OpenGL & Glut
Part I: Introduction

COSC 4431/5331
Computer Graphics

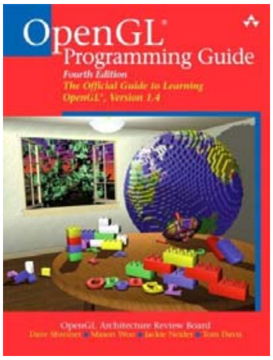
Bill Kapralos

Tuesday January 20, 2004

YORK UNIVERSITÉ UNIVERSITY

Essential Reference:

- **OpenGL Programming Guide (Third Edition):**
  - M. Woo, J. Neider, T. Davis & D. Shreiner
  - Extremely useful when developing OpenGL applications
  - Internet version – link from course web site
    - Some images in this presentation were taken from web site


OpenGL Programming Guide
Fourth Edition
The Official Guide to Learning OpenGL, Version 1.4
OpenGL Architecture Review Board
Dave Shreiner · Mason Woo · Jackie Neider · Tom Davis

Overview:

- **Introduction to OpenGL**
  - What is OpenGL ?
  - OpenGL command syntax
  - OpenGL as a state machine
  - OpenGL primitives
- **Introduction to GLUT**
  - What is GLUT ?
  - Initializing & creating a window
  - Handling window events
  - Sample GLUT code

Introduction to OpenGL

What is OpenGL? (1):

- **OpenGL is an API**
  - Software interface to the graphics hardware
  - Most widely used in the graphics industry
  - Supports both 2D and 3D
    - Can be used to produce interactive 3D applications
  - ~250 commands to specify objects and operations
- **Main Purpose of OpenGL → Rendering**
  - Conversion of object descriptions (geometric or mathematical) into images
  - Does not handle windowing or input tasks

What is OpenGL? (2):

- **Independent of Windowing System & OS**
  - Runs on Unix, Linux, Windows, Mac, OS/2 etc.
  - C/C++, Java, Fortran, Python, Perl, Ada
  - Scalable, portable, reliable, easy to use
    - Plenty of documentation freely available
- **Independent of Display Device**
  - Monitor, projector, HMD etc.

What is OpenGL ? (3):

- **OpenGL can only Render Primitives**
  - Low level commands only
    - High-quality color images composed of geometric and image primitives only
    - No commands to describe 3D objects
- **Geometric Primitives:**
  - Points, lines and polygons
- **Image Primitives:**
  - Bitmaps, images

What is OpenGL ? (4):

- **Example of OpenGL Rendered Scene**

What is OpenGL ? (5):

- **Another Example of OpenGL Rendered Scene**

What is OpenGL? (6):

- **Libraries Built on top of OpenGL**
  - Use OpenGL primitives to allow for high level commands describing complicated shapes and 3D objects/animations
  - OpenGL Utility Library (GLU)
    - Standard part of OpenGL (~50 commands)
    - Set up matrices for viewing transformations, polygon tessellation etc…
  - Fahrenheit Scene Graph (FSG)
    - Objects and methods for creating interactive 3D graphics applications

What is OpenGL? (7):

- **Libraries Built on top of OpenGL (cont…)**
  - OpenGL Extensions to allow display on specific windowing systems:
    - GLX → X-Windows
    - WGL → MS Windows 95/98/NT
    - AGL → Apple
  - OpenGL Utility Toolkit (GLUT) API
    - Interface to window system and input devices
    - Device independent unlike APIs listed above!
    - Most commonly used and also used in this course!

OpenGL Syntax (1):

- **OpenGL Argument Data Types:**

Suffix	Data Type	Corresponding C Type	OpenGL Type
b	8-bit integer	signed char	**GLbyte**
s	16-bit integer	short	**GLshort**
i	32-bit integer	int or long	**Glint, GLsizei**
f	32-bit floating point	float	**GLfloat, GLclampf**
d	64-bit floating point	double	**GLdouble, GLclampd**
ub	8-bit unsigned integer	unsigned char	**GLubyte, GLboolean**
us	16-bit unsigned integer	unsigned short	**GLushort**
ui	32-bit unsigned integer	unsigned int or unsigned long	**GLuint, GLenum, GLbitfield**

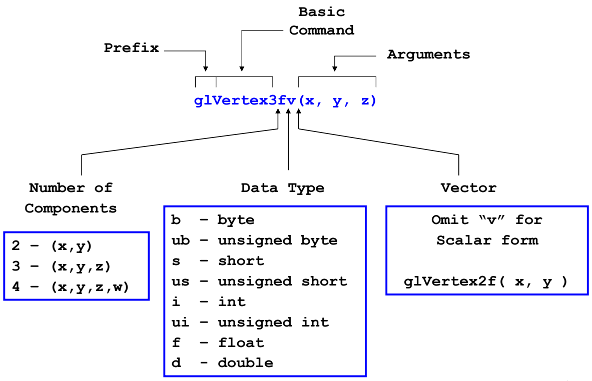
OpenGL Syntax (2):

- **Functions:**
  - Use the prefix "gl"
  - Each word after gl begins with capital letter
  - Example: glClearColor3f(), glBegin(), glEnd()
- **Constants:**
  - Upper-case letters only
  - Begin with "GL_"
  - Multiple words separated by "_"
  - Example: GL_COLOR_BUFFER_BIT, GL_DEPTH

OpenGL Syntax (3):

- **Functions May Contain Suffix as Well:**
  - Denotes the number of and type of arguments
  - Typically of the form: "xt"
    - "x" → number of arguments
    - "t" → argument type
  - Allows for "same" function name to be used with different arguments
  - Example: glColor3f(), glColor3i(), glColor2f(), glColor2i()

OpenGL Function Syntax:



```
2 - (x,y)
3 - (x,y,z)
4 - (x,y,z,w)
```

```
b  - byte
ub - unsigned byte
s  - short
us - unsigned short
i  - int
ui - unsigned int
f  - float
d  - double
```

```
Omit "v" for
Scalar form

glVertex2f( x, y )
```

OpenGL as a State Machine (1):

- **Various OpenGL Rendering Attributes are Treated as State Variables:**
  - Once set to specific state (value), OpenGL retains the state until state (value) is changed again
  - Each state variable has a default value – no need to explicitly set state unless needed
  - Some states have two values: activated or de-activated
  - Example state variables:
    - Current color, viewing & projection transformation, polygon drawing modes, lighting etc…

OpenGL as a State Machine (2):

- **Most Two-Value States are Initially De-activated**
  - May be costly to operate so activate only when needed - to turn state ON/OFF use:

```
glEnable(GLenum cap);
glDisable(GLenum cap);
```

  - Example states which can be activated/de-activated
    - GL_LIGHTING → lighting
    - GL_DEPTH_TEST → controls depth comparisons
    - GL_LINE_STIPPLE → patterned lines
    - GL_BLEND → controls blending of RGBA values

OpenGL as a State Machine (3):

- **State Querying Functions Available**
  - Find current value of a state

```
glGetBooleanv(GLenum pname, GLboolean *params);
glGetIntegerv(GLenum pname, GLint *params);
glGetFloatv(GLenum pname, GLfloat *params);
glGetDoublev(GLenum pname, GLdouble *params);
glGetPointerv(GLenum pname, GLvoid  **params);
```

  - *pname* → state variable to return value of
  - **params* → pointer to array where return data placed

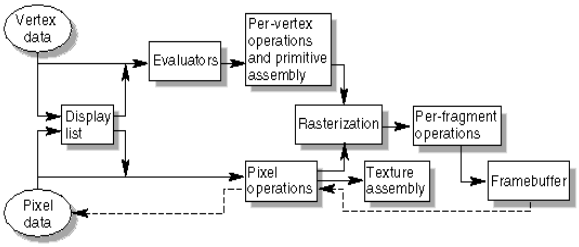
```
glGetFloatv(GL_CURRENT_COLOR, curColorValue);
```

OpenGL Rendering Pipeline (1):

- **Series of Processing Stages**
  - Not a "strict rule" but good predictor on what OpenGL will do
  - Geometric primitives:
    - Evaluators and per vertex operations
  - Pixel data (pixels, images, bit-maps):
    - Follow different path initially
  - Both data types undergo same "final steps" before final pixel data is written into the *framebuffer*
    - Rasterization and per-fragment operations

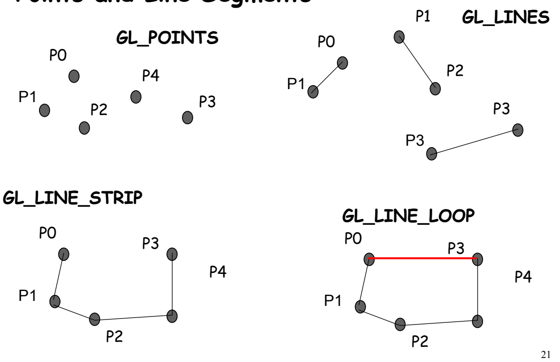
OpenGL Rendering Pipeline (2):

- **Graphical Illustration:**



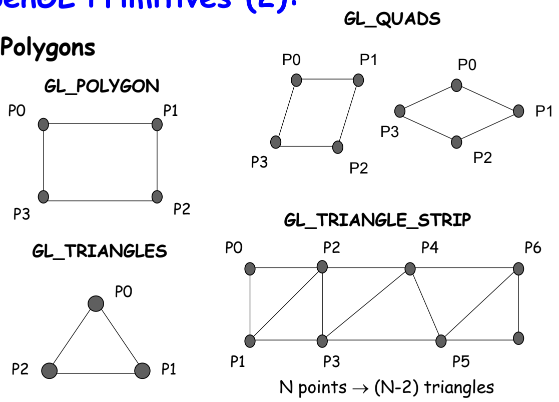
OpenGL Primitives (1):

- **Points and Line Segments**

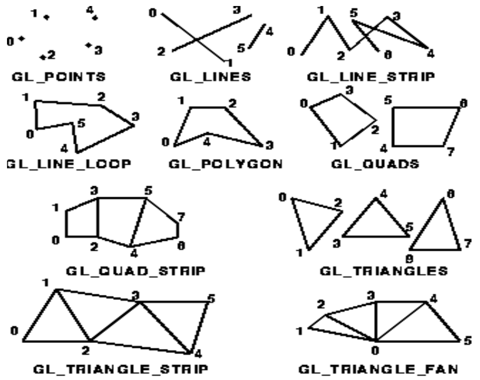


OpenGL Primitives (2):

- **Polygons**



OpenGL Primitives (3):



OpenGL Primitives (4):

- **Specifying Primitives/Geometry**
  - All primitives (geometric objects) are specified by a list of vertices between glBegin() and glEnd():
  - Usage:
    1. Begin with: glBegin(*primitive*) where *primitive* denotes the primitive type to draw (e.g. points, lines etc…)
    2. List vertices of primitive type
    3. End with: glEnd()

OpenGL Primitives (5):

- **A simple example: Rendering a triangle**

```
glBegin(GL_TRIANGLE);
   glVertex3f(x1, y1, z1);
   glVertex3f(x2, y2, z2);
   glVertex3f(x3, y3, z3);
glEnd();
```

- **A simple example: Rendering a polygon**

```
glBegin(GL_POLYGON);
   glVertex3f(x1, y1, z1);
   glVertex3f(x2, y2, z2);
   glVertex3f(x3, y3, z3);
   glVertex3f(x4, y4, z4);
glEnd();
```

OpenGL Primitives (6):

- **Example: Rendering a Red Triangle**

```
glBegin(GL_TRIANGLE);
   glColor3f(1.0, 0.0, 0.0);
   glVertex3f(x1, y1, z1);
   glVertex3f(x2, y2, z2);
   glVertex3f(x3, y3, z3);
glEnd();
```

- **Example: Rendering Different Colored Points**

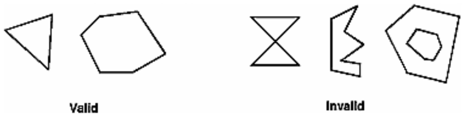
```
glBegin(GL_POINTS);
   glColor3f(1.0, 0.0, 0.0);
   glVertex3f(x1, y1, z1);
   glColor3f(0.0, 1.0, 0.0);
   glVertex3f(x2, y2, z2);
   glColor3f(0.0, 0.0, 1.0);
   glVertex3f(x3, y3, z3);
glEnd();
```

OpenGL Primitives (7):

- **Restrictions Regarding glBegin()/glEnd()**
  - Restricted set of OpenGL commands can be placed between glBegin/glEnd
  - Can specify vertices and vertex specific data for each vertex only (e.g. color, normal vector, texture coordinates etc.)
  - Any other programming language constructs are also allowed (e.g. loops, if/else etc.)

OpenGL Primitives (8):

- **Polygons and OpenGL**
  - Supports rendering of convex polygons only!
    - For any two points in interior, line joining them is also in the interior
    - No holes in polygons!
  - Polygons must be simple
    - Edges of polygon cannot intersect

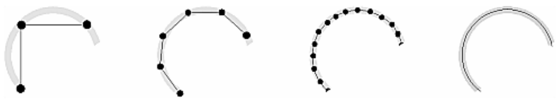


OpenGL Primitives (9):

- **Polygons and OpenGL (cont…)**
  - But many real-world surfaces consist of non-simple polygons, non-convex polygons or polygons with holes
    - Such polygons can be formed from unions of simple convex polygons
  - Routines to build more complex objects are provided in the GLU library
    - Tessellation: Take complex descriptions and break them down into groups of the simpler OpenGL polygons that can be rendered!

OpenGL Primitives (10):

- **Tessellation Example**
  - Any smooth curved line or surface can be approximatedm by short line segments or small polygonal regions.
  - Arbitrarily set accuracy of approximation
    - Decrease length of each segment → increase accuracy



OpenGL Primitives (11):

- **Every Polygon Has Two Sides: Front & Back**
  - Rendered differently depending which side is facing viewer
  - Allows for cut-away views of objects where there is difference between parts inside and those outside
  - By default, both front & back drawn same way
    - Change using:

```
void glPolygonMode(GLenum face, GL_enum mode);
```

- *face* → GL_FRONT_AND_BACK, GL_FRONT or GL_BACK
- *mode* → GL_POINT, GL_LINE_ or GL_FILL (indicates if polygon is drawn as points, outline or filled)

OpenGL Primitives (12):

- **Objects Drawn Independent of Color**
  - Object color is a state variable
  - Objects rendered using current color
- **Two Modes to Store *Bitplanes***
  - Bitplane → pixel colors stored in hardware
  - RGBA Color Mode
    - Store red, green, blue and alpha values directly in bitplane
  - Index Color Mode
    - Store single index that references color look-up table

OpenGL Primitives (13):

- **RGBA Color Mode**
  - Mixture of red, green and blue colors
  - Each r,g,b value is given value between 0.0 to 1.0
    - 0.0 → don't use any of specific component
    - 1.0 → use the maximum of specific component
    - In OpenGL use glColor*() command – for example:

```
void glColor3f(r, g, b);
```

- To set the current color to red:

```
void glColor3f(1,0, 0.0, 0.0);
```

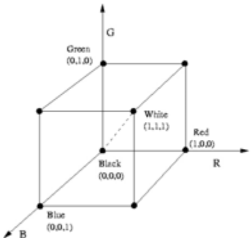
OpenGL Primitives (14):

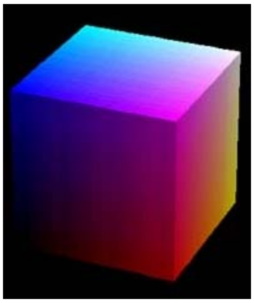
- **RGBA Color Mode (cont…)**

```
glColor3f(0.0, 0.0, 0.0);   /*black*/
glColor3f(1.0, 0.0, 0.0);   /*red*/
glColor3f(0.0, 1.0, 0.0);   /*green*/
glColor3f(0.0, 0.0, 1.0);   /*blue*/
glColor3f(1.0, 1.0, 0.0);   /*yellow*/
glColor3f(0.0, 1.0, 1.0);   /*cyan*/
glColor3f(1.0, 0.0, 1.0);   /*magenta*/
glColor3f(1.0, 1.0, 1.0);   /*white*/
```

OpenGL Primitives (15):

- **RGB Color Cube**
  - r,g,b colors define a cube of possible color mixtures





OpenGL Primitives (16):

- ***Clearing the Display Window***
  - Specify the background color in RGBA format
    - RGB → red, green, blue value (0.0 – 1.0)
    - A → alpha – transparency 0.0 – 1.0 (0.0 is opaque)
    - Background color is a state variable

```
void glClearColor(r, g, b, a);
void glClear(GL_COLOR_BUFFER_BIT);
```

- Color bitplane is one of several buffers maintained by OpenGL:
  - Depth buffer, accumulation buffer, stencil buffer – use glClear() to clear these too

Introduction to the OpenGL Utility Toolkit (GLUT)

What is GLUT? (1):

- **OpenGL Utility Toolkit**
  - Not officially part of OpenGL
  - Interface to window system and input devices
  - Written by Mark J. Kilgard initially for X-Windows
    - Ported to Microsoft by Nate Robins
  - Purpose:
    - Enable construction of OpenGL applications independent of any window system
    - Can write applications without knowing about X-Windows, Microsoft's or Apple's window system

What is GLUT? (2):

- **Event Based**
  - Open rendering window
  - Register callback functions for any specific window or input events of interest
    - Mouse, keyboard, window re-sizing, etc.
  - Create a *main loop* which never exits and continuously:
    - Scans for any of the registered events
    - When registered event detected, appropriate callback functions are executed
    - After completing callback function, back to main loop

Initializing & Creating a Window:

```
void glutInit(int argc, char *argv[]);
```

- Initializes the GLUT library
- Processes window system specific command line arguments

```
void glutInitDisplayMode(int mode);
```

- sets display mode (e.g. single buffer with RGB) to *mode*

```
Void glutInitWindowSize(int w, int h);
```

- sets window size to width = *w* and height = *h*

```
void glutInitPosition(int x, int y);
```

- sets upper left corner of window to position *x, y*

```
void glutCreateWindow(char *name);
```

- open window with title *name*

Handling Window & Input Events (1):

```
void glutDisplayFunc(void (* (func)(void));
```

- Specifies function to be called when window needs to be re-drawn (e.g. when window is initially opened, window is popped or damaged etc.)
- Can also be explicitly called using **glutPostRedisplay()**

```
void glutreshapeFunc((* (func)(int width, int height));
```

- Specifies function to be called when window is re-sized
- Two arguments specify new window dimensions

```
void glutKeyboardFunc(* (func)(int key, int x, int y);
```

- Specifies function to be called when a key which generates an ASCII character is pressed.
- *key* is the ASCII value of the pressed key
- *x,y* are the coordinates of mouse in the window when key was pressed

Handling Window & Input Events (2):

```
void glutMouseFunc(void (*(func)(button, state, x, y));
```

- Specifies function to be called when mouse botton is pressed or released.
- *button:* **GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON or GLUT_MIDDLE_BUTTON**
- *state*: **GLUT_UP** or **GLUT_DOWN**
- *x,y* are the coordinates of the mouse when event occurred

```
void glutMotionFunc((* (func)(int x, int y));
```

- Specifies function to be called when mouse pointer moves within the window while one or more mouse buttons are pressed
- *x,y* are coordinates of mouse when event occurred

```
void glutPostRedisplay(void);
```

- Calls **glutPostRedisplay**() in order to re-draw window

Handling Window and Event Functions (3):

- **Display Callback Function Example:**
  - All drawing is done in this function
  - Define Function:

```
void myDisplayFunction(void){
  // Insert any "drawing" specific commands here
  // e.g. viewing/model transformations

  glBegin(GL_POLYGON);
    glVertex3fv(x1, y1, z1);
    glVertex3fv(x2, y2, z2);
    glVertex3fv(x3, y3, z3);
    . . .
  glEnd();
}
```

  - Register callback:

```
glutDisplayFunc(myDisplayFunction);
```

Handling Window and Event Functions (4):

- **Keyboard Callback Function Example:**
  - Define Function:

```
void myKeyboardFunction(char key, int x, int y ){
  switch( key ) {
    case 'q':
     exit(1);
     break;
    case 'r':
     rotateObject = GL_TRUE;
     break;
  }
}
```

  - Register callback:

```
glutKeyboardFunc(myKeyboardFunction);
```

Drawing 3D Objects:

- **Several Drawing Routines for 3D Objects**
  - All graphics rendered in *immediate* mode (e.g. drawn immediately rather than at a latter time)
  - Two "flavors" for each 3D object:
    1. Wire-frame → no surface normals
    2. Solid → surface normals included – for lighting
  - Example functions for 3D objects:

```
glutWireCube(GLdouble size);
glutSolidCube(GLdouble size);
glutWireTeapot(GLdouble size);
glutSolidTeapot(GLdouble size);
```

Sample GLUT Code (1):

```
    // Initialization of GLUT, display mode and window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("Test");

    // Register any callback functions
    glutDisplayFunc(myDisplayFunction);
    glutReshapeFunc(myReshapeFunction);
    glutMouseFunc(myMouseFunction);
    glutKeyboardFunc(myKeyboardFunction);

    // Enter the GLUT main loop and wait for any events
    glutMainLoop( );
}
```

Getting Started (1):

- **Microsoft Windows (XP) & Visual Studio (C++)**
  - OpenGL included in newer versions of Windows OS
  - If using MS Visual Studio GLUT also installed
  - Compiling and linking – After creating project
    - From menu bar, go to

      "Project -> Settings -> … Link"
    - Add the following string to the "Objects/Library Modules" string

      `"opengl32.lib glu32.lib glut32.lib"`
    - Build & execute program

Getting Started (2):

- **Include Libraries**
  - For all OpenGL applications, include gl.h in every file
  - Almost all OpenGL applications use GLU, so include glu.h as well
  - If using Glut, you also need glut.h
  - OpenGL source file typically begins with

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
```

  - But glut.h includes the gl.h and glu.h so reall only glut.h is needed!

Getting Started (3):

- **Setting Up Project in Visual Studio/C++**

1. Load Visual Studio/C++
2. File->New a dialog box will appear - choose "Win 32 Console Application", give the project a name and press "OK"
3. Another dialog box will then appear: choose "A Simple Application" and click "Finish"
4. Your new project workspace will now be available and on the screen - add the three libraries as previously described
5. All necessary files etc. will be generated in addition to the file containing "main" method – this is entry point