



# IA725 – Computação Gráfica I

## Introdução ao OpenGL

Redbook, capítulos 1 e 2



## Visão Geral

- Histórico do OpenGL
  - Evolução e bibliotecas auxiliares
- Arquitetura
  - Fluxo de processamento
  - OpenGL como uma máquina de estados
- Comandos
  - Definição de geometria
  - Matrizes de transformação
  - Cor e iluminação
- Exemplos em C usando a biblioteca GLUT



## Histórico do OpenGL



- Antigas tentativas de desenvolver APIs padrão de indústria (1973-92)
  - GKS (2D), PHIGS (2D e 3D)
- Em 1982, a Silicon Graphics Inc. (SGI) implementa a API IrisGL (GL=*Graphics Library*) em hardware
- SGI cria o OpenGL em 1992 a partir do IrisGL
  - Independente de plataforma, linguagem e sistema de janelas
  - Abstração do *hardware* no nível mais baixo possível
  - Padrão de indústria



## Histórico do OpenGL



- Órgão mantenedor definido pela SGI
  - 1992-2006: *Architecture Review Board* (ARB)
  - 2006-hoje: *Khronos Group*
- Versões e implementações do OpenGL
  - v1.0 (1992) até 1.5 (2004)
  - v2.0 (2004): suporte a placas programáveis
  - v3.0 prevista para 2008
  - Centenas de extensões para hw proprietário
  - Mesa 3D (*free software*, licença MIT)



## Bibliotecas OpenGL e auxiliares



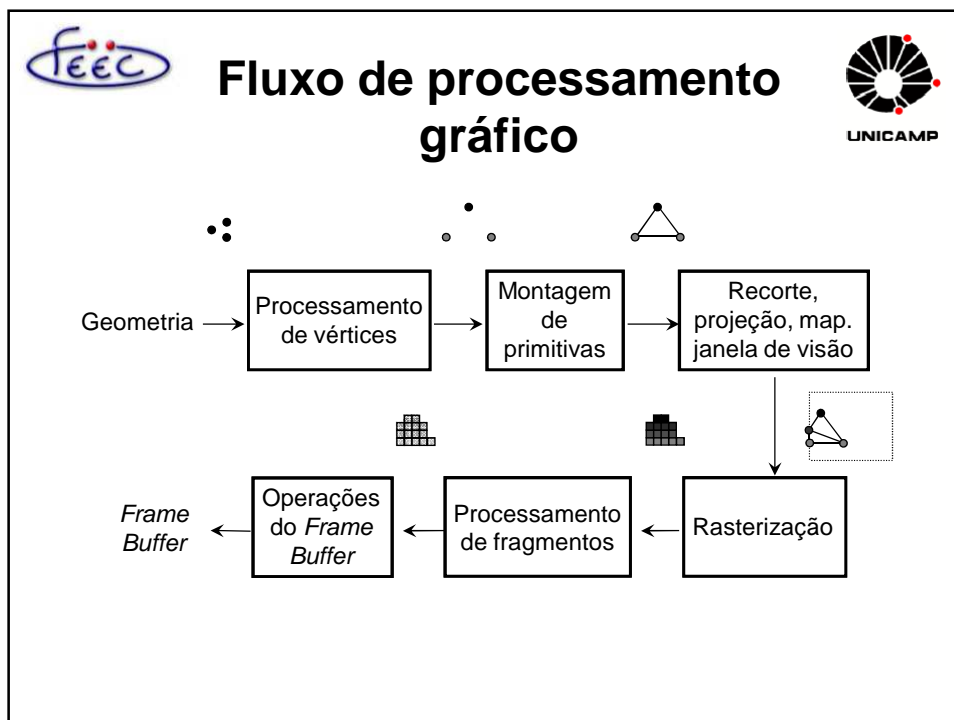
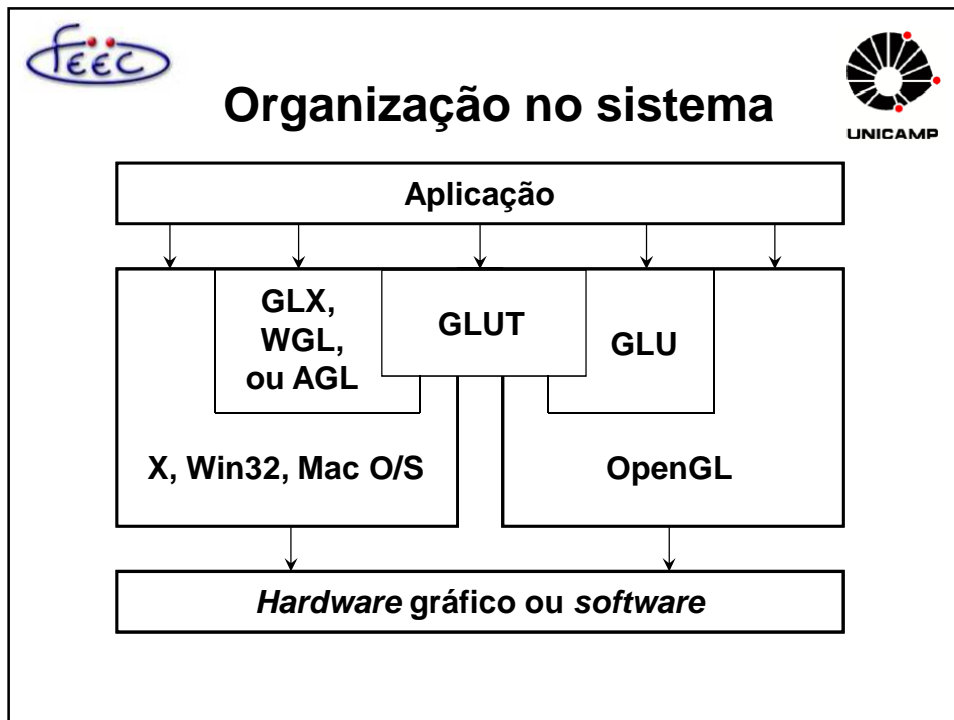
- Biblioteca OpenGL
  - Win32: `opengl32.lib`
  - Unix/Linux: GL (`libGL.a`) –IGL
  - Comandos começam com o prefixo `gl`
- *OpenGL Utility Library (GLU)*
  - Comandos para criar modelos 3D mais complexos
  - Win32: `glu32.lib`
  - Unix/Linux: GLU (`libGLU.a`) –IGLU
  - Comandos começam com o prefixo `glu`
- Bibliotecas de ligação com o sistema de janelas
  - GLX no *X Window System*, WGL no *Windows*, AGL no *Macintosh*



## OpenGL Utility Toolkit (GLUT)



- Biblioteca para usar OpenGL em diferentes sistemas de janelas de forma portátil.
  - Win32: `glut32.lib`
  - Unix/Linux: GLUT (`libglut.a`) –lglut
  - Comandos começam com o prefixo `glut`
- Funcionalidades:
  - Gerenciamento de janelas
  - Produção de objetos gráficos 3D
  - Controle de eventos
- Funcionalidades limitadas em relação a *toolkits* como GTK e Qt





## OpenGL - máquina de estados



- Comandos do OpenGL definem o estado atual
  - Ex.: `glColor3f(1.0f, 0.0f, 0.0f)` define o atributo de cor RGB atual (valores entre 0 e 1), isto é, a cor vermelha
- Vértices subseqüentes são afetados pelo estado atual
- Tipos de comandos
  - Comandos para gerar primitivas
  - Comandos para mudar o estado atual
    - Mudança de transformações
    - Mudança de atributos



## OpenGL - comandos



- Geração de primitivas
  - Pontos, segmentos de linha, polígonos
  - Criadas a partir de seus vértices
- Mudança de atributos
  - Cores, texturas, parâmetros de iluminação, matrizes de transformação
- Não é orientado a objetos
  - Múltiplos sufixos de comandos para um mesmo comando lógico. Ex.: `glVertex3f`, `glVertex2i`



## OpenGL - comandos



- `glVertex3fv` ← `v` se estiver presente, indica formato vetorial
  - tipo de dado: `f` float
  - `d` double float
  - `s` signed short integer
  - `i` signed integer
  - número de componentes: `2`, `3` ou `4`
- Ex.: `glVertex3f(2.3f, 5.0f, 1.5f);`
- Outros tipos de dados: `b` (byte), `ub` (unsigned byte), `us` (unsigned short integer), `u` (unsigned integer)



## OpenGL - comandos



- Exemplo de geração de um triângulo

```
glBegin(GL_TRIANGLES);
  glVertex2f(5.0f, 5.0f);
  glVertex2f(25.0f, 5.0f);
  glVertex2f(5.0f, 25.0f);
glEnd();
```





## OpenGL - comandos



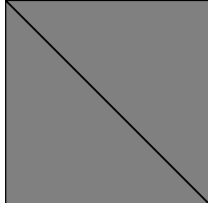
- Exemplo de geração de dois triângulos

```

glBegin(GL_TRIANGLES);
  glVertex2f(5.0f, 5.0f);
  glVertex2f(25.0f, 5.0f);
  glVertex2f(5.0f, 25.0f);
  glVertex2f(25.0f, 5.0f);
  glVertex2f(25.0f, 25.0f);
  glVertex2f(5.0f, 25.0f);
glEnd();

```

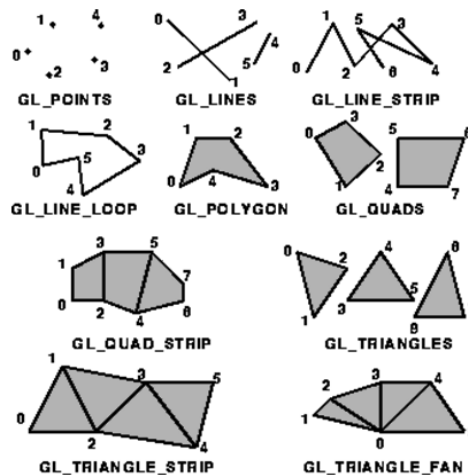
(5.0, 25.0) (25.0, 25.0)



(5.0, 5.0) (25.0, 5.0)



## OpenGL - primitivas possíveis





## OpenGL - *display list*



- Encapsula um conjunto de comandos para serem executadas posteriormente
  - É definida agrupando as instruções entre as funções `glNewList` e `glEndList`
  - Executada através do comando `glCallList`
- Vantagem
  - Geometria é armazenada na memória de vídeo, se possível
- Desvantagem
  - Não é possível modificar uma *display list*



## OpenGL - *display list*



- Exemplo do uso de uma *display list*

```
glNewList(0, GL_COMPILE);
glBegin(GL_TRIANGLES);
    glVertex2f(5.0f, 5.0f);
    glVertex2f(25.0f, 5.0f);
    glVertex2f(5.0f, 25.0f);
glEnd();
glEndList();
...
glCallList(0);
```





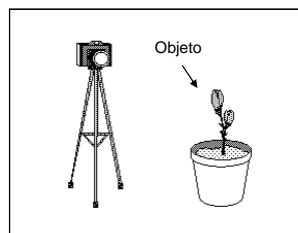


## OpenGL - transformações

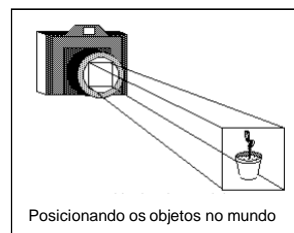


- Conversão de objetos 3D para um espaço 2D
- Analogia com uma câmera
  - Transformação de **modelagem**
    - Arranjar os objetos na cena

No mundo real



No OpenGL



## OpenGL - transformações

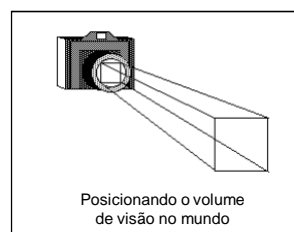


- Conversão de objetos 3D para um espaço 2D
- Analogia com uma câmera
  - Transformação de **visualização**
    - Posicionar a câmera na cena

No mundo real



No OpenGL



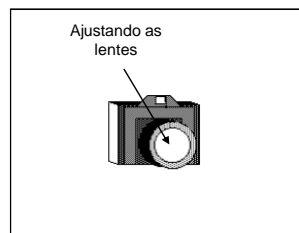


## OpenGL - transformações

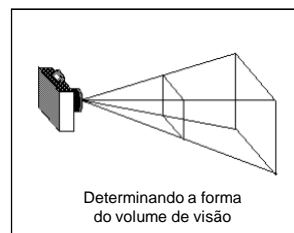


- Conversão de objetos 3D para um espaço 2D
- Analogia com uma câmera
  - Transformação de **projeção**
    - Definir as lentes e o *zoom* da câmera

No mundo real



No OpenGL

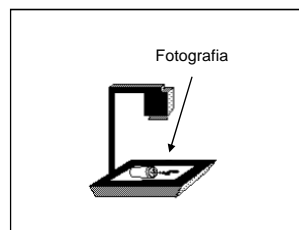


## OpenGL - transformações



- Conversão de objetos 3D para um espaço 2D
- Analogia com uma câmera
  - Transformação de **janela de visão**
    - Definir o tamanho da foto

No mundo real



No OpenGL





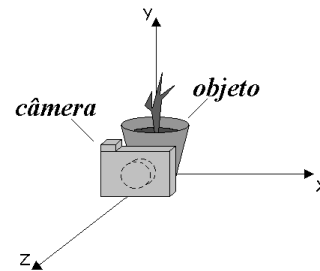
## OpenGL - transformações



- Transformações de modelagem e visualização
  - Câmera e objetos são originalmente posicionados na origem

comandos {

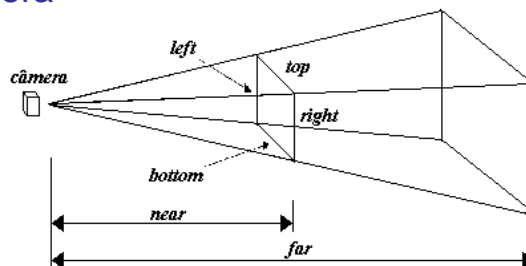
- `glTranslate*`
- `glRotate*`
- `glScale*`
- `glMultMatrix*`



## OpenGL - transformações



- Transformação de projeção perspectiva
  - Através da rotina `glFrustum`, define um volume de visualização no qual a projeção do objeto é reduzida a medida que ele é afastado da câmera

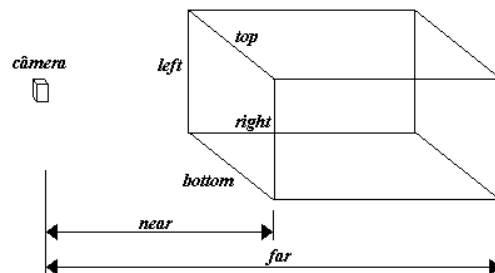




## OpenGL - transformações



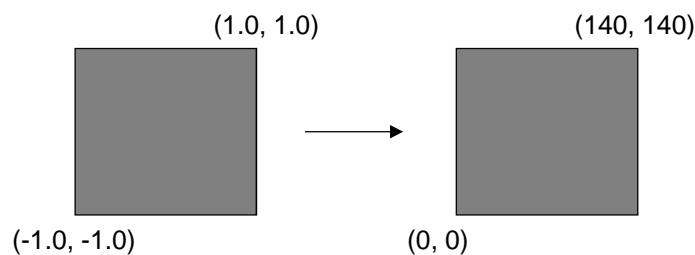
- Transformação de projeção ortogonal
  - Através da rotina `glOrtho`, define um volume de visualização onde a projeção do objeto não é afetada pela sua distância em relação à câmera



## OpenGL - transformações



- Transformação de janela de visão
  - Através da rotina `glViewport`, define uma correspondência entre as vértices projetados (normalizados entre -1.0 e 1.0) e os *pixels* da tela (*default* = janela inteira).

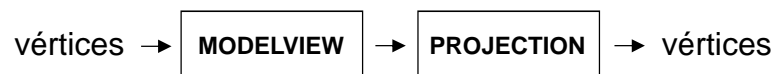




## OpenGL - transformações



- Operações de modelagem, visualização e projeção são realizadas através de duas matrizes:
  - **MODELVIEW**: modelagem e visualização
  - **PROJECTION**: projeção
- O comando `glMatrixMode` muda o estado da matriz a ser modificada, através do parâmetro `GL_MODELVIEW` OU `GL_PROJECTION`



## OpenGL - transformações



- Transformações e a metodologia de pilha
  - Organizar a seqüência de operações sobre as matrizes de transformação
  - Facilitar a construção de modelos hierárquicos
- O controle sobre a pilha é realizado através dos comandos `glPushMatrix` e `glPopMatrix`



## OpenGL - cores



- Possui dois modos de cor:
  - RGBA (*Red, Green, Blue, Alpha*)
  - Indexado
- A seleção do modo de cor é feita pela interface responsável pelo gerenciamento das janelas.
- Na biblioteca GLUT, a seleção do modo de cor é feita pelo comando `glutInitDisplayMode`.



## OpenGL - cores



- RGBA
  - As componentes variam de 0.0 a 1.0
  - A componente *alpha* é utilizada, e. g., em operações de transparência
  - O comando `glColor*` é utilizado para definir o estado da cor RGB dos vértices subsequentes
  - Interpolação das cores entre os vértices
    - Smooth: interpolação linear
    - Flat: uma cor para toda a primitiva



## OpenGL - cores



- Exemplo (RGBA, *smooth*)

```
glShadeModel(GL_SMOOTH);
glBegin(GL_TRIANGLES);
    glColor3f(1.0,0.0,0.0);
    glVertex2f(5.0,5.0);
    glColor3f(0.0,1.0,0.0);
    glVertex2f(25.0,5.0);
    glColor3f(0.0,0.0,1.0);
    glVertex2f(5.0,25.0);
glEnd();
```

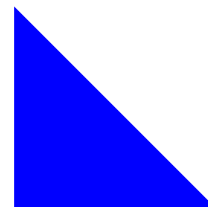


## OpenGL - cores



- Exemplo (RGBA, *flat*)

```
glShadeModel(GL_FLAT);
glBegin(GL_TRIANGLES);
    glColor3f(1.0,0.0,0.0);
    glVertex2f(5.0,5.0);
    glColor3f(0.0,1.0,0.0);
    glVertex2f(25.0,5.0);
    glColor3f(0.0,0.0,1.0);
    glVertex2f(5.0,25.0);
glEnd();
```

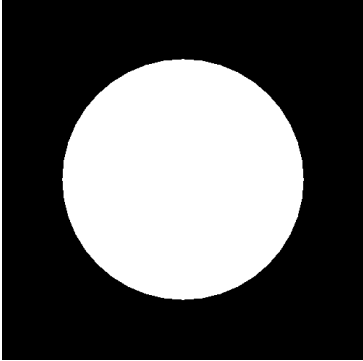


FEEC

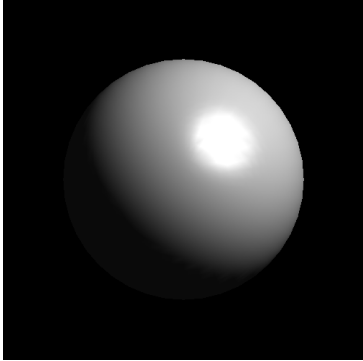
UNICAMP

## OpenGL - iluminação

- OpenGL utiliza o modelo de iluminação de Phong



`glDisable(GL_LIGHTING)`



`glEnable(GL_LIGHTING)`

FEEC

UNICAMP

## OpenGL - iluminação

- Componentes de iluminação


+

+

=


ambiente + difusa + especular = reflexão de Phong

- **Ambiente:** sem fonte de luz específica, proveniente das várias reflexões nas superfícies da cena
- **Difusa:** proveniente de uma direção específica, reflete em todas as direções
- **Especular:** proveniente de uma direção específica, reflete em uma direção específica





## OpenGL - iluminação



- Fontes de luz
  - Fonte direcional: irradia energia luminosa em uma determinada direção
  - Fonte pontual: irradia energia luminosa em todas direções
  - Fonte *spot*: tem uma direção principal na qual ocorre a máxima concentração de energia
- As rotinas `glLight*` são utilizadas para especificar o tipo, a posição e as componentes da fonte de luz.



## Downloads



- Mesa 3D:
  - [www.mesa3d.org](http://www.mesa3d.org)
- GLUT (Windows):
  - [www.opengl.org/resources/libraries/glut/glutdlls36.zip](http://www.opengl.org/resources/libraries/glut/glutdlls36.zip)