OpenGL: A Practical Introduction

(based on a talk by Mark Livingston)
Outline

- What is OpenGL?
- Auxiliary libraries
- Basic code structure
- Rendering
- Practical hints
- Virtual world operations
OpenGL Definitions

Software interface to graphics hardware
Model of client-server graphics
State machine
# Features of OpenGL

Basic features:
- Drawing primitives
- Transformations
- Color
- Lighting
- Display Lists

Advanced features:
- Texture mapping
- Vertex Arrays
- Blending effects
- Frame buffer manipulation
OpenGL Anti-definitions

Not a library of pre-defined 3D objects

Not a window system interface

Not a window system event manager

Not a user event manager
Auxiliary libraries

auxlib

glX

GLU

GLUT

Motif, Xt, X11
Features of auxiliary libraries

Most provide:

- Window system commands
- Events and callbacks
- More frame buffer management
- 3D drawing primitives

Some include:

- Some user interface items (e.g. menus)
- Improved support for fonts
- Overlay management
A typical OpenGL program

Definition of callback functions, including drawing and per-frame computations

 Initialization and window creation

Turn control over to the auxiliary library's event loop

(see cube.c handout)
Essential GLUT functions

- glutInitWindowSize
- glutInitWindowPosition
- glutInit
- glutInitDisplayMode
- glutCreateWindow
- glutDisplayFunc
- glutMainLoop
- glutSwapBuffers
Other GLUT Functionality

Event handling

- keyboard, mouse position, mouse buttons, window resize, etc.

Pop-up menus
### Primitives and Attributes

<table>
<thead>
<tr>
<th>“Open”</th>
<th>glBegin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normals</td>
<td>glNormal</td>
</tr>
<tr>
<td>Texture Coordinates</td>
<td>glTexCoord</td>
</tr>
<tr>
<td>Colors</td>
<td>glColor</td>
</tr>
<tr>
<td>Other material props</td>
<td>glMaterial</td>
</tr>
<tr>
<td>Vertex Coordinates</td>
<td>glVertex</td>
</tr>
<tr>
<td>“Close”</td>
<td>glEnd</td>
</tr>
</tbody>
</table>

Johns Hopkins Department of Computer Science
Course 600.460: Virtual Worlds, Spring 2000, Professor: Jonathan Cohen
Attributes and Current State

All drawing attributes have a current state maintained for each rendering context.

Calling `glVertex()` sets vertex position attribute and binds all necessary current state to the vertex.

`glColorMaterial` determines which material property is set by `glColor` “shortcut”

- usually `GL_AMBIENT_AND_DIFFUSE`
Lighting

Light properties
- Position or direction
- Color
- Attenuation

Enable lighting

Enable lighting
- glLight
- glEnable
  - GL_LIGHTING
  - GL_LIGHT0, GL_LIGHT1, etc.
# Textures

## Define (load)
- Image size
- Pixel format, data type

## Blend or replace?
- **glTexImage2D**
  - $2^M \times 2^N$

## Boundary handling
- **glTexEnv**
- **glTexParameter**

## Sampling

## Binding
- **glBindTextureEXT**

## Update “live” texture
- **glTexSubImage2DEXT**

---

Johns Hopkins Department of Computer Science
Course 600.460: Virtual Worlds, Spring 2000, Professor: Jonathan Cohen
Matrix stacks

Projection

• `glFrustum`, `gluPerspective`

Model-view

• `glRotate`, `glTranslate`, `glScale`, `glLoadIdentity`

Texture

Viewport (okay, no stack for this one)

• `glViewport`
Frame buffer configuration

Color
Alpha
Depth

Double-buffering
• glutSwapBuffers
Performance

Triangle/Quad Strips

Display lists

Vertex Arrays (man glIntro)

Performance

Render primitives with the “right” type

Lighting is slow

Don’t overload texture memory

Multiprocessing

• Not for feeding pipe, only for pre-processing
Some practical hints

Develop incrementally

Develop in wireframe

Develop without lighting, anti-aliasing, texturing, and other “extra” operations

Light positions get transformed

Lighting is per vertex

/usr/sbin/ogldebug <application>

Watch your modes -- state machine
Transformation matrices

Render axis tripods everywhere

Everything has a coordinate system!

- tracker, sensor, room, world, hand, eyes, etc.

Naming convention: foo2bar

A useful OpenGL paradigm

“Transform from object space to eye space.”
Column or row vectors?

\[ \mathbf{v'} = \mathbf{M} \ast \mathbf{v} \quad \iff \quad \mathbf{M}_3 \ast \mathbf{M}_2 \ast \mathbf{M}_1 \ast \mathbf{v} = \mathbf{M}_{321} \ast \mathbf{v} \]

\[
\begin{align*}
\mathbf{x'} &= \begin{bmatrix} a & b & c & d \end{bmatrix} \mathbf{x} \\
\mathbf{y'} &= \begin{bmatrix} e & f & g & h \end{bmatrix} \mathbf{y} \\
\mathbf{z'} &= \begin{bmatrix} i & j & k & m \end{bmatrix} \mathbf{z} \\
1 &= \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}
\end{align*}
\]

\[ \mathbf{v'} = \mathbf{v} \ast \mathbf{M} \quad \iff \quad \mathbf{v} \ast \mathbf{M}_1 \ast \mathbf{M}_2 \ast \mathbf{M}_3 = \mathbf{v} \ast \mathbf{M}_{123} \]

\[
\begin{align*}
\mathbf{x'} \mathbf{y'} \mathbf{z'} \mathbf{1} &= \begin{bmatrix} a & e & i & 0 \\
b & f & j & 0 \\
c & g & k & 0 \\
d & h & m & 1 \end{bmatrix} \\
\mathbf{x} \mathbf{y} \mathbf{z} \mathbf{1} &= \begin{bmatrix} a & e & i & 0 \\
b & f & j & 0 \\
c & g & k & 0 \\
d & h & m & 1 \end{bmatrix}
\end{align*}
\]
OpenGL Matrices

Written out using column vector notation

BUT: stored in memory in column-major order rather than row major

\[
\begin{pmatrix}
0 & 4 & 8 & 12 \\
1 & 5 & 9 & 13 \\
2 & 6 & 10 & 14 \\
3 & 7 & 11 & 15
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix}
\]
Manipulating transformations

Quatlib: library for common mathematical types and operations used in VEs

Source: Ken Shoemake, SIGGRAPH 1985; various UNC additions

Numerous operations and conversions

- affine matrix inversion, matrix multiplication, matrix-vector multiplication, vector magnitude, point-to-point distance, dot product, cross product
Conclusions

Reality: event-driven programming
Simple drawings are easy
Complex stuff is more complex
See the OpenGL and GLUT section of our course homework help page

• will be available soon