiro segundo Y e, em seguida, para o mesmo Y, segundo X.

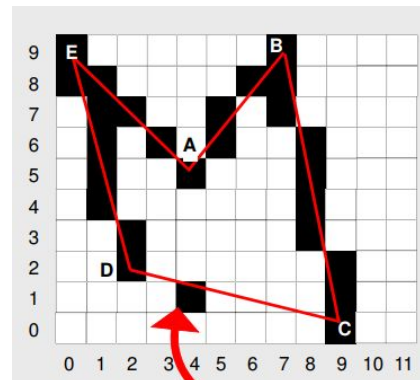
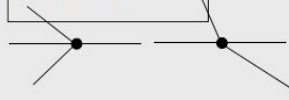
X_1, Y_1 precede X_2, Y_2 se $(Y_1 < Y_2)$ OR
se $(Y_1 = Y_2)$ AND $(X_1 \leq X_2)$

3. Os segmentos horizontais de preenchimento são agora especificados considerando pares de pontos consecutivos.

Os vértices duplos podem causar problemas:



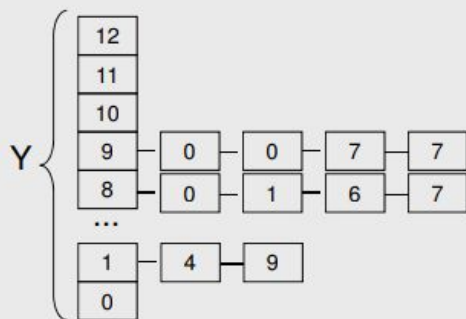
Vértices simples:



Neste caso os vértices duplos são o A, B, C e E.

Melhoramento: Algoritmo da tabela de listas de pontos ordenados

Consiste em construir uma lista ordenada de pontos para cada valor de Y.



Algoritmo:

1. Determinar as intersecções (x_i, y_i) para cada aresta. Para cada intersecção colocar x_i na lista y_i .
2. Em cada lista y_i , ordenar os valores X por ordem crescente.
3. Em cada lista y_i , considerar os pares de valores X consecutivos, que definem os segmentos horizontais a visualizar.

Em relação ao ponto 3 da primeira imagem, é o mesmo que dizer que tendo a lista ordenada, são pintados os sítios que se situam entre cada par de pontos ordenados.

Algoritmo da lista das arestas ativas:

Algoritmo da Lista das Arestas Ativas (AEL):

1. Constituição da Tabela das Arestas

- Para cada aresta é calculado e memorizado:
 - A coordenada X (valor inicial a partir do primeiro vértice).
 - DX, valor a adicionar a X, para encontrar o ponto seguinte da aresta quando se incrementa Y de 1.
 - LongY, altura da aresta segundo o eixo Y.

2. Para cada linha de varrimento:

- Verificar na tabela de arestas se existem novas arestas nesta linha. Em caso afirmativo, juntam-se à AEL.
- Ordenar AEL pelos valores de X.

3. Agrupa AEL aos pares de arestas, segundo os valores de X.

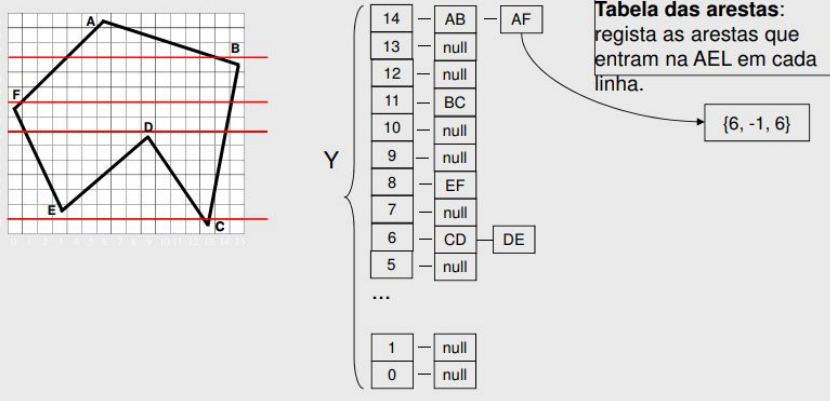
4. No final da linha preparar a informação para a linha seguinte:

Para cada Aresta Ativa:

Decrementar o valor LongY. Se LongY=0, então a aresta respetiva sai da lista das arestas ativas, senão é calculado o novo X, adicionando DX ao valor atual.

5. Voltar a 2.

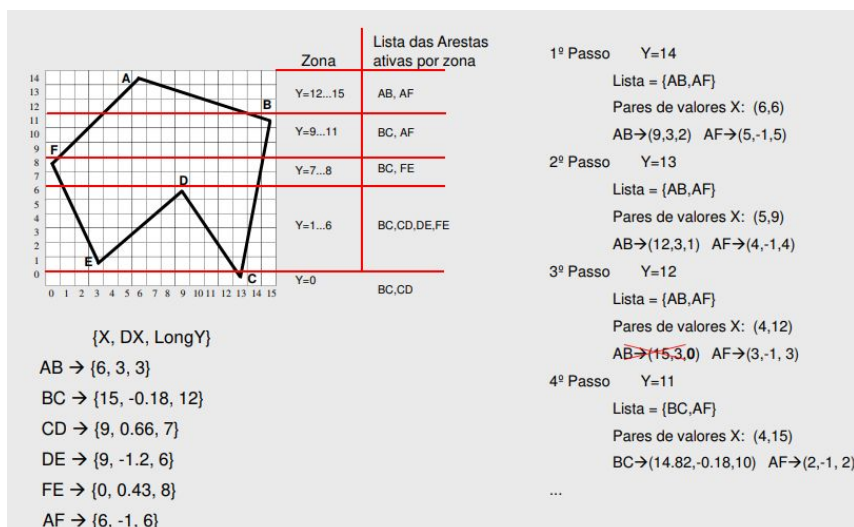
O primeiro passo do algoritmo será a classificação dos **vértices** em: **simples** ou **duplos**. A seguir constrói-se a tabela das arestas.



A análise é efetuada de cima para baixo e da direita para a esquerda.

- Vértice simples: $\text{longY} = y_2 - y_1$

- Vértice duplo: $\text{longY} = y_2 - y_1 + 1$



Perguntas Teóricas

Comente a afirmação: "A atenuação atmosférica no modelo de iluminação simplificado simula o escurecimento de objectos com o quadrado da distância à fonte de luz, mas usualmente utiliza-se uma variação linear".

Pelo modelo da física no mundo real, a iluminação reduz com o quadrado da distância, no entanto quando aplicamos isso no virtual a cena fica muito escura, pois o mundo virtual não tem em conta as imensas reflexões que acontecem entre os objectos.

Assim, muitas vezes acaba-se por usar uma variação linear, que faz com que os objectos não escureçam tão rapidamente, contrariando o facto de não haverem

reflexões, e cria assim uma aparência mais "natural" na cena, apesar de ser cientificamente incorreto.

Justifique a afirmação “Os algoritmos de cálculo de visibilidade do tipo lista de prioridade começam por trabalhar no espaço objecto e terminam no espaço imagem”.

Inicialmente trabalham no espaço objeto de modo a determinar a posição de cada objeto e saber qual a ordem que devem ser “pintados”. São depois desenhados trabalhando no espaço imagem

Comente a afirmação “Na modelação 3D, o modelo de varrimento espacial é utilizado, especialmente, como meio de interacção no acto de modelar; o produto resultante é normalmente convertido para o modelo de representação pela fronteira (Boundary Representation)”.

Na representação por varrimento, a utilização deste método pode resultar numa modelação ineficiente do objeto, exigindo, por exemplo um poder computacional muito elevado para o cálculo do volume ou pode, mesmo, gerar um sólido inválido. Por estas razões é comum fazer a conversão do modelo de varrimento para uma outra forma de representação. O modelo mais utilizado é o modelo de representação por fronteira.

Comente a afirmação “O algoritmo Ray-Tracing elimina o integral da equação de rendering, substituindo-o por uma ou duas parcelas”.

O ray-tracing substitui o rendering pelo cálculo da iluminação local e das reflexões recursivamente.

Comente a afirmação “O algoritmo de Atherton & Weiler permite reutilizar o cálculo de sombras numa sequência de várias imagens de vários pontos de vista desde que a cena 3D se mantenha estática, isto é, sem objectos em movimento”.

É verdadeira porque as sombras não dependem do visualizador, sendo possível reutilizar um cálculo de sombras em cálculos de imagem seguintes, desde que a cena se mantenha estática (fontes de luz e corpos sem se mexerem). mesmo que o observador se mexa, as sombras serão as mesmas, calculadas em função de onde estão as fontes luminosas.

Suponha que se encontra a projectar um sistema gráfico capaz de, entre outras facilidades, mapear Bump Textures. Diga, justificando, qual o sistema de Smooth Shading que utilizaria nesse sistema.

O método de Phong é o melhor porque o de gourand não tem em conta as normais e o de Phong tem. O bump texture simula textura criando normais.

Comente a afirmação no contexto do cálculo de visibilidade “Uma vantagem dos algoritmos do tipo Lista de Arestas Activas sobre o algoritmo Z-Buffer é o facto de não visitarem cada pixel mais do que uma vez; no entanto, apresentam a desvantagem de não serem conjugáveis com a técnica Back Face Culling”.

O algoritmo Z-Buffer visita cada pixel 2 vezes (Frame Buffer e Depth Buffer).

Comente a afirmação “Aherton & Weiller é um algoritmo de calculode projeção de sombras que só pode ser usado a cena comportar uma só fonte de luz”.

Falsa nas duas partes. É um algoritmo de cálculo de visibilidade e pode ser calculado com várias fontes de luz.

Comente a afirmação “O método de *smooth shading* de Gouraud é isento de qualquer efeito de *Mach Band*. Por esse motivo não é normalmente possível perceber a localização dos vértices e das arestas nas imagens produzidas”.

Falso. O efeito mach-band está presente no algoritmo de Gouraud.

Comente a afirmação “O algoritmo *Midpoint*, dado que só utiliza variáveis do tipo inteiro, facilita a sua implementação em *hardware*. No entanto, se implementado em *software*, torna-se mais lento que o algoritmo DDA (Digital Differential Analyser), dado que este último é bastante mais simples.

1a parte verdadeira (inteiros) e 2a parte falsa (mais rápido). Com int é mais rápido que com floats. O bresenham tem uma diferença mínima em relação ao Midpoint. Este usa a noção de ponto médio enquanto o Bresenham usa o erro entre o ponto real e o pixel, de resto é igual. O bresenham é bastante mais rápido que o DDA pois este usa aritmética com números reais.

Seja, em 2D, um espelho segundo a reta $y=2.x-1$. Determine, em notação simbólica, a matriz de transformação geométrica “reflexão” correspondente.

$T(0,b)*R(\arctan(m))*S(1,-1)*R(-\arctan(m))*T(0,-b)$

Comente a afirmação “Ray-Tracing é um algoritmo de iluminação global que dispensa a componente de iluminação ambiente no cálculo de iluminação de um ponto”.

Calcula a iluminação local num ponto e na iluminação local está implícita a ambiente.

Comente a afirmação “Os métodos de Iluminação Global produzem melhores resultados do que os métodos de Iluminação Local porque se baseiam, tal como o nome indica, no cálculo de iluminação na globalidade dos pixels que compõem a imagem”.

Os métodos de iluminação global são melhores que os de iluminação local mas não por terem mais pixels mas por terem em conta outras reflexões que os métodos de iluminação local não contabilizam. A iluminação local só tem em conta iluminação direta.

Comente a afirmação “RGB é um sistema de representação de cores bastante útil dado que, com apenas três componentes, consegue representar todo o espectro de cores visíveis”.

É falso, visto que o RGB apenas com 3 componentes não consegue representar todo o espectro de cores visíveis.