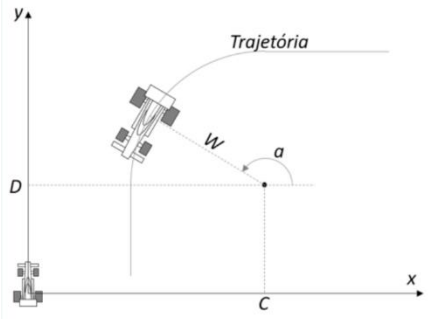
Computação gráfica

Transformações 2D + 3D

Garantir que:

- A **rotação** deve ser feita em (0,0) ou:
  - (**w,**0) / (0,**w**), caso se pretenda que a distância ao ponto seja **w**;
- **Scale** deve ser feito em (0,0);
- Aplicar **translação** no fim de todos os procedimentos



Modelos de Iluminação Locais

Atenuação com a distância:

- denominador = 1, sem atenuação
- denominador = d, atenuação linear
- denominador = d^2, atenuação quadrática

$$I = k_a I_a + \frac{k_d I_p}{d^2}\cos(\theta) + \frac{k_s I_p}{d^2} cos^n(\alpha)$$

Lei de Snell:

$$\sin(\theta_r) = \frac{\eta_i}{\eta_r}\sin(\theta_i)$$

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

(Observador)

Shading & Smooth Shading

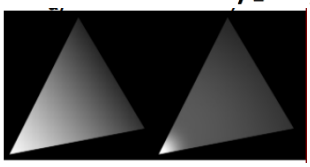
Nota-se a **malha poligonal** ⇒ Efeito de **Mach Band**

Algoritmo de Gouraud

$$I_a = I_1 - (I_1 - I_2)\,\frac{y_1 - y_s}{y_1 - y_2}$$

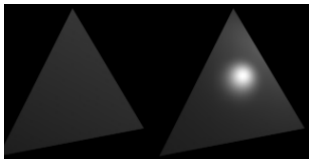
Preferência ao eixo dos **Y**'s, quando não é possível, usar os **X**'s.

Problema: O resultado depende da orientação do polígono



Método de Phong

Interpolação das normais, em vez da cor.



Texturas

- **Mapeamento de texturas**: Papel de parede
- **Bump Mapping Textures**: Sensação de relevo (rugoso)
- **Texturas 3D**: Evolui no interior dos objetos

Textura tem coordenadas normalizadas (u,v) Є [0,1] (**textels**)

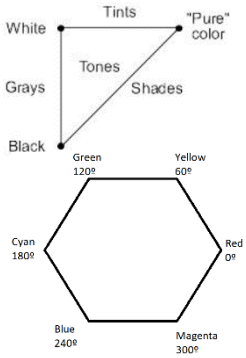
Projecção de Sombras

Algoritmo de Atherton & Weiller: Através da posição da luz, determinar as partes visíveis e pintá-las, o resto é sombra.

Ray Casting: Emitir um raio do observador (1..* pixéis), até chegar à fonte de luz. O primeiro objeto a intersetar o raio, define o objeto visível nesse pixel. A **sombra** é verificada se ao partir deste objeto interceta um outro objeto.

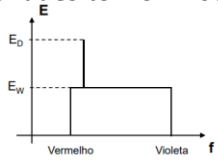
Luz & Cor

Luz Cromática: **HSV**



Todos os pontos na linha de ***Shades*** têm **S** = 100%, ***Tints*** têm **V** = 100%

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

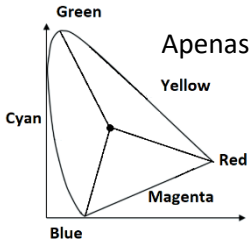
Quanto maior $E_D - E_W$, tanto mais pura será a cor emitida.

$E_W = 0$, pureza 100%

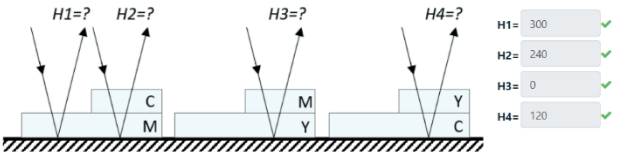
$E_W = E_D$, pureza 0% (branco)

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

Luminância = área abaixo da curva da energia total emitida



Apenas tem **H** e **S** (distância a White)



Se absorve **M**, resta **RB** = 300º; **CM** resta **B** = 240º

Cálculo de Visibilidade (*Rendering*)

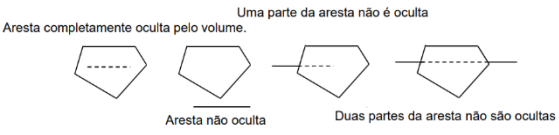
- **Algoritmos no espaço imagem**: Para cada pixel da imagem determinar qual o objeto visível;
- **Algoritmos no espaço objeto**: Comparar os objetos entre si de modo a selecionar a parte visível de cada um.

Backface culling: Tecnica para reduzir o nº de polígonos a processar (sensivelmente para 50%). **Front faces** ⇒ envia; **Back faces** ⇒ não envia; Apenas Verifica ∝ entre observador e polígonos.

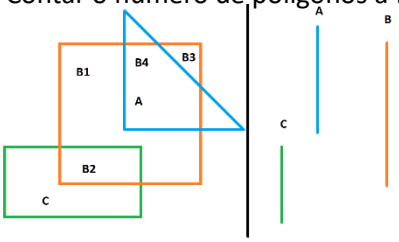
Algoritmos no espaço objecto

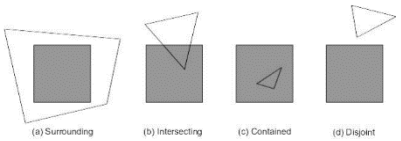
Ao Volume ⇒ supõe-se que uma aresta pode ser oculta pelo volume de um objeto. À Aresta ⇒ aresta contra aresta, verifica coerência permite determinar a invisibilidade de uma aresta a partir da invisibilidade de outra aresta que possua com ela um vértice comum. (**Coerência de aresta**: uma aresta só altera a sua visibilidade onde se cruza por trás de uma aresta visível.)

Quantitative Invisibility: Contar o número de polígonos à frente da aresta.



Atherton & Weiller:

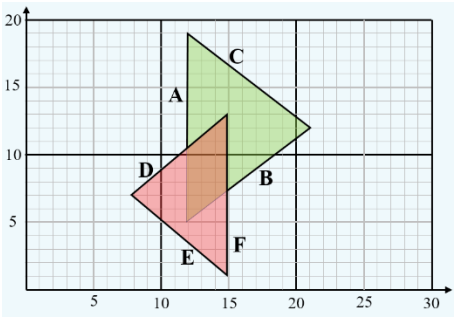




Algoritmo de Warnock: Divide a imagem em áreas retangulares e pinta de acordo com o maior polígono na área.

Algoritmos no espaço imagem

Algoritmo de Linha de Varrimento: **AEL** - Arestas Ativas, **ET** - Tabela de novas Arestas, **PT** - Tabela de Polígonos; Começa de baixo para cima, esquerda para a direita.



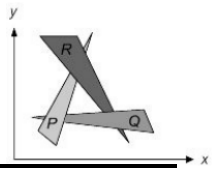
Algoritmo Z-Buffer: **Frame Buffer**: Cor em si; **Depth Buffer** / **Z-Buffer**: depth.
Cor inicial é a do background, depth é = 0.

Algoritmo Ray-casting: Verifica se a linha bate num polígono.

Algoritmos tipo Lista de Prioridades: Pinta as faces mais afastadas primeiro, e vai sobrepondo com as mais próximas.

Algoritmo de Newel, Newel & Sancha: Ordenar polígonos a Z, dividir polígonos se necessário e pintar do mais afastado ao mais próximo.

c)



Iluminação Global

O algoritmo de Ray Tracing: Parte do observador e vai somando as contribuições de reflexão/refração de cada polígono para o pixel final.
Vantagens: sombras, reflexões e refrações são facilmente incorporadas simula razoavelmente bem os efeitos especulares;
Desvantagens: tem custos computacionais elevados porque o custo de cálculo das intersecções é elevado não simula bem os efeitos de iluminação difusa;
Otimizações:

- diminuir o número de raios a processar: **Item Buffers**, **Adaptive Tree-Depth Control** (não vai a fundo nas arvores), **Light-Buffers** (a cada fonte de luz, associar os objetos que rodeiam: ***shadow feelers***)

- diminuir o número de interseções a testar: **Volumes Envolventes** (teste da caixa à volta do volume), **Organização Hierárquica dos Volumes Envolventes** (se um raio não interseta um volume, não interseta os que estão dentro dele), **Divisão Espacial em Grelhas Tridimensionais** (cada célula reconhece os objetos lá contidos)

Radiosity

Permite obter a quantidade de energia proveniente de **i** que atinge **j**.

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

energia expelida — energia emitida (produzida) — energia refletida

---

Representação de Curvas e Superfícies

Representações das curvas: **Malha poligonal**, Superficie paramétrica bicubica, Superficie quadrática;

Caracteristicas de uma malha poligonal: uma aresta liga 2 vertices; 1 poligono é uma sequencia fechada de arestas; uma aresta é ligada a 1 ou 2 poligonos (adjacentes); um vértice é partilhado pelo menos por 2 arestas; todas as arestas fazem parte de um polígono.

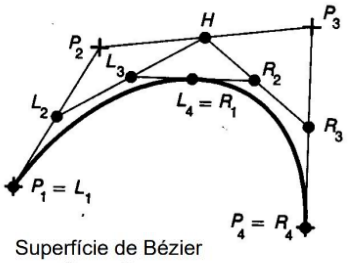
Representação por Apontadores para Lista de Vértices: cada polígono é representado por uma lista de índices.
Vantagens: Cada vértice da malha poligonal é guardado uma única vez na memória. A coordenada de um vértice é facilmente alterada;
Desvantagens: Difícil obter os polígonos que partilham uma dada aresta. As arestas continuam a ser desenhadas mais do que uma vez.

Representação por Apontadores para Lista de Arestas: cada polígono é representado por uma lista de apontadores para uma lista de arestas. Cada aresta aponta para os 2 vértices que a define, e os polígonos a que pertence.
Vantagens: O desenho é facilmente obtido percorrendo a lista de arestas. Não ocorre a repetição de arestas. Para o preenchimento dos polígonos trabalha-se com a lista de polígonos. Fácil efectuar a operação de clipping sobre os polígonos.
Desvantagens: Difícil determinar as arestas que incidem sobre o mesmo vértice.

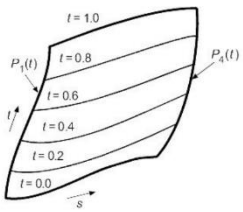
Curvas Cúbicas Paramétricas

G0 – continuidade geométrica zero
G1 – continuidade geométrica um
C1 – continuidade paramétrica 1

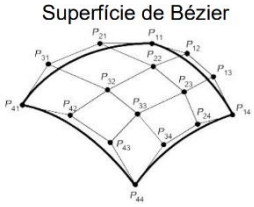
Hermite: $G_H = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}$ Bézier: $G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$ Conversão: $R_1 = 3.(P_2 - P_1)$ Algoritmo de Casteljau:



Superfície de Hermite

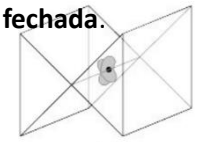


Superfície de Bézier



Modelação de Sólidos

Os sólidos são descritos pela sua superfície de fronteira: a representação mais comum é a **malha poligonal fechada**. Um solido tem fronteira ***2-manifolds***, se tiver **apenas** 2 faces a partilhar uma aresta.

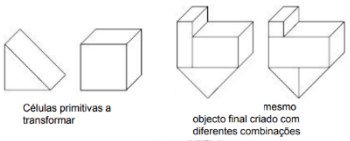

(c) não é ***2-manifold***

Formula de Euler**: V – E + F = 2**, F é as Faces, **não é condição suficiente para garantir que um objeto é um poliedro válido!! ⇒ requer que cada aresta seja partilhada por 2 faces e pelo menos 3 arestas partilham o mesmo vértice!!!**

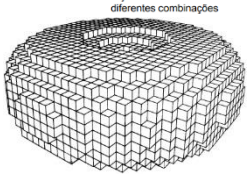
Generalização da Fórmula de Euler (para buracos): **V – E + F – H = 2 (C – G)**, onde **H** são os buracos (oco), **C** número de partes constituídas do objeto (por exemplo, dupla pirâmide, partilhando a base é 2), **G** número de buracos que atravessam totalmente.

Decomposição Espacial

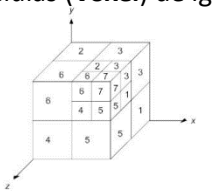
Decomposição Celular: União de células que não se intersetam!



Enumeração da Ocupação Espacial: dividido em células (**Voxel**) de igual dimensão



Octrees: divisão proporcional em octantes.



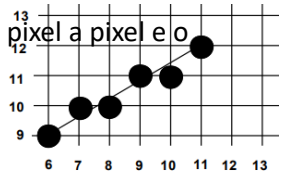
CSG – Constructive Solid Geometry: Objeto é obtido por combinação de células primitivas através de operadores booleanos

---

Desenho de Linhas

Algoritmo para desenho de Segmentos de Reta: tem de ser eficiente, a execução é chamada centenas/milhares de vezes. Devem criar linhas com aspeto satisfatório: **deve parecer retas.**

$y_{i+1} = y_i + m$, com **$\Delta x = 1$** e também tem de se garantir que **m<1**, desta forma, podemos desenhar pixel a pixel e o ponto a desenhar terá coordenadas (x_{i+1}, round(y_{i+1})).

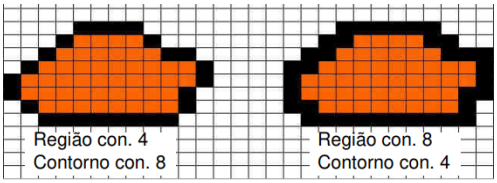


Algoritmo Midpoint: Apenas consideramos o 1º octante (0 <= m <= 1), se **M** estiver acima da reta, escolher **E** (abaixo), caso contrário escolher **ME** (acima). Obtém-se um error sempre inferior a ½.
Variável de decisão d_p como: $d_p = f(x_p + 1, y_p + ½)$, para calcular os pontos seguintes, apenas é preciso calcular o próximo d_p.

Otimização: $\mathbf{d_{p+1} = d_p + b - a}$, se $\mathbf{b_p} > 0$, se não $\mathbf{d_{p+1} = d_p + b}$, neste caso a vai representar se escolheu **E** ou **ME**;
Podemos fazer **2***(**b – a**) para garantir que não calha um valor não inteiro, isto não muda o algoritmo!

Para circunferências: O valor d_0 pode ser obtido considerando o primeiro ponto (0,R):
$d_0 = f(0 + 1, R - 1/2) = 1 + (R^2 – R + ¼) – R^2 = 5/4 - R$

Preenchimento de Regiões



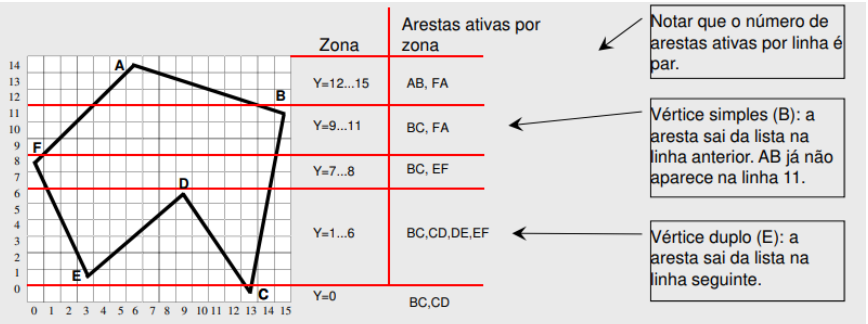
Por difusão [***flood-fill***] ou por análise do contorno [***boundary algorithm***]

Flood-fill: Preenche segundo o contorno existente, espalha como se fosse água. **Desvantagem**: Uso de stack. **Solução**: Não passar cor como parâmetro.

Garante-se que os cantos do ecrã estão já preenchidos pelo contorno!

Boundary Algorithm: 1. Colocar um ponto na pilha; 2. Retirar ponto da pilha; 3. Preencher na horizontal com cor (primeiro direita, depois esquerda) até ao contorno. Aponta como Xleft e Xright as extremidades; 4. Na linha abaixo (depois acima), **entre Xleft e Xright**, procura novos pontos de partida: antes de um contorno, ou em baixo de Xright.

Preenchimento por varrimento segundo descrição de contorno [*Scan Conversion*]:



- O preenchimento realiza-se por linha de varrimento do ecrã, pelo que será viável tratar memorizar apenas os pontos relativos a essa linha.
- Só as arestas ativas são usadas no preenchimento de uma linha de varrimento.
- → Interessa manter a **Lista das Arestas Ativas**.
- Esta lista é atualizada sempre que se entra numa nova zona vertical.

1º Passo Y=14
Lista = {AB,AF}
Pares de valores X: (6,6)
AB→(9,3,2) AF→(5,-1,5)

2º Passo Y=13
Lista = {AB,AF}
Pares de valores X: (5,9)
AB→(12,3,1) AF→(4,-1,4)

3º Passo Y=12
Lista = {AB,AF}
Pares de valores X: (4,12)
AB→(15,3,0) AF→(3,-1, 3)

4º Passo Y=11
Lista = {BC,AF}
Pares de valores X: (4,15)
BC→(14.82,-0.18,10) AF→(2,-1, 2)

Manter uma lista de arestas, com {**X, Δx, LongY**}, **Δx** representa a evolução a cada **y-1**, **LongY** o nº de linhas que vai estar ativo, sendo que: **LongY** = **y2 – y1** (**+1**, caso seja verticle duplo)