OpenL: A Practical Introduction

(Originally 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

glX, wgl
GLU
GLUT, FreeGLUT,
X11
glex, GLEW
gluv
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

“Open”

Normals

Texture Coordinates

Colors

Other material props

Vertex Coordinates

“Close”

glBegin

glNormal

glTexCoord

glColor

glMaterial

glVertex

glEnd

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
- `glLight`
- `GL_LIGHTING`
- `GL_LIGHT0`, `GL_LIGHT1`, etc.

**Textures**

Define (load)
- `glTexImage2D`
- Image size: `2^M x 2^N`
- Pixel format, data type

Blend or replace?
- `glTexEnv`

Boundary handling
- `glTexParameter`

Sampling

Binding
- `glBindTextureEXT`

Update “live” texture
- `glTexSubImage2DEXT`

**Matrix stacks**

Projection
- `glFrustum`, `gluPerspective`

Model-view
- `glRotate`, `glTranslate`, `glScale`, `glLoadMatrix`

Texture

Viewport (okay, no stack for this one)
- `glViewport`

**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?**

\[ v' = M * v \]

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

\[ v' = v * M \]

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

**OpenGL Matrices**

Written out using column vector notation

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

\[
\begin{bmatrix}
0 & 4 & 8 & 12 \\
1 & 5 & 9 & 13 \\
2 & 6 & 10 & 14 \\
3 & 7 & 11 & 15
\end{bmatrix}
\]

**Textures**

Define (load)
- `glTexImage2D`
- Image size: `2^M x 2^N`
- Pixel format, data type

Blend or replace?
- `glTexEnv`

Boundary handling
- `glTexParameter`

Sampling

Binding
- `glBindTextureEXT`

Update “live” texture
- `glTexSubImage2DEXT`
Frame buffer configuration

- Color
- Alpha
- Depth
- Double-buffering
  - `glutSwapBuffers`

Performance – CPU/API

- Minimize state changes
- Avoid flushing or stalling the pipe
  - Various gets and readbacks
- Use multi-processing for non-API functions

Performance – Vertex processing

- Vertex Arrays – reduce per-call overhead
- Vertex Buffer Objects – keep vertices in video/AGP memory
- Indexed vertex arrays – reduce data size
- Vertex re-ordering – reduce vertex processing
- Triangle Strips – reduce vertices and processing
- Display lists – opportunities for driver optimizations and storage in video memory
- Level of detail – reduce model quality, vertices

Performance – fragment processing

- Texture objects – allow indexing of texture data and state
- Mip-mapping – increase texture cache coherence
- Texture compression – fit more textures in video memory
- Pixel buffer object – increase readback speed
- `TexSubImage` – overwrite texture data rather than creating new texture

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
- Watch your modes — state machine

Conclusions

- Reality: event-driven programming
- Simple drawings are easy
- Complex stuff is more complex
For More Information

See the OpenGL and GLUT section of our course homework help page

• will be available soon