

# **Introduction to OpenGL**

**CSCI 4229/5229  
Computer Graphics  
Fall 2021**

# OpenGL by Example

- Learn OpenGL by reading
- [nehe.gamedev.net](http://nehe.gamedev.net)
- [lighthouse3d.com](http://lighthouse3d.com)
  - Excellent free tutorials
- OpenGL: A Primer (3ed) by Edward Angel
  - Short and sweet
- OpenGL Programming Guide (Vermillion Book)
  - Free older editions as PDF
- OpenGL Superbible
  - Theory and Applications

# What is OpenGL?

- Sometimes called a library, actually an Application Programming Interface (API)
- Specification is controlled by Khronos
- Multiple implementations by different vendors
  - Mesa & FreeGLUT free implementations
- OpenGL just does real time graphics
  - Need GLX/WGL/AGL for windowing and input
  - Limited font support (in GLUT)
  - No sound, printing, etc. support

# OpenGL Versions

- 1.0 Initial release (1992)
- 1.1 Major upgrade (1997)
  - Latest version on some Windows system
- 1.2 Improves textures (1998)
- 1.3-1.5 Incremental improvements (2001-2003)
- 2.0 Relaxes restrictions, adds shader (2004)
- 2.1-2.3 Incremental improvement (2006-7)
- 3.0 Support advanced hardware features (2008)
- 3.1-3.3 Improved shaders (2009)
- 4.0 Merge desktop and devices (2010)
- 4.1-4.x Additional shaders

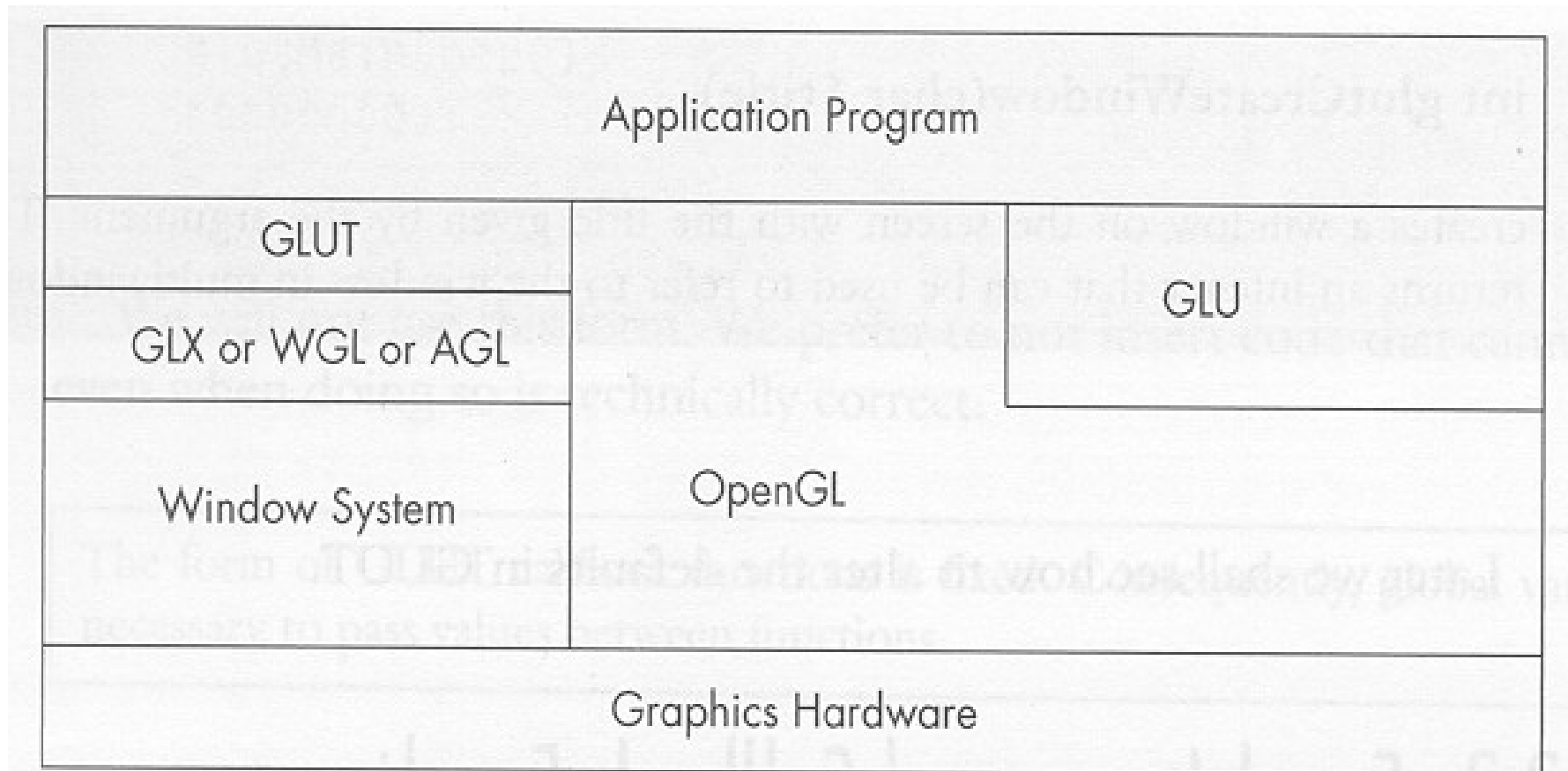
# OpenGL Deprecation

- I will mostly use OpenGL 2.0
  - Feature rich, flat learning curve
  - I will use GL3 or GL4 only as needed
- OpenGL Core Profile concentrates on rendering
  - Improved execution time performance
- User must provide deprecated functionality
  - Steepens the learning curve
  - Deprecated features in Compatibility Profile
  - Increases reliance on third party libraries
  - Adds development time until tools mature

# OpenGL APIs

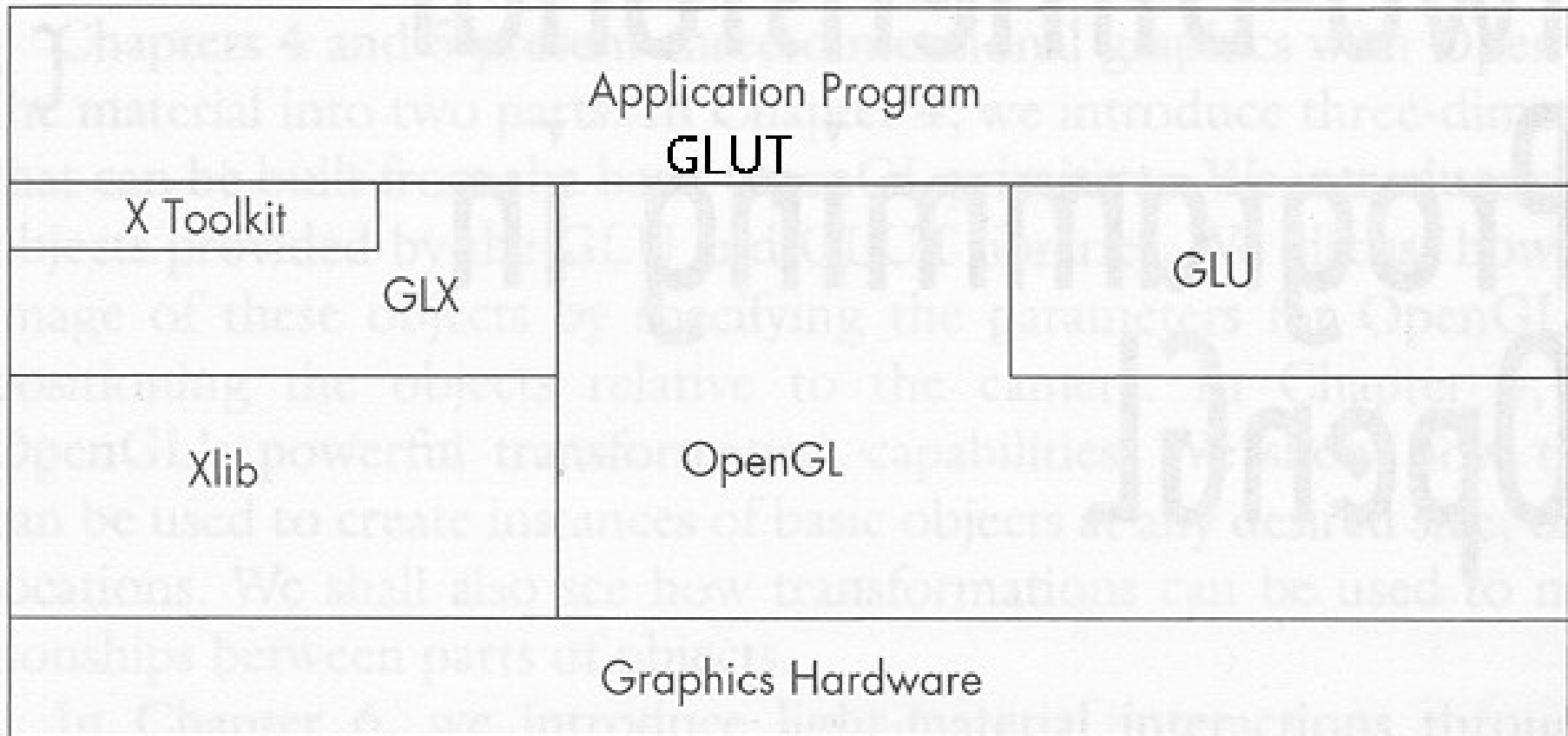
- Languages
  - C, C++, C#
  - FORTRAN
  - Java
  - Perl
  - Python
  - Ada
- Packages
  - Qt (QOpenGLWidget)
  - SDL, glfw, etc
  - Many others

# OpenGL and Friends



From *OpenGL: A Primer*

# OpenGL on X11



From *OpenGL: A Primer*



# GLU: OpenGL Utility

- Higher Level and Convenience Functions
  - Projections
  - Creating texture maps
  - NURBS, quadrics, tessalation
  - Predefined objects (sphere, cylinder, teapot)
- Collections of calls for convenience
- Standard with all OpenGL implementations

# GLUT: GL Utility Toolkit

- Provides access to OS and Window System
  - Open windows and setting size and capabilities
  - Register and triggers callbacks
  - Keyboard and mouse interaction
  - Elementary fonts
- Not part of OpenGL, but provides a portable abstraction of the OS
  - FreeGLUT
  - OpenGLUT
- Alternatives: SDL, Qt, glfw, ...

# Header Files and Libraries

- Usually you only need
  - `#define GL_GLEXT_PROTOTYPES`
  - `#include <GL/glut.h>`
- Header file locations
  - `/usr/include/GL` on most systems
- Linking may only need
  - `-l glut -l GLU -l GL`
- Special cases
  - OS/X separates GL and GLUT
  - Windows differs depending on the compiler

# OpenGL Naming Convention

- glSomethingNt()
  - *Something* is the name of the function
  - N is 2 or 3 or 4 for the dimension
  - t is for the the variable type
    - b GLbyte (signed char) 8 bit
    - s GLshort (signed short) 16 bit
    - i GLint (signed int) 32 bit
    - ub GLubyte (unsigned char) 8 bit
    - us GLushort (unsigned short) 16 bit
    - ui GLuint (unsigned int) 32 bit
    - f GLfloat (float) 32 bit
    - d GLdouble (double) 64 bit

# OpenGL Naming Example

- Vertex
  - glVertex3i(0 , 0 , 1)
  - glVertex2d(27.34 , 88.12)
  - glVertex3dv(array)
- Few functions return a value
- Most functions created by name mangling
- Constants are GL\_SOMETHING
- Variable types are GLsomething

# GLUT and GLU Naming

- Functions
  - glutSomething
  - gluSomething
- Constants
  - GLUT\_SOMETHING
  - GLU\_SOMETHING
- You can always tell by the name which API supplies a function or constant
- Avoid things starting with glx, wgl & agl

# GLUT: GL Utility Toolkit

- Supplies interface to OS
  - Windowing
  - Interaction
- Hello World in GLUT (well sorta)

```
int main(int argc,char* argv[ ])
{
    glutInit(&argc,argv);
    glutCreateWindow("Hello World");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

# Completing Hello World

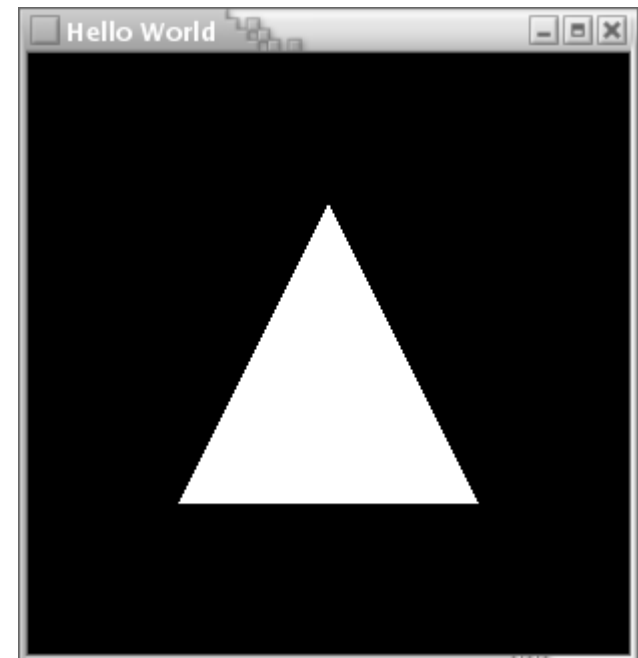
- Draw a triangle

```
#include <GL/glut.h>
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    glVertex2f(0.0,0.5);
    glVertex2f(0.5,-0.5);
    glVertex2f(-0.5,-0.5);
    glEnd();
    glFlush();
}
```



# Compile, link and run

- `gcc -Wall -o ex1 ex1.c -lglut -lGL`
- Heavily relies on defaults
  - Window
  - Viewport
  - Projection
  - Color



# Event Driven Programming

- Don't call us, we'll call you
  - register callbacks corresponding to events
  - similar to interrupt driven programs
- DO NOT explicitly call display()
  - request redisplay using glutPostRedisplay()
- NEVER call sleep()
  - use global/static variables and wall time for timing
  - use glutTimerFunc() for regular events
- Return control as soon as possible

# Types of Objects

- glBegin(type)
  - GL\_POINTS points
  - GL\_LINES lines between pairs of points
  - GL\_LINE\_STRIP series of line segments
  - GL\_LINE\_LOOP closed GL\_LINE\_STRIP
  - GL\_POLYGON simple polygon
  - GL\_TRIANGLES triangles between triples of points
  - GL\_TRIANGLE\_STRIP series of triangles
  - GL\_TRIANGLE\_FAN fan of triangles
- Set coordinates with glVertex
- glEnd()

# Qualifiers

- `glPointSize(float size)`
  - POINT size in pixels (default 1)
- `glLineWidth(float width)`
  - LINE width in pixels (default 1)
- `glLineStipple(int factor, unsigned short pattern)`
  - LINE type
  - Requires `glEnable(GL_LINE_STIPPLE)`

# Color

- Default is RGB color
  - X11 TrueColor
  - R,G,B 0-1 or integer range
    - glColor3f(1.0 , 0.0 ,0.0)
    - glColor3b(127 , 0 , 0);
    - glColor3ub(255 , 0 , 0);
    - glColor3fv(rgbarray);
- Color can also contain transparency (alpha)
  - glColor4f(1.0 , 0.0 , 0.0 , 0.5);
  - Default alpha=1 (opaque)
- Stays in effect until you change color

# Indexed Color

- X11 Direct Color
  - Based on a colormap
- Set color using `glIndexi(27)`
- Need to load colors into color map using `glutSetColor()`
- Use RGB color unless hardware constrain
- Deprecated in OpenGL 3 since it really is obsolete

# Displaying a scene

- Register using `glutDisplayFunc()`
- `glClear()`
- *Draw Something*
- `glFlush()`
- `glutSwapBuffers()`
- Schedule using `glutPostRedisplay()`

# Transformations

- Transformation apply to everything that follows
- Transformations are cumulative
  - Call `glLoadIdentity()` in `display()`
- Primitive operations
  - `glLoadIdentity();`
  - `glTranslate[fd](dx , dy , dz)`
  - `glScale[fd](Sx , Sy , Sz)`
  - `glRotate[fd](angle , Ux , Uy , Uz)`
- Compatibility profile in OpenGL4 still useful



```
glTranslate[fd](dx , dy , dz);
```

- Move an object in three dimensions
- Allows you to easily produce multiple copies of an object
- Always takes 3D coordinates (float or double)

# glScale[fd](Sx , Sy , Sz)

- Change the scale along the axes
- Multiplicative factors
  - $|S| < 1$  shrink
  - $|S| > 1$  expand
  - Negative values creates mirror image
- Allows you to easily create multiple copies of the same type at different sizes

# glRotate[fd](angle , Ux , Uy , Uz)

- Rotates around the origin and axis (Ux,Uy,Uz)
- Angle is measured in degrees
- The axis often a primary axis, but may be any axis
  - Undefined behavior if  $Ux=Uy=Uz=0$
- Allows you to create multiple copies of the same object viewed from different sides, or to view the scene from different positions

# Temporary Transformations

- `glPushMatrix()`
  - Saves the current transformation
- `glPopMatrix()`
  - Resets the transformation to what it was when you did the push
- Allows you to build complex transformations and then get them back

# Compound Transformations

- Rotate angle around the point  $(X,Y,Z)$  and axis  $(U_x,U_y,U_z)$ 
  - `glTranslated(-X,-Y,-Z)`
  - `glRotated(angle,U_x,U_y,U_z)`
  - `glTranslated(X,Y,Z)`
- OpenGL does this intelligently

# Projections

- Orthographic
  - `glOrtho(left,right,bottom,top,near,far)`
  - Same size regardless of distance
  - Easiest to use
- Perspective
  - `glFrustum(left,right,bottom,top,near,far)`
  - Closer objects are bigger
  - GLU convenience functions
    - `gluPerspective(fov,aspect,Znear,Zfar)`
    - `gluLookAt(Ex,Ey,Ez , Cx,Cy,Cz , Ux,Uy,Uz)`

# Text

- OpenGL provides only hooks for fonts
- Stroked fonts
  - Lines and fills write the characters
- Bitmap (raster) fonts
  - Characters are raster images
- Orientation, size, etc. treated just like any other drawing elements

# Text using GLUT

- `glutBitmapCharacter(GLUT_FONTTYPE,ch)`
  - Single character
  - Limited font selection
- `glRasterPos3d(x,y,z)`
  - Sets position to write text in (x,y,z) coordinates
- `glWindowPos2i(x,y)`
  - Sets position to write text in pixels coordinates



# Registering Callbacks

- Display
  - glutDisplayFunc() Draw the scene
  - glutReshapeFunc() Window resized
  - glutIdleFunc() Nothing more scheduled
- User input
  - glutKeyboardFunc() Key pressed
  - glutSpecialFunc() Special key pressed
  - glutMouseFunc() Mouse button
  - glutMotionFunc() Mouse motion
- Many more

# Keyboard Input

- `special(int key,int x,int y)`
  - Cursor keys `GLUT_KEY_LEFT`, `GLUT_KEY_UP`,...
  - Function keys `GLUT_KEY_Fx`
  - Basically anything not an ASCII key
- `keyboard(char ch,int x,int y)`
  - Regular ASCII keystrokes
- `(x,y)` is the mouse position in pixels

# Setting Modes

- `glutInitDisplayMode`
  - Interfaces with the window manager to get the right kind of window (BE CAREFUL ABOUT DEFAULTS)
- `glEnable()` & `glDisable()`
  - Switches OpenGL into various modes
    - `GL_DEPTH_TEST`
    - `GL_ALPHA_TEST`
    - `GL_CULL_FACE`
    - `GL_LIGHTING`
  - Different modes for different objects

# Checking for Errors

- OpenGL fails silently
- Functions do not return an error code
- `glGetError()` must be called explicitly to check for errors
- A black screen is a sure signal of an error