Introdução a Programação OpenGL

Luiz A. Pavão

introdução

OpenGL pode ser definida como um software de interface para o hardware gráfico.

Na essência, é uma biblioteca gráfica tridimensional extremamente portável.

histórico

- A OpenGL é o resultado do esforço da Silicon Graphics Inc. - SGI em aumentar a portabilidade da IRIS GL - API gráfica das workstations SGI.

- A nova biblioteca oferecia o poder da GL, mas seria aberta, permitindo assim fácil adaptabilidade para outras plataformas de hardware e sistemas operacionais.

Histórico (2)

Em julho de 1992 a versão 1.0 da OpenGL foi introduzida, quase que ao mesmo tempo da primeira Conferência de Desenvolvedores Win32.

Nesta mesma época a SGI e a Microsoft já estavam juntas trazendo a OpenGL para as próximas versões do Windows NT (v 3.5).

padronização

O objetivo da SGI era transformar a OpenGL em um padrão, para isto foi criada a ARB - Open Architecture Review Board, com os seguintes membros fundadores: SGI, DEC, IBM, Intel e Microsoft.

Todos os encontros da ARB são abertos ao público e companhias não-membros podem participar das discusões.

o que é a OpenGL

É uma biblioteca com características de linguagem, possuindo definições sintáticas e semânticas.

Apresenta-se como uma linguagem procedural ao invés de uma linguagem gráfica descritiva.

Ao invés de descrever a cena e como ela deve aparecer, descreve-se os passos necessários até chegar a imagem desejada.

como funciona a OpenGL

Os comandos e as funções são usados para desenhar primitivas gráficas (pontos, linhas, e polígonos) em três dimensões.

Adicionalmente, a OpenGL suporta iluminação, sombreamento, mapeamento de textura, animação e efeitos especiais.

como funciona a OpenGL (2)

Operações:

- mapeamento de coordenadas;
- janelamento;
- projeção;
- recorte;
- transformações geométricas.

a divisão de trabalho

A OpenGL, ou melhor a API da OpenGL é dividida em três bibliotecas distintas:

- OpenGL ou gl (opengl32.dll, gl.h)
- Auxiliary ou Toolkit (glaux.lib, glaux.h)
- Utility ou glu (glu32.dll, glu.h)

biblioteca gl

opengl32.lib

Aqui estão as funções e comandos que definem a especificação OpenGL conforme a ARB.

biblioteca auxiliar

glaux.lib

As funções contidas nesta biblioteca não fazem parte da especificação OpenGL. É na verdade um conjunto de ferramentas que criam um "framework" para as chamadas de funções OpenGL.

Foi criada com objetivo de facilitar o aprendizado de programar em OpenGL.

biblioteca auxiliar (2)

Com ela podemos escrever programas simples em OpenGL, sem nos preocuparmos com detalhes de criação de janelas, contextos de dispositivos e loops de mensagens, operações não suportadas pela OpenGL.

biblioteca utilitária

glu32.lib

Esta biblioteca contém funções que facilitam tarefas como desenhar esferas, discos e cilindros - objetos tridimensionais primitivos.

Ela foi escrita utilizando comandos OpenGL.

introdução a programação OpenGL
tipos de dados

Tipo OpenGL	Represent.Interna	Definição C	Sufixo
GLbyte	8 bits int.	signed char	b
GLshort	16 bits int.	short	s
GLint, GLsizei	32 bits int.	long	l
GLfloat, GLclampf	32 bits f.p.	float	f
GLdouble, GLclampd	64 bits f.p.	double	d
GLubyte, GLboolean	8 bits uns. int.	unsigned char	ub
GLushort	16 bits uns. int.	unsigned short	us
GLuint, GLenum,			
GLbitfield	32 bits uns. int.	unsigned long	ui

introdução a programação OpenGL
nome de funções

As funções OpenGL seguem uma convenção que nos informa de qual biblioteca pertence e quantos e que tipos de argumentos necessita.

Ex.: glColor3f( );

gl – função da biblioteca gl

Color - comando

3 - número de argumentos

f - tipo de dado (float)

introdução a programação OpenGL
GL state machine

A OpenGL mantém uma série de variáveis de estado que controlam o seu comportamento e a geração da imagem.

Com os comandos **glEnable( );** e **glDisable( );** controlamos os switches OpenGL, como p.ex. ativar ou desativar a iluminação.

Outros tipos de dados são obtidos através de variações do comando **glGet*( );** (**glGetBoolean**, **glGetFloat**, ...)

Coleções de variáveis podem ser armazanadas e obtidas através dos comandos **glPushAttrib( );** e **glPopAttrib( );**

introdução a programação OpenGL
1º programa

```
void main(void)
    {
    auxInitDisplayMode(AUX_SINGLE|AUX_RGBA);
    auxInitPosition(100,100,200,200);
    auxInitWindow(“ Programa OpenGL Básico”);
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
    cprintf(“pressione qualquer tecla para finalizar”);
    getch();
    }
```

introdução a programação OpenGL
1º programa (2)

auxInitDisplayMode (AUX_SINGLE|AUX_RGBA);

Esta função define o Display Mode - configuração da janela gráfica. Os flags definem uma janela "single buffered" e modo de cor "tipo RGBA".

As cores são definidas em termos de seus componentes RGB: vermelho, verde e azul. O A refere-se ao canal Alfa, utilizado entre outras coisas para o efeito de transparência.

Single buffered - todos os comandos de desenho são executados diretamente na janela.

A alternativa é o Double Buffer, usado normalmente em animação.

introdução a programação OpenGL
1º programa (3)

auxInitPosition (100,100,200,200);

Define aonde a janela será criada, e qual o seu tamanho.

A coordenada (100,100) é o canto esquerdo superior da janela.

Os valores (200,200) definem a largura e a altura da janela.

Valores são medidos em pixels.

Mapeamento de Coordenadas em OpenGL:
Origem - canto inferior esquerdo;
Crescimento positivo de X - esquerda para direita;
Crescimento positivo de Y - de baixo para cima.

introdução a programação OpenGL
1º programa (4)

auxInitWindow (“Programa básico OpenGL”);

Esta função cria a janela e define seu título.

glClearColor (0.0f, 0.0f, 1.0f, 0.0f);

Define a cor de limpeza (back-ground)

glClear(GL_COLOR_BUFFER_BIT);

Executa a limpeza

introdução a programação OpenGL
1º programa (5)

glFlush();

Internamente a OpenGL utiliza um “render pipeline” que processa sequencialmte os comandos.

Estes comandos são enfileirados, até que o OpenGL Server processe vários comandos de uma só vez - através do glFlush()

Isto aumenta a performance, principalmente quando desenhamos objetos complexos.

O desenho é acelerado porque o hardware gráfico é acessado menos vezes para um conjunto de instruções de desenho.

introdução a programação OpenGL
2º programa

Com algumas modificações, o primeiro programa se comportará como uma aplicação Windows e adicionalmente desenhará um retângulo.

Uma função CALL BACK é criada com o objetivo de podermos tratar as mensagens do Windows, ela reúne os comandos relativos a biblioteca gl, e é passada como parâmetro para a função da biblioteca auxiliar que trata as mensagens.

introdução a programação OpenGL
2º programa (2)

```
void CALLBACK RenderScene(void)
    {
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glRectf(100.0f, 150.0f, 150.0f, 100.0f);
    glFlush();
    }
void main(void)
    {
    auxInitDisplayMode(AUX_SINGLE|AUX_RGBA);
    auxInitPosition(100,100,250,250);
    auxInitWindow(“ Programa OpenGL Básico Melhorado”);
    auxMainLoop(RenderScene);
    }
```

introdução a programação OpenGL
2º programa (3)

RenderScene(void);

Nesta função é que é colocado todo o código responsável pelo desenho.

O processo de desenho OpenGL é sempre referenciado como Rendering.

A declaração CALLBACK é necessária porque iremos dizer à biblioteca AUX que deverá chamar esta função sempre que a janela precisar de atualização:

auxMainLoop(RenderScene);

introdução a programação OpenGL
2º programa – desenhando o retângulo

Primeiro define-se a cor corrente; todo o desenho após esta definição será feito nesta cor (vermelho)

```
glColor3f(1.0f, 0.0f, 0.0f);  // R=1.0, G=0.0, B=0.0
```

Os três parâmetros definem as três componentes R, G, B de maneira semelhante a glClearColor(), sem o componente alfa.

A função responsável pelo desenho do retângulo é:

```
glRect(x1, y1, x2, y2);
```

onde (x1,y1) representa o canto esquerdo superior do retângulo, e (x2,y2) o canto inferior direito.

introdução a programação OpenGL
2º programa – inicialização (main)

```
auxInitDisplayMode(AUX_SINGLE|AUX_RGBA);
auxInitPosition(100,100,250,250);
auxInitWindow(“ Programa OpenGL Básico Melhorado”);
auxMainLoop(RenderScene);
```

A função auxMainLoop() recebe como argumento um ponteiro para a função RenderScene(), que é chamada a cada vez que a janela necessita de atualização.

A função CALLBACK é sempre chamada, quando a janela é criada, movida, redimensionada ou quando é descoberta por outra janela.

introdução a programação OpenGL
3º programa

Nesta alteração do programa anterior, nosso objetivo é o de criar uma aplicação que tenha a habilidade de redimensionar o conteúdo da janela gráfica, mantendo as proporções originais do desenho.

Para isto utilizamos as funções de janelamento e mapeamento de coordenadas da OpenGL.

introdução a programação OpenGL
3º programa – sistema de coordenadas

Até agora os pontos P(x,y), utilizados por nós eram definidos em termos de um sistema de coordenadas baseado na tela física do computador - dependendo intimamente da resolução previamente definida.

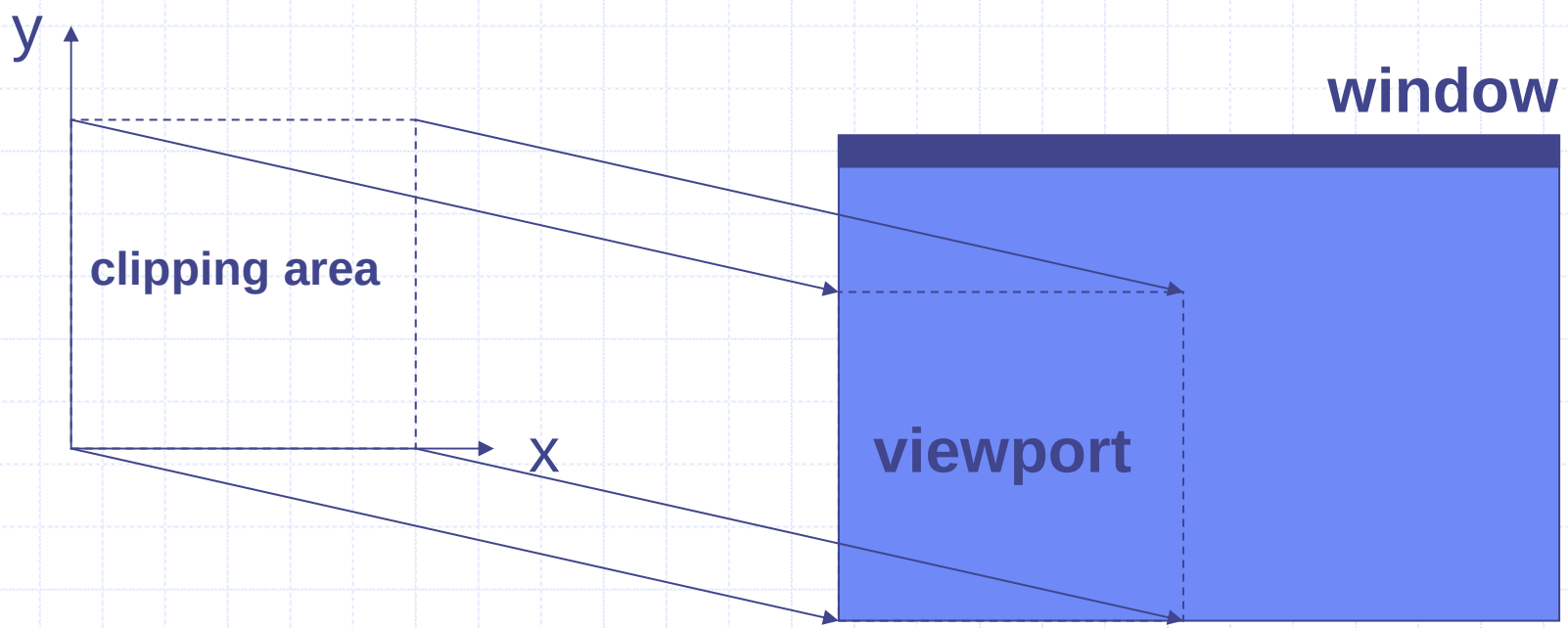
Em OpenGL, quando criamos uma janela em que desenharemos, precisamos especificar antes o "Sistema de Coordenadas" e como mapear (transformar) em coordenadas físicas da tela (pixels).

Este mapeamento é feito especificando a região do espaço cartesiano que a janela ocupa; esta região é denominada área de recorte (clipping area).

introdução a programação OpenGL
3º programa – VIEWPORT

Raramente a área de recorte (clipping area) é exatamente do tamanho da janela.

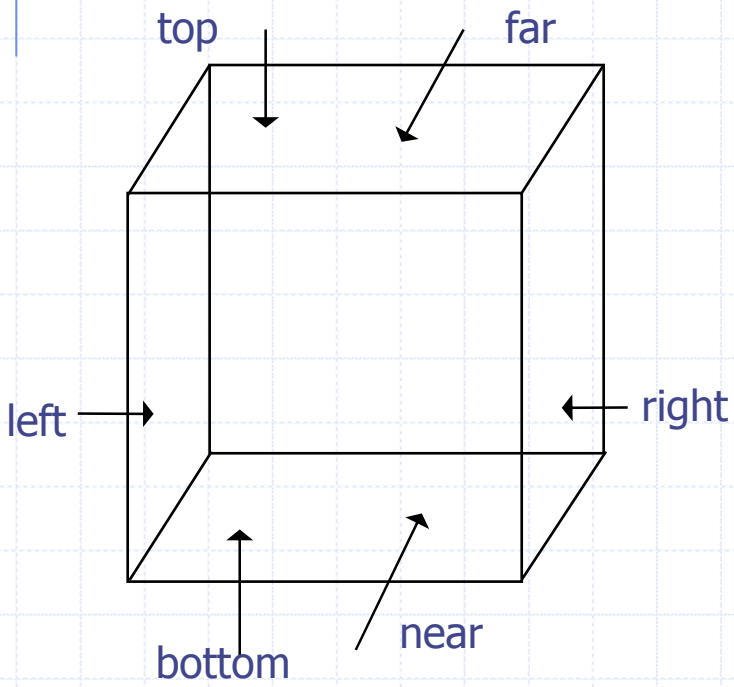
A viewport é a região dentro da janela que utilizamos para o desenho.



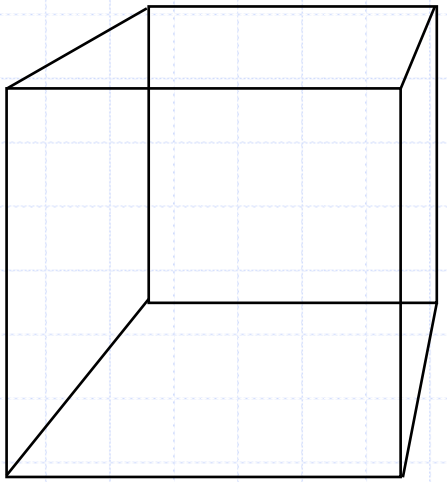
introdução a programação OpenGL
3º programa – Volume de Recorte

No espaço tridimensional a área de recorte vira volume de recorte:

Projeção Ortogonal



Projeção Perspectiva



introdução a programação OpenGL
3º programa – Conteúdo Redimensionável

auxReshapeFunc();

tem por objetivo detectar a mudança de tamanho da janela. Retorna os novos valores da largura e altura da janela (w e h).

void glViewport(GLint x, GLint y, GLsizei width, GLsizei);

define a configuração da viewport, x e y especificam o canto inferior esquerdo da viewport dentro da janela (valores especificados em pixels). Os valores width e height, definem a largura e altura.

introdução a programação OpenGL
3º programa – Definindo o Volume deRecorte

void glOrtho(GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top,
GLdouble near, GLdouble far);

Tem por objetivo a especificação dos limites do volume de recorte em uma projeção ortogonal.

introdução a programação OpenGL
3º programa – Controlando a Razão de Aspecto

Razão de Aspecto: Relação entre a largura e altura

Para manter a relação de maneira uniforme:

```
if (w<=h)
                glOrtho(0,250,0,250*h/w,1,-1);
else
                glOrtho(0,250*w/h,0,250,1,-1);
```

introdução a programação OpenGL
3º programa – O Código Final

```
void CALLBACK ChangeSize(GLsizei w, GLsizei h)
{
if (h==0)                       //prevenindo divisão por zero
        h=1;
glViewport(0,0,w,h);            //ajusta viewport  para as dimensões da Janela

glLoadIdentity();               //reseta sistema de coordenadas
if (w<=h)
                glOrtho(0,250,0,250*h/w,1.0,-1.0);
else
                glOrtho(0,250*w/h,0,250,1.0,-1.0);
}

void main(void)
{
        auxInitDisplayMode(AUX_SINGLE | AUX_RGBA);
        auxInitPosition(100,100,250,250);
        auxInitWindow("Janela Redimensionável");
        auxReshapeFunc(ChangeSize);
        auxMainLoop(RenderScene);
}
```

introdução a programação OpenGL
3º programa – Observações

A função CALLBACK RenderScene não é modificada.

A função auxReshapeFunc() recebe como parâmetro um ponteiro para a função que deverá ser chamada quando a janela é redimensionada.

A função auxMainLoop() recebe como parâmetro um ponteiro para a função que deverá ser chamada quando a janela precisa de atualização.

introdução a programação OpenGL
3º programa – Mapeamento de Coordenadas

IMPORTANTE: Não há necessidade de termos a clipping area numericamente igual a viewport, mas sim sua proporção.

O código anterior talvez nos leve a esta falsa interpretação, mas alterando o código, podemos entender um pouco melhor;

```
glOrtho(0,2500,0,2500*h/w,1.0,-1.0);

glRectf(1000.0f, 1000.0f, 1500.0f, 1500.0f);
```

introdução a programação OpenGL
3º programa – "centralizando"

No código anterior o ajuste é feito no lado direito e acima (parâmetros right e top da glOrtho) para centralizarmos o a origem do sistema de coordenadas no centro da janela fazemos o seguinte:

Antes:

```
if (w<=h)
        glOrtho(0,250,0,250*h/w,1.0,-1.0);
else
        glOrtho(0,250*w/h,0,250,1.0,-1.0);
```

Modificação:

```
nRange=125;
if (w <= h)
        glOrtho (-nRange, nRange, -nRange*h/w,
                nRange*h/w, -nRange, nRange);
else
        glOrtho (-nRange*w/h, nRange*w/h, -nRange,
        nRange, -nRange, nRange);
```

introdução a programação OpenGL primitivas

Estrutura básica para o desenho de primitivas:

```
glBegin(Parâmetro_Primitiva);
        glVertex3f(0.0f, 0.0f, 0.0f);
        glVertex3f(50.0f, 50.0f, 50.0f);
        ......
glEnd();
```

Parâmetro_Primitiva:

GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_FAN, GL_TRIANGLE_STRIP, GL_QUADS, GL_QUAD_STRIP

introdução a programação OpenGL
polígonos - triângulo

Ordem de construção de polígonos

A combinação da ordem e a direção na qual os vértices são especificados é de extrema importância, definem se a face é frontal ou posterior.

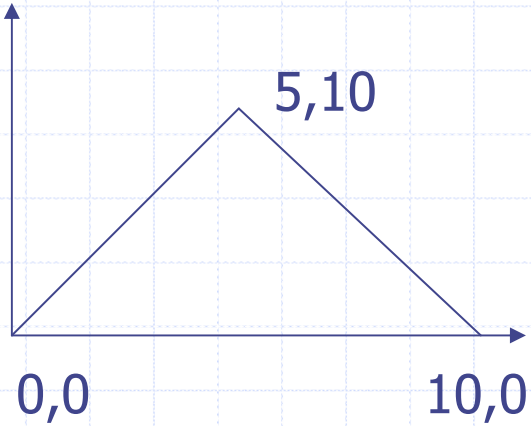
Por default a OpenGL considera polígonos construídos em sentido anti-horário como frontais.

A definição do que é frente ou trás é importante quando se deseja definir diferentes propriedades para cada um dos lados.

introdução a programação OpenGL
polígonos - triângulo

A seguinte porção de código desenha 2 polígonos; um frontal e um posterior.

```
glBegin(GL_TRIANGLES);
        glVertex2f(0.0f, 0.0f);
        glVertex2f(10.0f, 0.0f);
        glVertex2f(5.0f, 10.0f);

        glVertex2f(0.0f, 0.0f);
        glVertex2f(5.0f, 10.0f);
        glVertex2f(10.0f, 0.0f);
glEnd();
```



Horário – back face:

(0,0) (0,10) (10,0)

Anti-horário – front face:

(0,0) (10,0) (0,10)

introdução a programação OpenGL
ajustando as cores dos polígonos

Até agora nossos programas ajustavam apenas uma vez a cor corrente, e desenhavam uma forma simples.

Com múltiplos polígonos, podemos usar cores diferentes para cada polígono.

As cores são atualmente especificadas por vértice, não por polígono.

O Modelo de Sombreamento - Shading Model - afeta o polígono de duas formas, dependendo da configuração:

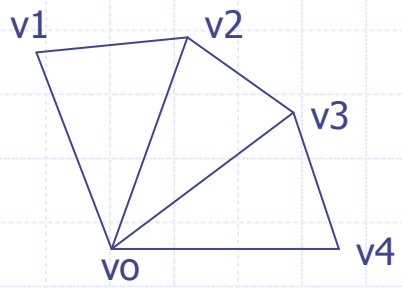
glShadeModel(GL_FLAT);

Colorido solidamente - utilizando a cor corrente quando da última especificação de vértice.

glShadeModel(GL_SMOOTH);

Colorido interpoladamente - utiliza as cores de cada vértice, criando uma variação linear de cor entre cada vértice.

introdução a programação OpenGL
construindo objetos - cone



lateral

```
glBegin(GL_TRIANGLE_FAN);
glVertex3f(0.0f, 0.0f, 75.0f);
for(angle=0.0f; angle<(2.0f*GL_PI);
angle+=(GL_PI/8.0f))
            {
            x=50.0f*sin(angle);
            y=50.0f*cos(angle);
            if((iPivot %2)==0)
                  glColor3f(0.0f, 1.0f, 0.0f);
            else
                  glColor3f(1.0f, 0.0f, 0.0f);
            iPivot++
            glVertex2f(x,y);
            }
glEnd();
```

base

```
glBegin(GL_TRIANGLE_FAN);
glVertex2f(0.0f, 0.0f);
for(angle=0.0f; angle<(2.0f*GL_PI);
angle+=(GL_PI/8.0f))
            {
            x=50.0f*sin(angle);
            y=50.0f*cos(angle);
            if((iPivot %2)==0)
                  glColor3f(0.0f, 1.0f, 0.0f);
            else
                  glColor3f(1.0f, 0.0f, 0.0f);
            iPivot++
            glVertex2f(x,y);
            }
glEnd();
```

introdução a programação OpenGL
transformações geométricas

Operações:

- Translação (dx, dy, dz);
- Rotação (θ, i, j, k);
- Escala (sx, sy, sz).

Objetivos / Contexto:

- Modelagem;
- Visualização.

introdução a programação OpenGL
transformações geométricas - Matrizes

Translação em coordenadas homogêneas (x,y,z,1):

$$T = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P'(x,y,z,1) = P(x,y,z,1) * T$$

introdução a programação OpenGL
transformações geométricas - Matrizes

Escala e Rotação:

S=

Sx	0	0	1
0	Sy	0	1
0	0	Sz	1
0	0	0	1

Rz=

Cos θ	-Sen θ	0	0
Sen θ	Cos θ	0	0
0	0	1	0
0	0	0	1

$P'(x,y,z,1) = P(x,y,z,1) * S$

$P'(x,y,z,1) = P(x,y,z,1) * Rz$

introdução a programação OpenGL
transformações geométricas - Matrizes

$$P'(x,y,z,1) = P(x,y,z,1) * T$$
$$P'(x,y,z,1) = P(x,y,z,1) * S$$
$$P'(x,y,z,1) = P(x,y,z,1) * Rz$$

Com o uso das coordenadas homogêneas pode-se utilizar a operação de multiplicação de vetor-matriz em todas as operações.

introdução a programação OpenGL
transformações - comandos

```
glTranslatef(dx, dy, dz);
glRotatef(δ, i, j, k);
glScalef(sx, sy, sz);
```

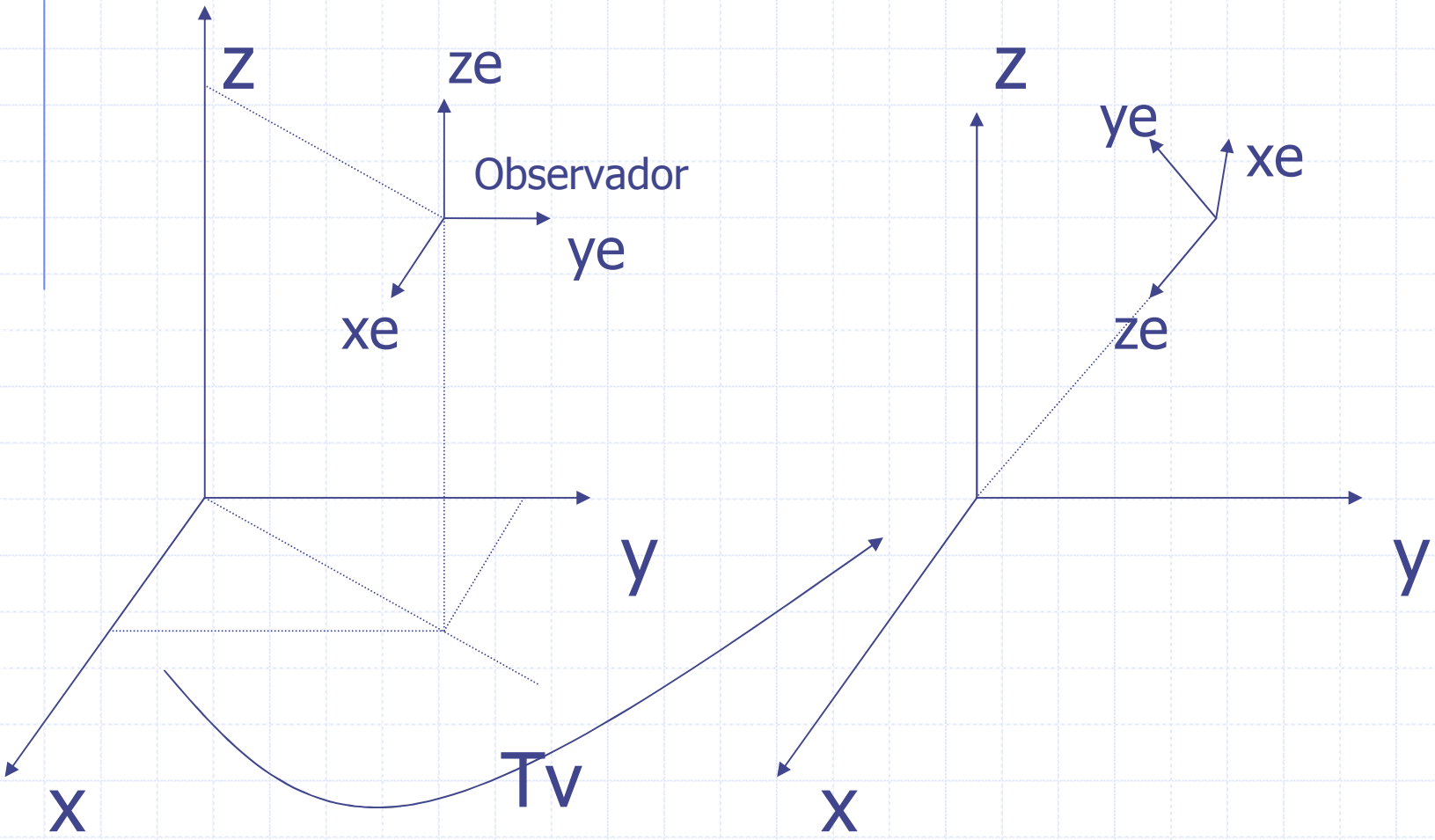
Com estes comandos definem-se as operações a serem efetuadas sobre as coordenadas subsequentes.

introdução a programação OpenGL
transformações de visualização

A transformação de visualização é a primeira aplicada a cena. Por default, o ponto de observação - alvo do observador, está na origem (0,0,0), e o observador está no eixo z (parte positiva).

A transformação de visualização permite-nos modificar o ponto de observação para qualquer outro lugar desejado. Desta forma o observador pode "olhar" para qualquer direção em relação a cena.

introdução a programação OpenGL
a transformação de visualização



introdução a programação OpenGL
a transformação de visualização

Tv = Translação para origem *
Rotação em x de 90º *
Rotação em y de Θº *
Rotação em x de δº *
Inversão do eixo z

P^e = P * Tv (ponto P no sistema de coordenadas do observador)

introdução a programação OpenGL
transformações de Modelagem

Transformações de modelagem são utilizadas para manipular a cena e os objetos nela contidos.

Estas transformações significam: mover, rotacionar e redimensionar objetos.

introdução a programação OpenGL
transformação ModelView

Transformações de visualização e modelagem são, de fato, o mesmo em termos de seus efeitos internos, bem como em relação a aparência final da cena. A distinção entre as duas é feita para ajudar o programador. Não há diferença real entre mover um objeto para trás, e mover o sistema de coordenadas para frente.

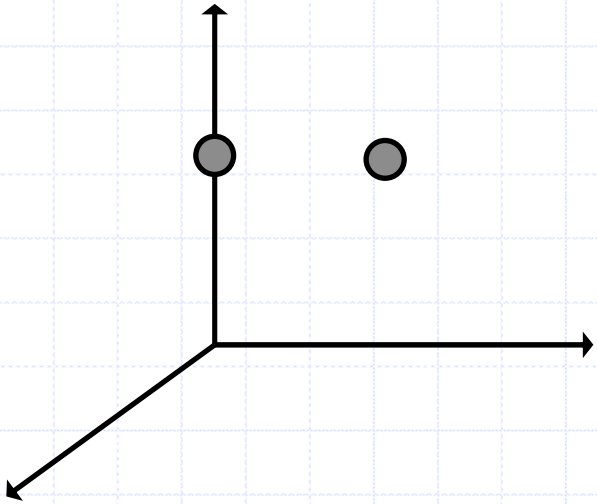
O termo ModelView é usado para indicar que podemos pensar na transformação tanto como uma transformação de modelagem ou como de visualização.

introdução a programação OpenGL
transformações - sintaxe

```
// 10 unidades a cima
glTranslatef(0.0f, 10.0f, 0,0f);

// Desenhando uma esfera
auxSolidSphere(1.0f);

// 10 unidades a direita
glTranslatef(10.0f, 0.0f, 0.0f);

// Desenhando uma segunda esfera
auxSolidSphere(1.0f);
```

introdução a programação OpenGL
transformações

Resultado:



?? Cumulatividade das operações ??

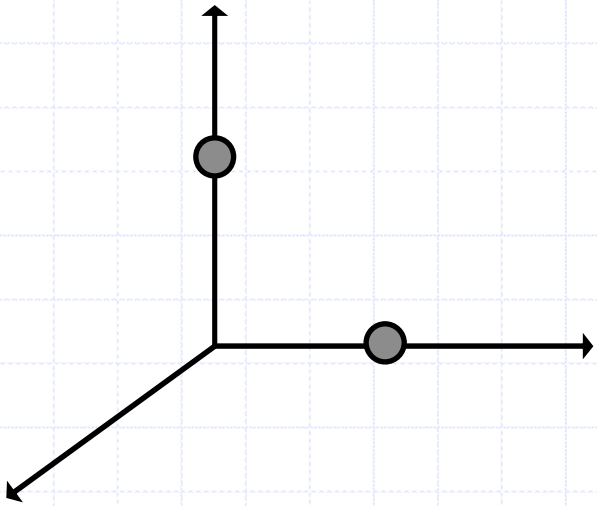
introdução a programação OpenGL
transformações

```
// 10 unidades a cima
glTranslatef(0.0f, 10.0f, 0,0f);

// Desenhando uma esfera
auxSolidSphere(1.0f);

// Reinicializa a matriz corrente
//(carrega matriz identidade)
    glLoadIdentity();

// 10 unidades a direita
glTranslatef(10.0f, 0.0f, 0.0f);

// Desenhando uma segunda esfera
auxSolidSphere(1.0f);
```

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

introdução a programação OpenGL transformações

Resultado:



introdução a programação OpenGL
remoção de superfícies não visíveis

Por default a OpenGL desenha na tela na ordem em que o código é escrito. Isto muitas vezes causa erros de visualização, pois dependendo da posição do observador alguns polígonos encobrem outros, independentemente da ordem em que estão em relação ao observador.

Para resolver esta peculiaridade, a OpenGL utiliza a técnica de "hidden surface removal" - remoção de superfícies escondidas (não visíveis) baseada na distância do pixel até o observador chamada de teste de profundidade (z-buffer ou depth-test).

introdução a programação OpenGL
remoção de superfícies não visíveis

O z-buffer é uma estrutura de dados utilizada para armazenar a coordenada Ze: distância / profundidade do pixel em relação ao observador.

Quando um novo pixel está para ser escrito na tela (vídeo buffer), sua coordenada Ze é comparada com a do pixel já armazenado (caso exista). Se a comparação indicar que está na frente deste, o novo pixel é considerado.

introdução a programação OpenGL
remoção de superfícies não visíveis

Para habilitarmos o teste de profundidade utilizamos o seguinte comando OpenGL:

```
glEnable(GL_DEPTH_TEST);
```

O depth-buffer, precisa ser limpado/atualizado a cada vez que a cena é redesenhada:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

introdução a programação OpenGL
desenhando seletivamente

O processo de esconder superfícies utilizando o depth-buffer tem obviamente vantagens visuais, mas com algum custo computacional. isto porque a cada pixel desenhado precisa ser testado, mesmo aquele que temos certeza de que não aparecerão, pois obviamente estão encobertos por outros.

Como vimos anteriormente, a ordem de construção de polígonos define sua face frontal e posterior. Isto é importante pois definem também o interior e o exterior do objeto por ele formado.

Esta informação permite-nos dizer a OpenGL para processar seletivamente cada polígono: frente, traz, ou ambos os lados.

introdução a programação OpenGL
desenhando seletivamente

Com a eliminação do lado posterior dos polígonos (interior dos objetos), pode-se reduzir drasticamente o custo computacional para gerar a imagem da cena - Esta eliminação é denominada “Back-Culling”.

Para habilitarmos o Back-Culling, utilizamos o seguinte comando:

glEnable(GL_CULL_FACE);

introdução a programação OpenGL
modo de desenho de polígonos

Polígonos não precisam ser preenchidos pela cor corrente. Por default, são desenhados sólidos; mas podemos modificar este comportamento especifificando que serão desenhadas somente as arestas ou somente os vértices.

A função: glPolygonMode();

permite estas definições, aliando a possibilidade da aplicação para cada face independentemente.

```
glPolygonMode(GL_BACK, GL_LINE);   //exemplo
```

GL_FRONT, GL_BACK, GL_FRONT_AND_BACK

GL_LINE, GL_FILL, GL_POINT

introdução a programação OpenGL
matrix stack – pilha de matrizes

Nem sempre é desejável reinicializar a matriz ModelView antes de criar cada objeto (glLoadIdentity).

As vezes é útil guardar o estado corrente, e então restaurá-lo após a criação de alguns objetos.

A OpenGL mantém estruturas para armazenar três diferentes estados - na verdade três pilhas:

ModelView;
Projection;
Texture.

introdução a programação OpenGL
matrix stack – pilha de matrizes

A matriz Projection é responsável pelas projeções 3D→2D utilizada para a conversão da cena em três dimensões para uma imagem bi-dimensional a ser desenhada na tela plana do computador.

A matriz Texture é responsável pelas transformações sobre as coordenadas dos mapas de textura aplicados sobre determinados objetos da cena.

introdução a programação OpenGL
matrix stack – pilha de matrizes

O funcionamento da pilha de matrizes é o convencional, ou seja: Push para colocar e Pop para resgatar um estado.

```
glPushMatrix( );
glPopMatrix( );
```

A matriz corrente é definida por:

```
glMatrixMode( );
```

Parâmetros:

GL_PROJECTION, GL_MODELVIEW, GL_TEXTURE

introdução a programação OpenGL
matrix stack – exemplo de código

```
void CALLBACK RenderScene(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
// Salva a matrix e faz a rotação do observador (toda a cena)
glPushMatrix();
   glLoadIdentity();
   glRotatef(45.0f+Angulo, 1.0f, 1.0f, 1.0f);   // toda a cena

   DesenhaEixos();
   DesenhaSolidos();

   glFlush();
glPopMatrix();
}
```

introdução a programação OpenGL
projeção perspectiva

Nos códigos anteriores somente a projeção ortogonal foi utilizada:

glOrtho( );

Para uma projeção perspectiva, devemos definir o volume de recorte através do comando:

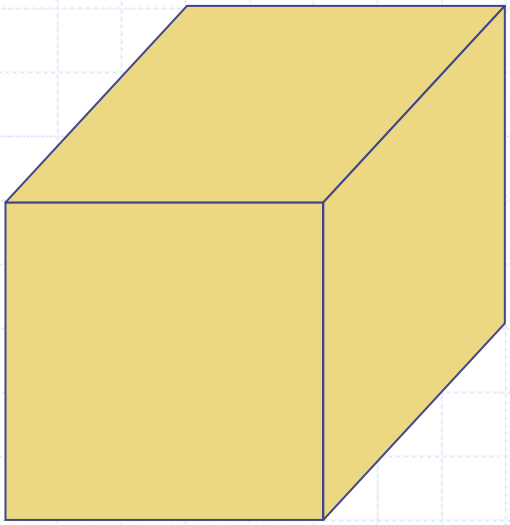
glFrustum( );

Os parâmetros são os mesmos:

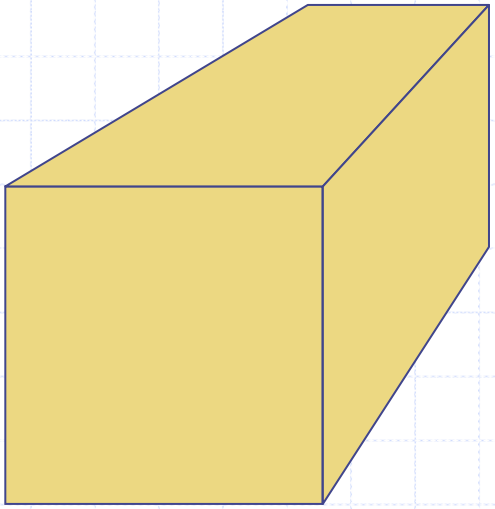
left, right, bottom, top, near, far

introdução a programação OpenGL
projeção perspectiva

Na projeção ortogonal a definição dos parâmetros é fácil, mas para a projeção perspectiva não é tão simples.



ortogonal



perspectiva

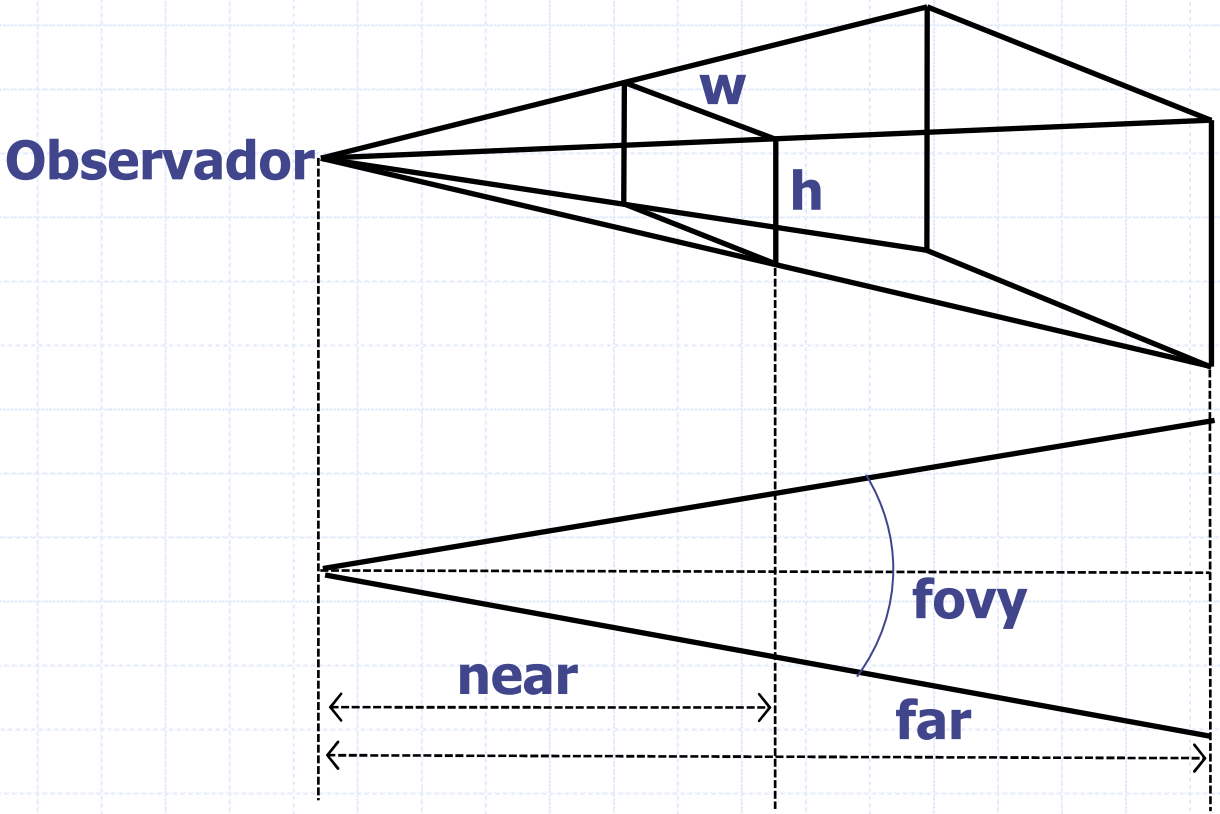
introdução a programação OpenGL
projeção perspectiva

Com a seguinte função a determnação dos parâmetros fica mais fácile e intuitiva:

```
void gluPerspective(    GLdouble fovy;
                        GLdouble aspect;
                        GLdouble zNear;
                        GLdouble zFar);
```

fovy = fiel-of-view (campo de visão em graus)
aspect = aspect-ratio (razão de aspecto) height/width
near e far = clipping planes

introdução a programação OpenGL
projeção perspectiva



introdução a programação OpenGL
projeção perspectiva - código

```
void CALLBACK ChangeSize(GLsizei w, GLsizei h)
    {
    GLfloat fAspect;
    if(h==0) h=1;                    //previne divisão por zero
    glViewport(0,0,w,h);//ajusta a viewport para as dim. da janela
    fAspect=(GLfloat)w/(GLfloat)h;
                    //define matriz corrente
    glMatrixMode(GL_PROJECTION);
                    //reseta o sistema de coordenadas
    glLoadIdentity();
                    //define a projeção perspectiva
    gluPerspective(60.0f, fAspect, 1.0, 400.0);
    glmatrixMode(GL_ModelView);
    glLoadIdentity();
    }
```

introdução a programação OpenGL
sombreamento - iluminação

Como foi visto, podemos definir cores diferentes para cada vértice de uma linha ou face.

Desta forma a cor da linha ou da face é calculada (interpolada) pela OpenGL a partir dos vértices.

Com Smooth Shading - sombreamento suavizado:

```
glShadeMode(GL_SMOOTH);
```

introdução a programação OpenGL
iluminação – tipos

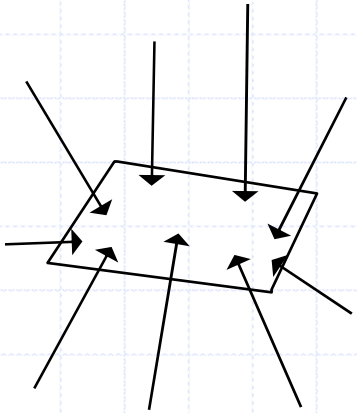
Luz Ambiente: é a luz que não possui fonte definida em termos de posicionamento. Um objeto iluminado pela luz ambiente possui todas as suas faces iluminadas.

Luz Difusa: vêm de uma posição definida, e reflete de maneira difusa a luz incidente (a reflexão não possui direção definida).

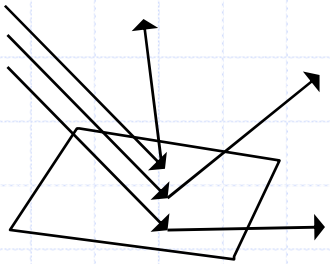
Luz Especular: como a luz difusa, possui uma fonte em posição definida, com uma direção também definida, mas a reflexão é uniforme em uma dada direção (definida em função do ângulo de incidencia).

introdução a programação OpenGL
iluminação – tipos

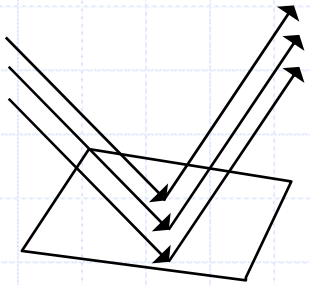
Luz Ambiente



Luz Difusa



Luz Especular



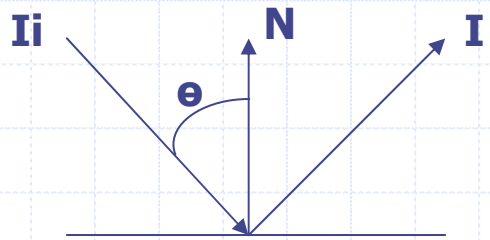
Luz Composta

Nenhuma luz real é somente ambiente, difusa ou especular. Na verdade é composta por uma mistura de diferentes intensidades diferentes de cada uma delas.

Desta forma uma fonte de luz na cena é composta pelas três componentes : Ambiente, Difusa e Especular.

introdução a programação OpenGL
iluminação – modelo

A lei de Lambert nos dá uma representação matemática de como a luz difusa se comporta quando atinge uma superfície:



$$I = Ii\ Kd \cos\theta \qquad 0 \leq \theta \leq \Pi / 2$$

Onde:

- ***I*** *- Intensidade da luz refletida*
- ***Ii*** *- Intensidade da luz incidente*
- ***Kd*** *- Coeficiente de reflexão difusa* $0 \leq Kd \leq 1$
- θ *- Ângulo entre a direção da luz e a Normal da superfície*

Kd é função do material e do comprimento de onda da luz (λ)

introdução a programação OpenGL
iluminação – modelo

Devemos também considerar a **iluminação ambiente**, caso contrário os objetos terão regiões onde não serão iluminados (dando uma impressão de descontinuidade geométrica), assim:

$$I = Ia\ Ka + Ii\ Kd \cos\theta \qquad 0 \leq \theta \leq \Pi / 2$$

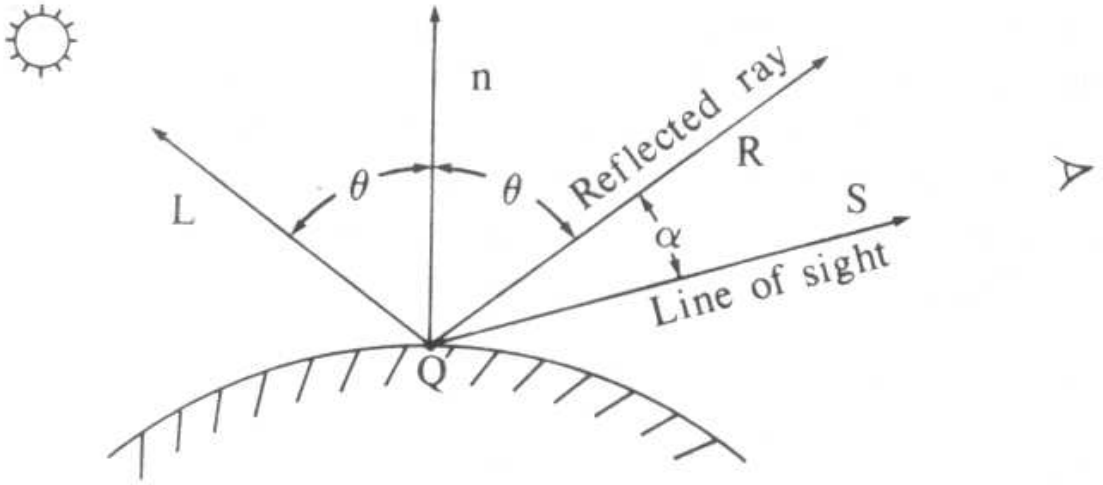
Onde:

Ia - Intensidade da luz ambiente
Ka - Coef. de reflexão difusa ambiente $0 \leq Ka \leq 1$

introdução a programação OpenGL
iluminação – modelo

A componente especular da iluminação é responsável pela criação de efeitos visuais para superfícies polidas, como plástico e metais.

Utilizamos a equação de Fresnel adaptada por Buy-Tuong Phong, que leva em consideração o ângulo de incidência da luz, o seu comprimento de onda, o coeficiente de distribuição espacial, e a posição do observador.



introdução a programação OpenGL
iluminação – modelo

Desta forma a Intensidade da luz especular refletida é dada por:

$$Is = Ii \; w(i,\lambda) \; \cos^n\alpha$$

Onde:

$w(i,\lambda)$- Curva de refletividade: função de i, ângulo de incidência e λ , comprimento de onda.

$\cos^n\alpha$ - Distribuição espacial da luz especular refletida.

introdução a programação OpenGL
iluminação – modelo

Para criarmos o efeito da **atenuação da iluminação** em função da distância do objeto em relação à luz, podemos incluir um termo de atenuação linear ou exponencial:

Exponencial: $1 / d2$
Linear: $1 / (d + K)$

Por ter uma variação muito rápida, a exponencial é inadequada para cenas em que há objetos muito próximos do observador.

O valor *K* é arbitrário e constante para toda a cena.

introdução a programação OpenGL
iluminação – modelo

Combinando todas as equações (**luz ambiente, difusa e especular**), além de considerar a atenuação pela distância, temos a seguinte equação para nosso modelo de iluminação:

$$I = Ia\,Ka + (Ii \;/\; (1 / (d + K)))\;(Kd \cos\theta + w(i,\lambda)\;\cos^{n}\alpha)$$

A função *w*, pela sua complexidade, normalmente é substituída por uma constante experimental *Ks*, desta forma temos:

$$I = Ia\,Ka + (Ii \;/\; (1 / (d + K)))\;(Kd \cos\theta + Ks\;\cos^{n}\alpha)$$

No caso de múltiplas fontes de luz, estas são somadas:

$$I = Ia\,Ka + \sum [\,(Ii \;/\; (1 / (d + K)))\;(Kd \cos\theta + Ks\;\cos^{n}\alpha)\,]$$

introdução a programação OpenGL adicionando luz à cena

Primeiro deve-se habilitar a iluminação:

```
glEnable(GL_LIGHTNING) ;
```

Esta habilitação informa à OpenGL para utilizar as propriedades dos materiais e os parâmetros de iluminação para determinar a cor para cada pixel da cena. Se não tivermos nenhum material definido, os objetos da cena ficarão negros.

A habilitação da iluminação, e todas as inicializações necessárias são efetuadas na função :

```
SetupRC();
```

introdução a programação OpenGL adicionando luz à cena

Após habilitar a iluminação, a primeira coisa que deve ser feita é a definição do modelo de iluminação. Os três parâmetros que afetam o modelo são definidos com a função:

glLightModel();

A seguinte porção de código especifica uma luz branca intensa:

```
// luz branca brilhante
GLfloat ambienteLight[]={1.0f,1.0f, 1.0f,1.0f};
// habilita a iluminação
glEnable(GL_LIGHTING);
// define o modelo de iluminação, utilizando a luz
// ambientLight[] como luz ambiente
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,
                    ambientLight);
```

introdução a programação OpenGL
iluminação – materiais

Até agora apenas definimos as cores dos objetos na cena, a partir das componentes R,G,B.

Com a definição de luz, temos agora a possibilidade de definir materiais.

O Material é então a definição de sua cor (R,G,B) e de seu comportamento ao ser exposto à iluminação - na verdade sua reflexividade para cada componente da luz (ambiente, difusa e especular).

introdução a programação OpenGL
materiais – sintaxe

```
.
.
GLfloat gray[]={0.75f, 0.75f, 0.75f, 1.0f};
.
.
.
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, gray);
.
.
glBegin(GL_TRIANGLES);
    glvertex3f(-15.0f, 0.0f, 30.0f);
    glvertex3f(0.0f, 15.0f, 30.0f);
    glvertex3f(0.0f, 0.0f, -56.0f);
glEnd();
```

introdução a programação OpenGL
a função SetupRC( )

```
void SetupRC( );
    {
    GLfloat ambientLight[] = {1.0f, 1.0f, 1.0f, 1.0f};

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glEnable(GL_LIGHTING);

    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientLight);

    glClearColor(0.0f, 0.0f, 0.5f, 1.0f);
    }
```

introdução a programação OpenGL
cálculo da normal

```
void calcNormal(float v[3][3], float out[3])
    {
    float v1[3], v2[3];
    static const int x = 0;  static const int y = 1;       static const int z = 2;

    v1[x] = v[0][x] - v[1][x];
    v1[y] = v[0][y] - v[1][y];
    v1[z] = v[0][z] - v[1][z];

    v2[x] = v[1][x] - v[2][x];
    v2[y] = v[1][y] - v[2][y];
    v2[z] = v[1][z] - v[2][z];

    out[x] = v1[y]*v2[z] - v1[z]*v2[y];
    out[y] = v1[z]*v2[x] - v1[x]*v2[z];
    out[z] = v1[x]*v2[y] - v1[y]*v2[x];

    ReduceToUnit(out); // vetor unitário
    }
```