SGRAI RESUMOS

ÍNDICE

PDF- “PREAMBULO”

Representações Gráficas

1. **Gráficos Vetoriais**(pontos,retas,curvas,planos,polígonos)
2. **Gráficos Matriciais**(amostragem em grelhas retangulares)

Representações Vetoriais

- Permitem uma série de operações quase sem perda de precisão:
    - Transformações lineares / afim
    - Deformações
    - complexidade de processamento = O(novértices / vectores)

Representações Matriciais

- Representação flexível e muito comum
- Muitas operações implicam em perda de precisão(**rotaçao**, **escala**)
- Técnicas para lidar com o problema(técnicas anti-discretização: **anti-aliasing**)
- Complexidade de processamento = O(node pixels)

Conversão entre Representações

Repr. Vectoriais → Rasterização,“Scan conversion” → **Repr. Matriciais**
Repr. Vectoriais ← Reconhecimento de padrões ← **Repr. Matriciais**

Dispositivos Gráficos**Dispositivos vetoriais -** traçadores(plotters)

Dispositivos virtuais - Linguagens de descrição de página(HPGL / Postscript), rasterização implícita
Dispositivos matriciais - sinónimo de dispositivo gráfico, impressoras, displays

Displays **(dispositivo para apresentação de informação visual)**

Resolução: 640x480 até 1600x1200, tendência de aumento
Resolução no espaço de cor:

- Monocrómatico (preto e branco)
- Tabela de cores (cada pixel tem tipicamente 8 bits)
- RGB, tipicamente 24 bits(8 para cada componente)

Processador (acelerador) gráfico

- **Uso de paralelismo para atingir alto desempenho**
- **Alivia o CPU do sistema de algumas tarefas:**
    - Transformações(Rotaçao,translaçao,escala,etc)
    - Recorte(Supressão de elementos fora da janela de visualização)
    - Projeçao(3d → 2d)
    - Mapeamento de texturas
    - Rasterizaçao

- **Normalmente usa memó0ria separada da do sistema**

PDF- “Open GL Básico”

Open GL - o que é?

- **API para gerar gráficos:**
  - **3d e 2d**
  - **primitivas vetoriais e matriciais**
  - **independente de plataforma e sistema de janelas**

Sistemas de Janela

- **Principal meio de interacção humano-máquina em ambientes de computação modernos**
- **Interação do utilizador e do próprio sistema de janelas é comunicada à aplicação através de eventos**(rato foi ativado, janela foi redimensionada)
- **Eventos são tratados por rotinas callback da aplicação**(redesenhar o conteúdo da janela, mover um objeto de um lado para outro da janela)

Desenhar com OpenGL

- **OpenGL funciona como uma máquina de estados**
- **API tem rotinas para**
  - Desenhar primitivas geométricas e imagens
  - Alterar variáveis de estado (ex.: cor, material, fontes de iluminação,etc.)
  - Consultar variáveis de estado

Anatomia de um programa OpenGL/GLUT

```
#include <GL/glut.h>                          Cabeçalhos
/* Outros headers */

void display (void) {                         Rotinas Callback
...
}
/* Outras rotinas callback */

int main (int argc, char *argv[]) {
glutInit (argc, argv);                        Inicialização do GLUT
glutInitDisplayMode( modo);                   Inicialização  da janela
glutCreateWindow( nome_da_janela);            Inicialização  da janela
glutDisplayFunc( displayCallback );           Registo de Callbacks
glutReshapeFunc( reshapeCallback );           Registo de Callbacks
/* Registo de outras rotinas callback */
```

```
glutMainLoop();
return 0;
}
```

Ciclo principal

Cabeçalhos OpenGL/GLUT

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

Já inclui automaticamente os cabeçalhos do OpenGL:

```
#include <GL/gl.h>
#include <GL/glu.h>
```

GLUT - Registando Callbacks

- Callbacks sao rotinas que serão chamada para tratar eventos
- Para uma rotina callback ser chamada é preciso registá-la numa função

Exemplo:

- glutXXXFunc (callback), onde XXX designa uma classe de eventos e callback é o nome da rotina

GLUT - Callback de desenho

- Rotina chamada automaticamente sempre que a janela ou parte dela necessita de ser redesenhada
- Exemplo:

```
void display ( void )
{
        glClear( GL_COLOR_BUFFER_BIT );
        glBegin( GL_TRIANGLE_STRIP );
        glVertex3fv( v[0] );
        glVertex3fv( v[1] );
        glVertex3fv( v[2] );
        glVertex3fv( v[3] );
        glEnd();
        glutSwapBuffers(); /* double-buffering! */
}
```

GLUT - Callback de redimensionamento

- Chamada sempre que a janela é redimensionada
- tem a forma, void reshape (int width, int height) { . . . }
- width/height sao a nova largura / altura da janela (em pixeis)

- se nao for especificada, o GLUT usa uma rotina de redimensionamento por omissao, que ajusta o viewport de modo a usar toda a área útil da janela

GLUT - Callbacks

Callbacks frequentemente usadas:
- void keyboard (unsigned char key, int x, int y)
- void mouse(int button, int state, int x, int y)
- void motion (int x , int y)
- void passiveMotion(int x, int y)

Programa OpenGL / GLUT - Inicialização

Inicialização do GLUT
glutInit(int * argc, char ** argv) //Estabelece contacto com sistema de janelas
Inicialização da Janela
glutInitDisplayMode(int modo)
glutInitWIndowPosition (int x, int y)
glutInitWindowSize (int width, height)

Criação da janela
int glutCreateWindow (char * nome)
- Cria janela;
- “nome” é o nome da janela
- o int é para identificar a janela

Programa OpenGL/GLUT – Ciclo Principal

Depois de registados os callbacks, o controlo é entregue ao sistema de janelas:
- glutMainDisplayLoop(void)

OpenGL - Primitivas de desenho

glBegin (primitiva);
- especificação de vértices, cores, coordenadas de textura, material

glEnd();

Entre glBegin e glEnd podem ser usados:
- glMaterial, glNormal e glTexCoord;
- - glColor, glVertex

OpenGL - Convenções de Nome

glVertex3fv (v)

3 - Número de componentes: duas (x,y), três (x,y,z), quatro (x,y,z,w)

f - Tipo de dados (f - float, d - double, etc…)
v - vector (omitir o “v” quando coordenadas dadas uma a uma)

OpenGL - Controlando as cores

Diretamente:

- cores: glColorIndex() ou glColor()

Calculadas a partir de um modelo:

- Ligar a iluminação: glEnable (GL_LIGHTING);
- Escolher modelo de sombreamento:
    - Constante por face: glShadeModel (GL_FLAT);
    - Gouraud (default): glShadeModel (GL_SMOOTH);
- Ligar pelo menos uma fonte de luz. Ex.: glEnable(GL_LIGHT0);
- Especificar propriedades da(s) fonte(s) de luz: glLight();
- Especificar propriedades de material de cada objecto:glMaterial()
- Especificar normais de cada face ou de cada vértice: glNormal()

PDF - “SGRAI-2009-OpenGL_1”

Esqueleto de programa

```
#include <teste.h>
main()
{
        InitializeAWindowPlease();
        glClearColor (0.0, 0.0, 0.0, 0.0);                  //limpar o ecrã
        glClear (GL_COLOR_BUFFER_BIT);                      //limpar o ecrã
        glMatrixMode(GL_PROJECTION)                         //definir sistema de coordenadas
        glLoadIdentity();                                   //reset das coord. p/ matriz identidate
        glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);             //definir sistema de coor
        glColor3f (1.0, 1.0, 1.0);                          //desenhar
        glBegin(GL_POLYGON);                                //desenhar
        glVertex3f (0.25, 0.25, 0.0);                       //desenhar
        glVertex3f (0.75, 0.25, 0.0);                       //desenhar
        glVertex3f (0.75, 0.75, 0.0);                       //desenhar
        glVertex3f (0.25, 0.75, 0.0);                       //desenhar
        glEnd();                                            //desenhar
        glFlush();
        UpdateTheWindowAndCheckForEvents();
}
```

glBegin - define um objeto da cena
glVertex - define um vertice do objeto a desenhar
glFlush - força o pipeline do OpenGL a terminao o processamento e desenhar os pixeis

Link: http://pt.wikipedia.org/wiki/OpenGL#Arquitetura_do_OpenGL

Inicialização

- glutInit (argc, argv) - inicializa a biblioteca GLUT
- glutInitDisplayMode (mode) - indica qual o modo de funcionamento
- glutInitWindowSize(width, height) - indica o tamanho da janela
- glutInitWindowPos (x , y) - indica a posição inicial da janela
- glutCreateWindow (titulo) - cria a janela da aplicação

Ciclo Principal

- glutMainLoop - processa os eventos do sistema gestor de janelas(rato, teclado..) e invoca as callbacks registadas

Renderer

- glutDisplayFunc (callback) - indica qual a função a invocar sempre que for necessário desenhar o conteúdo da janela

Input

- glutKeyboardFunc (keyboard)
- glutSpecialFunc (special)
- glutMouseFunc (mouse)
- glutMotionFunc (motion)
- glutTimerFunc (unsigned int millis, onTimer, int value)

Outras callbacks GLUT

- glutReshapeFunc (reshape)
- glutIdleFunc (idle)

Forçar redesenho

- glutPostRedisplay

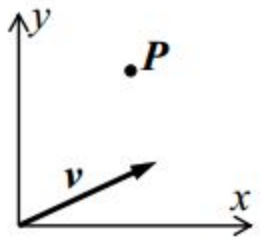
PDF - “Geometria”

Pontos e Vetores (2D)

- Ponto : marca posição no plano
- Vetor: marca deslocamento, inclui noção de direção e magnitude
- São ambos expresso por pares de coordenadas (em 2D) mas **não são a mesma coisa**

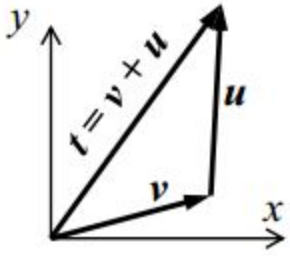
P = (Xp, Yp)

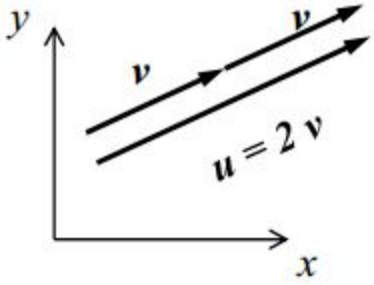
$\vec{v}$ = (Xv, Yv)

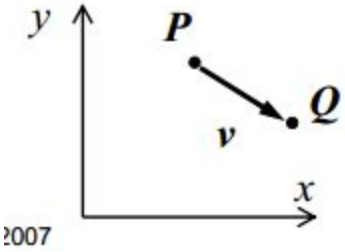


Operações com Pontos e Vetores (2D)

1. **Soma de vetores:** t = v + u
2. **Multiplicação de vetor por escalar :** u = 2 v
3. **Subtração de pontos :** v = Q - P
4. **Soma de ponto com vetor :** Q = P + v







Transformações

- **Transformação é uma função que faz corresponder pontos de um espaço a outros pontos do mesmo espaço**
- **Se for Transformação Linear então :**
    - Um conjunto de pontos pertencentes a uma reta, depois de transformados vão eles também pertencer a uma reta;
    - Um ponto P guarda uma relação de distancia com dois outros pontos Q e R, então essa relação de distância é mantida pela transformação
- Transformação Linear Afim = Transformação Linear + translação

Transformações Lineares em 2D

- **Transformação linear:**
    - x' = ax + by
    - y' = cx + dy

- **Transformação linear Afim:**
    - x' = ax + by + e
    - y' = cx + dy + f

Transformações de Vetores

Uma transformação linear afim aplicada a um vetor não inclui translação.

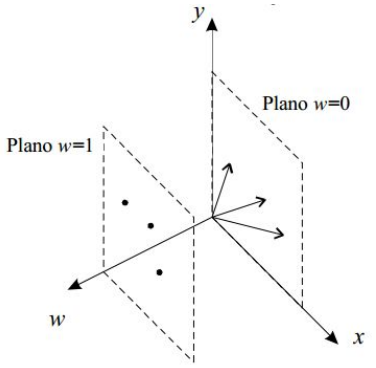
Coordenadas Homogéneas

A transformação de vetores é operacionalmente diferente da de pontos
Problema é levado para uma dimensão superior:

- w = 0 - **vetores**
- w = 1 - **pontos**

Termos independentes formam uma coluna extra na matriz de transformação

Coordenadas Homogéneas Interpretação



Sistema de Coordenadas

Um sistema de coordenadas para R^n é definido por um ponto (origem) e n vectores

Transformações em 3D

• **Vectores e pontos em 3D**

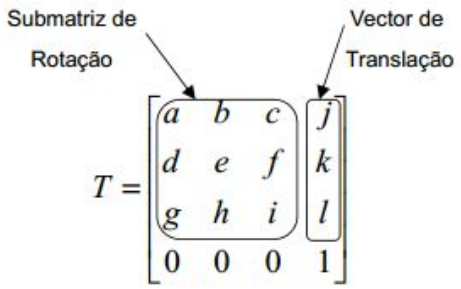
$$\vec{V} = \begin{bmatrix} V_x \\ V_y \\ V_z \\ 0 \end{bmatrix} \qquad P = \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

• **Transformação linear afim**

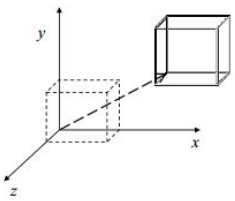
$$T = \begin{bmatrix} a & b & c & j \\ d & e & f & k \\ g & h & i & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformações Rígidas

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



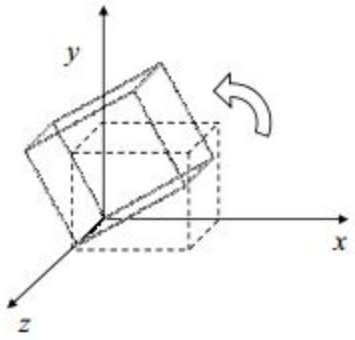
Translação

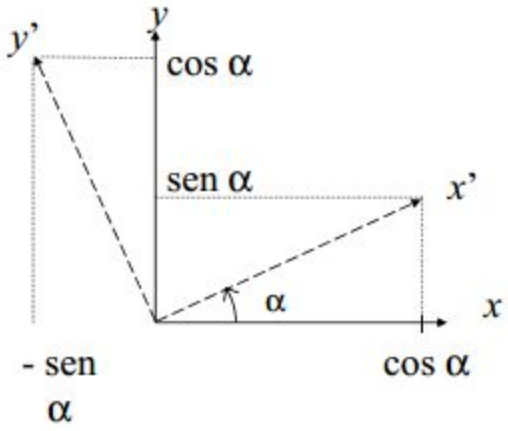


As translações são **comutativas**: P + t + v = P + v + t

Rotação em torno do eixo Z

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





Rotação em torno dos eixos coordenados

Rotação em torno de Z é dada pela matriz:

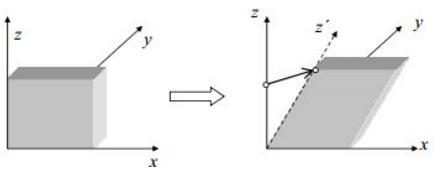
$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotação em torno de Y e X é dada pela matriz:

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

Inclinação (“Shear”)

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



Se o vector unitário do eixo z é transformado em [Shx Shy 1 0]T, então a matriz de transformação é dada por:

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

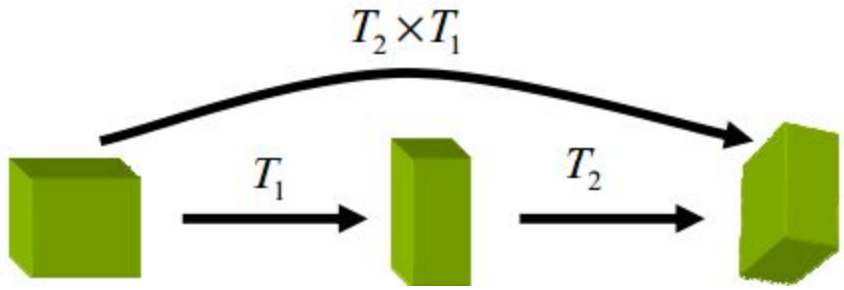
Escalamento

- Especificado por 3 fatores (Sx, Sy , Sz) que multiplicam os vectores unitários x, y, z
- Escalamento **não é** uma transformação rígida;

Composição de transformações em 3D

Para compor 2 transformações temos:

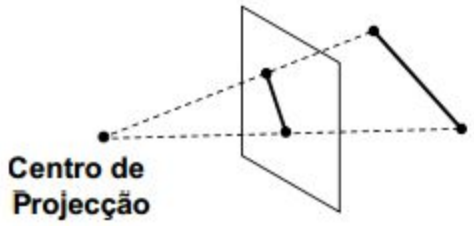
- Se P' = T1 x P e P'' = T2 x P' , então, P'' = T2 x T1 x P



PDF- “Projeções”

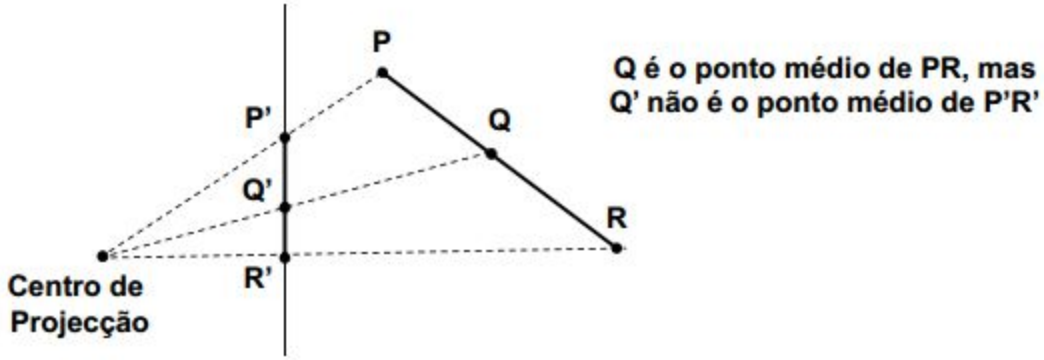
Perspectiva

O problema consiste em projectar pontos do espaço d-dimensional no plano d-1 dimensional usando um ponto especial chamado de centro de projecção



Transformações Projectivas

Transformações projectivas transformam rectas em rectas mas não preservam as combinações afim



Geometria Projectiva

Geometria **euclidiana**: duas rectas paralelas **não se encontram;**
Geometria **projectiva**: assume-se a existência de **pontos ideais** (no infinito):

- Rectas paralelas encontram-se num ponto ideal;

Coordenadas homogéneas em espaço projectivo

Representamos apenas pontos (não vectores);

Em 2D, um ponto (x,y) será representado em coordenadas homogéneas pela matriz-coluna $[x \cdot w \; y \cdot w \; w]^T$, para $w \neq 0$

- Dado um ponto com coordenadas homogéneas $[x \; y \; w]^T$, a sua representação canônica é dada por $[x/w \; y/w \; 1]^T$

Perspectiva - Sumário

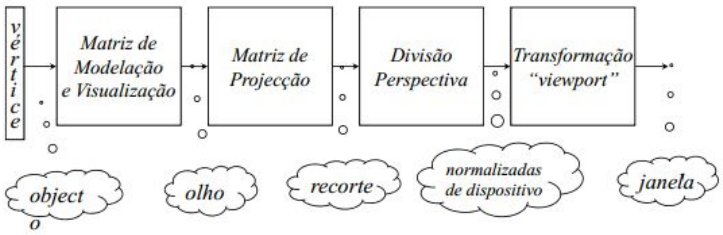
Para fazer projecção perspectiva de um ponto P, seguem-se os seguintes passos:

- P é levado do espaço **euclidiano** para o **projectivo**(mesmas coordenadas homogéneas)
- P é **multiplicado** pela matriz de transformação perspectiva resultando em P'.
- P' é levado novamente ao espaço euclidiano (operação de divisão perspectiva)

Transformações em OpenGL

Modelação (mover / deformar objetos)
Visualização (mover e orientar a câmara)
Projeção (ajuste da lente / objectiva da câmara)
"**Viewport** " (aumentar ou reduzir a fotografia)

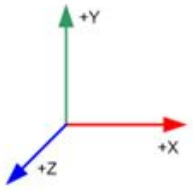
Pipeline OpenGL de Transformações



Link ajuda: http://www.songho.ca/opengl/gl_transform.html

Estado Inicial do Pipeline

- Inicialmente, as matrizes "modelview" e "projection" são matrizes-identidade (diagonal da matriz só com 1)
    - Vértices não são transformados;
    - Projeção é paralela ao plano xy;
    - Mundo visível restrito ao cubo -1 <= x,y,z <= 1
- A transformação "viewport" transforma o quadrado -1 <= x.y <= 1 (em coordenadas normalizadas de dispositivo) m na superfície total da janela



Link ajuda: http://pontov.com.br/site/opengl/179-aplicando-as-transformacoes
Link ajuda: http://www.pcs.usp.br/~pcs5748/pdf/Sistemas_de_Coordenadas.pdf
Link ajuda: http://www.di.ubi.pt/~agomes/cg/teoricas/03-janelas.pdf

Especificação do Viewport

glViewport (x0, y0, largura, altura)

- Especifica a área da janela na qual será transformado o quadrado do plano de projeção
- x e y especificam o canto inferior esquerdo;
- Por defeito, o visor ocupa a área gráfica total da janela do ecrã.

Especificação de transformações

- As matrizes **Modelview** e **Projection** usadas no pipeline são aquelas que se situam no topo de 2 pilhas que são usadas para fazer operações com matrizes.
- Para selecionar em qual pilha queremos operar usamos:
  - glMatrixMode(**GL_MODELVIEW** ou **GL_PROJECTION**)
- Funções para operar com a pilha corrente:
  - glLoadIdentity ();
  - glLoadMatrix* ();
  - glMultMatrix* ();
  - glPushMatrix ();
  - glPopMatrix ().

Transformação de objectos

Usam-se **funções** para multiplicar o topo da pilha de matrizes por transformações especificadas por parâmetros:

- glTranslate* (x, y, z);
- glRotate* (ângulo, x, y, z);
- glScale* (x, y, z);

Cuidado: a ordem é importante:

```
glTranslatef (10, 5, 3);
glRotatef (10, 0, 0, 1);
glBegin (GL_TRIANGLES);
```

O objecto é rodado e depois transladado!

Nota: OpenGL takes these transformations, combines them together and applies them to each vertex that we pass in with glVertex commands. The transformations get combined *in **reverse order***, so each vertex gets spun (rotated) around the vertical axis, then moved (translated) forward.

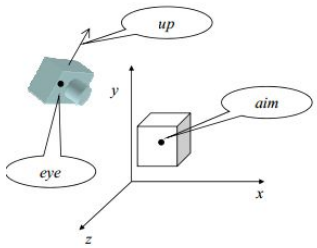
Link: http://www.basic4gl.net/index.php?page=Tutorial&tutorial=Introduction+to+OpenGL&subpage=1

Transformações de visualização

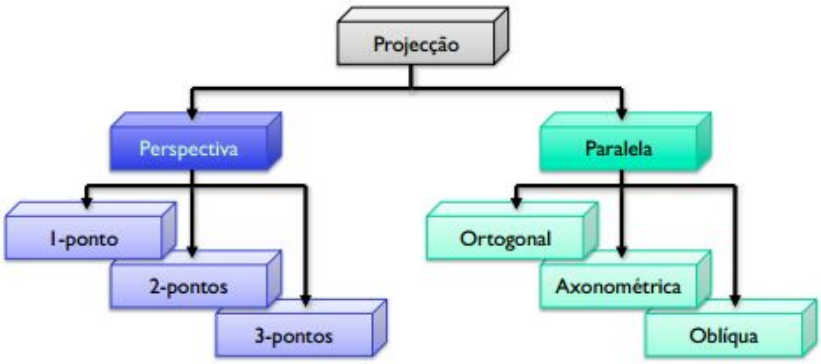
- Levam **a câmara até a cena** que se quer visualizar;
- Levam os objectos da cena até uma **câmara estacionária**

gluLookAt (eye x, eye y , eye z , aim x, aim y, aim z , up x, up y, up z);

- eye = ponto onde a câmara será posicionada;
- aim = ponto para onde a câmara será apontada;
- up = vector que dá a direção “para cima” da câmara;

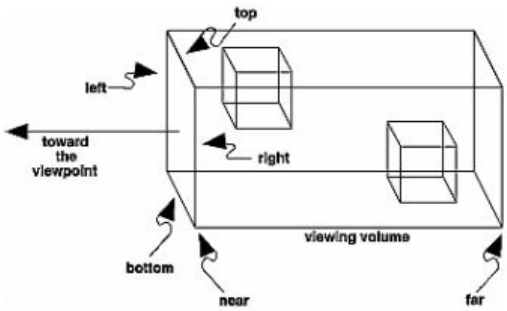


Projecção Paralela



Para ajustar o volume visível, a matriz de projeção é inicializada com:

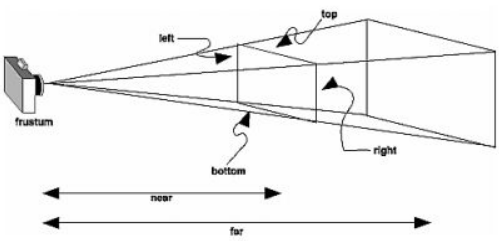
- **glOrtho** (left, right, bottom, top, near, far);
- near e far são valores **positivos** tipicamente;



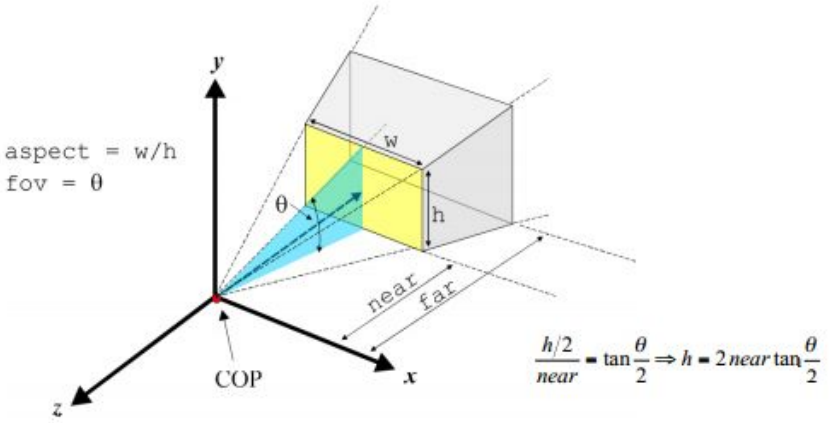
Link: http://www.di.ubi.pt/~agomes/cg/teoricas/04-visualizacao.pdf

Projecção em Perspectiva

- O observador está a uma distância finita do/a objeto/cena.
- As projetantes não são paralelas e convergem para um ou mais pontos (obser**vadores).**
- Volume de visualização especificado com:
    - **glFrustum**(left,right,bottom,top,near,far);
    - **frustum** = pirâmide truncada do campo de vista



- Alternativamente pode-se usar a rotina:
    - **gluPerspective** (fovy, aspect, near, far);



Evitar "ecrãs pretos"

- Matriz de **projecção** especificada com **gluPerspective**();
- Tentar levar em conta a razão de aspecto da janela (parâmetro aspect);
- Usar sempre **glLoadIdentity**() antes;
- Não colocar nada depois;
- Matriz de **visualização** especificada com **gluLookAt**;
- Usar sempre **glLoadIdentity** () antes;
- Outras transformações usadas para mover / instanciar os objectos aparecem **depois;**

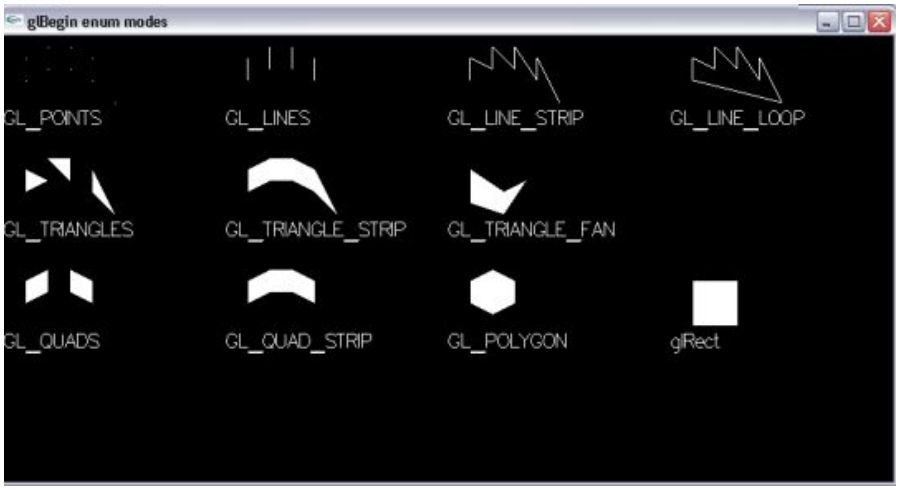
Exemplo

```
void resize( int w, int h )
{
        glViewport( 0, 0, (GLsizei) w, (GLsizei) h );
        glMatrixMode( GL_PROJECTION );          //matriz de projeção
        glLoadIdentity();
        gluPerspective( 65.0, (GLdouble) w / h, 1.0, 100.0 );
        glMatrixMode( GL_MODELVIEW );           //matriz de visualização
        glLoadIdentity();
        gluLookAt( 0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0 );
}
```

PDF “SGRAI-2009-OpenGL_2.pdf”

Desenho de objetos simples

Entre glBegin(mode) e glEnd():



Instruções possíveis em glBegin / glEnd

- glColor;
- glIndex (cor em modo indexado);
- glVertex;
- glNormal (perpendicular à superfície, utilizado para iluminação;
- glMaterial , tipo de material do objeto;
- glCallList, glCallLists - Objetos pré-construídos

Vértices

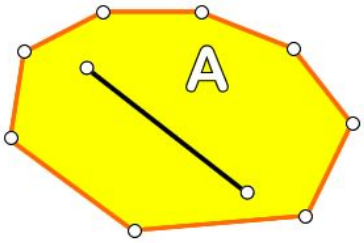
- glVertex {2 | 3 | 4}
- 2D - x,y
- 3D - x,y,z
- Coordenadas homogéneas (x,y,z,w)
- **Indicar os vértices de cada face no mesmo sentido** (CW ou CCW)

Modo de polígonos

- glPolygonMode(face, mode);
- **face** pode ser, GL_FRONT, GL_BACK , GL_FRONT_AND_BACK;
- **mode** pode ser, GL_POINT, GL_LINE , GL_FILL;

Polígonos convexos

- Qualquer linha que “atravesse” um polígono só tem um segmento dentro do polígono.



Polígonos não convexos

- Dividir em polígonos convexos (ex., triângulos) e usar glEdgeFlag para indicar os vértices que pertencem a arestas de bordo (processo chama-se “tesselation”)

GLU quadrics

- Objectos descritos por uma equação quadrática
- Criar um objecto gluNewQuadric().
- Especificar atributos de desenho(gluQuadricOrientation(), gluQuadricDrawStyle(), gluQuadricNormals(), gluQuadricTexture())
- Registar callback de erros em gluQuadricCallback();
- Construir o objecto gluSphere(), gluCylinder(), gluDisk(), ou gluPartialDisk().

Iluminação básica

```
void init()
{
        glEnable(GL_LIGHTING);  //ligar luz 0 ( branca por omissão) e teste de profundidade
        glEnable(GL_LIGHT0);
        glEnable(GL_DEPTH_TEST);
        glFrontFace(GL_CW); // problema nas normais do teapot
        glEnable(GL_COLOR_MATERIAL);
}
```

Display lists

- Sequências pré-compiladas de comandos OpenGL;
- Melhoram performance ao evitar cálculos repetitivos;
- Facilitam leitura de código

Criar lista

- GLuint glGenLists (quantas);
- void glNewList(list, mode);
    - **mode** pode ser **GL_COMPILE** ou **GL_COMPILE_AND_EXECUTE**
- void glEndList()

Utilizar lista

- glCallList(list) // **executa uma display list**
- glCallLists(GLsizei n, GLenum type, const GLvoid *lists)

Animações

Double buffer

- Desenhar próximo frame num buffer escondido e não no buffer de ecrã;
- Quando a cena estiver completa, trocar o buffer de ecrã pelo buffer escondido;
- glutInit(GLUT_DOUBLE) // **select a double buffered window**
- **glutSwapBuffers()** em vez de **glFlush()** na callback de display

PDF “Modelacao.pdf”

Descrição de Sólidos

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

Representação por Fronteira

- Sólido definido indiretamente através da **superfície que o delimita;**
- Superfícies são descritas parametricamente por **parametrização**:
  - $\phi : U \subset \Re \rightarrow \Re^3$

Parametrização

- **Estabelece um sistema de coordenadas sobre a superfície** herdado de um sistema de coordenadas no plano;
- Em geral, não é possível cobrir (descrever) toda a superfície com uma única parametrização:
  - Usam-se várias parametrizações que formam um **Atlas;**

Parametrizações Válidas

- Sólido deve estar bem definido;
- Normal é usada para determinar o interior e o exterior do sólido;

Parametrização do Círculo

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

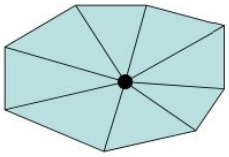
$$x(t)=\frac{1-t^2}{1+t^2}; y(t)=\frac{2t}{1+t^2}; t\in\left(-\frac{\pi}{2},\frac{\pi}{2}\right)$$

Representação Linear por Partes

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

Operações sobre Malhas Poligonais

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



Codificação Explícita

- A mais **simples;**
- Cada face armazena explicitamente **a lista ordenada das coordenadas** dos seus vértices;
- Muita **redundância** de informação;

Desenho da Malha

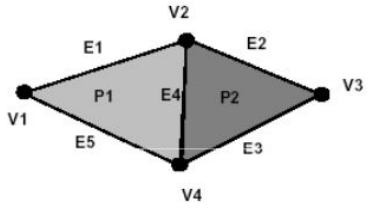
- Cada aresta é desenhada **duas** vezes - pelas duas faces que a compartilham;

Ponteiros para Lista de Vértices

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

Exemplo:

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

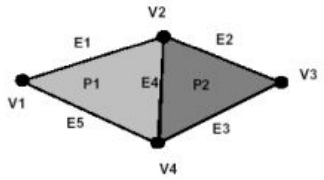


Ponteiros para Lista de Arestas

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

Exemplo:

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



Winged-Edge

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

Face-Edge

- Representa objectos non-manifold (não variedades);
- **Armazena a lista ordenada de faces** incidentes em uma aresta;

Representação Implícita

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

Funções Implícitas

- Uma superfície definida de forma implícita pode apresentar **auto-intersecção**

Teorema da Função Implícita

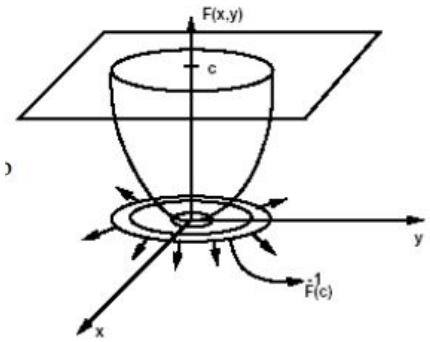
- Seja F : $\Re$ n → $\Re$ definida num conjunto aberto U;
- Se F possui derivadas parciais contínuas em U e ∇ F ≠ 0 em U, então F é uma **subvariedade** de dimensão n -1 do $\Re^n$
    - Superfície **sem auto-intersecção**

Valores Regulares

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

Exemplo 1:

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



Exemplo 2:

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

Exemplo 3:

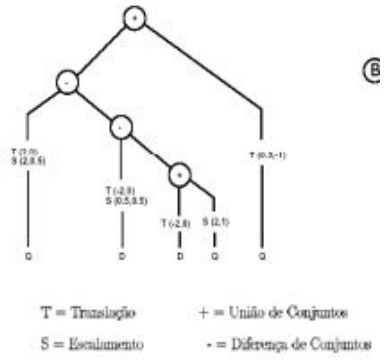
- $F(x,y) = y^2 – x^2 – x^3$, $\nabla F = (2y, -3x^2 – 2x)$;
- Na forma paramétrica:
  - $x(t) = t^2 - 1$ e $y(t) = t (t^2 - 1)$
- Curva de nível 0 é um laço, com uma singularidade na origem:
  - $z = F(x,y) = y^2 - x^2 – x^3 = 0$

Esquema de Representação CSG

- Operações CSG definem objectos através de **operações regularizadas** de conjuntos de pontos: **União, Intersecção e Diferença;**
- Um objecto é **regular** se o 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;**
  - Os nós internos contêm operações de conjunto ou transformações lineares afim;
  - Folhas contêm **objectos primitivos** (tipicamente, quádricas)



CSG com Objectos Implícitos

- Primitivas CSG são definidas por $F_i (X) \leq 0$
- Operações booleanas são definidas nesse caso por:
    - $F 1 \cup F2 = \min (F 1, F2)$;
    - $F 1 \cap F2 = \max (F 1, F2)$;
    - $F 1 / F2 = F 1 \cap F2 = \max (F 1, - F2)$

Prós e Contras de Representações

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

Representações por Campos Escalares

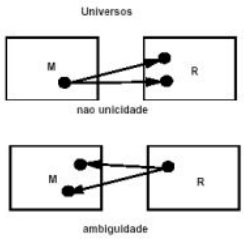
- Em tais representações **a classificação de um ponto é imediata**, bastando avaliar o sinal do valor do campo no ponto;
- Exibir um ponto sobre a superfície do objecto **requer a solução de uma equação**, a qual pode ser complicada;
- Operações booleanas são **avaliadas facilmente**;

Representações por Células

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

Ambiguidade e Unicidade

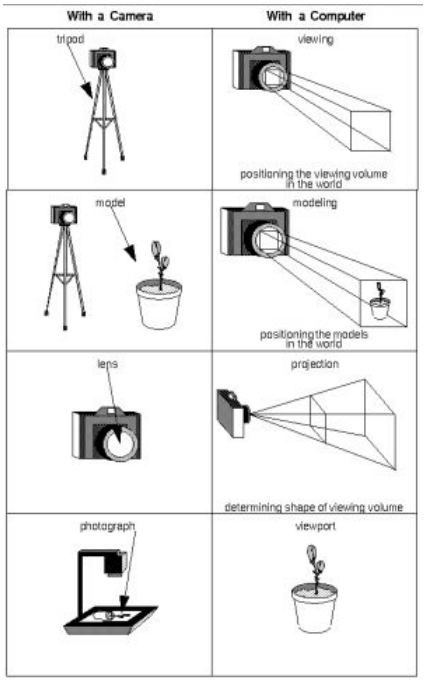
- Uma representação é **única** quando o modelo associado possui uma única representação;
- Uma representação é **ambígua** quando pode representar mais de um modelo.
- Representação ambígua é catastrófica (wireframe);



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

PDF “SGRAI-2009-OpenGL_3.pdf”



Analogia da câmara fotográfica

- Apontar a câmara à cena (viewing transformation).
- Compor a cena (modeling transformation).
- Escolher o tipo de lente e acertar o zoom (projection transformation).
- Determinar o tamanho físico da cena (viewport transformation).

Tipos de transformações

- **View e Model**
  - glMatrixMode(GL_MODELVIEW)
  - Posicionamento da câmara
  - Rotações, translações, escalamentos
- **Projection**
  - glMatrixMode(GL_PROJECTION)
  - Definição do volume de projecção
- **Viewport**
  - glViewport (x, y, width, height)

Esqueleto de código

```
void reshape(int w, int h)
{
        // viewport transformation
        glViewport(0, 0, w, h);
        // projection transformation
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        projeccao();
        …
}
```

```
void display()
 {
        // modelview transformation
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        // posicionamento da câmara
        camara();
        // transformações do modelo
        ...
}
```

Inicializar as matrizes

- Antes de definir transformações deve-se inicializar a matriz de transformação usando **glLoadIdentity**
- Deve-se evitar cálculos acumulados nas matrizes:
    - **Exemplo**: não ir aplicando rotações na mesma matriz, mas sim ter uma variável com o angulo de rotação

Rotações

- void glRotate{ f | d }(angle, x, y, z)

Translações

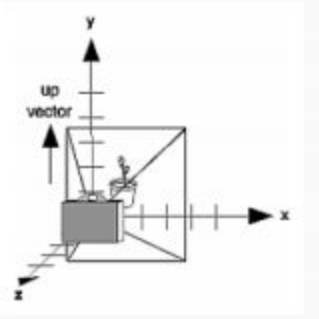
- void glTranslate{ f | d}(x, y, z);

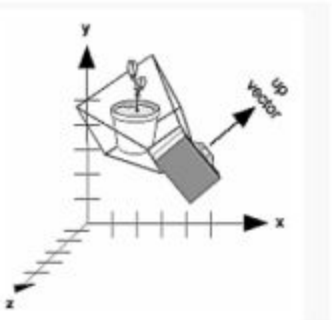
Escalamento

- void glScale{ f | d }(x, y, z)
- a evitar pois altera vectores normal

Posicionamento da câmara

- void gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)

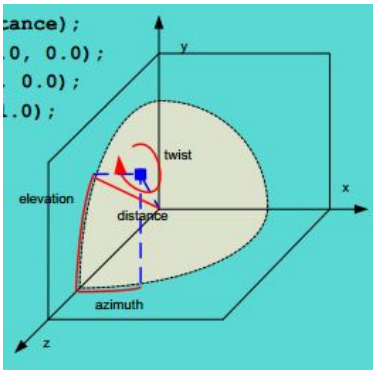




- Mover a câmara ou mover a cena obtém o **mesmo resultado**
    - gluLookAt(0, 0, +5, 0, 0, 0, 0, 1, 0)
    - glTranslatef(0, 0, -5)

Câmara “polar”

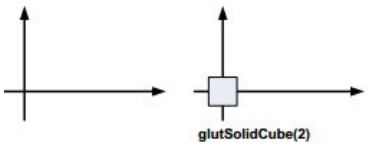
```
void polarView(GLdouble distance,
               GLdouble twist,
               GLdouble elevation,
               GLdouble azimuth)
{
        glTranslated(0.0, 0.0, -distance);
        glRotated(elevation, 1.0, 0.0, 0.0);
        glRotated(azimuth, 0.0, 1.0, 0.0);
        glRotated(twist, 0.0, 0.0, 1.0);
 }
```

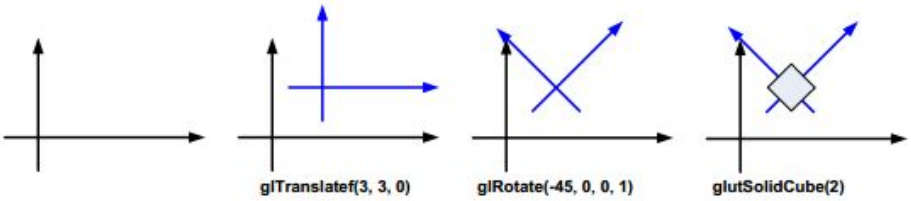


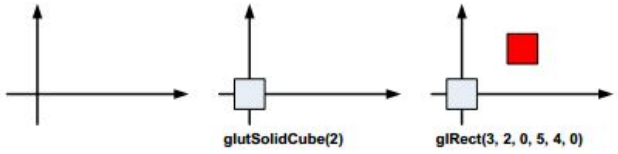
Link: http://www.glprogramming.com/red/chapter03.html

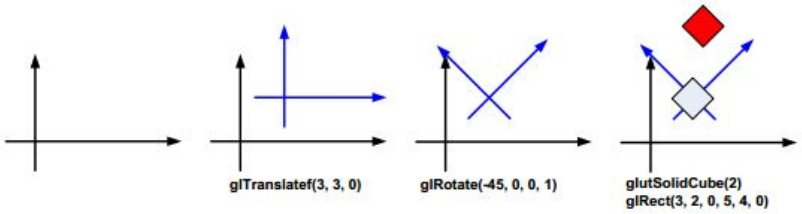
O sistema de coordenadas local

- Ao aplicar uma transformação estão na realidade **a mover um sistema de coordenadas** “agarrado” ao modelo



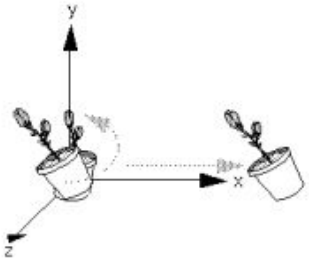




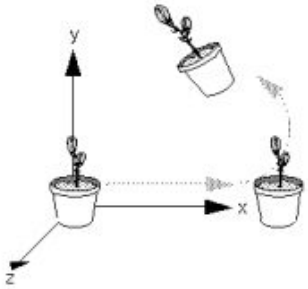


Efeito cumulativo de transformações

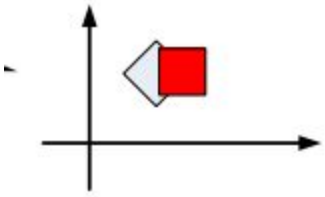
- **Translação + Rotação:**



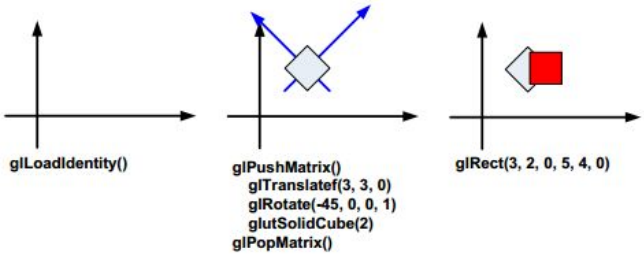
- **Rotação + Translação:**



- **Cena final desejada:**



- **Passos:**



Transformações “locais”

- Guardar a matriz de transformação usando glPushMatrix;
- Recuperar a matriz anterior usando glPopMatrix;
- Push e pop matrix podem ser usados para a matriz de projecção ou para a matriz de modelo/vista;

Texto em GLUT

- Desenhar um caracter:
  - glutStrokeCharacter(fonte, caracter);
  - glutBitmapCharacter(fonte, caracter)
- Desenhado **usando o sistema de coordenadas locais** e todas as transformações em vigor;
- O tamanho de cada caracter é tipicamente **120 unidades**;
  - Necessário escalar;
  - Proteger a matriz de transformação usando push e pop matrix;
- Ciclo para imprimir uma string completa;

Exemplo:

```
void drawString(GLfloat x, GLfloat y, GLfloat z,
        GLfloat scale,
        char* msg)
{
        glPushMatrix();
                glTranslatef(x, y, z);
                glScalef(scale, scale, scale);
                for (int i = 0; i < strlen(msg); i++)
                        glutStrokeCharacter(GLUT_STROKE_ROMAN, msg[i]);
        glPopMatrix();
}
```

PDF “Iluminacao.pdf”

Iluminação

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

Modelos de Iluminação em CG

- **Modelos locais**
  - Apenas caminhos do tipo fonte luminosa → superfície → olho são tratados
  - **Simples**
- **Modelos globais**
  - Muitos caminhos (**ray tracing**, radiosidade) ;
  - **Complexos**

Iluminação em OpenGL

- Assume fontes pontuais de luz:
  - Omnidireccionais (em todas as direções)
  - Spot
- Interacções de luz com superfície modeladas em componentes (**modelo de Phong**):
  - **Emissão**
  - **Ambiente**
  - **Difusa**
  - **Especular**
- Suporte de efeitos atmosféricos como
  - Fog (nevoeiro)
  - Atenuação
- Modelo de iluminação é computado **apenas nos vértices das superfícies;**
  - Cor dos restantes pixels é interpolada linearmente (**sombreamento de Gouraud**)

Fontes de Luz

- Para ligar uma fonte: **glEnable** (source);
  - **source** é uma constante cujo nome é **GL_LIGHTi** , começando com GL_LIGHT0;
  - Quantas? Pelo menos 8, mas para ter certeza:
    - glGetIntegerv(GL_MAX_LIGHTS, &n);
- Não esquecer de ligar o cálculo de cores pelo modelo de iluminação:
  - **glEnable** (GL_LIGHTING);

- Para configurar as propriedades de cada fonte usa-se:
    - **glLightfv**(source, **property**, value);
    - **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)

Propriedades de Material

Especificados por:

- glMaterialfv (face, property, value);
- **face**, designa quais **os lados da superfície** que se quer configurar (GL_FRONT, GL_BACK, GL_FRONT_AND_BACK) ;
- **Property** designa a propriedade do modelo de iluminação (GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, GL_SHININESS)

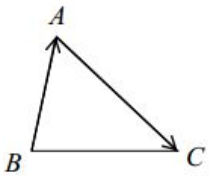
Geometria

Além das propriedades da luz e do material, a **geometria** do objecto é também importante.

- **A posição dos vértices** em relação ao olho e à fonte luminosa contribui para o cálculo dos efeitos atmosféricos;
- A **normal** (perpendicular a uma superfície) é fundamental
    - Não é calculada automaticamente ;
    - Precisa de ser especificada com **glNormal** ();

Cálculo do Vector Normal

- Triângulo
- Dados 3 vértices,
    - $\vec{n}$ (vetor normal) = normalizar((A - B) * (C-A))



Link: http://www.cin.ufpe.br/~marcelow/Marcelow/calculovetornormal.html

- Polígono planar
- Uma opção é usar a fórmula do triângulo para quaisquer 3 vértices:
    - Sujeito a erros (vectores pequenos ou quase colineares)
- Outra opção é determinar a equação do plano
    - **ax + by + cz + d = 0**
    - **Normal tem coordenadas (a, b, c);**
- Coeficientes a, b, c da equação do plano são proporcionais às áreas do polígono projectado nos planos yz, zx e xy

Cálculo do Vector Normal de Superfícies Implícitas

- Normal é dada pelo vector gradiente

Cálculo do Vector Normal de Superfícies Paramétricas

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

Componentes do Modelo de Phong (EADE)

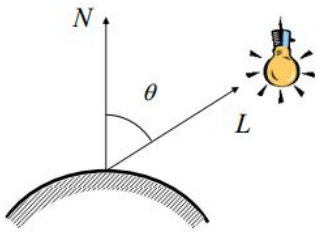
- **E**missão: contribuição que não depende de fontes de luz (fluorescência) ;
- **A**mbiente: contribuição que não depende da geometria;
- **D**ifusa: contribuição correspondente ao espalhamento da reflexão lambertiana (independente da posição do observador);
- **E**specular: contribuição referente ao comportamento de superfícies polidas;

Iluminação Ambiente

- Componente que modela como uma constante o efeito da reflexão de outros objectos do ambiente;
- Depende dos **coeficientes GL_AMBIENT** tanto das fontes luminosas quanto dos materiais;
- É ainda possível usar luminosidade ambiente não relacionada com fontes luminosas: (glLightModelfv (GL_LIGHT_MODEL_AMBIENT, params)

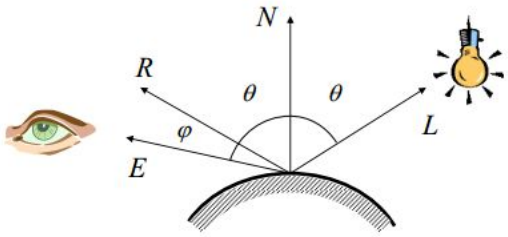
Iluminação Difusa

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



Iluminação Especular

- Simula a reflexão à maneira de um espelho (objetos altamente polidos);
- **Depende da posição do observador, objecto e fonte de luz**;
- Num espelho perfeito, a reflexão dá-se em ângulos iguais;
    - Observador só veria a reflexão de uma fonte pontual se estivesse na direção certa



- No modelo de Phong simulam-se reflectores imperfeitos, assumindo que luz é reflectida segundo um cone cujo eixo passa pelo observador.

Coeficiente de Especularidade

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

Atenuação

- Para fontes de luz **posicionais** (w = 1), é possível definir um factor de atenuação que leva em conta a distância d entre a fonte de luz e o objecto iluminado.
- Coeficientes são definidos pela função **glLight** ();
- Por omissão não há atenuação (c0=1, c1 =c2=0);

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

PDF “SGRAI-2009-OpenGL_3b-projeccoes.pdf”

O que é a transformação de projecção?

- O objetivo é definir um volume de visualização, que é usado de duas maneiras:
    - O volume de visualização determina como é que um objeto é projetado para o ecrã(usando um projeção em perspetiva ou ortogonal);
    - E define quais os objetos ou porção dos objetos que é retirada para fora da imagem final;

Ortográfica

- void glOrtho(
  GLdouble left, GLdouble right,
  GLdouble bottom, GLdouble top,
  GLdouble zNear, GLdouble zFar);
- zNear e zFar definem os **planos de corte** (clipping) relativos à posição da câmara
- Exemplo:
    - zNear = 1 & zFar = 3
    - Câmara (0, 0, 0)
    - Só são visíveis objectos com coordenada z no intervalo [-1, -3]
    - Mover câmara para (0, 0, 3)
    - Só são visíveis objectos com coordenada z no intervalo [2, 0]

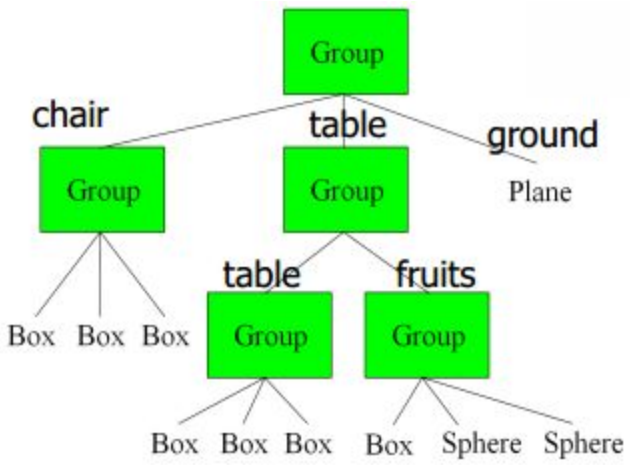
PDF “SGRAI-2009-OpenGL_4_hierarquicos.pdf”

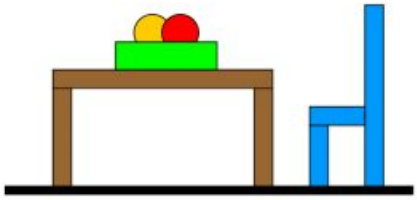
Modelos hierárquicos

- Objectos compostos por vários objectos.
- **Estruturas em árvore** para representar o modelo.
- Cada nó é uma transformação ou objecto
- Nós de um mesmo ramo representam **transformações acumuladas**;
- Nós de ramos diferentes são **transformações independentes;**

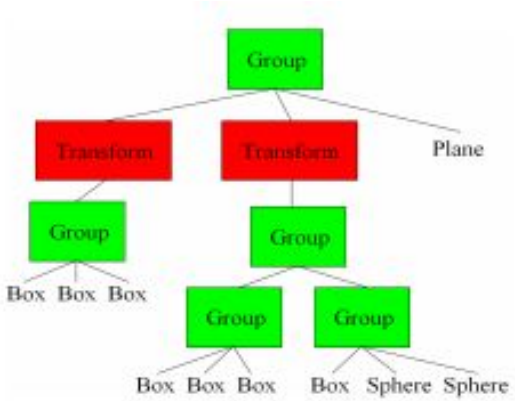
Hierarchical Grouping of Objects

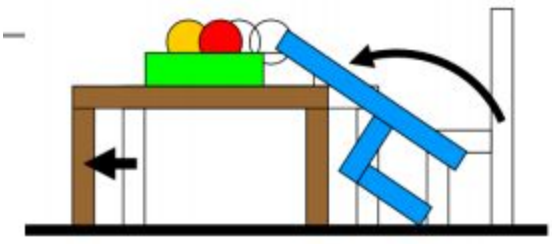
- Logical organization of scene





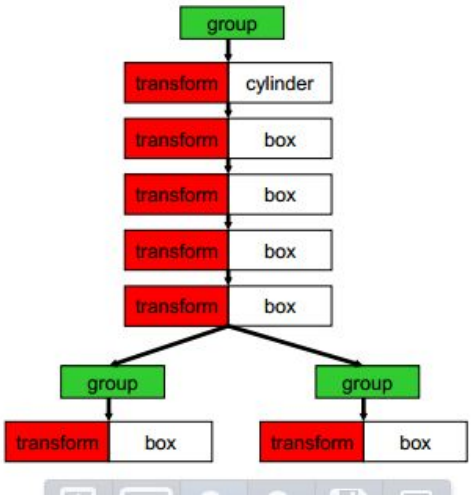
Hierarchical Transformation of Objects





Exemplo: braço robot

- BASE - rotação no plano
- 1º segmento - Rotação no eixo
- 2º segmento - Rotação no eixo
- Pulso - Rotação no eixo
- Garra - rotação no plano e Fecha/abre



Classe RobotArm

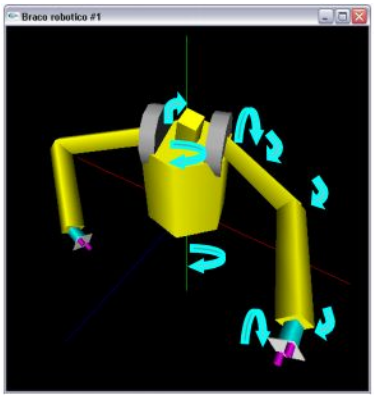
```
class RobotArm
{
        GLfloat rotBase;
        GLfloat rotSeg1;
        GLfloat rotSeg2;
        GLfloat rotWrist;
        GLfloat rotClaw;
        bool clawOpened;
}
```

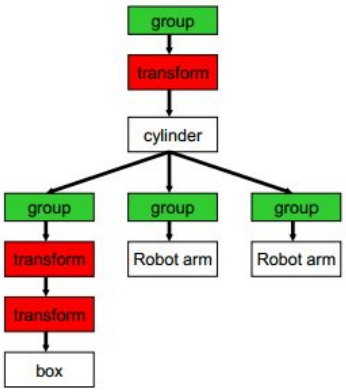
```
glPushMatrix();
        //base
        glRotatef(rotBase, 0, 0, 1);
        cylinderWithTopAndBottom(mode, BASE_RADIUS, BASE_HEIGHT, 12, 2);

        //segmento 1
        glTranslatef(0, 0, BASE_HEIGHT);
        glRotatef(rotSeg1, 1, 0, 0);
        box(mode, SEG1_WIDTH, SEG1_LENGTH);

        //segmento 2
        glTranslatef(0, 0, SEG1_LENGTH);
        glRotatef(rotSeg2, 1, 0, 0);
        box(mode, SEG2_WIDTH, SEG2_LENGTH);

        //pulso
        glTranslatef(0, 0, SEG2_LENGTH);
        glRotatef(rotWrist, 1, 0, 0);
        box(mode, WRIST_WIDTH, WRIST_LENGTH);

        //garra

        glTranslatef(0, 0, WRIST_LENGTH);
        glRotatef(rotClaw, 0, 0, 1);
        box(mode, CLAW_BASE_WIDTH, CLAW_BASE_LENGTH);
```

```
        // pinças
        glTranslatef(0, 0, CLAW_BASE_LENGTH);
        float d = (clawOpened ? CLAW_BASE_WIDTH/2 : CLAW_WIDTH/2);

        // pinça "direita"
        glPushMatrix();
                glTranslatef(-d, 0, 0);
                box(mode, CLAW_WIDTH, CLAW_LENGTH);
        glPopMatrix();

        // pinça "esquerda"
        glPushMatrix();
                glTranslatef(+d, 0, 0);
                box(mode, CLAW_WIDTH, CLAW_LENGTH);
        glPopMatrix();
glPopMatrix();
```

Exemplo: braço robot

- Tronco - rotação no plano
- Pescoço - Rotação no plano e Rotação no eixo
- Dois braços - Rotação no eixo






```
class Robot
{
        GLfloat rotTorso; ;
        GLfloat rotKneck; ;
        GLfloat rotHead; ;
        RobotArm left;
        RobotArm right;
}
```

```
glPushMatrix();

        // torso
        glRotatef(rotTorso, 0, 0, 1);
        cylinderWithTopAndBottom(mode, TORSO_WIDTH, TORSO_HEIGHT, 6, 2);

        glPushMatrix();
                // kneck
                glTranslatef(0, 0, TORSO_HEIGHT);
                glRotatef(rotKneck, 0, 0, 1);

                // head
                glRotatef(rotHead, 0, 1, 0);
                box(mode, HEAD_WIDTH, HEAD_HEIGTH);
        glPopMatrix();

        // left arm
        glPushMatrix();
                glTranslatef(-TORSO_WIDTH/2, 0, TORSO_HEIGHT);
                glRotatef(-90, 0, 1, 0);
                left.Display(mode, showAxis);
        glPopMatrix();

        // rigth arm
        glPushMatrix();
                glTranslatef(+TORSO_WIDTH/2, 0, TORSO_HEIGHT);
                glRotatef(+90, 0, 1, 0);
                right.Display(mode, showAxis);
        glPopMatrix();
glPopMatrix();
```

PDF “SGRAI-2009-OpenGL_5_iluminacao.pdf”

Iluminação

- OpenGL aproxima luz do mundo real em componentes **RGB**;
    - Fontes de luz, **emitem** luz;
    - Objectos (materiais) **reflectem** luz
        - Espalhando-a genericamente;
        - Numa direcção preferencial
- Luz de uma cena é proveniente de várias fontes de luz;
    - **Posicionais** ou **ambiente**

Tipos de iluminação

- **Ambiente**: Luz espalhada uniformemente em todas as direcções; resulta da luz a bater e ser reflectida em superfícies;
- **Difusa**: Luz vinda de uma determinada direcção; ao bater numa superfície a luz é espalhada uniformemente;
- **Especular**: Luz vinda de uma determinada direcção; ao bater numa superfície a luz é reflectida numa direcção especifica

Iluminação em OpenGL

- Passos:
    - **Definir normais** para cada vértice (irão determinar a orientação do objecto em relação às fontes de luz);
    - Configurar e posicionar uma ou mais fontes de luz ;
    - Configurar e escolher um modelo de iluminação (nível de luz ambiente e posicionamento do ponto de vista);
    - Definir propriedades dos materiais que compõem os objectos da cena

Vector normal

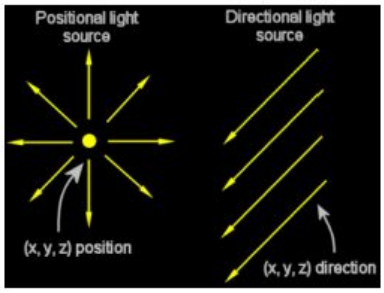
- **glNormal**(x, y, z) :
    - Utilizado para calcular a maneira como a luz incide na superfície do objecto;
    - Define um **vector perpendicular** à superfície/vértice;
    - Invocado dentro de glBegin/glEnd antes de glVertex;
    - Deve ter comprimento unitário;
    - **glEnable**(GL_AUTONORMALIZE)

Luzes

- glLightfv(luz, parâmetro, valor)
- **luz** : GL_LIGHT0 .. GL_LIGHT7
- parâmetro: GL_AMBIENT GL_DIFFUSE GL_SPECULAR GL_POSITION;

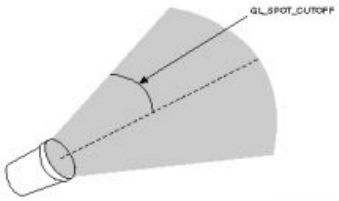
GL_POSITION

- **Luz direccional ou posicional**
  - (x, y, z, w);
  - **w = 0** direccional ;
  - **w ≠ 0** posicional;



Focos

- Por omissão uma luz emite em **todas as direcções;**
- É possível definir um foco indicando a direcção e a abertura;
- É possível definir a "concentração" de luz no cone GL_SPOT_EXPONENT



Ligar/desligar iluminação

- glEnable(GL_LIGHTING);
- glEnable(GL_LIGHTn);
- glDisable(GL_LIGHTn);

Modelo de iluminação

- glLightModel(parâmetro, valor);
- Parâmetro
  - GL_LIGHT_MODEL_AMBIENT - **Default**: (0.2, 0.2, 0.2, 1.0)
  - GL_LIGHT_MODEL_LOCAL_VIEWER - **Default**: GL_FALSE
  - GL_LIGHT_MODEL_TWO_SIDE - **Default**: GL_FALSE

Materiais

- Possuem componente **ambiente**, **difusa**, **especular** e **emissora;**
- **Ambiente** e **difusa** definem a cor do objecto e são **normalmente iguais;**
- **Especular** é normalmente **branco** para garantir a cor do foco de projecção;
- **Emissora** simula uma fonte de luz dentro do próprio objecto

- **Percentagem** de reflexão de cada componente de cor RGB
- Exemplo:
  - R = 1, G= 0.5, B = 0;
  - **Reflecte todo** o vermelho (R = 1)
  - **Reflecte** 50% do verde (G = 0.5)
  - **Absorve** todo o azul (B = 0);
- Ou seja:
- Fonte de Luz (LR, LG, LB)
- Material (MR, MG, MB)
- "Cor" visível = (LR*MR, LG*MG, LB*MB)

Materiais: exemplo

- Objecto vermelho (R=1, G=0, B=0)
- Luz vermelha(1,0,0) - objecto vermelho (1*1, 0*0, 0*0)
- Luz branca(1,1,1) - objecto vermelho(1*1,1*0,1*0)
    - Componente vermelha da luz branca é reflectida
- Luz verde(0,1,0) - objecto preto(0*1,1*0,0*0)
    - Todo o verde é absorvido

Posicionar luzes na cena

- Em OpenGL a posição de uma fonte de luz (posicional) é tratada como uma **primitiva**, sendo por isso transformada pela matriz de modelo/vista;
- Luz fixa
    - Definir posição da luz após as transformações de vista ;
- Luz móvel
    - Aplicar transformação antes de definir posição da luz
- Luz que acompanha o ponto de vista
    - Definir posição da luz antes de qualquer transformação (i.e., a seguir a **glLoadIdentity**)
    - Modificar o ponto de vista usando gluLookAt

Luz fixa

```
void reshape(int w, int h)
{
glViewport (0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode (GL_PROJECTION);
glLoadIdentity();
if (w <= h)
        glOrtho (-1.5, 1.5, -1.5*h/w, 1.5*h/w, -10.0, 10.0);
else
        glOrtho (-1.5*w/h, 1.5*w/h, -1.5, 1.5, -10.0, 10.0);
glMatrixMode (GL_MODELVIEW);
glLoadIdentity();

//viewing transformation
/* later in init() */
GLfloat light_position[] = { 1.0, 1.0, 1.0, 1.0 };                  //definir a posição da luz
glLightfv(GL_LIGHT0, GL_POSITION, position);
}
```

Luz móvel

```
static GLdouble spin;

void display(void)
{
GLfloat light_position[] = { 0.0, 0.0, 1.5, 1.0 };
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
        // viewing transformation
        gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
        // light position
        glPushMatrix();
                glRotated(spin, 1.0, 0.0, 0.0);                 //aplicar transf. antes da posição
                glLightfv(GL_LIGHT0, GL_POSITION, light_position);
        glPopMatrix();

// model
glutSolidTorus (0.275, 0.85, 8, 15);
glPopMatrix();
glFlush();
}
```

Luz no ponto de vista

```
void reshape (int w, int h)
{
        glViewport(0, 0, (GLint) w, (GLint) h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(40.0, (GLfloat) w/(GLfloat) h, 1.0, 100.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        //light position before viewing transformation
        GLfloat light_position() = {0.0, 0.0, 0.0, 1.0};        //Definir posição a seguir ao loadId.
        glLightfv(GL_LIGHT0, GL_POSITION, light_position);
}
```

```
static GLdouble ex, ey, ez, upx, upy, upz;
void display(void)
{
glClear(GL_COLOR_BUFFER_MASK | GL_DEPTH_BUFFER_MASK);
glPushMatrix();
        gluLookAt (ex, ey, ez, 0.0, 0.0, 0.0, upx, upy, upz);          //modificar o ponto de vista
        glutSolidTorus (0.275, 0.85, 8, 15);
glPopMatrix();
glFlush();
}
```

PDF “Texturas.pdf”

Detalhes de Superfícies

PDF “SGRAI-2008-OpenGL_7_feedback.pdf”

Selecção & feedback

Modos do OpenGL

- glRenderMode(mode)
  - GL_RENDER
    - **Modo normal de funcionamento**: desenho das primitivas no ecrã
  - GL_SELECTION
    - **Modo de selecção**: **não desenha no ecrã** mas devolve informação (nome simbólico) sobre os objectos que seriam desenhados;
    - **Modo picking**: idêntico mas com base na posição de dispositivo de input;
  - GL_FEEDBACK
    - **Modo feedback**: não desenha no ecrã mas devolve **informação** sobre os elementos gráficos que seriam desenhados no ecrã (**vértices**, **cores**, ...)

Selecção

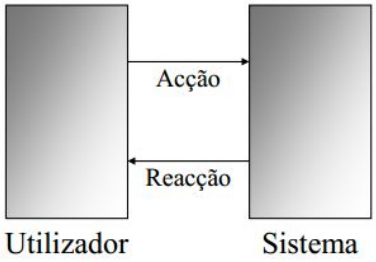
Passos a seguir:

- **Inicializar buffer** de retorno;
- Entrar modo de **selecção**;
- **Inicializar stack** de nomes simbólicos;
- Definir **volume de visualização**;
- “Desenhar” a cena contendo o nome simbolico dos objectos;
- **Sair do modo selecção** e processar os registos do buffer de retorno

Selecção

PDF “Interaccao_Pessoa-Maquina.pdf”

Sistema Interativo



Importancia da UI

- Para o utilizador, **a interface é o sistema;**
- A **comunicação com o sistema** é pelo menos **tão importante** como a **computação realizada pelo mesmo;**
- O **sucesso** de uma aplicação depende da **qualidade da interface** com o utilizador;
- **48% (valor médio)** a quase **100% (valor máximo)** do código de um sistema interactivo está relacionado com o **suporte da interface**

Definição de Usabilidade

- Combinação de características centradas no **utilizador**:
  - facilidade de **aprendizagem**;
  - **rapidez** na execução de tarefas;
  - **taxa de erros reduzida**;
  - satisfação subjectiva do utilizador;
  - retenção ao longo do tempo;

Desenvolvimento da Interacção

- Necessidades especiais não partilhadas com o desenvolvimento de software;
- Conflito não susceptível de ser evitado:
  - o que é melhor para o utilizador raramente é fácil de levar à prática pelo programador;

Domínios de Desenvolvimento da UI

	Comportamental	Estrutural
Objecto de desenvolvimento	Interacção	Software que suporta a interacção
Ponto de vista adoptado	Utilizador	Sistema
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

Princípios Orientadores

- **Estudo dos factores humanos:**
  - ciência da determinação dos princípios do comportamento humano, baseada na realização de testes empíricos com participantes humanos;
  - **objetivo**: optimização da performance humana, designadamente a redução da taxa de erros e o aumento do rendimento, satisfação e conforto do utilizador

Design centrado no Utilizador

- **Conhecer/envolver o utilizador:**
  - entrevistas**;**
  - observação no trabalho;
  - análise de necessidades;
  - análise do perfil dos utilizadores;
  - análise de tarefas;
  - análise do fluxo de informação;
- **Avaliação de usabilidade**
- **Prevenir contra os erros do utilizador:**
  - edição baseada na sintaxe: por exemplo, parêntese esquerdo/parêntese direito
  - inibição, em função do contexto, de opções ilegais;
  - requerer a confirmação de acções potencialmente destrutivas
- **Optimizar as operações realizadas pelo utilizador:**
  - teclas aceleradoras;
  - teclas de função;
  - macros (pré-definidos e definidos pelo utilizador);
  - abreviaturas;
- **Manter o controlo do lado do utilizador:**
  - O utilizador deve sentir que comanda o sistema e não o contrário;
- **Ajudar o utilizador a familiarizar-se com o sistema:**
  - de um modo geral, o utilizador não deverá necessitar de mais do que uma página de informação para começar a trabalhar com um sistema que não conhece

Modelo do Sistema

- **Dar ao utilizador um modelo mental, consistente, do sistema, baseado nas tarefas a efectuar:**
  - Paradigmas e metáforas de interacção:
    - linha de comandos (acção-objecto; amo/escravo);
    - manipulação directa (objecto-acção; tampo da secretária);
    - desenho (simultaneidade acção/objecto; papel/lápis)

Consistência

- **Princípio do menor espanto:**
  - o utilizador espera que tarefas similares sejam efectuadas de forma similar;

    - para semânticas similares deverão ser usadas sintaxes similares e vice-versa

Simplicidade

Sistemas interactivos complexos → Interfaces complexas

- Tarefas simples deverão ser simples de efectuar
- Tarefas complexas deverão ser divididas em subtarefas mais simples

Limitações da Memória Humana

- Hierarquização versus linearização na decomposição de tarefas grandes e 01-03-2007 19 complexas em subtarefas mais pequenas e simples
- Reconhecer em vez de relembrar

Aspectos Cognitivos

- Mnemónicas
- Analogias com o mundo real (metáforas):
    - tampo da secretária: pastas, documentos, cesto de reciclagem
    - folha de cálculo

Feedback

- **Feedback informativo:**
    - articulatório
    - semântico
- **Indicadores de estado para tarefas potencialmente demoradas:**
    - forma do cursor
    - barra de progressão
- **Tempo de resposta do sistema adequado à tarefa que está a ser efectuada:**
    - dactilografia, movimentação do cursor, 01-03-2007 22 accionamento de um botão do rato: 50 a 150 ms;
    - tarefas simples e frequentes: inferior a 1 s;
    - tarefas vulgares: 2 a 4 s
    - tarefas complexas: até 12 s

Mensagens do Sistema

- **Centradas no utilizador e nas tarefas a efectuar:**
    - “505 hex 0001F9 double words of storage were not recovered”
    - “Hit any key to continue”
- **Positivas e não ameaçadoras**
    - “Fatal error, run aborted”
    - “Disastrous string overflow, job abandoned”
    - “Catastrophic error, logged with operator”
- **Termos informativos e construtivos nas mensagens de erro**
    - “Invalid entry”
    - “Inventory part number is out of allowable range”
    - “Inventory numbers range from 0000 to 9999”

- **O sistema deve assumir a culpa dos erros**
    - “Illegal command” versus “Unrecognized command”

Antropomorfização

- Não antropomorfizar (não atribuir características humanas a objectos não humanos tais como, por exemplo, automóveis e computadores)

Modalidade

- Um modo de interacção é um estado da interface no qual uma acção do utilizador tem um significado diferente (e um resultado diferente) do que teria noutro modo/estado qualquer
- Indicadores de modo

Reversibilidade

- **A possibilidade de “desfazer” acções com facilidade encoraja os utilizadores no sentido de explorarem o sistema:**
    - existência de um comando “undo”:
    - navegação pelo sistema;

Chamada de atenção

- Os avanços da tecnologia proporcionam inúmeras maneiras de chamar a atenção do utilizador. Esta prática deve ser usada criteriosamente, de forma a evitar usos incorrectos ou excessivos;
- Texto(tipos,tamanhos,negritos, sublinhados, intermitências,maiúsculas);
- Áudio(sons suaves/ásperos)
- Cores (quantidade, códigos)

Visualização

- Manter a inércia;
- Organizar o conteúdo das janelas de modo a gerir a complexidade:
    - texto conciso;
    - baixa densidade global;
    - baixa densidade local;
    - aspeto geral equilibrado;

Utilizadores

- Ter em conta as preferências pessoais:
    - personalização da interface
- Ter em conta os diferentes níveis de experiência
    - inexperiente;
    - intermitente
    - experiente