Computação Gráfica
Módulo IV – OpenGL

UniverCidade - Prof. Ismael H F Santos

Considerações Gerais

- **Objetivo:** *Discutir os principais conceitos eos princípios básicos dos Sistemas Gráficos e a Programação em OpenGL.*
- **A quem se destina :** *Alunos e Profissionais que desejem aprofundar seus conhecimentos sobre Computação Grafica e suas aplicações.*

Bibliografia

- *Computação Gráfica Volume 1. Jonas Gomes e Luiz Velho. Instituto de Matemática Pura e Aplicada – IMPA.*
- *Introdução a Computação Gráfica - Paulo Roma*
  - http://www.lcg.ufrj.br/compgraf1/downloads/apostila.pdf
  - http://www.lcg.ufrj.br/compgraf1/downloads/apostila.ps.gz
- *Notas do Curso ministrado na Universidade de Maryland pelo Prof. David Mount*
  - ftp://ftp.cs.umd.edu/pub/faculty/mount/427/427lects.ps.gz
  - http://www.lcg.ufrj.br/~esperanc/CG/427lects.ps.gz
- *Apostila Fundamentos da Imagem Digital – Antonio Scuri*
- *Computer Graphics: Principles and Practice, Second Edition. James Foley, Andries van Dam, Steven Feiner, John Hughes. Addison-Wesley.*
- *OpenGL Programming Guide, 2nd Edition. Mason Woo, Jackie Neider, Tom Davis. Addison Wesley.*

Bibliografia OpenGL

- OpenGL® Programming Guide, 2nd Edition. Mason Woo, Jackie Neider, Tom Davis. Addison Wesley.
  - http://www.lcg.ufrj.br/redbook
- Manual de referência online
  - http://www.lcg.ufrj.br/opengl
- Sítio oficial do OpenGL
  - www.opengl.org

Ementa

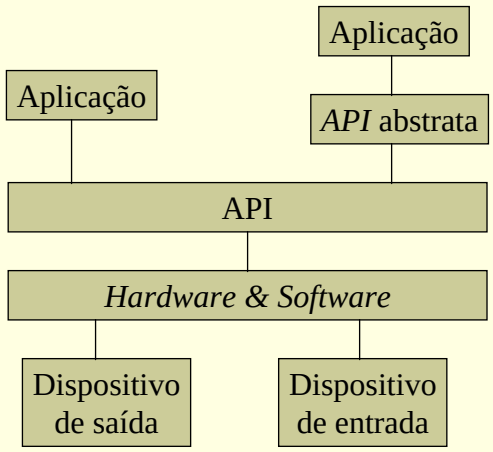
- **Introdução ao OpenGL**
  - *Geometria*
    - Exemplos de Transformações 2D
    - Fórmulas e cálculos das transformações 2D
    - Usando matriz de transformação (por que?)
    - Coordenadas Homogêneas
    - Concatenação de transformações
    - Transformações 3D
  - *Projeções*
  - *Histórico*

CG – CO023

Transformações Geométricas

OpenGL: o que é?

- API
  - Interface para programador de aplicação



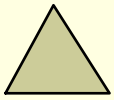
Por que OpenGL?

- primitivas geométricas e imagens
- arquitetura bem definida
- relativamente simples
- boa performance (sw & hw)
- bem documentado
- independente de sistemas de janelas
- padrão
  - disponível em diversas plataformas

Primitivas geométricas básicas

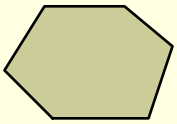
Ponto

Linha



Triângulo

Quadrado



Polígono (convexo)

Objetos 3D

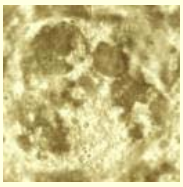
Polyhedra

Sphere

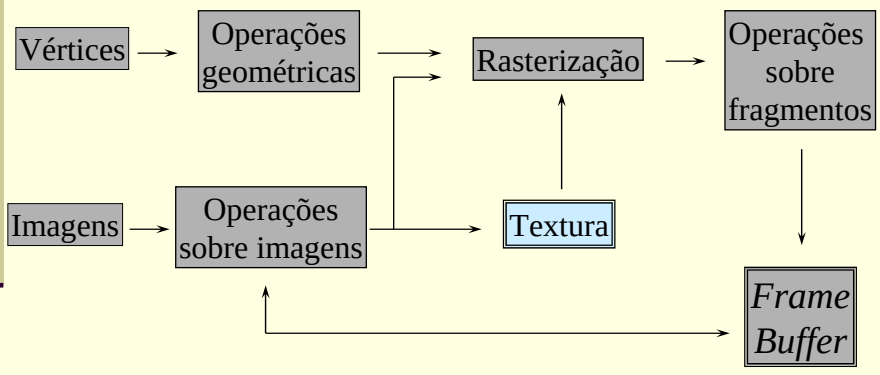
Bezier Surfaces

Quadric

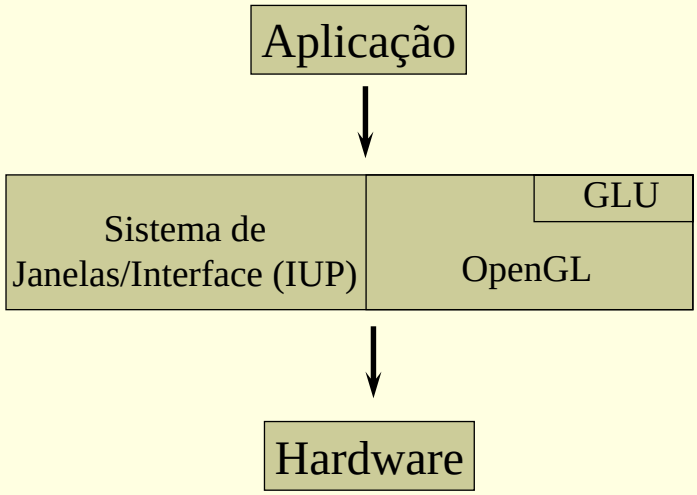
Imagem e Textura



OpenGL rendering pipeline



Aplicação típica

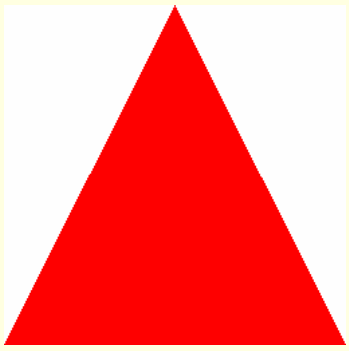


Programa simples (usando GLUT)

```
#ifdef _WIN32
  #include <windows.h>
#endif
#include "GL/gl.h"
#include "GL/glu.h"
#include "GL/glut.h"
int main (int argc, char* argv[]) {
 glutInit(&argc, argv); /* openg GLUT */
 glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

 glutInitWindowSize (250, 250);
 glutCreateWindow ("simple"); /* create window */
 glutDisplayFunc(display);
 glutMainLoop(); /* interact ... */
 return 0;
}
```

Programa simples (usando GLUT) - cont.

```
void display (void) {
 /* clear window */
 glClearColor(1,1,1,1);
 glClear(GL_COLOR_BUFFER_BIT);

 /* draw red triangle */
 glColor3d(1,0,0);
 glBegin(GL_TRIANGLES);
  glVertex2d(-1,-1);
  glVertex2d(1,-1);
  glVertex2d(0,1);
 glEnd();

 /* update screen */
 glFlush();
}
```



OpenGL: máquina de estado

- Trabalha com o conceito de valor corrente
  - Iluminação
  - Shading
  - Textura
  - etc.

glEnable / glDisable

OpenGL: inicializações

Inicialização da área de desenho

```
glClearColor(red,green,blue,alpha);
glClear(GL_COLOR_BUFFER_BIT);
```

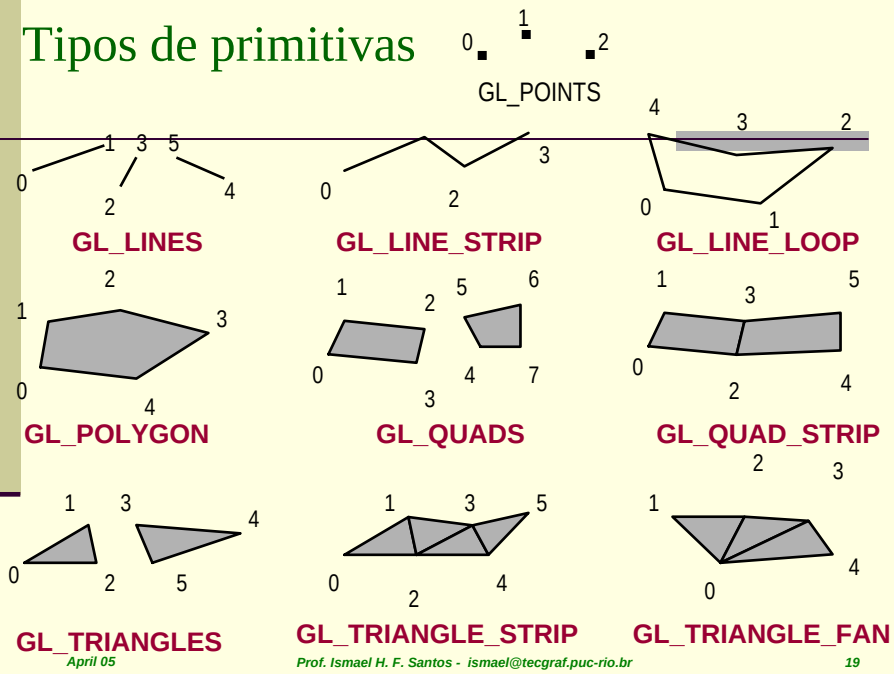
Atualização da área de desenho

```
glFlush( );
glFinish( );     // modal
```


Primitivas geométricas

```
glBegin(tipo_de_prim

  …define attributo de vértice
  …define vértice

glEnd
```

Tipos de primitivas



Especificação de vértice

glVertex{tam}{tipo}{vetor} (…);

exemplo:

```
GLdouble pos[ ] = {0.4,9.0,2.0};
glVertex3dv(pos);
```

ou

```
glVertex3d(0.4,9.0,2.0);
```

- *OpenGL trabalha com coordenadas homegêneas*

Especificação de atributos: Cor

- Modelo de cor
  - RGB

    glColor3d(red,green,blue);

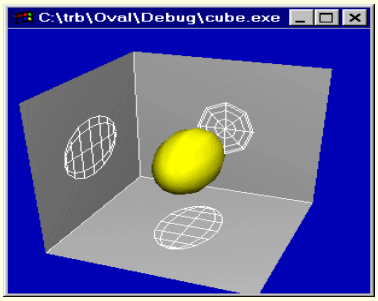
  - *Color index*
    - Paleta previamente definida

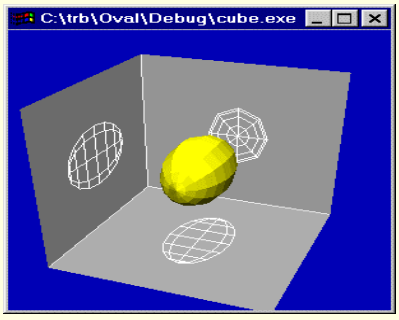
      ...

      glIndexi(index);

Interpolação de cores

```
void glShadeModel (GL_SMOOTH);
```



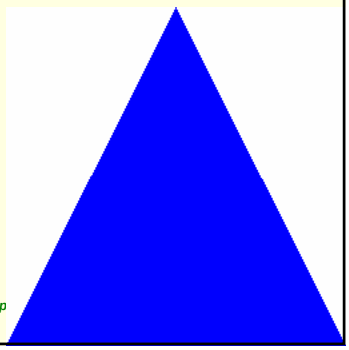



```
void glShadeModel (GL_FLAT);
```

Modelo de Shading

- *Flat*

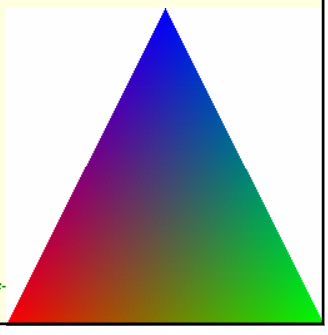
```
glShadeModel(GL_FLAT);
glBegin(GL_TRIANGLES);
  glColor3f(1.0,0.0,0.0);  // red
  glVertex2f(-1.0,-1.0);
  glColor3f(0.0,1.0,0.0); // green
  glVertex2f(1.0,-1.0);
  glColor3f(0.0,0.0,1.0); // blue
  glVertex2f(0.0,1.0);
glEnd( );
```



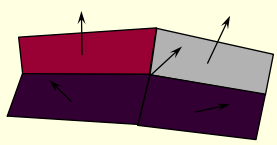
Modelo de Shading

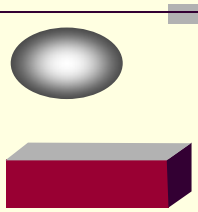
- *Smooth (Gouraud)*

```
glShadeModel(GL_SMOOTH);   // default
glBegin(GL_TRIANGLES);
  glColor3f(1.0,0.0,0.0);  // red
  glVertex2f(-1.0,-1.0);
  glColor3f(0.0,1.0,0.0); // green
  glVertex2f(1.0,-1.0);
  glColor3f(0.0,0.0,1.0); // blue
  glVertex2f(0.0,1.0);
glEnd( );
```

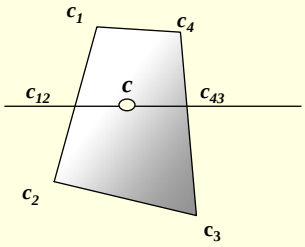


Suavização da tonalização

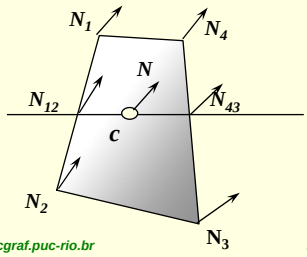




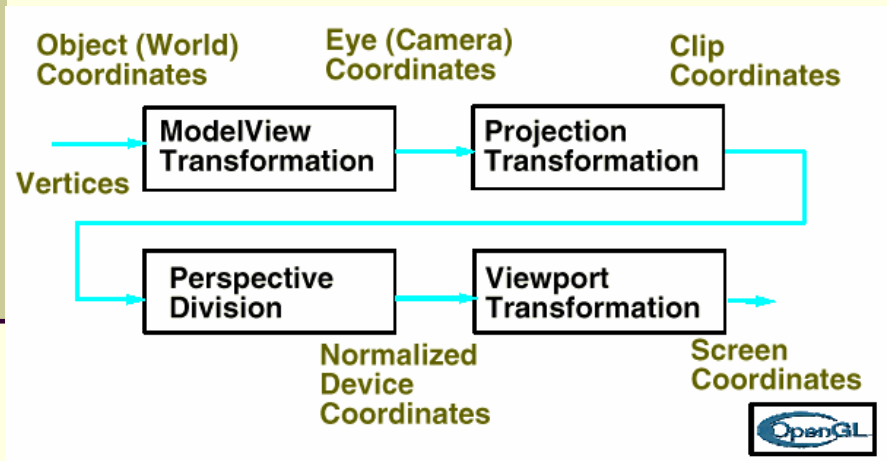
Gouraud



Phong

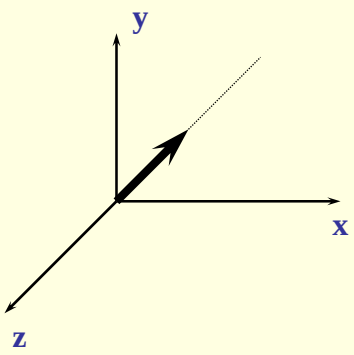


Transformações 3D e Sistemas de Coordenadas



Visualização 3D

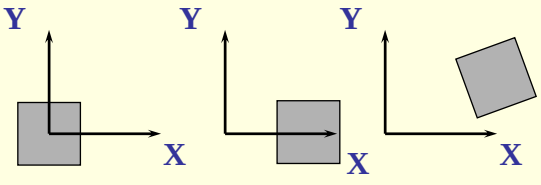
- Camera
  - Posição fixa: (0.0,0.0,0.0)
  - Direção: -z
- Composição da cena
  - move ~~câmera~~ *ou*
  - move objetos



Transformação de Modelagem X Visualização

- Transformação de modelagem - MODELVIEW
  - Sistema global fixo
  - Ordem inversa para especificação

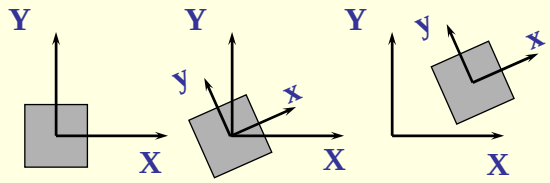
```
...
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(30,0,0,1);
glTranslatef(10,0,0);
...
```



Transformação de Modelagem X Visualização (cont.)

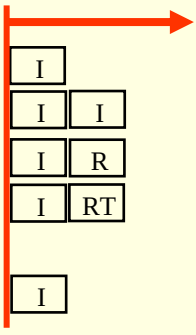
- Transformação de visualização
  - Sistema local móvel
  - Ordem natural para especificação

```
...
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(30,0,0,1);
glTranslatef(10,0,0);
...
```



Manipulação da pilha de matrizes

```
...
glMatrixMode(GL_MODELVIEW);
glLoadIdentity( );
glPushMatrix( );
glRotate(30,0,0,1);
glTranslate(10,0,0);
draw_object_1( );
glPopMatrix( );
...
```



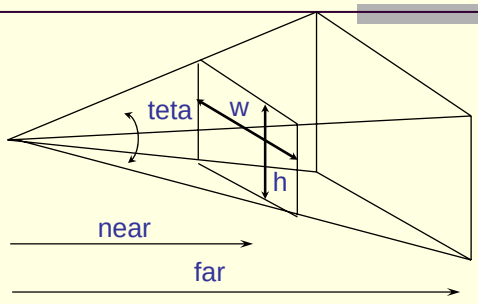
Posicionamento do mundo em relação à camera

- Função auxiliar

```
...
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

gluLookAt(eye_x, eye_y, eye_z,
          center_x, center_y, center_z,
          up_x, up_y, up_z
         );
```


Projeção: tipo de camera

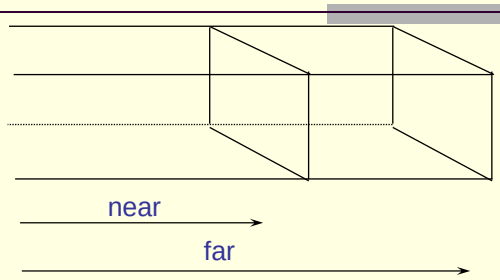
- Perspectiva




```
...
glMatrixMode(GL_PROJECTION);
glLoadIdentity( );
gluPerspective (teta_y,aspect,znear,zfar);
...
```

Projeção: tipo de camera (cont.)

- Ortográfica




```
…
glMatrixMode(GL_PROJECTION);
glLoadIdentity( );
glOrtho (xleft,xright,ybottom,ytop,znear,zfar);
...

2D:
gluOrtho2D (xleft,xright,ybottom,ytop);
```


Transformação de *viewport*

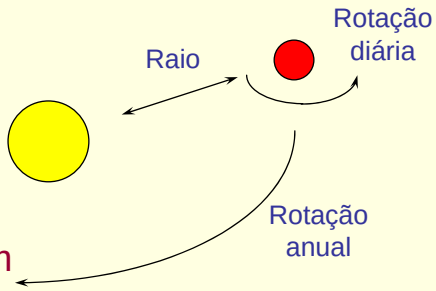
```
…
glViewport (x, y, width, height);
...
```

GLUT:

• A função default de "resize" define a *viewport* como sendo a área total do canvas.

Exemplo: sistema solar

- Sol e um planeta



Sol: desenhado na origem
Planeta:
Pensando em sistema local
- Rotação anual
- Translação em x
- Rotação diária

Remoção de superfícies ocultas

- Z-BUFFER
  - Inicializa window (default)
  - Habilita teste em Z

    glEnable (GL_DEPTH_TEST);
  - Define teste

    glDepthFunc (GL_LESS);
  - Limpa buffer

    glClear (GL_DEPTH_BUFFER_BIT);

Animação

- *Double color buffer: BACK & FRONT*
  - Inicialização - GLUT

    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

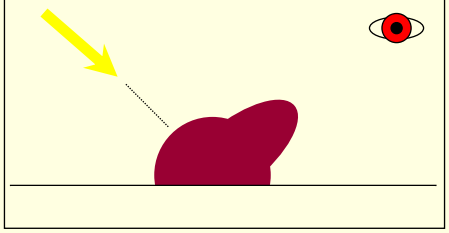
  - Atualização da tela - GLUT

    glutSwapBuffers();

Rendering

- Cor do objeto depende de:
  - fonte de luz
  - orientação da superfície
  - posição do observador
  - reflexividade do material
    - ambiente
    - difusa
    - especular

Modelo de iluminação: Phong



Especificação da orientação

- Vetor normal em cada vértice

  glNormal3d(nx,ny,nz);

- Se não for normalizado

  glEnable(GL_RESCALE_NORMAL);

  ou

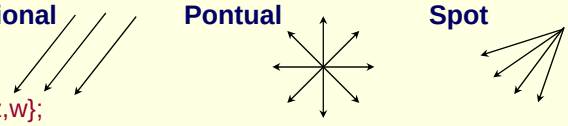
  glEnable (GL_NORMALIZE);

Obs: cálculo de normal é caro!

Fontes de luz

- Tipos

Direcional **Pontual** **Spot**



```
Glfloat pos[ ] = {x,y,z,w};
glLightf (GL_LIGHT0, GL_POSITION, pos);
```

- Cor e intensidade: ambiente, difusa, especular

```
Glfloat dif[ ] = {red,green,blue,alpha};
glLightf (GL_LIGHT0, GL_DIFFUSE, dif);
```

- Habilitação

```
glEnable (GL_LIGHTING);
glEnable (GL_LIGHT0);
```

Parâmetros adicionais de iluminação

- Luz ambiente global

```
GLfloat amb[ ] = {0.2,0.2,0.2,1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, amb);
```

- Posição do observador: local ou infinito

```
glLightModeli (GL_LIGHT_MODEL_VIEWER, GL_TRUE);
```

- Iluminação de faces: *back e front*

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

- Iluminação especular em separado (p/ texturas)

```
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,
              GL_SEPARATE_SPECULAR_COLOR);
```


Culling de faces

- Define orientação das faces

```
glFrontFace (GL_CCW);
```

- Descarta faces (*culling*)

```
glCullFace (GL_BACK);
glEnable(GL_CULL_FACE);
```

Material

- Cor (reflexividade)
  - Ambiente
    - não depende de orientação
  - Difusa
    - depende da orientação da superfície e da posição da fonte de luz
  - Especular
    - depende da orientação da superfície, da posição da fonte de luz e posição do observador
  - Brilho (*shininess*)
    - fator de brilho da reflexão especular
  - Emissão
    - para representação de fontes de luz na cena

```
Glfloat color [ ] = { red, green, blue, alpha };
glMaterialf (GL_BACK_AND_FRONT,
             GL_AMBIENT_AND_DIFFUSE, color);
```


Cor como material

- Usando cor para definição de material

```
glColorMaterial (GL_BACK_AND_FRONT,
                 GL_AMBIENT_AND_DIFFUSE);

glEnable (GL_COLOR_MATERIAL);

…
glColor3f (red, green, blue);
...
```

Propriedades dos vértices em arrays

- Array de vértices
  - glEnableClientState (GL_VERTEX_ARRAY);
  - glVertexPointer (*size, type, stride, pointer*);
    - size: 2, 3 ou 4 (coordenadas)
    - type: GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
    - stride: *byte offset* entre vértices consecutivos
    - *pointer*: ponteiro para área de memória
- Arrays de normais e cores tem API análoga

Acessando arrays de propriedades

- Acessando um elemento
  - glBegin (GL_TRIANGLES);
  - ...
  - glArrayElement (i);
  - ...
  - glEnd ( );
- Acessando um conjunto elemento
  - glDrawElements (*mode, count, type, indices*);
    - *mode*: GL_LINES, GL_TRIANGLES, etc.
    - *count*: número de elementos a ser desenhados
    - *type*: tipo do vetor de índices: GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, GL_UNSIGNED_INT
    - *indices*: vetor de índices
  - glDrawRangeElements (*mode, start, end, count, type, indices*);
    - *start, end*: delimitam valores dos índices para permitir pre-processamento

Pilha de grupos de atributos

- Permite restaurar atributos eficientemente
- Pilha de atributos (do servidor)

```
glPushAttrib ( GL_FOG_BIT | GL_LIGHTING_BIT | etc );
glPopAttrib (   );
```

- Pilha de atributos do cliente

```
glPushClientAttrib ( GL_CLIENT_PIXEL_STORE |
                     GL_CLIENTE_VERTEX_ARRAY_BIT );
glPopClientAttrib (   );
```


Sistema solar: implementação

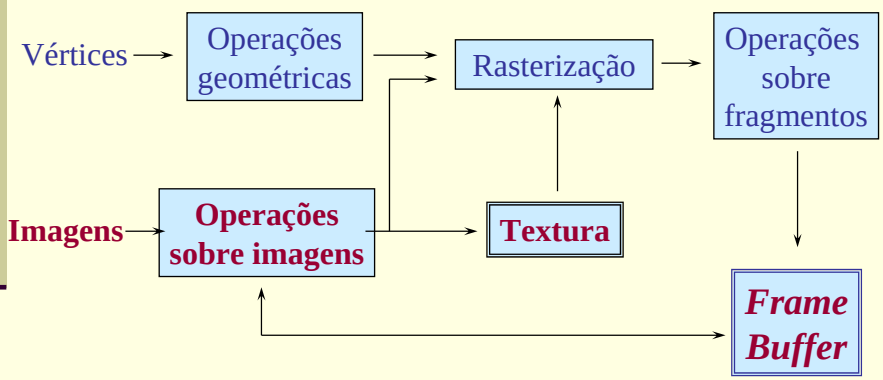
```
int main (int argc, char **argv) {
  /* GLUT - Initialization */
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowSize(500, 500);
  glutCreateWindow("CG2001-T1");

 /* Registrando callbacks */
  glutDisplayFunc(display);
  glutReshapeFunc(redraw);
 //  glutMouseFunc(mouseCall);
 //  glutMotionFunc(motionCall);
 //  glutKeyboardFunc(keyboardCall);
  glutIdleFunc(display);

  /* GLUT main loop */
  glutMainLoop();
  return 0;
}
```

Programa: Lua_scene.cpp

Blending

- Combinação da cor do fragmento sendo processado com a cor do pixel
  - depois da rasterização e antes do fragmento ser desenhado no *framebuffer*.
- Aplicações
  - transparência
  - composição digital
  - pintura

OpenGL rendering pipeline



Fatores de combinação

- Fonte (*source*)
  - representa fragmento
  - S_R, S_G, S_B, S_A
- Destino (*destination*)
  - representa pixel
  - D_R, D_G, D_B, D_A
- Fatores
  - R_S, G_S, B_S, A_S
  - R_D, G_D, B_D, A_D
- Resultado
  - $R_S S_R + R_D D_R$, $G_S S_G + G_D D_G$, $B_S S_B + B_D D_B$

Especificação dos fatores

glBlendFunc (fator_frag, fator_pixel);

GL_ZERO	src *or* dst	(0,0,0,0)
GL_ONE	src *or* dst	(1,1,1,1)
GL_DST_COLOR	src	(R_d,G_d,B_d,A_d)
GL_SRC_COLOR	dst	(R_s,G_s,B_s,A_s)
GL_ONE_MINUS_DST_COLOR	src	(1,1,1,1) - (R_d,G_d,B_d,A_d)
GL_ONE_MINUS_SRC_COLOR	dst	(1,1,1,1) - (R_s,G_s,B_s,A_s)
GL_SRC_ALPHA	src *or* dst	(A_s,A_s,A_s,A_s)
GL_ONE_MINUS_SRC_ALPHA	src *or* dst	(1,1,1,1) - (A_s,A_s,A_s,A_s)
GL_DST_ALPHA	src *or* dst	(A_d,A_d,A_d,A_d)
GL_ONE_MINUS_DST_ALPHA	src *or* dst	(1,1,1,1) - (A_d,A_d,A_d,A_d)
GL_SRC_ALPHA_SATURATE	src	(f ,f ,f ,1); f = $\min(A_s,1-A_d)$

Blending: exemplos de uso

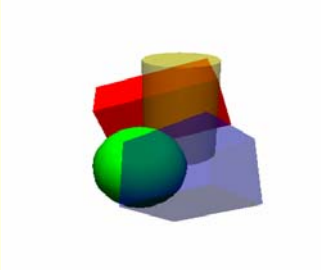
- Desenho temporário sobre imagem

```
glEnable (GL_BLEND);
glBlendFunc (GL_ONE_MINUS_DST_COLOR,
             GL_ZERO);
glColor3d (1.0, 1.0, 1.0);
```

- Objetos transparentes
  - Cor dada por: (*red, green, blue, opacity*)

```
glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA,
  GL_ONE_MINUS_SRC_APHA);
```


Transparência em cena 3D



- habilita-se z-buffer
- desenha-se objetos opacos
- define-se z-buffer como *read-only*

```
glDepthMask (GL_FALSE);
```

- desenha-se objetos com transparência em ordem

Buffers

- Color
  - onde desenha-se
- Depth
  - z-buffer
- Stencil
  - usado para restringir desenho a uma área do color buffer.
- Accumulation
  - usado para composição de imagem.
  - também é um RGBA buffer

Antialiasing

- Cobertura dos pixels é multiplicada na componente alpha da cor

Controle da qualidade:
glHint (GL_*primitive*_SMOOTH_HINT, GL_NICEST);

Antialiasing

- Pontos e linhas

```
glEnable (GL_POINT_SMOOTH);
glEnable (GL_LINE_SMOOTH);

glEnable (GL_BLEND);
glBlendFunc
   (GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
```


Antialiasing

- Polígonos

```
glEnable (GL_POLYGON_SMOOTH);

glEnable (GL_BLEND);
glBlendFunc
   (GL_SRC_ALPHA_SATURATE, GL_ONE);
```

  - Desenha polígonos em ordem (*front to back*)!

Fog

- Decaimento

```
glFogi (GL_FOG_MODE, GL_LINEAR);
glFogf (GL_FOG_START, zstart);
glFogf (GL_FOG_END, zend);
```

- Cor

```
glFogfv (GL_FOG_COLOR, color);
```

- Qualidade

```
glHint (GL_FOG_HINT, GL_NICEST);  // per pixel
```


Polygon Offset

glPolygonOffset (*factor*, *units*)

o = r . units + m . Factor

r = menor valor que garante diferença em z_w

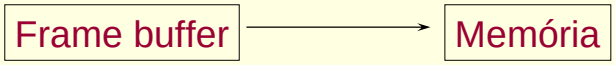
m = declividade na profundidade

Imagens

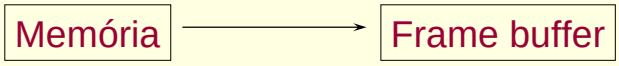
- Representa uma área retangular de valores associados aos pixels
- Fatores complicantes
  - existem diferentes dados associados aos pixels
  - existem diferentes formas de armazenar uma imagem
  - existem diferentes conversões de dados quando operamos sobre pixels

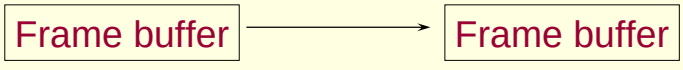
Operações sobre imagens

- Read



- Draw
- Copy





Formato de cada pixel

- De 1 a 4 elementos representam um pixel

GL_RGB
GL_RGBA
GL_RED
GL_GREEN
GL_BLUE
GL_ALPHA
GL_LUMINANCE
GL_LUMINANCE_ALPHA
GL_DEPTH_COMPONENT
GL_STENCIL_INDEX
GL_COLOR_INDEX

Tipo de cada elemento

UNSIGNED_BYTE ..... 8 bits
BYTE ..... 7 bits
UNSIGNED_SHORT ..... 16 bits
SHORT ..... 15 bits
UNSIGNED_INT ..... 32 bits
INT ..... 31 bits
FLOAT ..... [0.0,1.0]

BITMAP..... 1 bit

Desenhando imagens

- Posição da imagem
  - glRasterPos* (x, y, z, w);
- Especificação da imagem
  - glDrawPixels (width, height, format, type, pixels)
- Especificação de bitmap
  - Projetado para suportar desenho de fontes raster

  glBitmap (width, height, x_0, y_0, x_{inc}, y_{inc}, bitmap);

Zoom de imagem

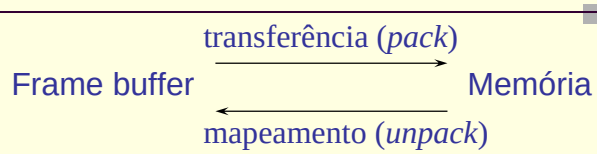
Image pixel → Screen pixel

glPixelZoom ($zoom_x$, $zoom_y$);

↳ podem ser fracionários

Pixel —*rasterização*→ Quadrilateral

Modos de armazenamento de pixel



- Exemplos

```
glPixelStorei (GL_UNPACK_SWAP_BYTES, GL_TRUE);

glPixelStorei (GL_PACK_ALIGNMENT, 4);
```

- Também permite especificação de subáreas da imagem

Textura

- Mapeamento de imagens sobre primitivas
- Imagem composta por *texels*
- Largura e altura: 2^n

```
gluScaleImage (format,
               width_in, height_in, type_in, data_in,
               width_out, height_out, type_out, data_out);
```

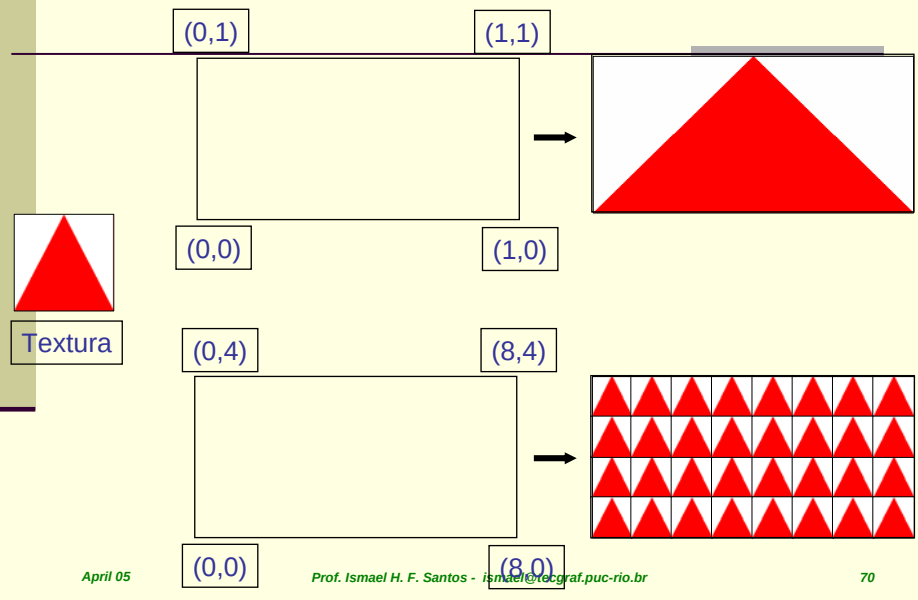
- Especificação: 1D e 2D

```
glEnable (GL_TEXTURE_2D or GL_TEXTURE_1D);
glTexImage2D (GL_TEXTURE_2D, level, components,
               width, height, border, format, type, pixels);
glTexImage1D (GL_TEXTURE_1D, level, components,
               width, border, format, type, pixels);
```

Coordenada de textura

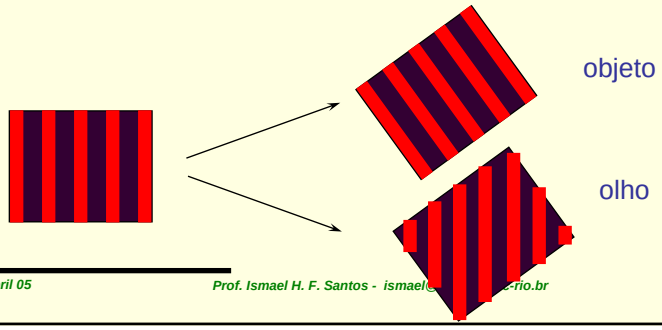
- Para cada vértice
  - coordenada: identifica qual o pixel no buffer
  - coordenada de textura: identifica qual o texel
- Coordenadas de textura: s, t, r, q
- Coordenadas de textura são linearmente interpoladas entre vértices

Mapeamento de textura



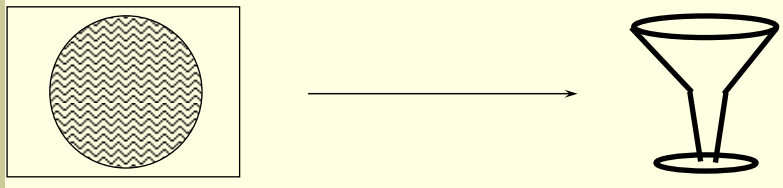
Geração automática de coordenadas de textura

- Coordenadas definidas pela distância dos vértices a um plano
  - Em relação às coordenadas do objeto
  - Em relação às coordenadas do olho



Geração automática de coordenadas de textura (cont.)

- Mapeamento de ambientes
  - representação de objetos reflexivos



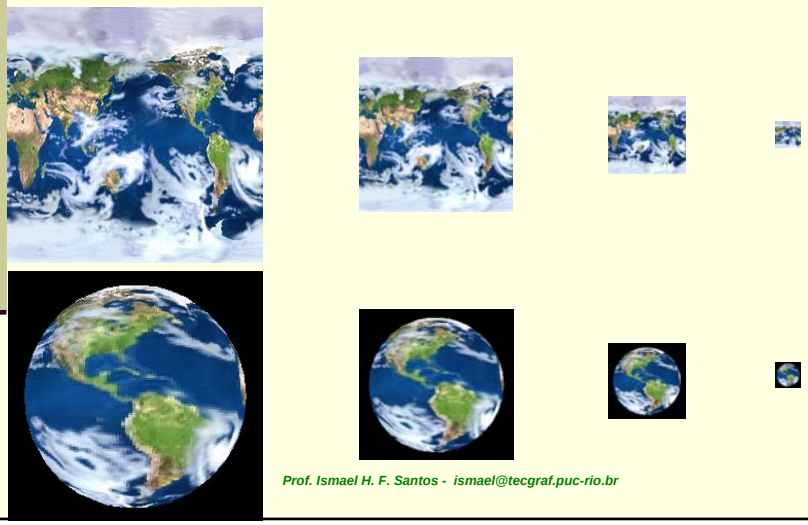
- Exemplo de especificação

```
glEnable (GL_TEXTURE_GEN_S);
glTexGeni (GL_S, GL_TEXTURE_GEN_MODE,
           GL_OBJECT_PLANE);
glTexGenfv (GL_S, GL_OBJECT_PLANE, plane);
```

LOD: multiresolução

mipmaps

mip := latim: “multim im parvo”; muitas coisas num mesmo lugar



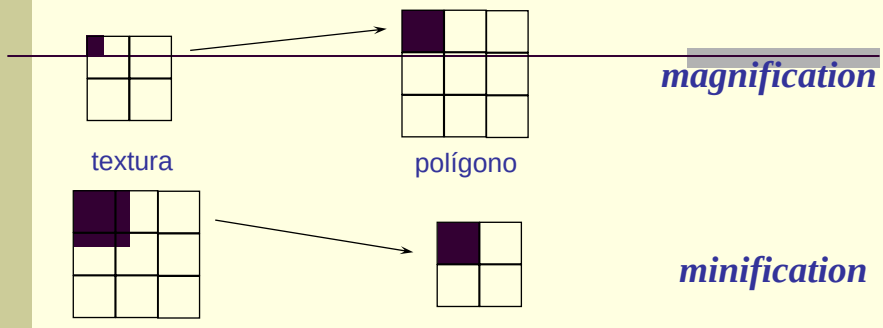
Construção de *mipmaps*

- Dado a imagem de maior resolução, pode-se construir e definir a pirâmide de mipmaps

```
gluBuild2DMipmaps (GL_TEXTURE_2D, components,
                   width, height, format, type, data);
```

level 0 ⟶ 64 x 16
32 x 8
16 x 4
8 x 2
4 x 1
2 x 1
1 x 1

Filtragem

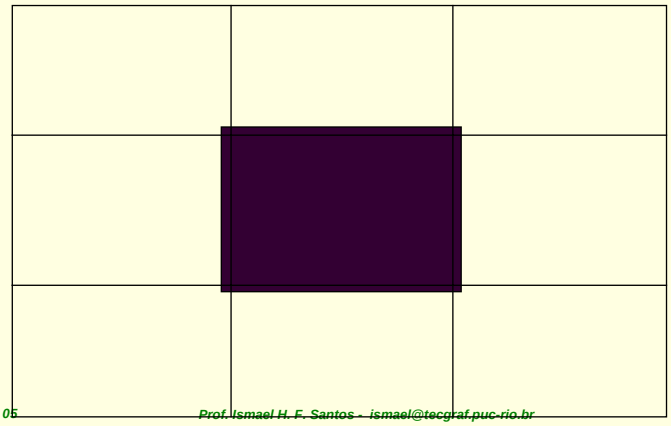


- Filtragem: GL_NEAREST, GL_LINEAR, etc
- Especificação

```
glTexParameteri (GL_TEXTURE_2D,
                 GL_TEXTURE_MAG_FILTER,
                 GL_LINEAR);
```


Bordas

- Garantir repetição de padrões
- Definir como truncar mapeamento da textura



Combinação de pixel com texel

- *Decal*
  - Cor definida pelo texel
- *Modulate*
  - Cor do pixel é modulada pela cor do texel
- *Blend*
  - Cor combinada com uma cor adicional de ambiente
- Exemplo
  - Modular com a cor branca para dar iluminação

Exemplo de modulação

DECAL MODULATE

Referências

- "The Red Book"
  - *OpenGL: Programming Guide*
    - Release 1.1
    - M. Woo, J. Neider, T. Davis
- Web sites
  - *The official OpenGL web page*
    - http://www.opengl.org
  - *SGI's OpenGL WWW Center*
    - http://www.sgi.com/Technology/openGL
  - *Gateway to OpenGL*
    - http://reality.sgi.com/mjk_asd/opengl-links.html