

## 1. INTRODUÇÃO À COMPUTAÇÃO E SISTEMAS GRÁFICOS

A **computação gráfica** baseia-se em **modelos matemáticos (modulação)**, resultantes do **reconhecimento de padrões (processamento de imagem)**, para fazer **síntese de imagem (rendering)** e vice-versa. Tem múltiplas **aplicações** como **CAD** (Desenho Assistido por Computador), **CAGD** (Desenho Geométrico Assistido por Computador), **SIG** (Sistemas de Informação Geográfica), **visualização científica/ médica/ educativa, lazer, etc.**

- **Gráficos "vetoriais":** representados por **coleções de objetos geométricos**, como pontos, retas, curvas, planos, polígonos. Permitem executar operações quase sem perda de precisão (devido a arredondamentos) e apresentam uma complexidade de processamento O ( $n^o$  vértices / vetores).
- **Gráficos "matriciais":** (amostragem em grelhas retangulares, imagens digitais, dados volumétricos). É uma representação **flexível** e **comum** com uma **complexidade de processamento igual a O (no de pixels)**, no entanto algumas operações envolvem **perda de precisão**.
- **Conversão de representações:** converter de **vetorial para matricial** designa-se de **rasterização ("scan conversion")** e ao **contrário** designa-se **reconhecimento de padrões**.
- **Processador (acelerador) gráfico:** é **hardware especializado**, que recorre a **parallelismo** para atingir **alto desempenho**, aliviando o **CPU** do sistema, no que toca a transformações, recortes (clipping), supressão de elementos fora da janela de visualização, projeção (3D para 2D), mapeamento de texturas, rasterização, amostragem de curvas e superfícies paramétricas, etc.

## 2. OPENGL BÁSICO

O **OpenGL** é uma **API** para **geração de gráficos** (2D, 3D, primitivas, imagens, etc.) **independente** da **plataforma e sistema de janelas** (meio de interação humano-máquina).

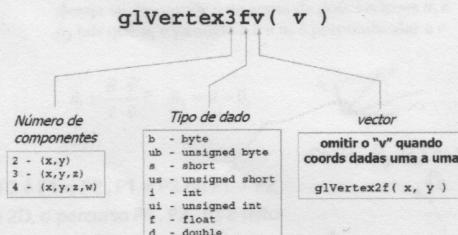
**Desenhar** com OpenGL **funciona como uma máquina de estados**, sendo que este tem rotinas para desenhar primitivas geométricas e imagens, Alterar/ consultar variáveis de estado (ex.: cor, material, fontes de iluminação, etc.). Apresenta **APIs relacionadas** como **GLU (OpenGL Utility Library)**, **AGL**, **GLX**, **WGL** e **GLUT (OpenGL Utility Toolkit)**.

- **Cabeçalhos:** os cabeçalhos do OpenGL são incluídos automaticamente, no entanto se o GLUT não for usado estes têm de ser incluídos explicitamente.
- **Registando CallBacks (GLUT):** as callbacks são rotinas para **tratamento de eventos**, tendo de ser registradas através da função **glutXXXfunc (callback)**, onde XXX é uma classe de eventos e callback é o nome da rotina.
- **Callback de desenho:** é uma rotina chamada automaticamente para efetuar o redesenho da janela ou parte dela. (ex: void display(void) {...}).
- **Callback de redimensionamento:** chamada sempre que a janela é redimensionada. Se não for especificada o GLUT apresenta uma rotina por defeito.
- **Outras Callback importantes:** keyboard, mouse, motion, passivemotion, idle.
- **Inicialização do GLUT:** primeiro é necessário estabelecer contacto com o sistema de janelas do SO (**glutInit (int\* argc, char\*\* argv)**), para se puder **iniciar as janelas**, especificando o tipo de recursos necessários (**glutInitDisplayMode (int modo)**), a **posição inicial do canto sup. Esq.** (**glutInitWindowPosition (int x, int y)**) e o **tamanho** (**glutInitWindowSize (int width, height)**). De seguida, **cria-se as janelas** (**int glutCreateWindow (char\* nome)**).
- **Ciclo principal:** registados os callbacks, o **controlo é entregue ao sistema de janelas**: **glutMainDisplayLoop (void)** – **rotina despachante de eventos**.
- **Primitivas de desenho:** como se pode abaixo entre **glBegin ()** e **glEnd ()** apenas alguns comandos podem ser usados. Ex.: **glMaterial**, **glNormal**, **glTexCoord**. Uma vez emitido um vértice (**glVertex**), este é desenhado com as propriedades pré-definidas (cor, material, normal, coordenadas de textura etc.) nas variáveis de estado correspondentes.

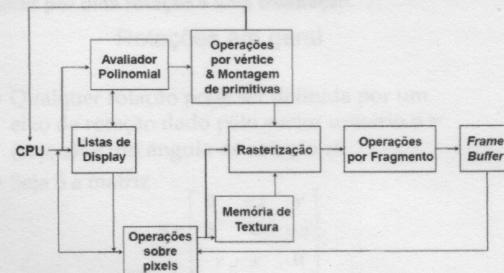
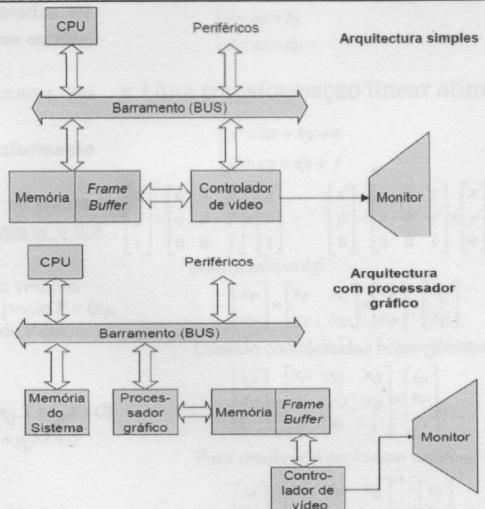
```
glBegin( PRIMITIVA );
    especificação de vértices, cores, coordenadas de textura, propriedades
    de material
glEnd();
```

### Exemplo de desenho:

```
void drawRhombus( GLfloat color[] )
{
    glBegin( GL_QUADS );
    glColor3fv( color );
    glVertex2f( 0.0, 0.0 );
    glVertex2f( 1.0, 0.0 );
    glVertex2f( 1.5, 1.118 );
    glVertex2f( 0.5, 1.118 );
    glEnd();
```



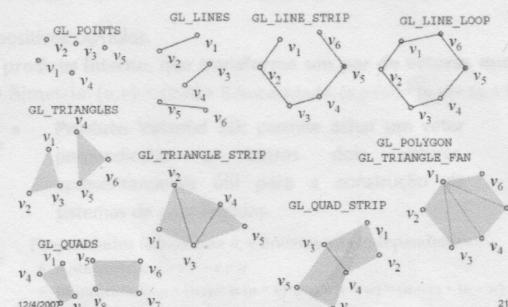
- **Controlando as cores:** se as cores forem **especificadas diretamente (default)** usa-se: **glColorIndex ()** ou **glColor ()**. Se forem **calculadas a partir de um modelo de iluminação** é necessário: **ligar a iluminação** com  **glEnable (GL\_LIGHTING)**; **escolher modelo de sombreamento**, que pode ser **constante por face** (**glShadeModel (GL\_FLAT)**) ou **Gouraud** (**glShadeModel (GL\_SMOOTH)**); **ligar** pelo menos uma **fonte de luz** ( **glEnable(GL\_LIGHT0)**); **especificar propriedades da(s) fonte(s) de luz** com  **glLight()**; **especificar propriedades de material de cada objeto**: **glMaterial()**; **especificar normais de cada face ou de cada vértice** com  **glNormal()**.



```
#include <GL/glut.h>
/* Outros headers */
void display (void)
{
    ...
}
/* Outras rotinas callback */
int main (int argc, char *argv[])
{
    glutInit (argc, argv);
    glutInitDisplayMode ( modo );
    glutCreateWindow ( nome_da_janela );
    glutDisplayFunc( displayCallback );
    glutReshapeFunc( reshapeCallback );
    /* Registo de outras rotinas callback */
    glutMainLoop();
    return 0;
}
```

Arquivos associados:
 - **Cabeçalhos**: glut.h, glutint.h, glut/glut.h, glut/glutint.h.
 - **Rotinas callback**: display, reshapeCallback, displayCallback, reshapeCallback.
 - **Inicialização do GLUT**: glutInit.
 - **Inicialização da janela**: glutCreateWindow.
 - **Registo de callbacks**: glutDisplayFunc, glutReshapeFunc.
 - **Ciclo principal**: glutMainLoop.

## OpenGL – Primitivas de desenho



## OpenGL – Exemplo de Inicialização

```
void myinit(void)
{
    GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
    GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
    glShadeModel (GL_SMOOTH);
```

### 3. GEOMETRIA

- Pontos e vetores 2D:** um ponto denota de posição no plano e um vetor denota de deslocamento, ou seja, direção e magnitude. São  $P = (x_p, y_p)$  e  $\vec{v} = (x_v, y_v)$  ambos descritos por pares de coordenadas. Há várias operações que podem ser aplicadas tais como soma de vetores, multiplicação de vetores por escalar, subtração de pontos e soma de ponto com vetor.
- Transformação:** é uma função que faz corresponder pontos de um espaço Euclídeo a outros (ou possivelmente os mesmos) pontos do mesmo espaço. Ex: transformação linear.
  - Transformação de vetores:** um vetor não está aplicado a um ponto no espaço, logo uma transformação linear afim aplicada a um vetor não inclui translação.
  - Coordenadas homogêneas:** a transformação de vetores é operacionalmente diferente da de pontos, sendo que coordenadas homogêneas permitem unificar o tratamento. A Coordenada extra  $w = 0$  é usada para vetores e  $w = 1$  para pontos.
- Sistema de coordenadas:** um sistema de coordenadas para  $R^n$  é definido por um ponto (origem) e  $n$  vetores. Ex.: Seja um sistema de coordenadas para  $R^2$  definido pelo ponto  $O$  e os vectores  $X$  e  $Y$ . Então
  - Um ponto  $P$  é dado por coordenadas  $x_p$  e  $y_p$  tais que  $P = x_p \cdot X + y_p \cdot Y + O$
  - Um vetor  $V$  é dado por coordenadas  $x_V$  e  $y_V$  tais que  $V = x_V \cdot X + y_V \cdot Y$

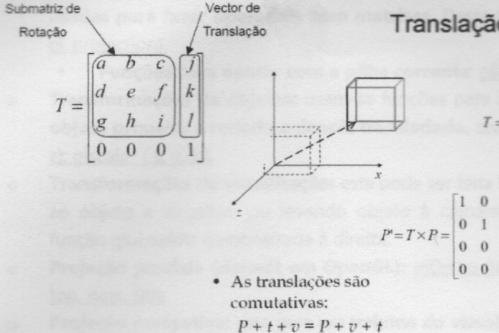
- Como calcular as coordenadas de um ponto  $P = (x_p, y_p)$  em  $O/X/Y$  dadas as coordenadas de  $P$  em  $Q/T/U$ , isto é,  $(t_p, u_p)$ ?

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

Logo,

$$\begin{aligned} x_p &= t_p \cdot x_T + u_p \cdot x_U + x_Q \\ y_p &= t_p \cdot y_T + u_p \cdot y_U + y_Q \end{aligned}$$

- Transformações rígidas:** não modificam a forma (dimensões/ângulos) do objeto, sendo compostos por uma rotação e uma translação.



- Inclinação:** é uma transformação de deformação em que o eixo é entortado, como por exemplo quando o vetor unitário do eixo z é transformado em  $[Sh_x \ Sh_y \ 1, 0]^T$ .
- Escalamento:** é especificado por três fatores que multiplicam os vetores unitários, sendo que esta transformação não é considerada rígida. Designa-se escalamento uniforme quando os fatores tomam todos o mesmo valor.
- Composição:** quando se submete várias transformações em conjunto.
- Geometria afim:** é a geometria composta dos elementos básicos (escalares, pontos, vetores) e de operações básicas (escalar . vector = vector; vector + vector ou vector - vector = vector; ponto - ponto = vector; ponto + vector ou ponto - vector = ponto).
- Combinações afim:** é uma maneira especial de combinar pontos  $\alpha_1 P_1 + \alpha_2 P_2 + \dots + \alpha_n P_n$  onde  $\sum_{i=1}^n \alpha_i = 1$ . Para 2 pontos  $P$  e  $Q$  poderíamos ter uma combinação 1 onde afim  $R = (1 - \alpha)P + \alpha Q = P + \alpha(Q - P)$ .
  - Combinações convexas:** combinação afim onde se garante que todos os coeficientes são positivos ou nulos.
- Geometria Euclidiana:** extensão da geometria afim pela adição de um operador chamado **produto interno**, que transforma um par de vetores num escalar. Apresenta as seguintes propriedades:
  - Positividade:  $(u, u) \geq 0$  e  $(u, u) = 0$  sse  $u = 0$
  - Simetria:  $(u, v) = (v, u)$
  - Bilinearidade:  $(u, v+w) = (u, v) + (u, w)$
  - $(u, \alpha v) = \alpha(u, v)$
- Normalmente usamos o produto escalar como operador de produto interno:
- Distância entre dois pontos  $P$  e  $Q$  é  $|Q - P|$
- O ângulo entre dois vectores pode ser determinado por
- Projecção ortogonal: dados dois vectores  $u$  e  $v$ , deseja-se decompor  $u$  na soma de dois vectores  $u_1$  e  $u_2$  tais que  $u_1$  é paralelo a  $v$  e  $u_2$  é perpendicular a  $v$
- Comprimento de um vector é definido como:  $\|\vec{v}\| = \sqrt{\vec{v} \cdot \vec{v}}$
- Vector unitário (normalizado):  $\hat{v} = \frac{\vec{v}}{\|\vec{v}\|}$
- Or (P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>) = -1
- Or (P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>) = +1
- Or (P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub>) = sign  $\begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}$
- Or<sub>2</sub>(P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>) = sign  $\begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}$
- Or<sub>3</sub>(P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub>) = sign  $\begin{pmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{pmatrix}$

- Uma transformação linear**

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

- Uma transformação linear afim**

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

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

Matricialmente:

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} x_T & x_U & x_Q \\ y_T & y_U & y_Q \end{bmatrix} \times \begin{bmatrix} t_p \\ u_p \\ 1 \end{bmatrix} + \begin{bmatrix} x_Q \\ y_Q \end{bmatrix}$$

Usando coordenadas homogêneas:

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_T & x_U & x_Q \\ y_T & y_U & y_Q \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} t_p \\ u_p \\ 1 \end{bmatrix}$$

Para resolver o problema inverso:

$$\begin{bmatrix} t_p \\ u_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_T & x_U & x_Q \\ y_T & y_U & y_Q \\ 0 & 0 & 1 \end{bmatrix}^{-1} \times \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix}$$

### Rotações em geral

- Qualquer rotação pode ser definida por um eixo de rotação dado pelo vetor unitário  $u = (x, y, z)^T$  e um ângulo de rotação  $\alpha$

- Seja  $S$  a matriz

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

- Então a submatriz de Rotação  $M$  é dada por  $M = uu^T + (\cos\alpha)(I - uu^T) + (\sin\alpha)S$

- Qualquer rotação pode ser definida por um eixo de rotação dado pelo vetor unitário  $u = (x, y, z)^T$  e um ângulo de rotação  $\alpha$
- Seja  $S$  a matriz
- Então a submatriz de Rotação  $M$  é dada por  $M = uu^T + (\cos\alpha)(I - uu^T) + (\sin\alpha)S$

- Qualquer rotação pode ser definida por um eixo de rotação dado pelo vetor unitário  $u = (x, y, z)^T$  e um ângulo de rotação  $\alpha$
- Seja  $S$  a matriz
- Então a submatriz de Rotação  $M$  é dada por  $M = uu^T + (\cos\alpha)(I - uu^T) + (\sin\alpha)S$

- Qualquer rotação pode ser definida por um eixo de rotação dado pelo vetor unitário  $u = (x, y, z)^T$  e um ângulo de rotação  $\alpha$
- Seja  $S$  a matriz
- Então a submatriz de Rotação  $M$  é dada por  $M = uu^T + (\cos\alpha)(I - uu^T) + (\sin\alpha)S$

Propriedades (assume-se  $u, v$  linearmente independentes):

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

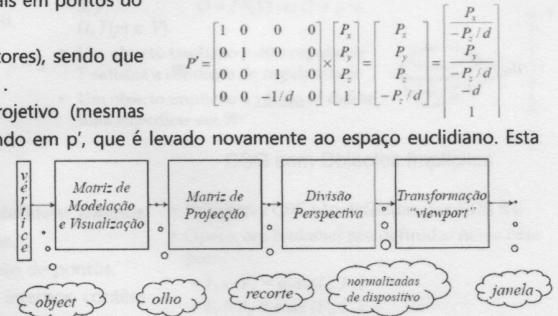
Or (P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>) = +1

Or (P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub>) = 0

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{pmatrix}$$

## 4. PROJEÇÕES

- Perspetiva** é o estudo das **transformações projetivas**, projetando mo espaço dimensional usando **um ponto chamado de centro de projeção**. Estas transformações não preservam as combinações afim.
- Geometria projetiva:** assume a existência de dois pontos ideais (no infinito), sendo que: retas paralelas encontram-se num ponto ideal; para não haver mais de um ponto ideal para cada inclinação de reta, assume-se que o plano projetivo se fecha sobre si mesmo; em 2d o plano projetivo tem uma fronteira que é uma reta no infinito (feita de pontos ideais); transformações projetivas podem transformar pontos ideais em pontos do plano euclidiano e vice-versa; problemas: o plano projetivo é uma variedade não orientável.
- Coordenadas homogéneas em espaço projetivo:** representa-se apenas pontos (e não vetores), sendo que estes em coordenadas homogéneas  $[x \ w]^T$ , representam-se canonicamente por  $[x/w \ y/w \ 1]^T$ .
- Projeção perspetiva de um ponto:** o ponto é levado do espaço euclidiano para o projetivo (mesmas coordenadas homogéneas), é multiplicado pela matriz de transformação perspetiva resultando em  $p'$ , que é levado novamente ao espaço euclidiano. Esta operação é designada de **operação de divisão perspetiva**.
- Transformações em OpenGL:** **modelação** (mover /deformar os objetos); **visualização** (mover e orientar a câmera); **projecção** (ajuste da lente / objetiva da câmera); "**viewport**" (aumentar ou reduzir a fotografia). A imagem ao lado é o **pipeline de transformações**.
  - Estado inicial do pipeline:** as matrizes "modelview" e "projection" são **matrizes-identidade**. O mundo visível é restrito ao cubo  $-1 \leq x, y, z \leq 1$  e a transformação "viewport" transforma o quadrado  $-1 \leq x, y \leq 1$  (em coordenadas normalizadas de dispositivo) na **superfície total da janela**.
  - Especificação do "viewport":** para especificar a área da janela na qual será **transformado o quadrado do plano de projeção**, utiliza-se `glviewport(x0, y0, largura, altura)`.
  - Especificação de transformações:** as matrizes **modelview** e **projection usadas no pipeline** são aquelas que se situam no topo de duas pilhas que são usadas para fazer **operações com matrizes**. Desta forma para **selecionar em qual pilha queremos operar**, usamos `glmatrixmode(gl_modelview ou gl_projection)`.
    - Funções para operar com a pilha corrente:** `glLoadIdentity()`; `glPushMatrix()`; `glLoadMatrix*` (); `glPopMatrix()`; `glMultMatrix*` ()
  - Transformações de objetos:** usam-se funções para **multiplicar o topo da pilha de matrizes por transformações especificadas por parâmetros**. O objeto primeiro é **rodado e depois transladado**, sendo que as **funções são executadas** na seguinte **ordem**: `gltranslate*(x, y, z)`; `glrotate*(ângulo, x, y, z)`; `glScale*(x, y, z)`.
  - Transformações de visualização:** esta pode ser feita levando a câmera ao objeto a visualizar ou levando objeto à câmera. Utilizando-se a função `gluLookAt` demonstrada à direita.
  - Projeção paralela (default em OpenGL):** `glOrtho(left, right, bottom, top, near, far)`;
  - Projeção perspetiva:** esta gera um **volumen de visualização simétrico centrado sobre o eixo z**. Utiliza-se a função: `gluPerspective(fovy, aspect, near, far)`:



## Transformação perspetiva em coordenadas homogéneas

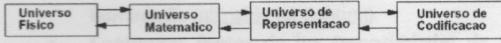
Não existe matriz  $4 \times 4$  capaz de realizar tal transformação em espaços euclidianos, mas se assumirmos que o ponto está no espaço projectivo, então

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

- Exemplo:**
- void resize( int w, int h )**
- glViewport( 0, 0, (GLsizei) w, (GLsizei) h );**
- glMatrixMode( GL\_PROJECTION );**
- glLoadIdentity();**
- gluPerspective( 65.0, (GLdouble) w / h, 1.0, 100.0 );**
- glMatrixMode( GL\_MODELVIEW );**
- glLoadIdentity();**
- gluLookAt( 0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0 );**
- Cuidado com casos degenerados**

## 5. MODELAÇÃO

- A necessidade de paradigmas (Ari Requicha).



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

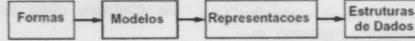
- Descrição de sólidos:** é um **conjunto tridimensional de pontos**, descritos por fronteiras ou por campos escalares (equações/ amostrados). Originam três tipos de representações: por **fronteira (B-rep – Boundary Representation)**; por **operações de conjuntos (CSG – Constructive Solid Geometry)**; por **enumeração do espaço em células (BSP-trees, Octrees, etc.)**.

- Representação por fronteira:** sólido definido pela superfície que o delimita, sendo estas compactas e sem bordas, **descritas por parametrização** - sistema de coordenadas sobre a superfície herda de um sistema de coordenadas no plano. Normalmente toda a superfície não pode ser coberta por apenas uma parametrização.
  - Parametrização válida:** a superfície não se pode intersetar a si mesma; o vetor normal não se anula à superfície; normal é utilizado para determinar o interior e exterior do sólido.
- Parametrização do círculo:** a forma implícita é:  $y = tx + t$ ;  $x^2 + y^2 = 1$ . Resolvendo-se esse sistema chega-se a uma parametrização alternativa:  $x(t) = \frac{1-t^2}{1+t^2}$ ;  $y(t) = \frac{2t}{1+t^2}$ ;  $t \in (-\frac{\pi}{2}, \frac{\pi}{2})$
- Representação linear por partes:** uma superfície com geometria complexa pode ser aproximada por uma superfície linear por partes, particionado o domínio da parametrização em polígonos. (Exemplo: esfera desenhada com triângulos (polígono), ligados através dos vértices adjacentes).
  - Malha poligonal:** definida por um conjunto adjacente de vértices, arestas, designado por **topologia da superfície**.

### • Codificação:

- Codificação explícita:** é a mais simples, sendo que **cada face armazena uma lista ordenada dos seus vértices** (as coordenadas), o que resulta em **redundância de informação e em consultas complicadas**.
- Ponteiros para listas de vértices:** os **vértices são armazenados separadamente numa lista**, sendo que as **faces os referenciam por ponteiros**. Esta forma resulta em economia de memória, apesar de achar adjacências ainda ser complicado e as arestas serem desenhadas em duplicado.
- Ponteiros para listas de arestas:** as arestas são armazenadas numa lista, sendo estas referenciadas pelas faces por ponteiros.
- Winged-Edge:** armazena a informação na estrutura associada as arestas. Todos os **9 tipos de adjacência** (imagem à direita) entre **vértices, arestas e faces** são determinados em tempo constante.
- Radial-Edge:** representa **objetos non-manifold (não variedades)** e armazena a lista ordenada de faces incidentes numa aresta, sendo mais complicado que winged-edge.

## Esquemas de Representação

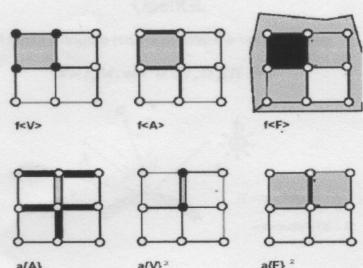
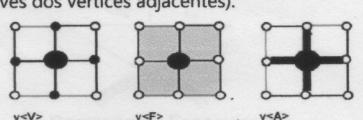


• Parametrização da esfera de raio unitário, centrada na origem

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

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

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



- **Representação implícita:** o sólido é definido por um conjunto de valores que caracterizam os seus pontos, sendo as superfícies dos objetos definidas implicitamente por equações, sendo **F** designada a função **implícita**. Sendo que uma superfície definida de forma implícita pode apresentar auto interseção.

### Teorema da Função Implícita

- Seja  $F : \mathbb{R}^n \rightarrow \mathbb{R}$  definida num conjunto aberto  $U$
- Se  $F$  possui derivadas parciais contínuas em  $U$  e  $\nabla F \neq 0$  em  $U$ , então  $F$  é uma subvariedade de dimensão  $n - 1$  do  $\mathbb{R}^n$ 
  - Superfície sem auto-intersecção
- **Classificação dos pontos no espaço da função F:**
  - $F > 0 \Rightarrow p \in$  exterior de  $O$ .
  - $F = 0 \Rightarrow p \in$  fronteira de  $O$ .
  - $F < 0 \Rightarrow p \in$  interior de  $O$ .

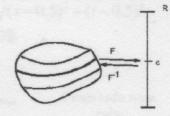
- **Esquema de representações CSG (Constructive Solid Geometry):** as operações CSG definem objetos através de operações (união, intersecção e diferença) regularizadas de conjuntos de pontos.

- **Objeto regular:** o fecho do interior do seu conjunto de pontos é igual ao próprio conjunto de pontos.
- **Árvore CSG:** um **modelo CSG** é codificado por uma árvore, em que os objetos internos contêm operações de conjunto ou transformações lineares afins e as folhas contêm objetos primitivos.
- **Representações por campos escalares:** numa b-rep as intersecções estão representadas explicitamente e é mais fácil exibir um ponto sobre a superfície do objeto, porém é difícil determinar este se está no interior, fronteira ou exterior do objeto. As operações booleanas também se tornam complicadas.
- **Representações por células:** dividem o espaço em sub-regiões convexas, que podem ser **grelhas** (cubos de tamanho igual), **octrees** (cubos cujos lados são potências de 2) e **bsp-trees** (poliedros convexos). As células são atribuídas valores de um campo escalar  $f(x, y, z)$ , que é assumido como constante dentro desta. Desta forma, um sólido é definido como o conjunto de pontos tais que a  $f(x, y, z) < b$  para valores  $a$  e  $b$  estipulados
- **Ambiguidade e unicidade:** uma representação é única quando o modelo associado possui uma única representação e é ambígua quando pode representar mais de um modelo. De notar que uma representação ambígua é catastrófica (wireframe), inviabilizando máquinas de controlo numérico.

### Valores Regulares

- Um valor  $c$  é dito **regular** se  $F^{-1}(c)$  não contém pontos onde  $\nabla F = 0$  (pontos singulares)
- $\forall p \in F^{-1}(c) \Rightarrow \nabla F_p = \left( \frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z} \right)|_p \neq 0$ .
- Neste caso interessam apenas os casos em que  $n = 2$  ou 3 (curvas e superfícies implícitas)

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



### CSG com Objectos Implícitos

- Primitivas CSG são definidas por  $F_i(X) \leq 0$
- Operações booleanas são definidas nesse caso por:

$$\begin{aligned} F_1 \cup F_2 &= \min(F_1, F_2) \\ F_1 \cap F_2 &= \max(F_1, F_2) \\ F_1 / F_2 &= F_1 \cap F_2 = \max(F_1, -F_2) \end{aligned}$$

Conversão entre Representações

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

## 6. ILUMINAÇÃO

A iluminação é o estudo de como a luz interage com os objetos de uma cena (emissão, transmissão, absorção, refração e reflexão). Em **modelos físicos** a luz é modelada como radiação eletromagnética e leva em conta as intersecções, o que se torna intratável computacionalmente.

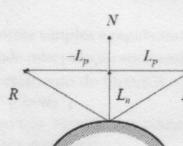
- **Modelos de iluminação em CG:** a luz é amostrada num número discreto de primárias (cor).
- **Modelos locais:** apenas caminhos do tipo fonte luminosa → superfície → olho são tratados. Ex: OpenGL.
- **Modelos globais:** muitos caminhos (*ray tracing*, radiosidade), logo complexos.
- **Iluminação em OpenGL:** assume **fontes pontuais de luz**, sendo as **interações** desta com as superfícies **modeladas nas componentes emissão, ambiente, difusa e especular**. O OpenGL dá suporte a efeitos atmosféricos como fog e atenuação.
  - **Ligar fontes de luz:** usa-se `glEnable(source)`, sendo **source** uma **constante** cujo nome é `gl_light`, começando com `gl_light0`. É necessário também **ligar o cálculo de cores pelo modelo de iluminação** (`glEnable(gl_lighting)`).
  - **Configurar fontes de luz (propriedades da fonte):** `glLightfv(source, property, value)`, sendo **property** uma **constante** (ver usos na imagem à direita).
  - **Propriedades do material:** especificados por `glMaterialfv(face, property, value)`, sendo que **Face** designa quais os **lados da superfície que se quer configurar** (`GL_FRONT`, `GL_BACK`, `GL_FRONT_AND_BACK`) e **Property** designa a **propriedade do modelo de iluminação** (`GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`, `GL_EMISSION`, `GL_SHININESS`).
  - **Geometria:** a posição dos vértices em **relação ao olho e à fonte luminosa contribui para o cálculo dos efeitos atmosféricos**, sendo a normal fundamental. No entanto esta não é calculada automaticamente e precisa de ser especificada com `glNormal()`.
  - **Componentes do modelo de Phong:**
    - **Emissão:** contribuição que não depende de fontes de luz (fluorescência).
    - **Ambiente:** contribuição que não depende da geometria.
      - **Iluminação ambiente:** componente que modela como uma constante o efeito da reflexão de outros objetos do ambiente. Está dependente dos coeficientes `GL_AMBIENT` tanto das fontes luminosas quanto dos materiais. `LightModelfv(GL_LIGHT_MODEL_AMBIENT, params)`.
    - **Difusa:** contribuição correspondente ao espalhamento da reflexão *lambertiana* (independente da posição do observador).
      - **Iluminação Difusa (reflexão lambertiana):** recebida por uma superfície e que é **refletida uniformemente em todas as direções**. É **característica de materiais baços ou foscos**, sendo que a luminosidade aparente da superfície não depende do observador, mas apenas do coseno do ângulo de incidência da luz.
    - **Especular:** contribuição referente ao comportamento de superfícies polidas.
      - **Iluminação especular:** simula a **reflexão à maneira de um espelho** (objetos altamente polidos), sendo que **depende da posição do observador**, do objeto e da fonte de luz. No **modelo de Phong** simulam-se **refletores imperfeitos**, assumindo que luz é refletida segundo um cone cujo eixo passa pelo observador.
      - **Coeficiente de especularidade:** indica o quanto polida é a superfície.
  - **Atenuação:** para fontes de luz posicionais ( $w=1$ ), é possível definir um fator de atenuação que leva em conta a distância  $d$  entre a fonte de luz e o objeto iluminado. Coeficientes são definidos pela função `glLight()`, por omissão não há atenuação ( $c0=1$ ,  $c1=c2=0$ ).
  - **Cor = Amb + E +  $\sum A_i + \text{aten}(D_i + S_i)$**

**Property** é uma constante designando:

- Coeficientes de cor usados no modelo de iluminação
  - `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`
- Geometria da fonte
  - `GL_POSITION`, `GL_SPOT_DIRECTION`, `GL_SPOT_CUTOFF`, `GL_SPOT_EXPONENT`
- Coeficientes de atenuação
  - `GL_CONSTANT_ATTENUATION`, `GL_LINEAR_ATTENUATION`, `GL_QUADRATIC_ATTENUATION`

### Cálculo do vector da luz reflectida

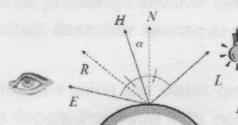
$$R = L_n - L_p = (N \cdot L)N - (L - L_n) = 2(N \cdot L)N - L$$



Componente Especular em OpenGL

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

$$S = I_s M_s \cos^\alpha \alpha = I_s M_s (H \cdot N)^\alpha$$



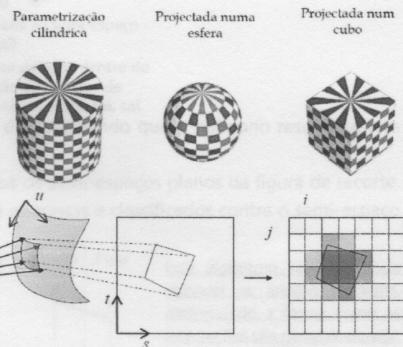
$$H = \text{normalizar}\left(\frac{E + L}{2}\right)$$

## 7. TEXTURAS

Os modelos de iluminação não são apropriados para descrever todas as diferenças de cor observáveis numa superfície, sendo estas diferenças/ efeitos modelados usando uma técnica chamada **mapeamento de textura**.

- **Propriedades mapeáveis:** cor (coeficientes de reflexão difusa), coeficientes de **reflexão especular** e **difusa** ("environment mapping"), perturbação do **vector normal** ("bump mapping"), perturbação da **superfície na direção da normal** ("displacement mapping") e **transparência / opacidade**.
- **Espaço de textura:** texturas 2D são funções  $T(s, t)$  cujo domínio é um espaço bidimensional e o contradomínio pode ser cor, opacidade, etc. É comum ajustar a escala da imagem de forma que fique enquadrada no intervalo  $0 \leq s, t \leq 1$ . A função de textura pode ser definida algebraicamente:  $T(s, t) = \sqrt{(s - 0.5)^2 + (t - 0.5)^2}$
- **Função de mapeamento:** devolve o ponto do objeto correspondente a cada ponto do espaço de textura  $(x, y, z) = F(s, t)$ . Corresponde à forma com que a **textura é usada para "embrulhar" (wrap) o objeto**, sendo que recorrentemente é **necessária** uma **função** que permita "desembrulhar" (unwrap) a textura do objeto, isto é, a inversa da função de mapeamento. Se a superfície do objeto puder ser descrita na forma paramétrica, esta pode servir como base para a função de mapeamento.
- **Parametrização de objetos genéricos:** quando os objetos não têm uma parametrização natural é usado um **mapeamento em duas fases** (Bier e Sloan): **mapear textura** sobre uma superfície simples como cilindro, esfera, etc. **que englobe aproximadamente o objeto**; mapear superfície simples sobre a superfície do objeto (raios passando pelo centroide do objeto; raios normais à superfície do objeto; raios normais à superfície simples; raios refletidos (**environment mapping**)).
- **Processo de mapeamento de texturas:** projeta-se o pixel sobre a superfície (pontos da superfície correspondentes aos vértices do pixel), parametriza-se (coordenadas paramétricas dos vértices do pixel projetados), mapeia-se inversamente (coordenadas dos vértices no espaço de textura) e faz-se a média (cor média dos "Texels" proporcional à área coberta pelo quadrilátero).

### Exemplos



Em OpenGL além das coordenadas dos vértices e do vetor normal é possível também especificar coordenadas de textura.

- **Mapeamento de texturas em polígonos:** projeta-se os vértices do polígono na imagem, sendo que a cada vértice projetado corresponde um ponto no espaço de textura. Cada pixel do polígono na imagem e a cor do textel obtido é dado por uma combinação afim.  $P = \alpha_1 P_1 + \alpha_2 P_2 + \alpha_3 P_3$

### • Mapeamento de texturas em OpenGL:

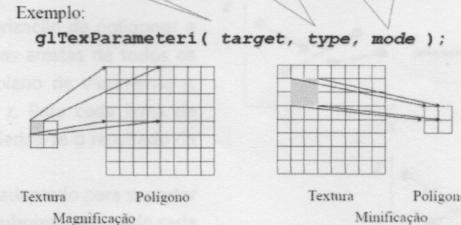
- Passos:
  - Ligar o mapeamento de texturas: `glEnable(GL_TEXTURE_2D)`.
  - Especificar a textura: usar `glTexImage2D`.
  - Configurar diversos parâmetros: **modos de filtragem** (magnificação/minificação, filtros mipmap de minificação); **modos de repetição de padrões** (cortar ou repetir), **funções de aplicação de textura** (como misturar a cor do objeto com a da textura).
  - Especificar coordenadas da textura: **por vértice** com `glTexCoord*` ou **por coordenadas** computadas automaticamente com `glTexGen*`.
- **Especificação da imagem da textura:** é normalmente carregada a partir de um array de textels na memória principal através da função: `glTexImage2D` (`target`, `level`, `components`, `w`, `h`, `border`, `format`, `type`, `*texels`);
- **Conversão da imagem de textura:** se o tamanho da imagem não for uma potência de 2 usa-se a função: `gluScaleImage` (`format`, `w_in`, `h_in`, `type_in`, `*data_in`, `w_out`, `h_out`, `type_out`, `*data_out`), sendo `*_in` a imagem original e `*_out` a imagem de destino.
- **Mapeamento da textura:** é baseado nas coordenadas paramétricas de textura, chamando-se `glTexCoord*()` para cada vértice.
- **Geração automática das coordenadas da textura:**
  - Habilitar a geração automática de coordenadas de textura `glEnable(GL_TEXTURE_GEN_[STRQ])`;
  - Especificar parâmetros
 

```
void glTexGen[ifd](GLenum coord, GLenum pname, TYPE param);
void glTexGen[ifd]v(GLenum coord, GLenum pname, TYPE *param);
```

    - Qual coordenada de textura?
      - `Coord = GL_S / GL_T / GL_R / GL_Q`
    - Plano de referência
      - `Pname = GL_OBJECT_PLANE / GL_EYE_PLANE`
      - `Param = coeficientes A/B/C/D do plano`
    - Modos de geração de coordenadas
      - `Pname = GL_TEXTURE_GEN_MODE`
      - `Param = GL_OBJECT_LINEAR / GL_EYE_LINEAR / GL_SPHERE_MAP`
- **Texturas Minimap:** permite que **texturas de diferentes níveis de resolução sejam aplicadas de forma adaptativa**, reduzindo a discretização devido a problemas de interpolação. O nível da textura na hierarquia mipmap é especificada durante a definição da textura com a função: `glTexImage2d` (`gl_texture *d, level, ...`).

### ○ Filtragem:

`glTexParameter{i}( target, type, mode );`



Exemplo:

`glTexParameteri( target, type, mode );`

- **Modos de repetição:** Exemplo:
 

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



## 8. VISIBILIDADE E RECORTE

Numa cena tridimensional não quer que objetos ou partes destes apareçam na imagem, a resolução deste tipo de problemas envolve: **culling** (descartar objetos que não podem ser vistos); **clipping** (recortar objetos de forma a apresentar apenas as partes que podem ser vistas); **desenhar apenas as partes visíveis; sombras**.

- **Espaço do objeto vs. espaço da imagem:** existem métodos que trabalham no espaço do objeto, sendo que entrada e saída são dados geométricos, são independentes da resolução da imagem, apresentam uma menor vulnerabilidade a aliasing e a rasterização ocorre depois. Por outro lado, nos métodos que trabalham com imagens, a entrada é vetorial e a saída matricial, são dependentes da resolução da imagem, apresentam visibilidade apenas em pontos e podem aproveitar aceleração de hardware.

### ○ Modos de aplicação da textura:

- Controla como a cor da textura afecta a cor do pixel
 

```
glTexEnv{f1}[v](GL_TEXTURE_ENV, prop, param)
```
- **Modos** (`prop = TEXTURE_ENV_MODE`)
  - GL\_MODULATE
  - GL\_BLEND
  - GL\_REPLACE
- Cor a ser misturada (`GL_BLEND`)
  - Especificada com `prop = GL_TEXTURE_ENV_COLOR`

### ○ Correção de perspetiva:

- Mapeamento de texturas em polígonos pode ser feito:
  - Da forma simples e rápida (interpolação linear)
  - Usando interpolação em coordenadas homogêneas
- Comportamento do OpenGL é influenciado por "dicas" ("hints")
 

```
glHint(GL_PERSPECTIVE_CORRECTION_HINT, hint)
```

 onde `hint` pode ser
  - GL\_DONT\_CARE
  - GL\_NICEST
  - GL\_FASTEST
- O OpenGL não obedece necessariamente!

- **Recortes ("Clipping"):** o problema é definido por geometria recortada, apresentando restrições de recorte como a janela (2D), o volume de visibilidade e os polígonos usados.

- **Recorte de segmentos de reta:** por norma feito através dos métodos **Cohen-Shutherland**, **Liang-Barsky / Cyrus-Beck** e **Nicholl-Lee-Nicholl**.

- **Cohen-Shutherland:** os vértices do segmento são classificados em relação a cada semi-espaco plano que delimita a janela, sendo que se ambos os vértices forem classificados como fora descarta-se o segmento, se como dentro testa-se o próximo semi-espaco, **se um como dentro e outro como fora computa-se o ponto de interseção e continua-se com o algoritmo, como o segmento recortado**.
- **Liang-Barsky:** consistem em representar a reta na sua forma paramétrica, o que descarta a necessidade de computar pontos de intersecção irrelevantes.

$$x_{\min} \leq x_1 + t \Delta x \leq x_{\max} \quad \Delta x = x_2 - x_1$$

$$y_{\min} \leq y_1 + t \Delta y \leq y_{\max} \quad \Delta y = y_2 - y_1$$

Linha infinita intercepta semi-espacos planos para os seguintes valores do parâmetro  $t$ :

$$t_1 = \frac{q_1}{p_1} \quad \text{onde} \quad p_1 = -\Delta x \quad q_1 = x_1 - x_{\min}$$

$$p_2 = \Delta x \quad q_2 = x_{\max} - x_1$$

$$p_3 = -\Delta y \quad q_3 = y_1 - y_{\min}$$

$$p_4 = \Delta y \quad q_4 = y_{\max} - y_1$$

- Se  $p_i < 0$ , à medida que  $t$  aumenta, recta entra no semi-espaco plano

- Se  $p_i > 0$ , à medida que  $t$  aumenta, recta sai do semi-espaco plano

- Se  $p_i = 0$ , recta é paralela ao semi-espaco plano (recorte é trivial)

- Se existe um segmento da recta dentro do retângulo, classificação dos pontos de intersecção deve ser entra, entra, sai, sai

- **Recorte de polígono contra retângulo:** este problema inclui o problema de recorte de segmentos de reta, sendo que o polígono resultante tem vértices que são da janela, do polígono original ou pontes de intersecção entre estes dois.

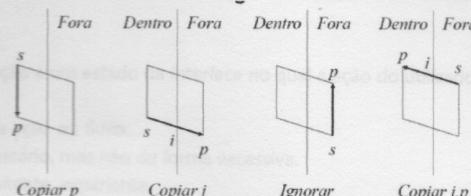
- **Algoritmo de Sutherland-Hodgman:** o objetivo é recortar o polígono sucessivamente contra todos os semi-espacos planos da figura de recorte. O polígono é assim dado como uma lista circular de vértices, que são processados juntamente com as arestas e classificados contra o semi-espaco corrente.

- Vértice:

- Dentro: copiar para a saída
- Fora: ignorar

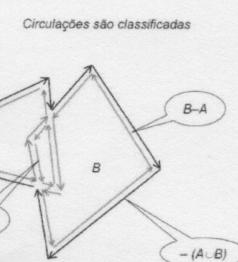
- Aresta

- Intercepta semi-espaco plano (vértice anterior e posterior têm classificações diferentes): Copiar ponto de intersecção para a saída
- Não intercepta: ignorar



Este algoritmo também pode remover as arestas fantasma, distinguindo a forma como as intersecções são geradas, ou seja, de dentro para fora (alfa) ou de fora para dentro (beta). Ao encontrar um alfa ligar com o ultimo beta visto.

- **Algoritmo de Weiler-Atherton:** este algoritmo recorte qualquer polígono contra qualquer polígono, sendo que pode ser usado para fazer uniões, intersecções e diferenças. Cada polígono divide o seu espaço como dentro, fora e borda (duplicada).

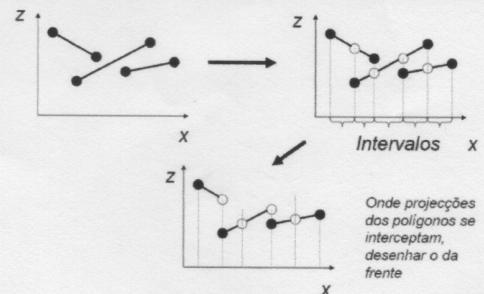


- **Algoritmos de visibilidade:**

- **Backface Culling:** reduz o número de faces aproximadamente a metade, sendo que as que ficam para trás não são pintadas, em OpenGL usa-se glEnable(Culling);

- **Z-Buffer:** opera no **espaço da imagem**, mantendo para **cada pixel um valor de profundidade, resultante de um teste de profundidade**. Se a profundidade do pixel for menor que a registada no z-buffer pinta-se o pixel. Em OpenGL usa-se glEnable(GL\_DEPTH\_TEST). Quando há objetos com transparência a ordem de renderização é importante, esta situação é resolvida com o método A-Buffer.

- **A-Buffer:** permite a **implementação de transparência e de filtragem** (anti-aliasing). Numa **primeira fase os polígonos são rasterizados**. Se o pixel for completamente coberto pelo polígono é opaco, sendo inserido na lista removendo os polígonos mais profundos. Se o polígono é transparente ou não cobre totalmente o pixel é inserido na lista. **Numa segunda fase gera-se a imagem**, onde as **máscaras de subpixels são misturadas para obter cor final do pixel**.



Onde projeções dos polígonos se interceptam, desenhar o da frente

- **Algoritmo scan-line:** o objetivo é aplicar o algoritmo de rasterização de polígonos a todos os polígonos em simultâneo. Para tal ordena-se todas as arestas de todos os polígonos pelo y mínimo. Para cada polígono e para cada plano de varrimento y, determina-se os intervalos de intersecção x e ordena-se por z. Para cada linha de varrimento z, insere-se as arestas na linha de varrimento e renderiza-se o resultado da linha de varrimento.

- **Algoritmo Warnock:** explora a coerência da área, podendo ser adaptado para suportar transparência. Pode ser levado por recursão até ao tamanho de subpixels, pintando cada pixel sozinho uma vez.

- **Algoritmo do pintor/prioridade em z:** pintar os objetos mais distantes antes de pintar os mais próximos, tendo estes de estar ordenados em Z.

- **Algoritmos de recorte sucessivo:** algoritmo do pintor, mas de frente para trás.

- **BSP-Trees:** são estruturas de dados que **representam uma partição recursiva do espaço (célula convexa - politopo)**. O ponto mais complicado é a orientação dos planos de partição, uma vez que se procura uma partição equitativa. Permitem obter uma ordem de desenho baseada em profundidade (algoritmo do pintor, dos cortes sucessivos, outros).

**Divide-se a coleção em duas subcoleções (além do próprio polígono usado como suporte): os polígonos na frente do plano e os polígonos atrás do plano**, o que pode ser feito recorrendo a recorte.

- **Células e portais:** a visibilidade é determinada convencionalmente dentro de cada compartimento/ célula, sendo que visibilidade entre células requer que a luz atravesse as partes.

- **Visibilidade pré computada:**

Operações de recorte são complexas

- Volume recortado pode ter um grande número de faces

Ideia: Pré-computar dados de visibilidade

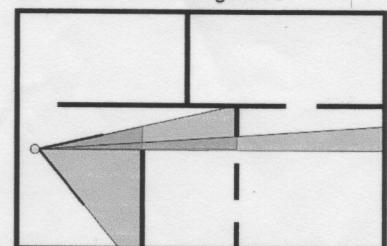
Conceito de **observador genérico**

- Observador que tem liberdade para se deslocar para qualquer ponto da célula e olhar em qualquer direção

Informação de visibilidade

- Célula a Região (estimativa exata)
- Célula a Célula (estimativa grossa)
- Célula a Objeto (estimativa fina)

### Células e Portais - Algoritmo



Desenhar célula C (paredes, objectos) onde o observador está

Para cada célula  $V_i$  vizinha à célula do observador por um portal, recortar o volume de visão pelo portal

Se volume recortado não for nulo,

- Desenhar célula vizinha restrita à região não recortada do volume de visão

- Repetir o procedimento recursivamente para as células vizinhas de  $V_i$

## 9. INTERAÇÃO PESSOA-MÁQUINA

Para o utilizador, a interface é o sistema, sendo que a comunicação com o sistema é pelo menos tão importante como a computação realizada pelo mesmo. Desta forma o sucesso de uma aplicação depende da qualidade da interface com o utilizador.

- **Usabilidade:** combinação de características centradas no utilizador, tais como: facilidade de aprendizagem; rapidez na execução de tarefas; taxa de erros reduzida; satisfação subjetiva do utilizador; retenção ao longo do tempo (Shneiderman, 1992).
- **Design centrado no utilizador:** entrevistas, observação no trabalho, análise de necessidades, análise do perfil dos utilizadores, análise de tarefas, análise do fluxo de informação. Prevenir erros contra os utilizadores, otimizando as suas operações de uma forma eficiente e familiar, mantendo o controlo do seu lado.
- **Modelo do sistema:** dar ao utilizador um modelo mental, consistente, do sistema, baseado nas tarefas a efetuar.
- **Consistência:** princípio do menor espanto.
- **Limitações da memória humana:** reconhecer em vez de lembrar.
- **Feedback:** indicadores de estado para tarefas potencialmente demoradas (ex: cursor, barra de progressão, etc.); tempo de resposta do sistema adequado à tarefa em questão.
- **Mensagem do sistema:** centradas no utilizador e nas tarefas a efetuar; positivas e não ameaçadoras; termos informativos e construtivos nas mensagens de erro; o sistema deve assumir a culpa dos erros;
- **Antropomorfização:** não atribuir características humanas a objetos.
- **Modalidade:** utilização de indicadores de modo; um modo e interação é um estado da interface no qual a ação do utilizador tem um significado diferente do que teria noutro modo ou estado.
- **Reversibilidade:** possibilidade de desfazer com facilidade uma dada ação ou fluxo.
- **Chamada de atenção:** chamar à atenção do utilizador quando necessário, mas não de forma excessiva.
- **Utilizadores:** ter em conta o tipo de utilizador : inexperiente, intermitente, experiente.

Objecto de desenvolvimento	Comportamental	Estrutural
Ponto de vista adoptado	Interacção	Software que suporta a interacção
Objecto de descrição	Acções do utilizador, percepções e tarefas	Reacções do sistema face às acções do utilizador
Aspectos envolvidos	Factores humanos, cenários, representações detalhadas, especificações de usabilidade, avaliação	Algoritmos, estruturas de dados, programação, widgets, callbacks
Teste	Procedimentos efectuados pelo utilizador	Procedimentos efectuados pelo sistema