Computer Graphics

by Ruen-Rone Lee

ICL/ITRI

OpenGL at a Glance

What is OpenGL

OpenGL Processing Pipeline

OpenGL Shaders

OpenGL Shading Language

OpenGL Utilities

A Simple OpenGL Framework with GLUT

What is OpenGL

Standard Graphics APIs

What is OpenGL

- OpenGL (Open Graphics Library) is an open standard for cross-language, cross-platform API specification
- OpenGL is not a programming language
- OpenGL is a set of APIs (Application Programming Interface) that is used to write 2D/3D graphics applications

OpenGL Evolution

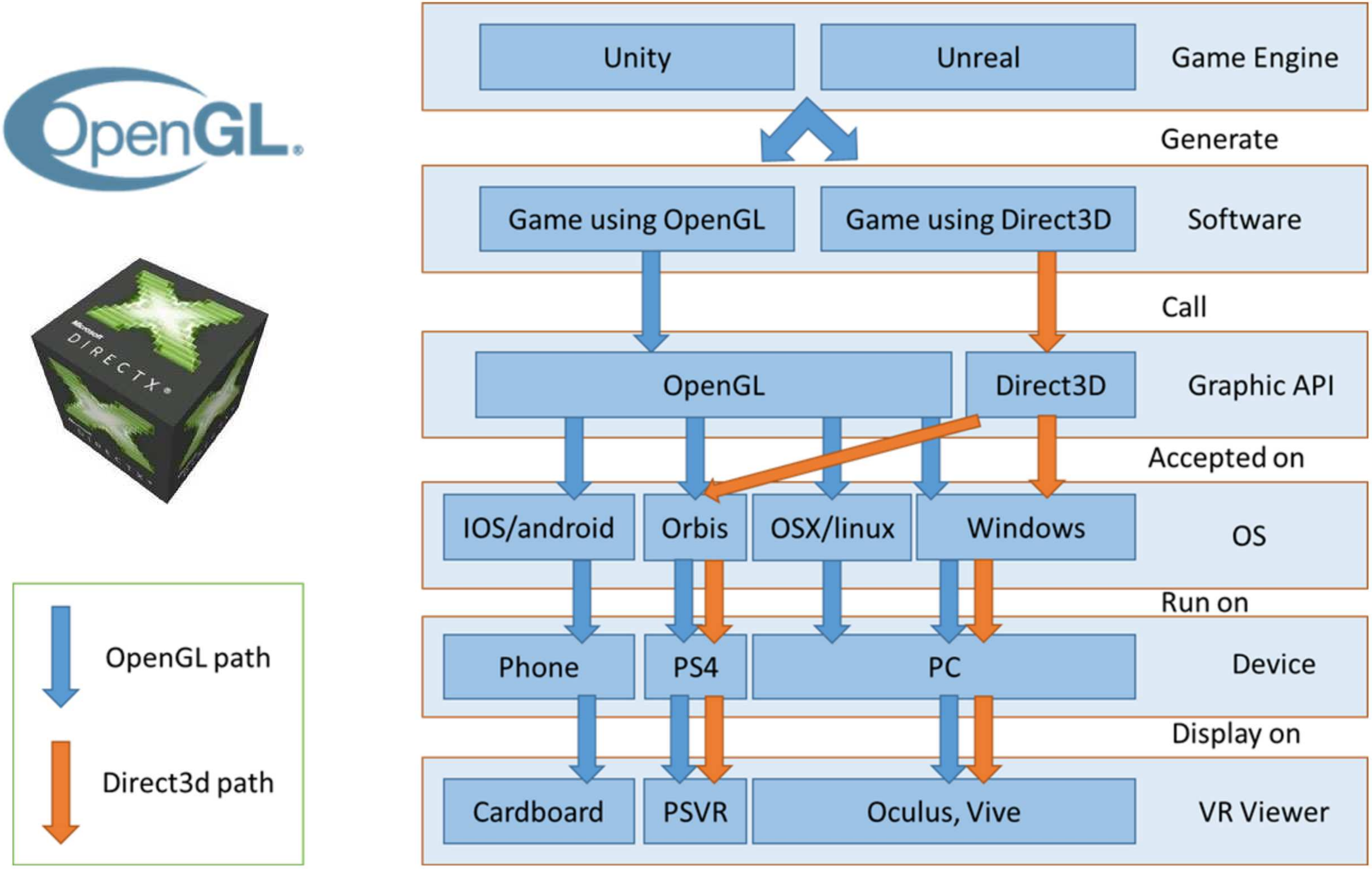
Source: Wikipedia

- **Fixed Function Pipeline (1992~2003)**
  - OpenGL 1.1 - Texture objects
  - OpenGL 1.2 - 3D textures, BGRA and packed pixel formats[20]
  - OpenGL 1.3 - Multitexturing, multisampling, texture compression
  - OpenGL 1.4 - Depth textures
  - OpenGL 1.5 - Vertex Buffer Object (VBO), Occlusion Queries[21]
- **Programmable Pipeline (2004~present)**
  - OpenGL 2.0 - GLSL 1.1, MRT, Non Power of Two textures, Point Sprites,[22] Two-sided stencil[21]
  - OpenGL 2.1 - GLSL 1.2, Pixel Buffer Object (PBO), sRGB Textures[21]
  - OpenGL 3.0 - GLSL 1.3, Texture Arrays, Conditional rendering, Frame Buffer Object (FBO)[23]
  - OpenGL 3.1 - GLSL 1.4, Instancing, Texture Buffer Object, Uniform Buffer Object, Primitive restart[24]
  - OpenGL 3.2 - GLSL 1.5, Geometry Shader, Multi-sampled textures[25]
  - OpenGL 3.3 - GLSL 3.30 Backports as much function as possible from the OpenGL 4.0 specification
  - OpenGL 4.0 - GLSL 4.00 Tessellation on GPU, shaders with 64-bit precision,[26]
  - OpenGL 4.1 - GLSL 4.10 Developer-friendly debug outputs, compatibility with OpenGL ES 2.0,[27]
  - OpenGL 4.2 - GLSL 4.20 Shaders with atomic counters, draw transform feedback instanced, shader packing, performance improvements
  - OpenGL 4.3 - GLSL 4.30 Compute shaders leveraging GPU parallelism, shader storage buffer objects, high-quality ETC2/EAC texture compression, increased memory security, a multi-application robustness extension, compatibility with OpenGL ES 3.0,[28]
  - OpenGL 4.4 - GLSL 4.40 Buffer Placement Control, Efficient Asynchronous Queries, Shader Variable Layout, Efficient Multiple Object Binding, Streamlined Porting of Direct3D applications, Bindless Texture Extension, Sparse Texture Extension,[29]
  - OpenGL 4.5 - GLSL 4.50 Direct State Access (DSA), Flush Control, Robustness, OpenGL ES 3.1 API and shader compatibility, DX11 emulation features
  - OpenGL 4.6 - GLSL 4.60 More efficient geometry processing and shader execution, more information, no error context, polygon offset clamp, SPIR-V, anisotropic filtering

Other Graphics APIs

- Direct3D
  - Proprietary Microsoft Windows 3D graphics API
- Vulkan
  - New cross-platform 3D graphics and compute API by Khronos group
- OpenGL ES
  - OpenGL for Embedded Systems
- Web-based OpenGL
  - JavaScript interface for OpenGL-ES-2.x API
- …

OpenGL vs. Direct3D



Source: “OpenGL vs. Direct3D – Who is the Winner of Graphics API” by Hunter Lin, 2016.

Capability of OpenGL

- Flight Simulator

Flight Gear

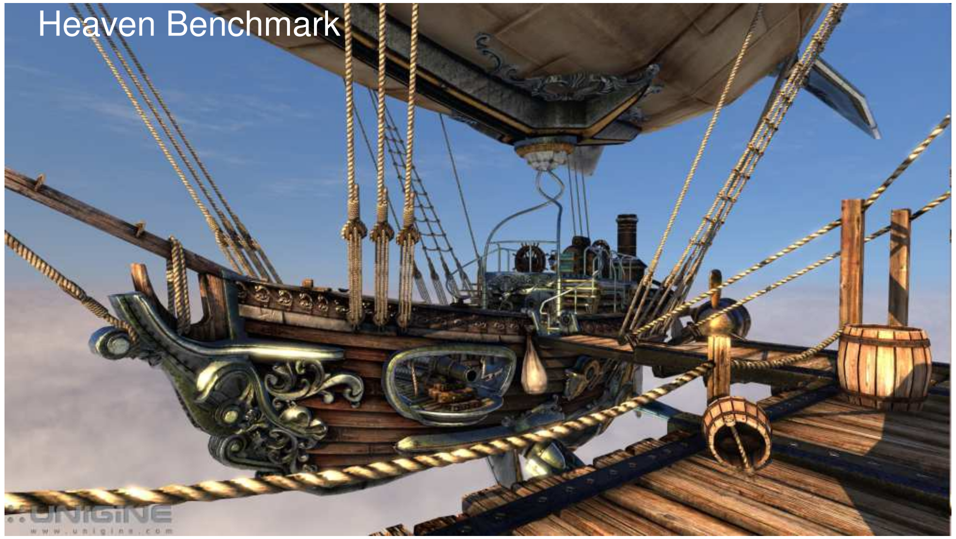
Flight Gear

X-Plane

Capability of OpenGL

- **Benchmarks**
  - **Unigine's Benchmarks**

Tropics Benchmark

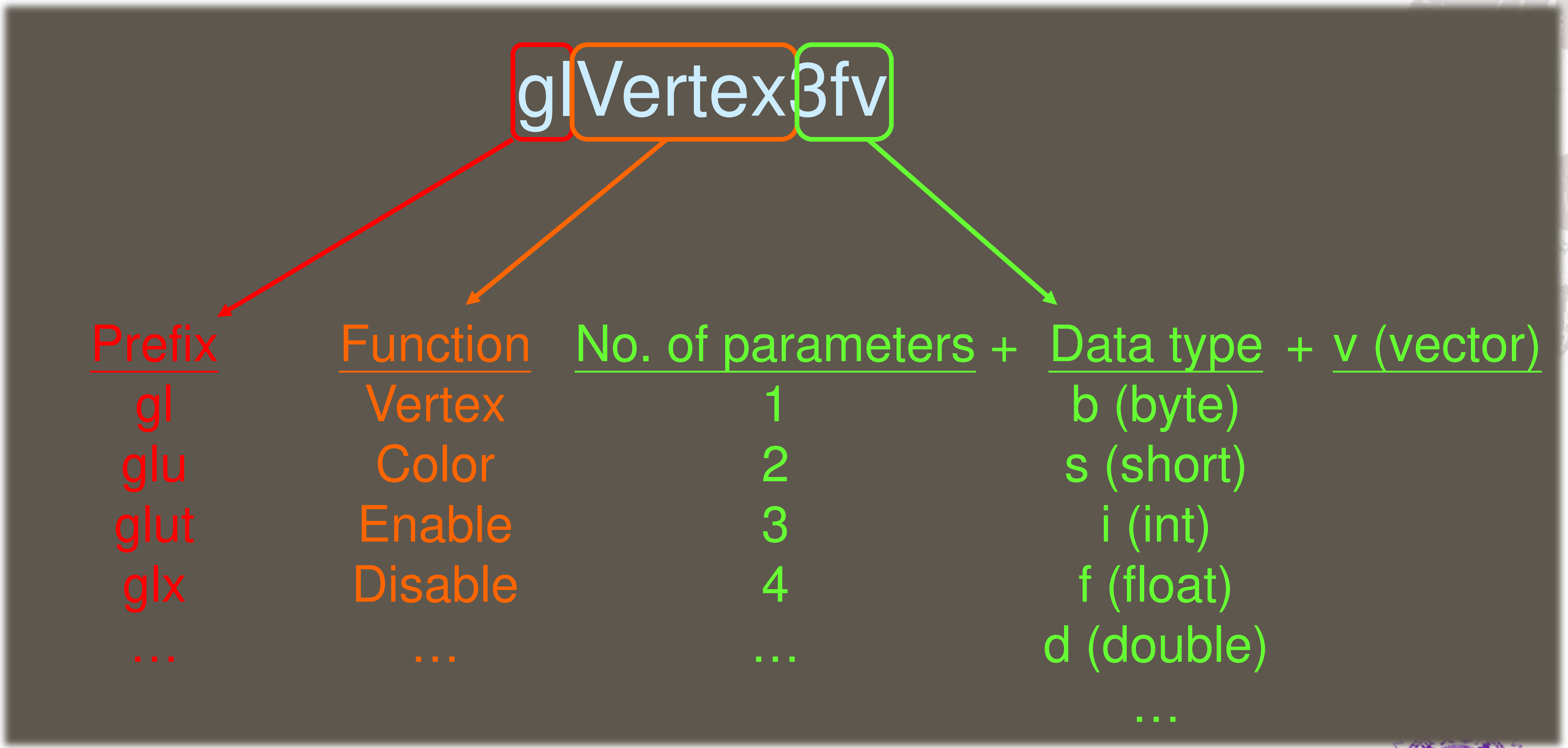

Heaven Benchmark

Sanctuary Benchmark

Why OpenGL

- Cross-platform
  - Windows, Mac OSX, Linux, …
- Portable
- Better backward compatibility
- Run on various hardware platforms
  - PC, embedded device, smart phone, game console, …
  - OpenGL ES was adopted in iOS, Android, and Symbian OS

OpenGL Command Syntax



glVertex3fv

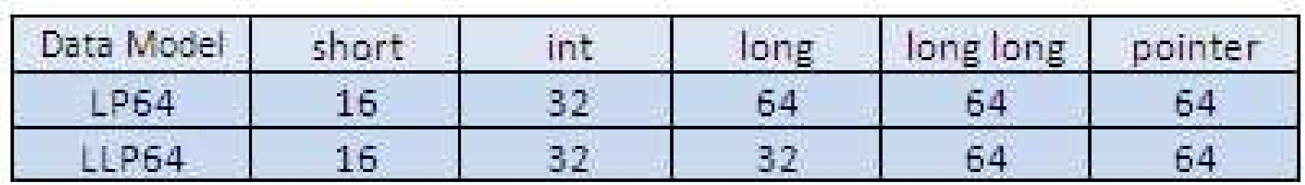
Prefix	Function	No. of parameters +	Data type	+ v (vector)
gl	Vertex	1	b (byte)	
glu	Color	2	s (short)	
glut	Enable	3	i (int)	
glx	Disable	4	f (float)	
…	…	…	d (double)	
			…	

OpenGL Data Types

Suffix	Data type	Typical C-Language Type	OpenGL Type Definition
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 Data Types

- In consideration of portability, OpenGL defined data types should be used throughout the application
  - Be careful when you mixed the usage of C++ data types and OpenGL data types



Data Model	short	int	long	long long	pointer
LP64	16	32	64	64	64
LLP64	16	32	32	64	64

Many 64-bit compilers today use the LP64 model (including Solaris, AIX, HP, Linux, Mac OS X, and IBM z/OS native compilers). Microsoft’s VC++ compiler uses the LLP64 model.

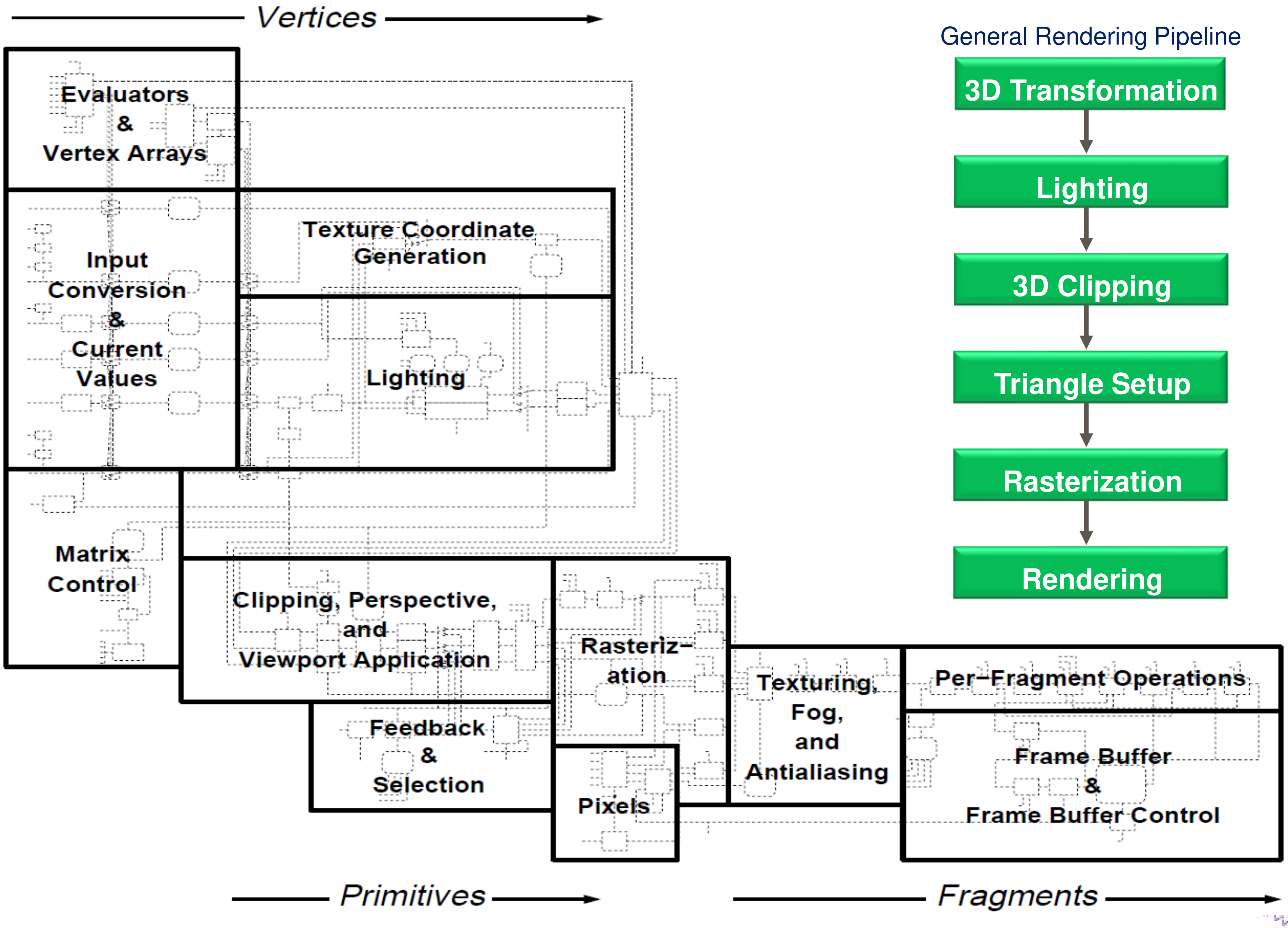
long ≠ GLint in Linux and Mac OS!!
Also, there is no “long long” in OpenGL data types

OpenGL Pipeline

Fixed Function Pipeline

Programmable Pipeline

OpenGL Pipeline (Fixed Function)


Vertices
Evaluators & Vertex Arrays
Input Conversion & Current Values
Texture Coordinate Generation
Lighting
Matrix Control
Clipping, Perspective, and Viewport Application
Rasteriz–ation
Feedback & Selection
Pixels
Texturing, Fog, and Antialiasing
Per–Fragment Operations
Frame Buffer & Frame Buffer Control
Primitives
Fragments

General Rendering Pipeline

3D Transformation

Lighting

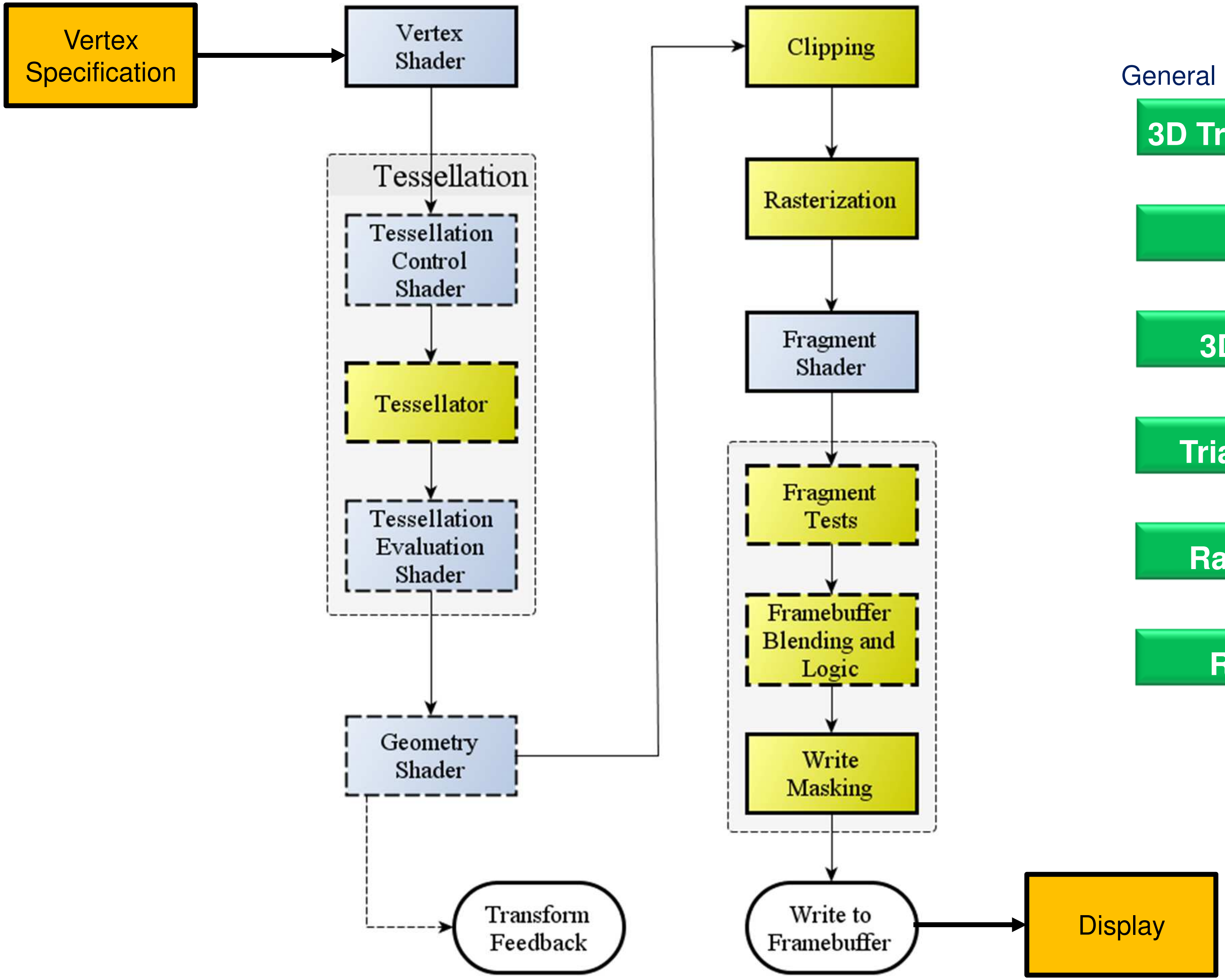
3D Clipping

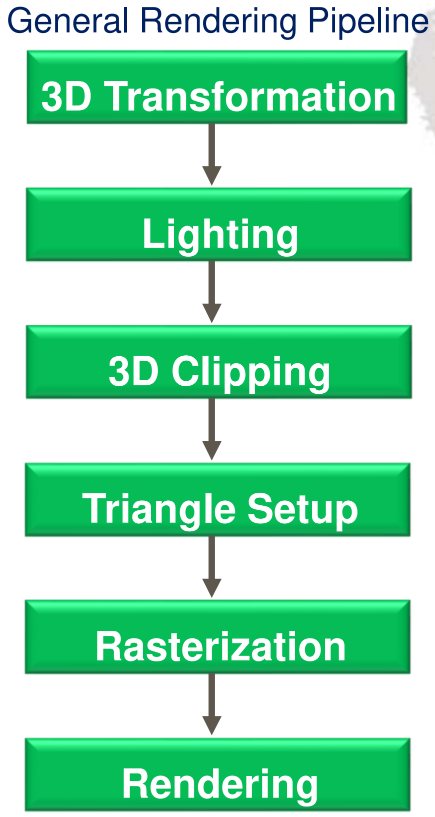
Triangle Setup

Rasterization

Rendering

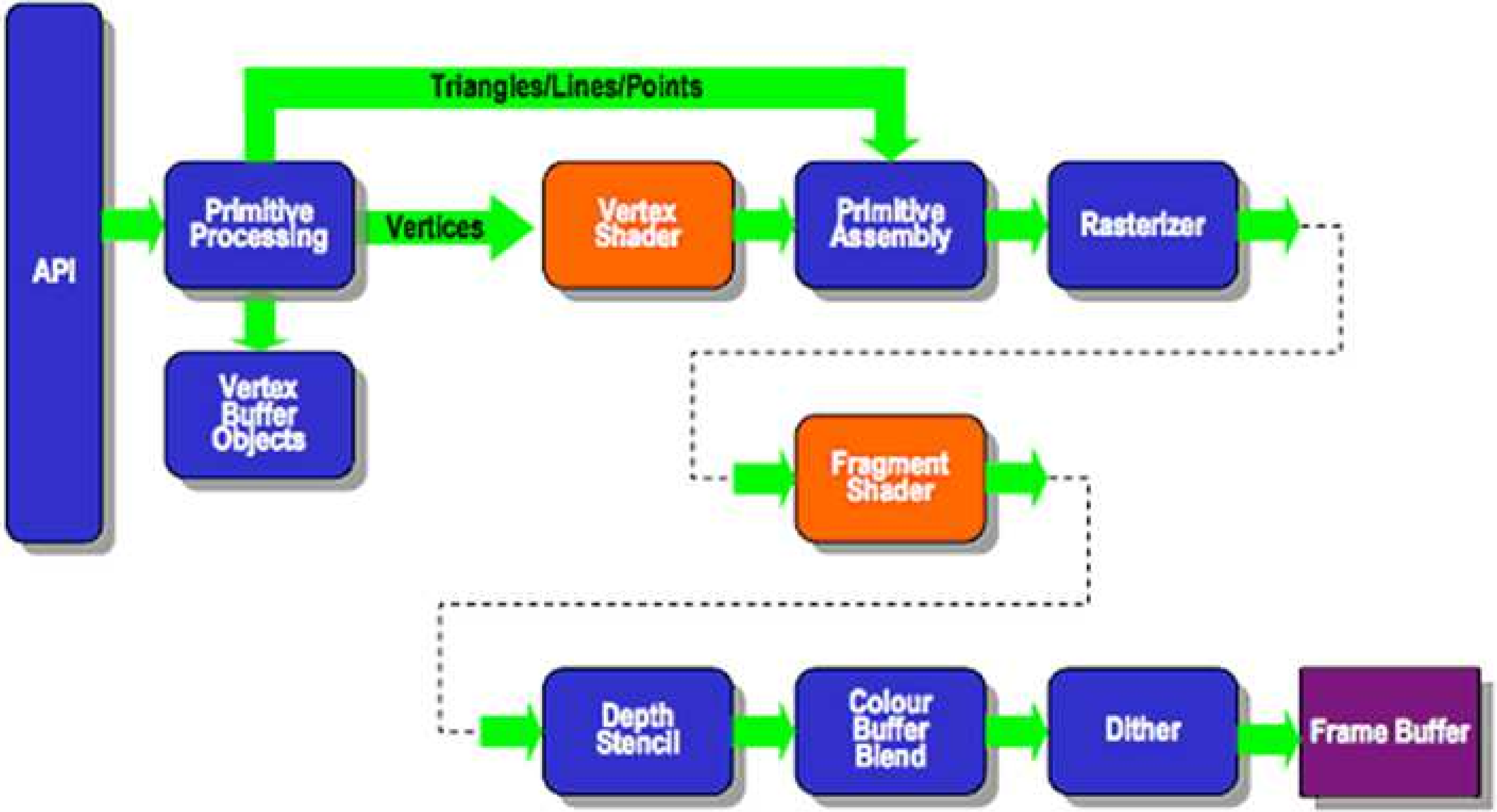
OpenGL Pipeline (Programmable)





OpenGL Pipeline

- **Difference between fixed and programmable**



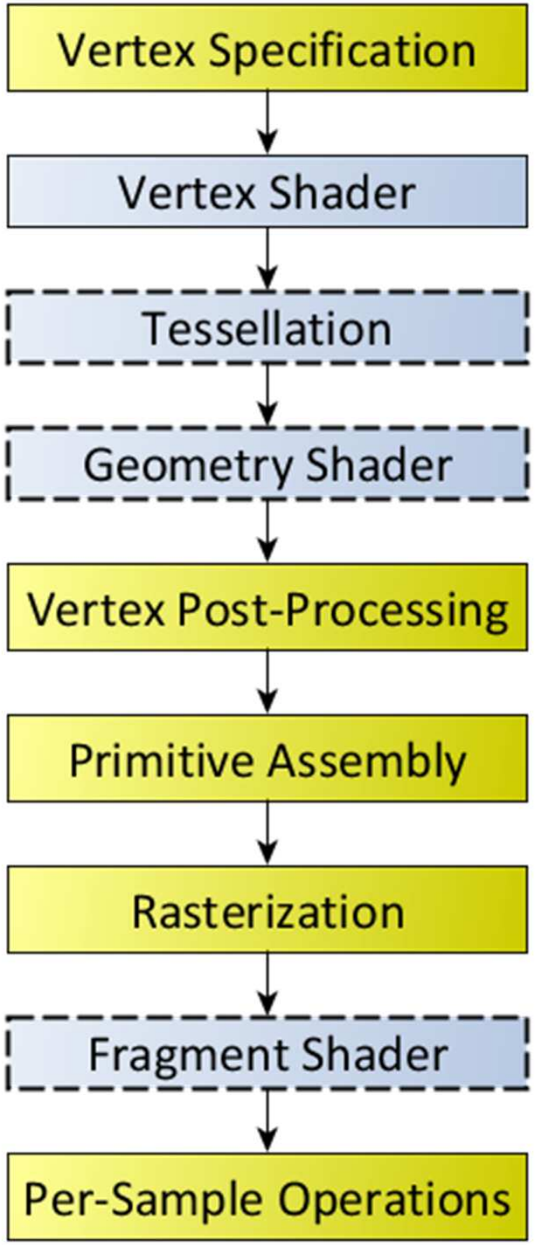
OpenGL Shaders and GLSL

Various Shaders

OpenGL Shading Language

OpenGL Shaders

- A user-defined program designed to run on some stage of a graphics processor
  - Vertex Shader
  - Tessellation
    - Tessellation Control Shader
    - Tessellation Evaluation Shader
  - Geometry Shader
  - Fragment Shader



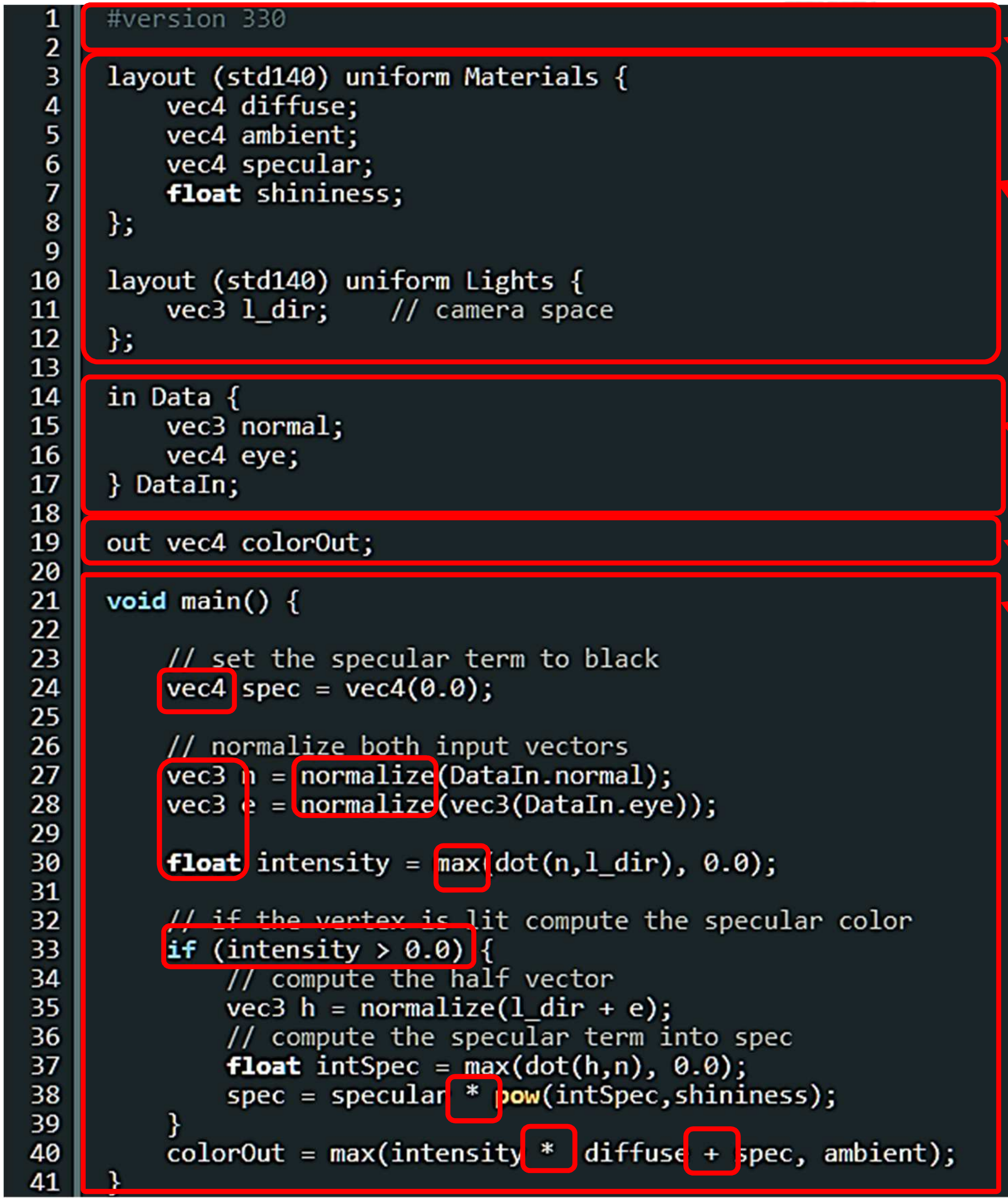
OpenGL Shading Language

- **High-Level shading Language based on C programming language**
- **Similar to NVIDIA's Cg and Microsoft's HLSL**
- **Hardware vendors will provide shader compiler to optimize for their hardware architecture**

OpenGL Shading Language

GLSL Version	OpenGL Version	Date	Shader Preprocessor
1.10.59	2.0	April 2004	#version 110
1.20.8	2.1	September 2006	#version 120
1.30.10	3.0	August 2008	#version 130
1.40.08	3.1	March 2009	#version 140
1.50.11	3.2	August 2009	#version 150
3.30.6	3.3	February 2010	#version 330
4.00.9	4.0	March 2010	#version 400
4.10.6	4.1	July 2010	#version 410
4.20.11	4.2	August 2011	#version 420
4.30.8	4.3	August 2012	#version 430
4.40	4.4	July 2013	#version 440
4.50	4.5	August 2014	#version 450
4.60	4.6	July 2017	#version 460

A GLSL Shader Example



```
#version 330

layout (std140) uniform Materials {
    vec4 diffuse;
    vec4 ambient;
    vec4 specular;
    float shininess;
};

layout (std140) uniform Lights {
    vec3 l_dir;    // camera space
};

in Data {
    vec3 normal;
    vec4 eye;
} DataIn;

out vec4 colorOut;

void main() {

    // set the specular term to black
    vec4 spec = vec4(0.0);

    // normalize both input vectors
    vec3 n = normalize(DataIn.normal);
    vec3 e = normalize(vec3(DataIn.eye));

    float intensity = max(dot(n,l_dir), 0.0);

    // if the vertex is lit compute the specular color
    if (intensity > 0.0) {
        // compute the half vector
        vec3 h = normalize(l_dir + e);
        // compute the specular term into spec
        float intSpec = max(dot(h,n), 0.0);
        spec = specular * pow(intSpec,shininess);
    }
    colorOut = max(intensity * diffuse + spec, ambient);
}
```

Version no.

Constant variables

Input Variables

Output Variable

Main()

C-like language

- Data types
- Structure
- Assignment
- Function
- Conditional branch
- Loop
- Augmented operators
  - Vector
  - Matrix
- …

OpenGL Useful Libraries

GLUT

GLEW

GLM

OpenGL Useful Libraries -- GLEW

- The OpenGL Extension Wrangler Library
- A cross-platform open-source C/C++ extension loading library
- GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform
- Current version supports
  - OpenGL 4.6
  - OpenGL, WGL, GLX extensions
  - http://glew.sourceforge.net/

OpenGL Extension

- A mechanism to provide additional functionality
  - New functions
  - New constants
  - Relax or remove restrictions on existing OpenGL functions

OpenGL Extension

- Advantages
  - Develop new functionality before new API spec is released
  - Hardware vendors can expose their new hardware features via extension first
  - Extension becomes core function (or extension) after being approved by ARB

OpenGL Extension

- Disadvantages
  - It is vendor specific before the extension becomes an ARB extension or core API
  - You must query the existence of specific extension before you use it
  - Compatibility might be an issue if an application was using a vendor specific extension
- GLEW can help in querying and loading OpenGL extensions

OpenGL Useful Libraries – glm

- Wavefront OBJ model file format reader/writer/manipulator
  - Not OpenGL Mathematics Library
- Includes routines for generating smooth normals with preservation of edges, welding redundant vertices & texture coordinate generation (spheremap and planar projections) + more.
- Used to load 3D models in OBJ file format
- http://devernay.free.fr/hacks/glm/

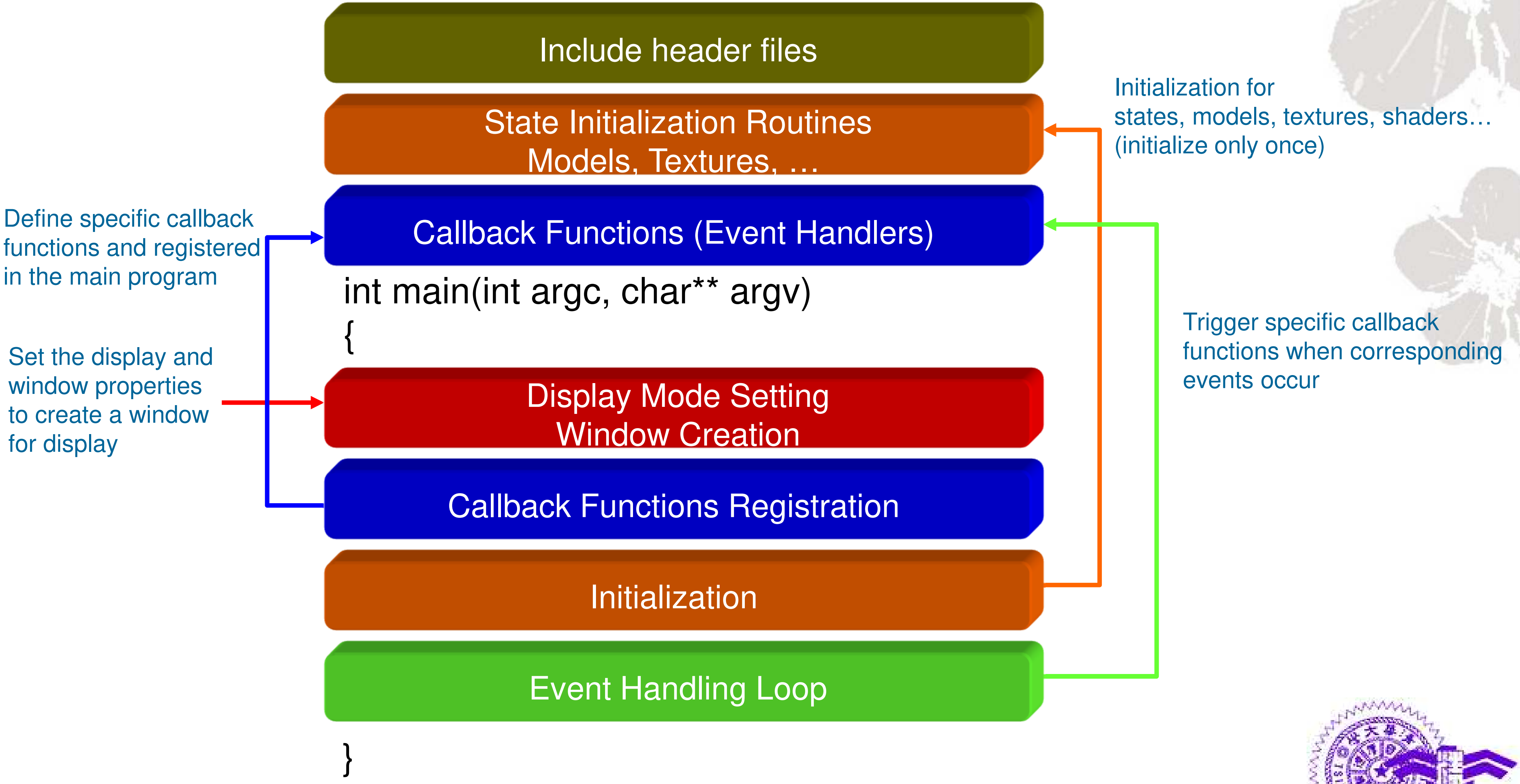
OpenGL Useful Libraries -- GLUT

- OpenGL Utility Toolkit
- A window system independent toolkit for writing OpenGL programs
- GLUT is not open source
- Alternative:
  - Freeglut: http://freeglut.sourceforge.net/
- Application frameworks provide support to control the platform's Windowing system and event handling.

OpenGL Framework

A Simple OpenGL Framework with GLUT

OpenGL Application Framework



Include header files

State Initialization Routines
Models, Textures, …

Callback Functions (Event Handlers)

```
int main(int argc, char** argv)
{
```

Display Mode Setting
Window Creation

Callback Functions Registration

Initialization

Event Handling Loop

```
}
```

Define specific callback functions and registered in the main program

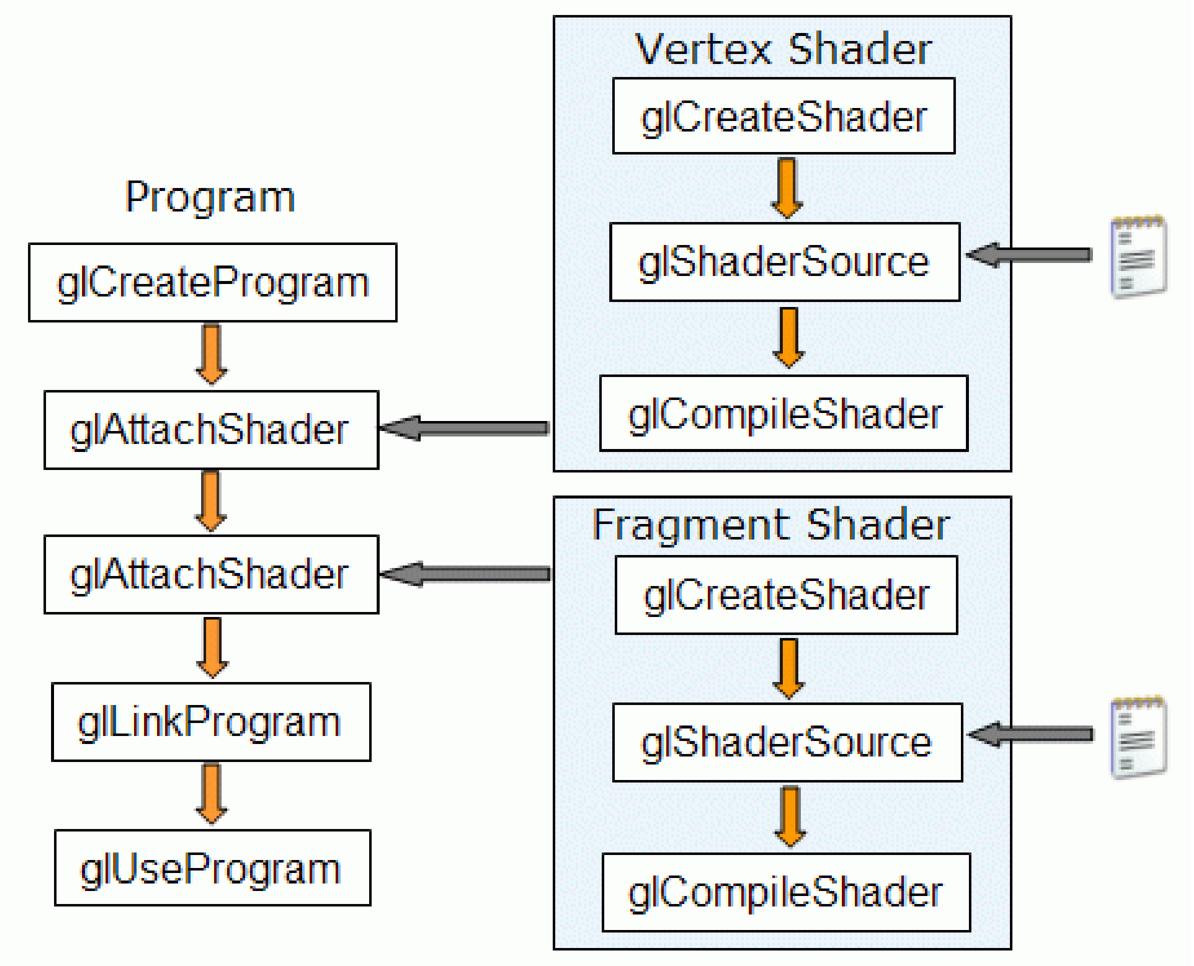
Set the display and window properties to create a window for display

Initialization for
states, models, textures, shaders…
(initialize only once)

Trigger specific callback functions when corresponding events occur

Shader Creation Flow

- Compilation – Check for syntax error
- Link – Check for Resource availability



Source: Lighthouse3d.com

OpenGL Using GLUT

◆ Example (Render a rectangle)

```
#include <freeglut/glut.h>  ←  Include header file

int main(int argc, char* argv[])
 {
                        ⎧ glutInit(&argc, argv);
Display mode setting    ⎪ glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
Window creation     →   ⎨ glutInitWindowSize(800, 600);
                        ⎪ glutInitWindowPosition(0, 0);
                        ⎩ glutCreateWindow("GLRect");

      glutDisplayFunc(RenderScene);  }  ←  Callback functions registration
      glutReshapeFunc(ChangeSize);   }

      SetupRC();  ←  State initialization

      glutMainLoop();  ←  Event handling loop

    return 0;
  }
```

GLUT functions

- `glutInit` **is used to initialize the GLUT library**
- `glutInitDisplayMode` **set the initial display mode with color model and various buffer modes**
- `glutWindowSize` **set the window size in pixels**
- `glutCreateWindow` **create window**
- `glutDisplayFunc` **set the display callback**
- `glutReshpeFunc` **set the reshape callback**
- `glutMainLoop` **enter the GLUT event processing loop**

OpenGL Using GLUT

- Example (Render a rectangle)
  - State Initialization

```
// Setup the rendering state
void SetupRC(void)
{
    // Set clear color to blue
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}
```

R G B A

Set the clear color
but not yet do the clear operation

OpenGL Using GLUT

- Example (Render a rectangle)
  - Display callback function

```
// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Set current drawing color to red
    //         R    G     B
    glColor3f(1.0f, 0.0f, 0.0f);

    // Draw a filled rectangle with current color
    glRectf(-25.0f, 25.0f, 25.0f, -25.0f);

    // Flush drawing commands
    glFlush();
}
```

OpenGL Using GLUT

- Example (Render a rectangle)
  - Equivalent functions to glRectf(…)

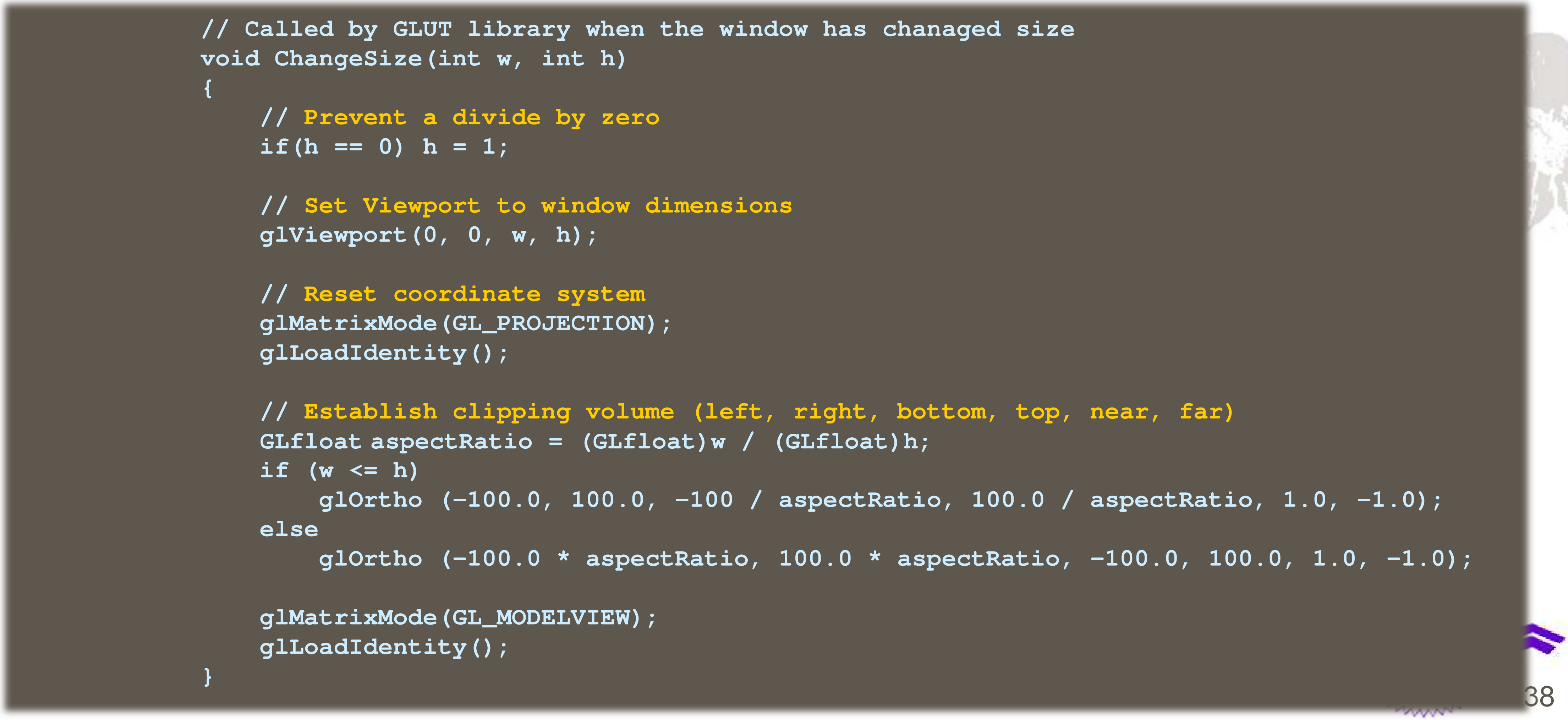
```
// Called to draw scene
void RenderScene(void)
{
    …

    // Draw a filled rectangle with current color
    // glRectf(-25.0f, 25.0f, 25.0f, -25.0f);
    glBegin(GL_POLYGON);
        glVertex2f(-25.0f, 25.0f);
        glVertex2f(25.0f, 25.0f);
        glVertex2f(25.0f, -25.0f);
        glVertex2f(-25.0f, -25.0f);
    glEnd();

    // Flush drawing commands
    glFlush();
}
```

OpenGL Using GLUT

- Example (Render a rectangle)
  - Reshape callback function



```
// Called by GLUT library when the window has chanaged size
void ChangeSize(int w, int h)
{
    // Prevent a divide by zero
    if(h == 0) h = 1;

    // Set Viewport to window dimensions
    glViewport(0, 0, w, h);

    // Reset coordinate system
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Establish clipping volume (left, right, bottom, top, near, far)
    GLfloat aspectRatio = (GLfloat)w / (GLfloat)h;
    if (w <= h)
        glOrtho (-100.0, 100.0, -100 / aspectRatio, 100.0 / aspectRatio, 1.0, -1.0);
    else
        glOrtho (-100.0 * aspectRatio, 100.0 * aspectRatio, -100.0, 100.0, 1.0, -1.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

The display callback

- The display callback is executed when
  - Window is first opened
  - Window is reshaped
  - Window is exposed
  - Post a redisplay message
- Every GLUT program must have a display callback

OpenGL Reference Links

- Khronos OpenGL web page
  - http://www.khronos.org/opengl/

- Official OpenGL web page
  - http://www.opengl.org/

Q&A