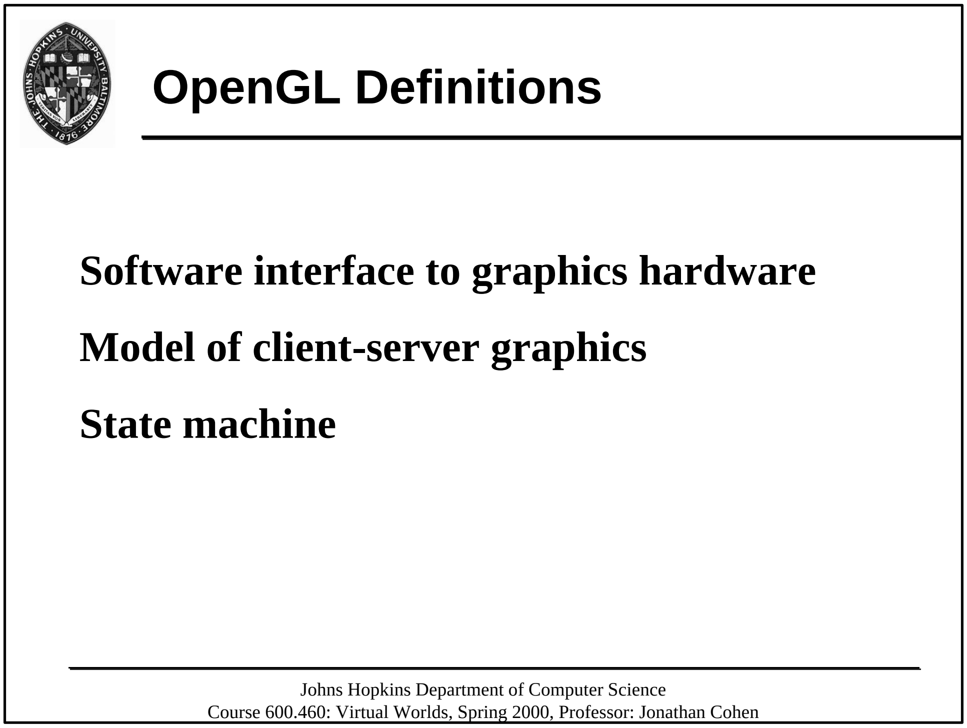
OpenGL:
A Practical Introduction

(based on a talk by Mark Livingston)

Outline

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

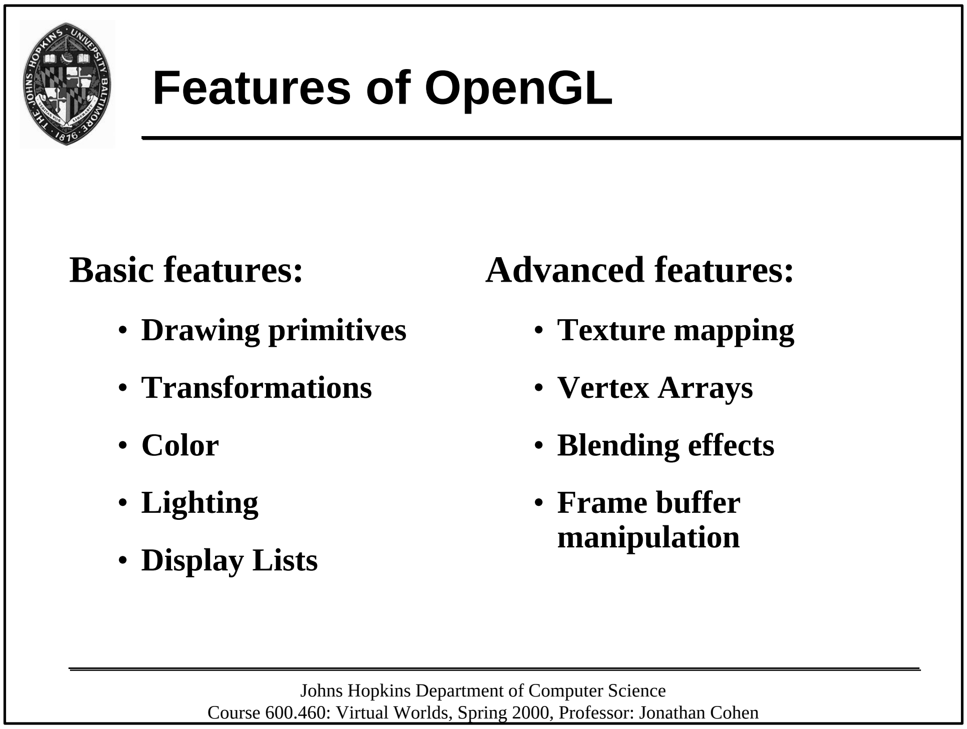


OpenGL Definitions

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**

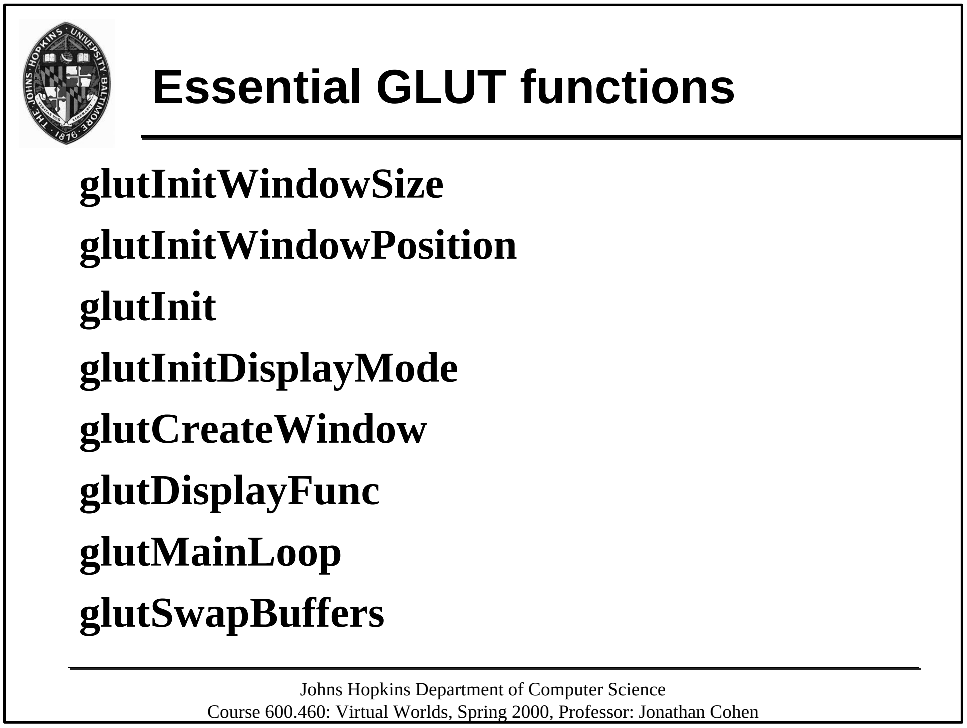
A typical OpenGL program

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

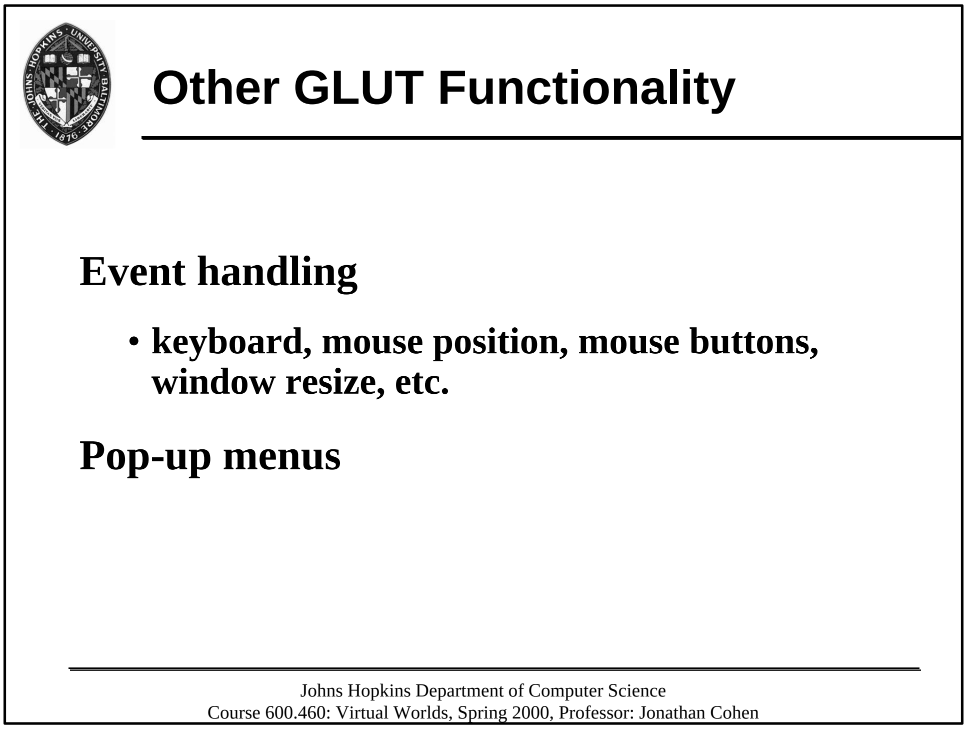
glutInitDisplayMode

glutCreateWindow

glutDisplayFunc

glutMainLoop

glutSwapBuffers



Other GLUT Functionality

Event handling

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

Pop-up menus

Primitives and Attributes

“Open”	**glBegin**
Normals	**glNormal**
Texture Coordinates	**glTexCoord**
Colors	**glColor**
Other material props	**glMaterial**
Vertex Coordinates	**glVertex**
“Close”	**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	**glLight**
• **Position or direction**	
• **Color**	
• **Attenuation**	
Enable lighting	**glEnable**
	• **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**

Frame buffer configuration

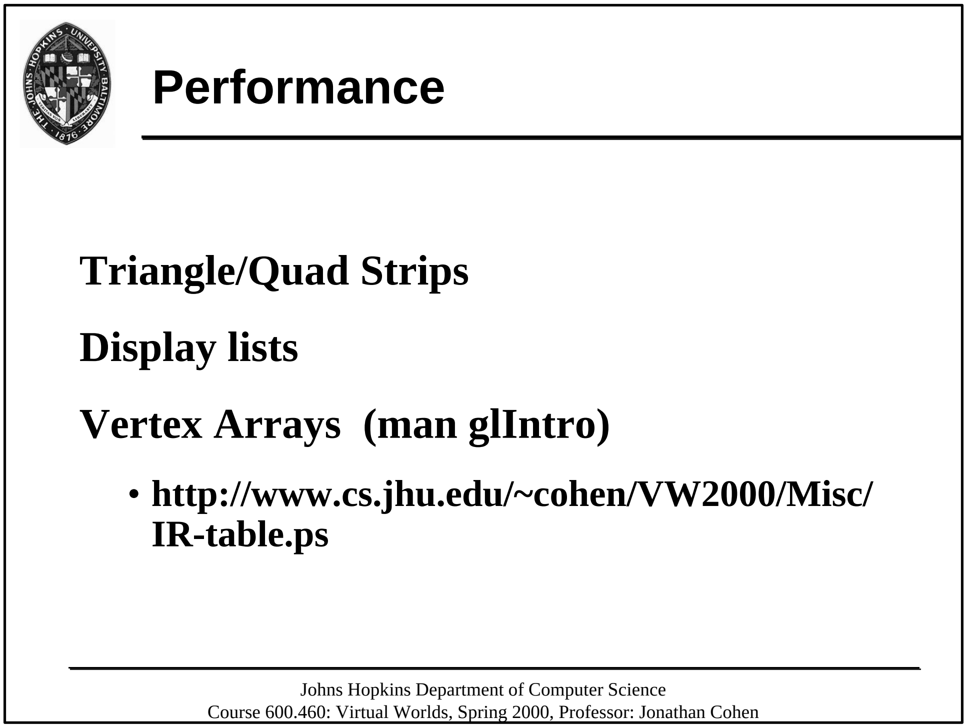
Color

Alpha

Depth

Double-buffering

- **glutSwapBuffers**



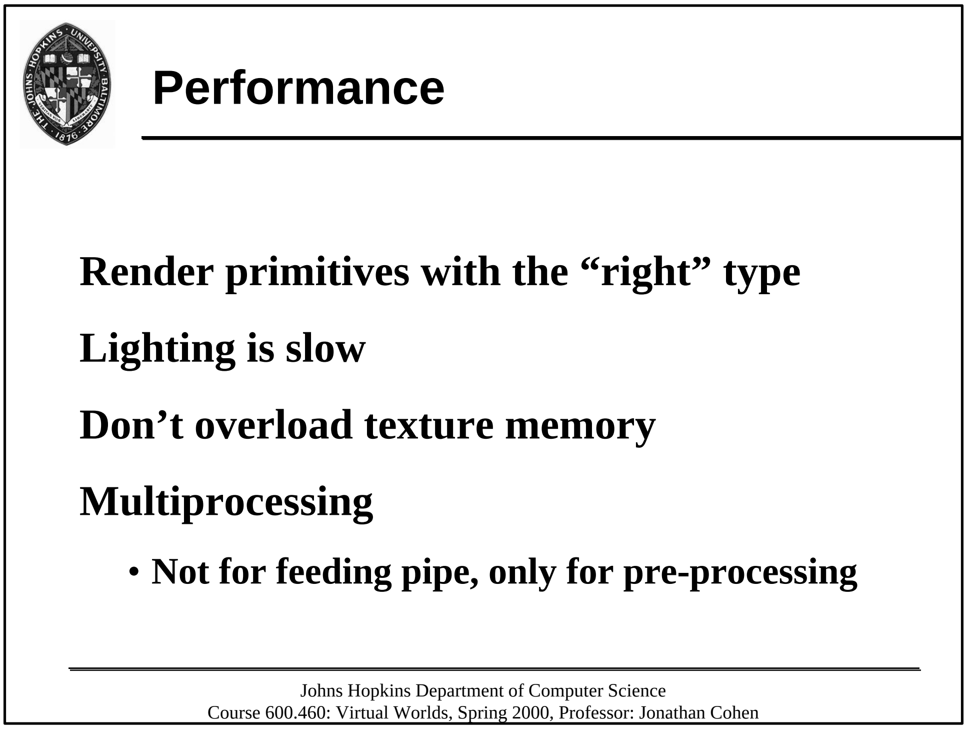
Performance

Triangle/Quad Strips

Display lists

Vertex Arrays (man glIntro)

- http://www.cs.jhu.edu/~cohen/VW2000/Misc/IR-table.ps



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?

v’ = M * v ⇨ **M3*M2*M1*v=M321*v**

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

v’ = v * M ⇨ **v*M1*M2*M3=v*M123**

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

OpenGL Matrices

Written out using column vector notation

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

$$\textbf{float M[16]} \quad \begin{matrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{matrix} * \begin{matrix} x \\ y \\ z \\ 1 \end{matrix}$$

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

For More Information

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

- will be available soon