

# **Introduction to OpenGL**

**Computer Graphics**  
**Instructor: Sungkil Lee**

# OpenGL

- **IRIS GL (Graphics Library):**

- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline approach in hardware (1982).

- **OpenGL (Open Graphics Library):**

- The success of IRIS GL led to OpenGL (1992).
- A platform-independent **rendering** API
- **Close enough to the hardware/driver** to get excellent performance
- Extensible for platform-specific features through extension mechanics
- Still, it is an industry standard for 3D graphics.

- **Originally controlled by Architectural Review Board (ARB)**

- Members includes: SGI, MS, NVIDIA, HP, Apple, 3DLabs, IBM, ...

- **Now managed by Khronos group ([www.khronos.org](http://www.khronos.org))**

- Current promoter members (board of directors)



- Active standards



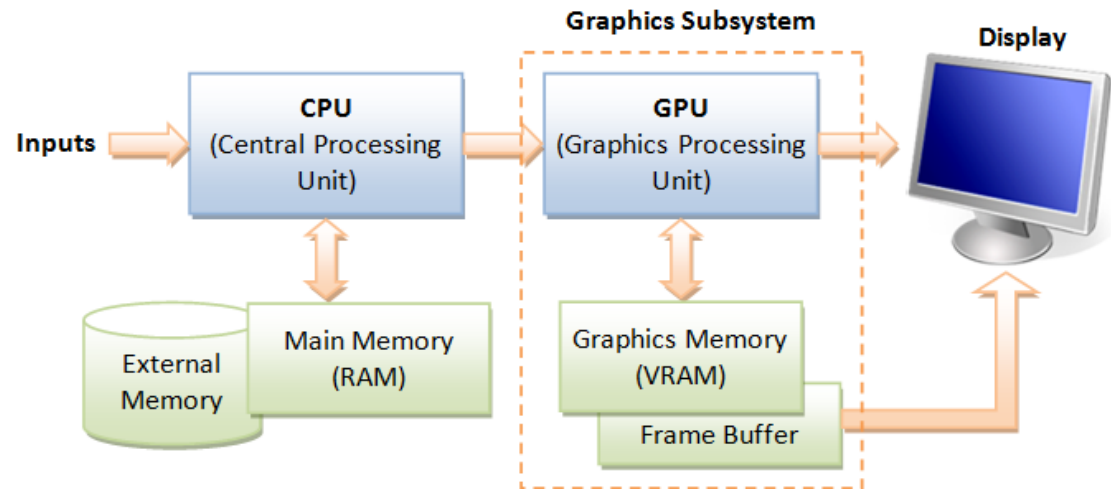
# Old-Style OpenGL

- **Through version 2.5 (more strictly up to version 3.1)**
  - was relatively *stable* and *backward-compatible*
  - The features were fixed, and impossible to modify
    - So, the pipeline is called the "*fixed-pipeline*," which simulates the basic transformation/projection (in vertex shader) and Blinn-Phong shading (in fragment shader).
- Now, many of the architecture are deprecated in modern-style OpenGL (since version 3.2).

# Graphics Hardware

- **GPU (Graphics Processing Unit)**

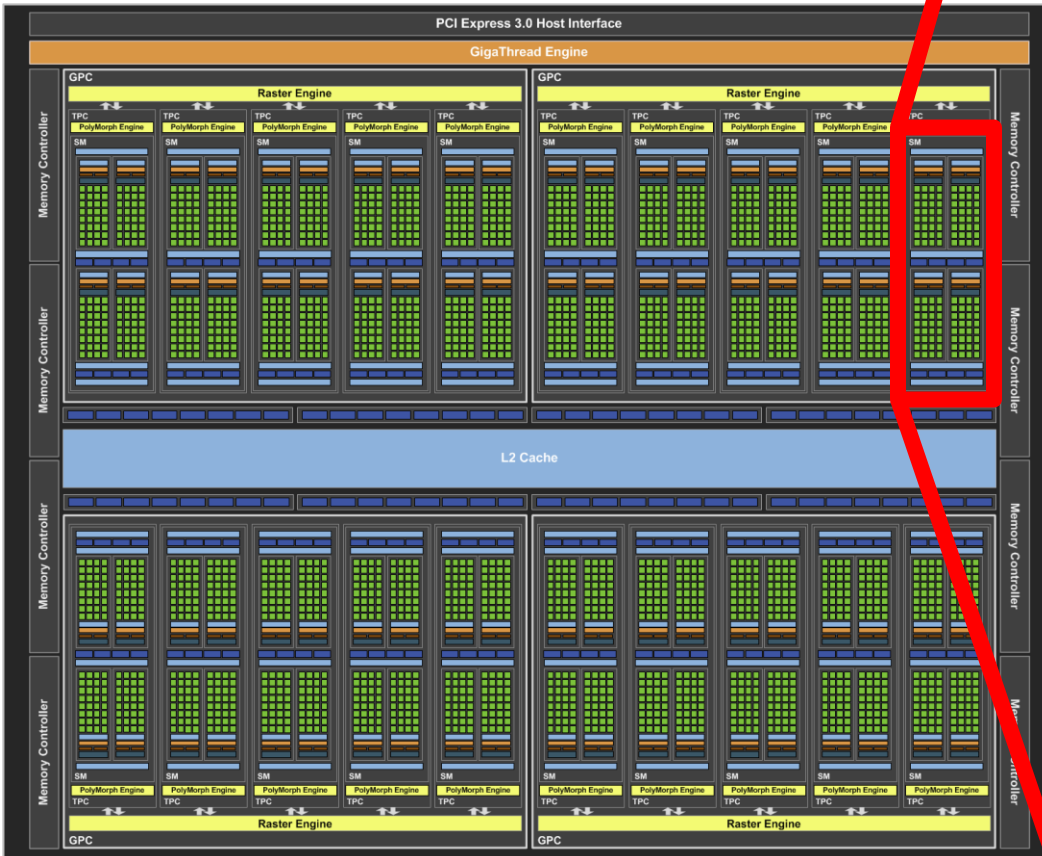
- Modern computer has dedicated Graphics Processing Unit (GPU) to produce images for the display.
- GPU is a special complete parallel computing system which has thousands of cores with its own graphics memory hierarchy (or Video RAM or VRAM).



# Graphics Hardware

- **Example: NVIDIA GTX 1080**

- 2560 CUDA Cores (1.6 GHz)
- 160 Texture units, 8GB Memory



SM



# Modern-Style OpenGL

- **Modern-style OpenGL:**

- OpenGL *since version 3.2* is called "modern-style OpenGL".

- **Using Powerful GPUs**

- Intensively using GPU rather than CPU for high performance
- *Application's job is only to send data to GPU.*
- GPU does all rendering (as well as some computing).

- **This course will focus only on modern-style OpenGL.**

- Basically, old-style OpenGL stuffs (many available on the web) are not allowed here.
- Even, I will not let you know what the old-style was to avoid confusion.

# Modern-Style OpenGL

- **User-*programmable* pipeline**

- Now, user can modify the vertex and fragment processing stages.
- This is possible by GPU programs called *shaders*.

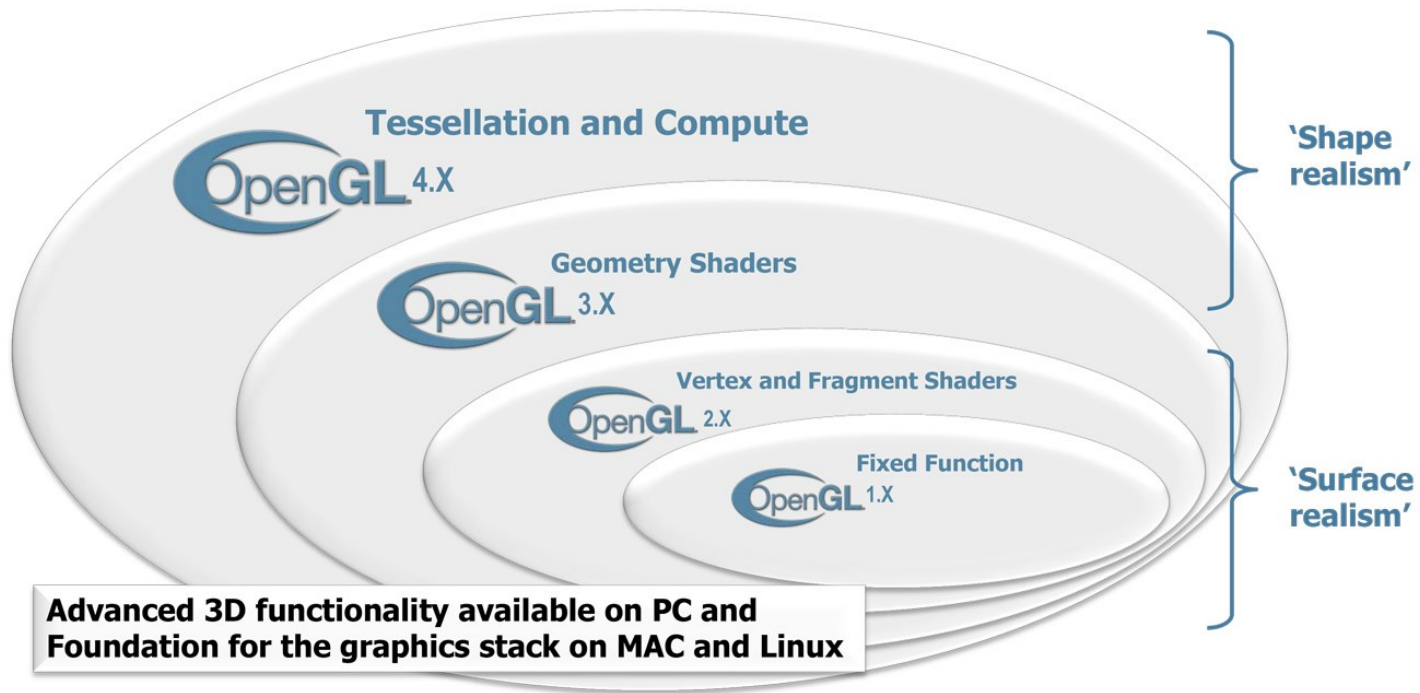
- **Totally shader-based**

- No default shaders (as fixed pipeline) available
- Each application must provide both a vertex and a fragment shader
- *Most 2.5 and previous functions deprecated.*



# Modern-Style OpenGL

- OpenGL for each Hardware generation



# **Introduction to OpenGL API**

# API?

---

- **Application Programming Interface (API)**

- A protocol intended to be used as an interface by software/hardware components to communicate with each other.
- A library that may include specification for routines, data structures, object classes, and variables.

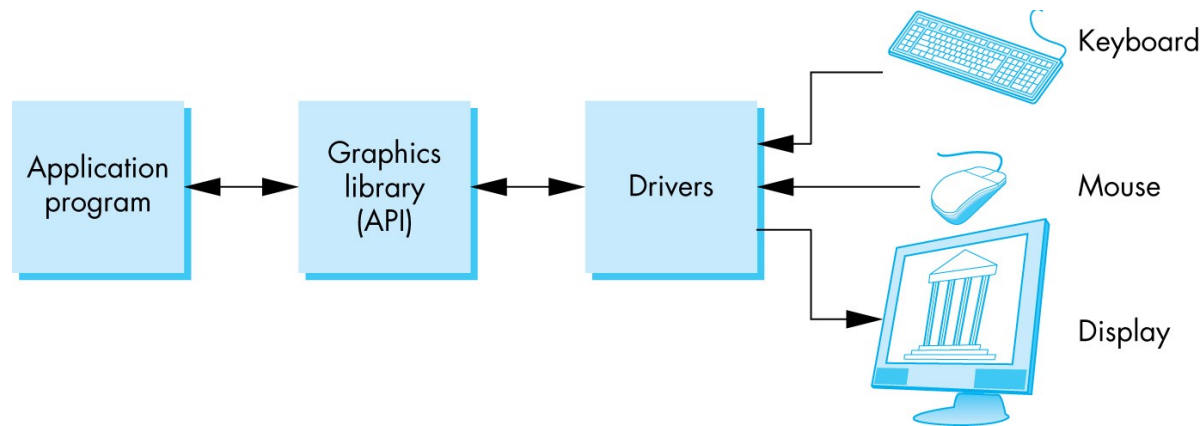
- **Examples:**

- POSIX (for Unix-like programming)
- Microsoft Windows API (for windows programming)
- C++ Standard Template Library (STL)

# OpenGL API

- **OpenGL API:**

- Allows us to interact with graphics hardware and other software platform via abstract forms of function calls.
- Cross-language, cross-platform API
  - Languages: **C**/C++, Java, C#, Fortran 90, Perl, Python, Delphi, Ada, ...
  - Platforms: Windows, Linux, Apple, ...
- Windowing support not exists for cross-platform API
  - 3rd-party libraries necessary (e.g., GLUT, freeGLUT, GLFW)



# OpenGL API: Lack of Object Orientation

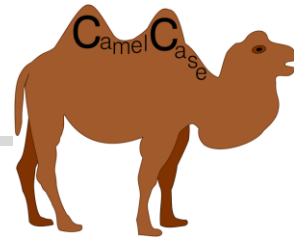
- **It's a pure "C" API.**

- There are multiple functions for a given logical function.
  - e.g., glUniform3f(), glUniform2i(), glUniform3dv()
- Underlying storage mode is the same
- Easy to create overloaded functions in C++ but an issue is efficiency.
- However, in practice, many of third-party libraries use OpenGL API with their C++ APIs.

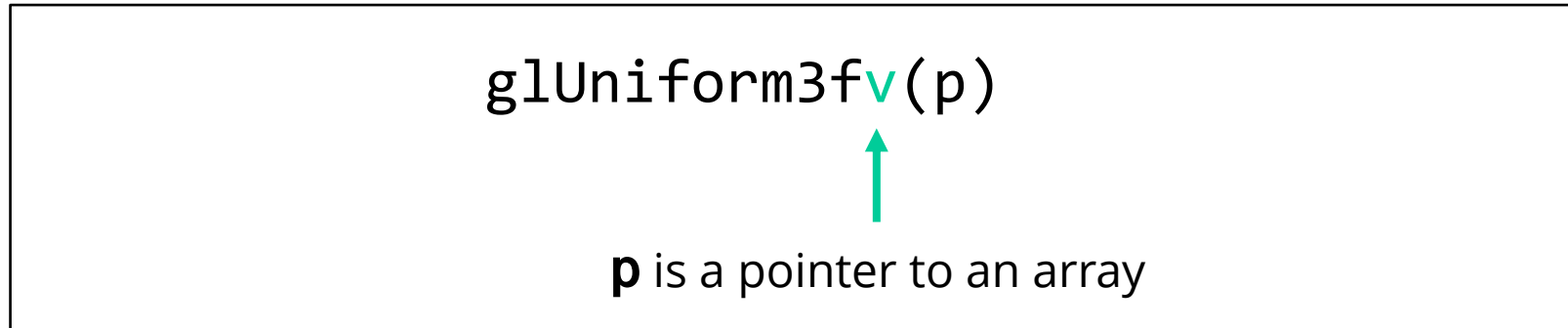
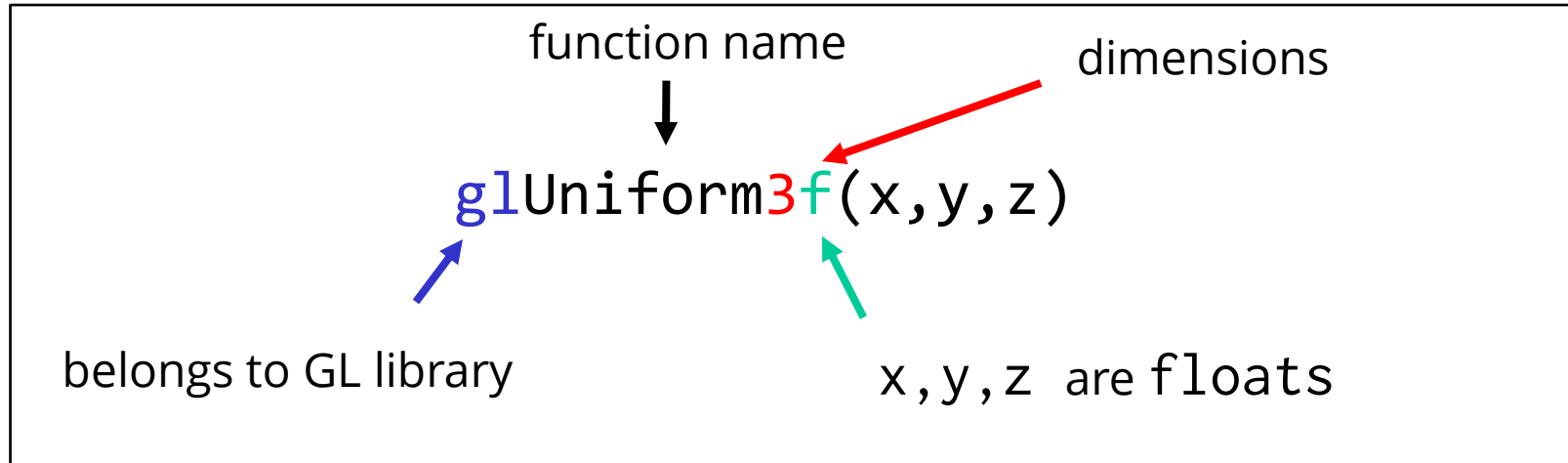
- **OpenGL is a state machine**

- Since it does not use OOP, many of the API states are stored internally.
- You can access it by query functions (e.g., glGetSomething())

# OpenGL API: Function Format



- It uses a (lower) camel-case style with variable types.



# **Prerequisites: How to Use Third-Party Libraries**

# Precompiled Binary Distribution

- **Most of binary distribution of 3rd-party libraries are structured like this:**
  - include/libname.h
  - lib/libname.lib (libname.a for Linux)
  - bin/libname.dll (libname.so for Linux)
- **DLL (or SO in Linux)?**
  - Dynamic Linking Library (Shared Object)
  - Precompiled binary objects that can be used for other programs.
- **Example:**
  - include/glfw3.h      **// have declaration of functions**
  - lib/glfw3dll.lib      **// have linking entry points of functions**
  - bin/glfw3.dll      **// have binary objects of functions**



# Installation

---

- **Assumed platform:**

- Windows + Visual Studio 2019 (32-bit builds)

- **How to get Visual Studio 2019 (Community Edition)**

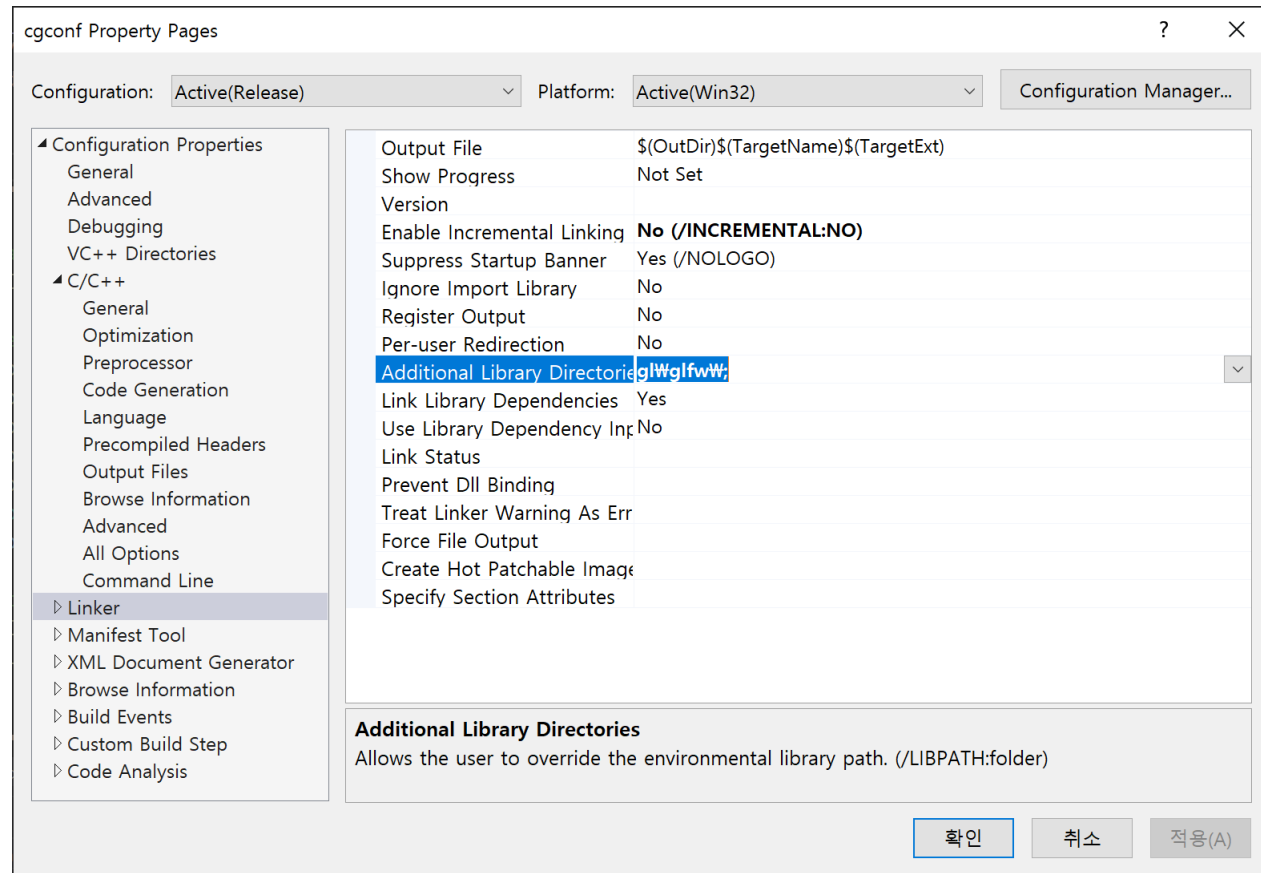
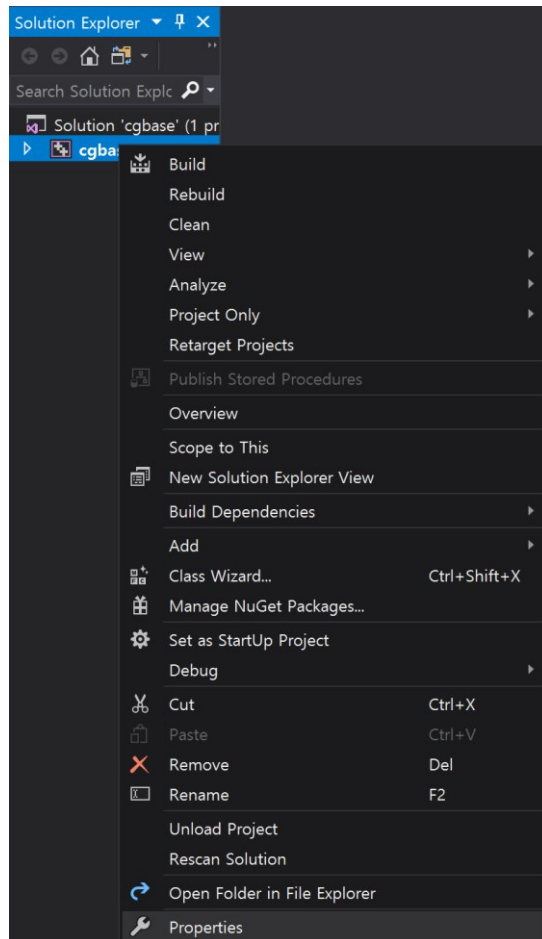
- Download a community edition of Visual Studio from Microsoft, freely available for college students.
- The community version is fully functional as a professional version does.
- Never use an illegal copy of the software.

# Installation: Local Copy

- **Global installation to VC's h/lib/bin directories is *not recommended*, because:**
  - You will not be able to re-compile the source code on other platforms.
  - You also need to distribute DLL files, too.
- **Instead, copy your library files to your project directory.**
  - Put LIBNAME.h and LIBNAME.lib to .\hello\src\gl\
  - Put LIBNAME.dll to .\hello\bin\
    - Binary files (\*.dll) are loaded at run time.
- **This local copy is *recommended*, because:**
  - You do not need to worry about a compiler system where the libraries are not installed.
  - You can easily distribute your binaries in /bin/ directory (just copy it).

# Things to Do in Visual Studio

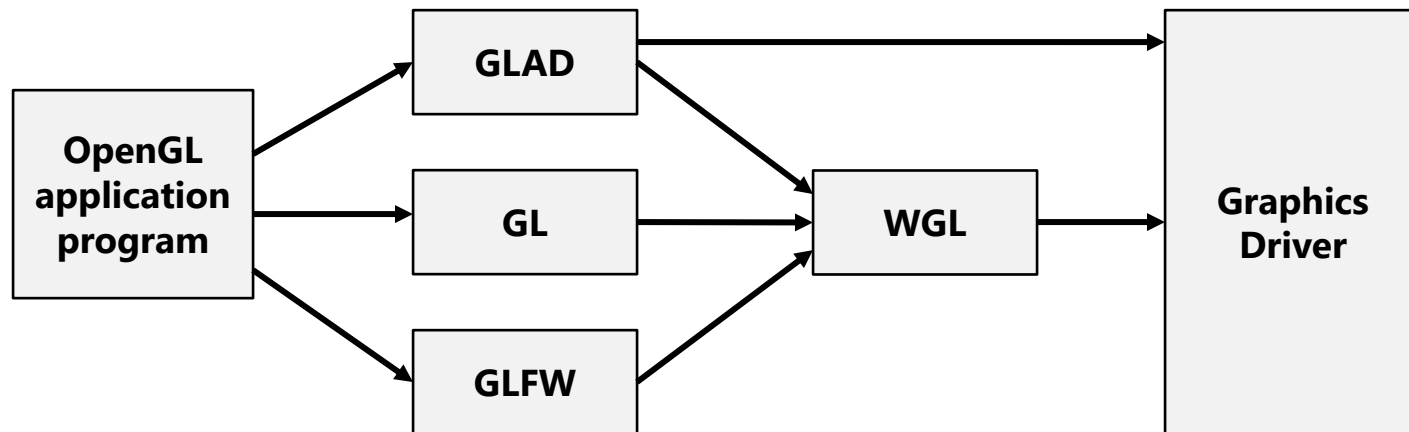
- Add additional library directories that have \*.lib files
  - In Visual Studio:



# **Installation of GL and 3rd-Party Libraries**

# OpenGL Software Organization

- OpenGL Software organization on Windows



# OpenGL Core

---

- **OpenGL core library (i.e., drivers)**

- It's already installed on your compiler systems when you are installing display drivers for your OS.
- Implementations are available through graphics drivers.

- **Linking with window system**

- WGL for Windows
- GLX for Linux

- **In your code,**

- You do not need to do anything, because GLFW and GLAD do everything instead.

- **GLFW: An Open Source, multi-platform library for**
  - creating windows with OpenGL contexts and receiving input and events.
  - written in C and has native support for Windows, OS X and many Unix-like systems using the X Window System, such as Linux and FreeBSD.
  - Modern alternative to old-school OpenGL Utility Toolkit (GLUT)
  - You can download the source and build on your own.
    - However, the course example will provide in-house pre-built binaries.
- **In your code,**
  - `#include "GL/glfw3.h"`
  - `#pragma comment( lib, "glfw3.lib" )` // or `glfw3dll.lib` for dynamic lib
  - copy "glfw3.dll" to your program binary directory (/bin/)
    - *This is only necessary when you are using dll version.*

# **GLAD:** <https://github.com/Dav1dde/glad/>

- **GLAD: OpenGL Extension Loader Generator Library**
  - Web service: <https://glad.dav1d.de/>
  - Makes it easy to access OpenGL extensions
  - Avoids having to have specific entry points in Windows code
- **In your code,**
  - #include "GL/glad.h"
  - embed "glad.c" into your source project for implementation
  - Application needs only to run `gladLoadGL()`
  - That's it; very simple C library



# **How to configure OpenGL for Linux**

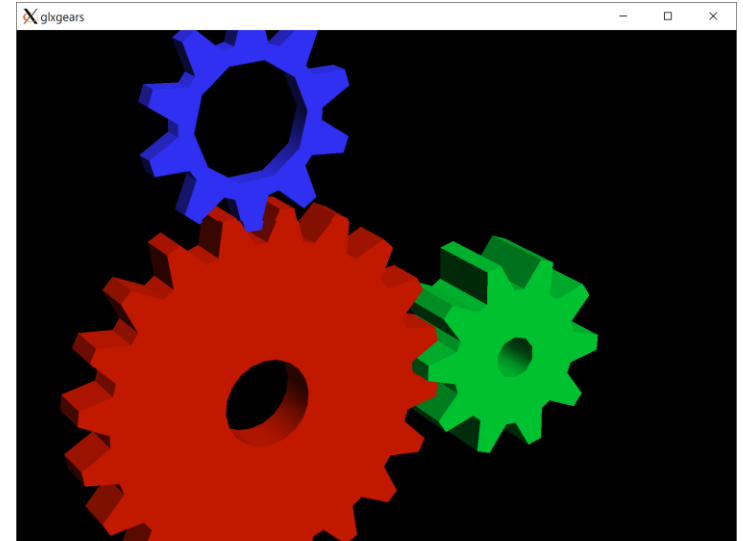
# Install MESA and GLFW

---

- **Install/update essential packages**
  - >> `sudo apt-get install -y build-essential`
  - >> `sudo apt-get update`
- **Install the latest MESA via ppa (personal package archive)**
  - >> `sudo add-apt-repository ppa:ubuntu-x-swat/updates`
  - >> `sudo apt-get dist-upgrade`
  - >> `sudo apt-get install mesa-utils`
- **Install GLFW**
  - >> `sudo apt-get install libglfw3-dev`

# Install MESA and GLFW

- **Find GL version and test glxgears**
  - `>> glxinfo | grep version`
  - `>> glxgears`
- **If you find version is 1.4 or similar,**
  - Try this in `~/.bashrc`
    - `export LIBGL_ALWAYS_INDIRECT=0`
    - this bypasses X's handling of OpenGL



# Makefile

---

- **Most of the source files are distributed with makefile.**
  - You can simply copy the source code directories to your Linux env.
- **Simply run 'make'**
  - >> cd ~/gl-01-hello/src/
  - >> make
  - >> ../bin/hello.out
- **You can find the resulting executable in /bin/**

# **Putting it all together: Hello OpenGL (cgconf)**

# Example

```
#include "gl/glad/glad.h"    // https://github.com/Dav1dde/glad
#define GLFW_INCLUDE_NONE
#include "gl/glfw/glfw3.h"   // http://www.glfw.org
#include <stdio.h>

int main()
{
    printf( "Hello OpenGL!\n\n" );

    // initialization
    if(!glfwInit()){ printf( "[error] failed in glfwInit()\n" ); return 1; }

    // create invisible window for OpenGL context
    glfwWindowHint( GLFW_VISIBLE, GL_FALSE );
    glfwWindowHint( GLFW_CONTEXT_VERSION_MAJOR, 3 );           // minimum requirement for modern OpenGL (3)
    glfwWindowHint( GLFW_CONTEXT_VERSION_MINOR, 3 );           // minimum requirement for modern OpenGL (3.3)
    glfwWindowHint( GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE );      // core profile (>=3.3) allow only forward-compatible profile
    glfwWindowHint( GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE ); // create core profile; all legacy deprecated
    GLFWwindow* window = glfwCreateWindow( 100, 100, "cgconf - Hello OpenGL", nullptr, nullptr );
    if(!window){ printf( "Failed to create GLFW window.\n" ); glfwTerminate(); return 1; }

    // make the current context and load GL extensions
    glfwMakeContextCurrent(window);
    if(!gladLoadGL()){ printf( "Failed in gladLoadGLLoader()\n" ); glfwTerminate(); return 1; }

    // check renderer and vendor
    printf( "You are using\n" );
    printf( " - OpenGL %s\n", glGetString(GL_VERSION) );
    printf( " - OpenGL Shading Language %s\n", glGetString(GL_SHADING_LANGUAGE_VERSION) );
    printf( " - GPU: %s\n", glGetString(GL_RENDERER) );
    printf( " - Vendor: %s\n", glGetString(GL_VENDOR) );

    glfwDestroyWindow(window);
    glfwTerminate();

    return 0;
}
```

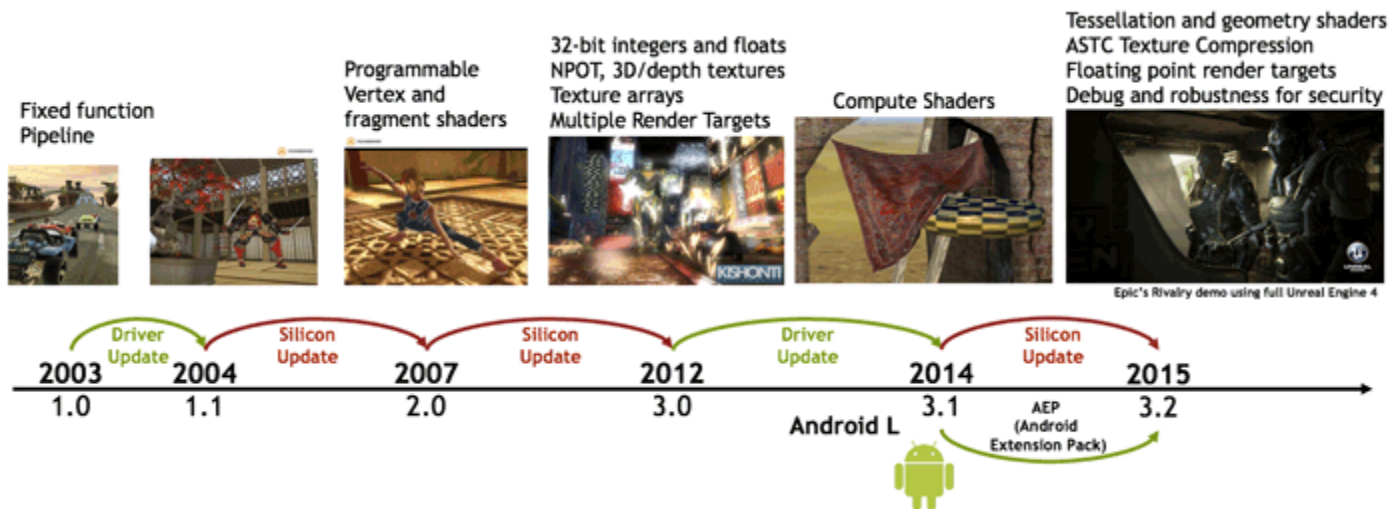
# **Advanced: OpenGL ES and Other APIs**

# OpenGL ES

- **OpenGL ES (Embedded Systems)**

- Well-defined subsets of desktop OpenGL, essential in mobile applications.
- Includes profiles for floating-point and fixed-point systems and EGL.

- **Roadmap for OpenGL ES**



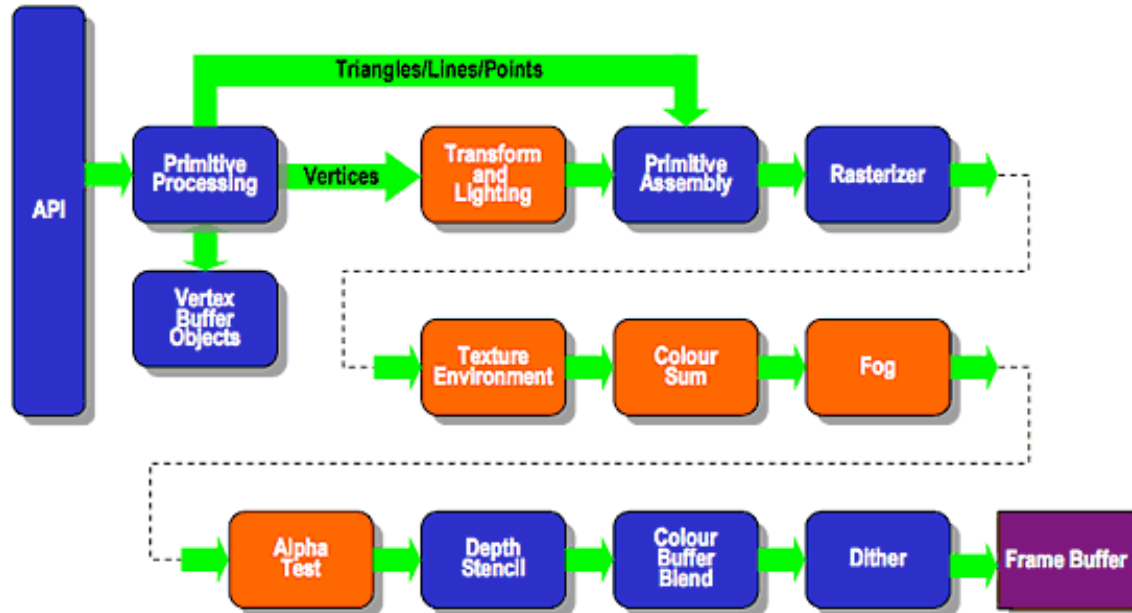


# OpenGL ES 1.x

- **OpenGL ES 1.x**

- Defined for fixed-function hardware, but now deprecated in later versions.
- OpenGL ES 1.0 is derived from OpenGL 1.3 (old style GL)
- OpenGL ES 1.1 is derived from OpenGL 1.5 (old style GL)

## Existing Fixed Function Pipeline

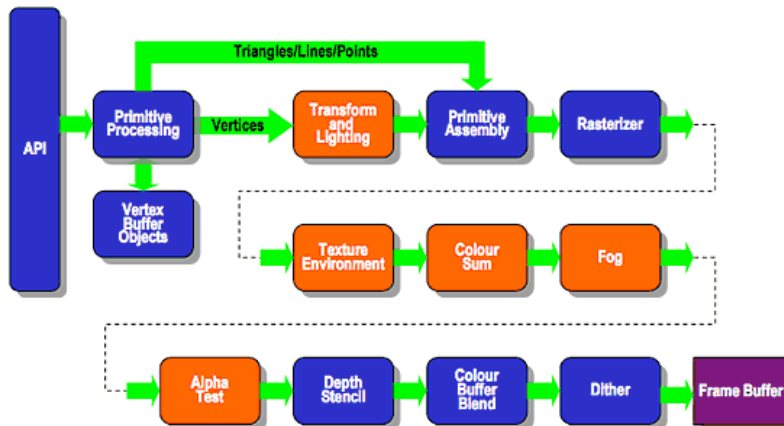


# OpenGL ES 2.0

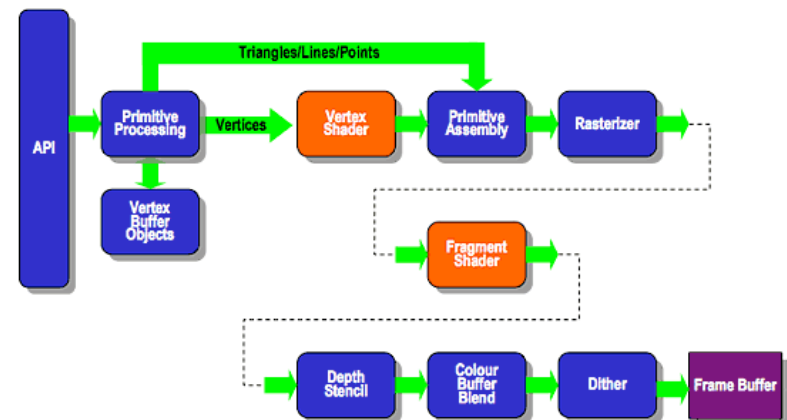
- **OpenGL ES 2**

- Defined relative to OpenGL 2.0 (old style but near to modern style)
- Programmable pipeline with vertex/fragment shaders.
- Does not support fixed-function transformation and fragment pipeline of OpenGL ES 1.x

**Existing Fixed Function Pipeline**



**ES2.0 Programmable Pipeline**



# OpenGL ES 2.0

- **OpenGL ES Shading Language 1.0 (ESSL 1.0, 2009)**
  - Adds the same shading language used in OpenGL 2.0 but adapted for embedded platforms.
  - Precision qualifier should be specified in the shader program.
- **Minimum precisions required in any platforms**
  - Vertex shader: 16-bit precision (in range  $[-2^{62}, +2^{62}]$  for fp)
  - Fragment shader: 10-bit precision (in range  $[-2^{14}, +2^{14}]$  for fp)

# OpenGL ES 2.0

- **Precision Qualifiers**

- *highp, mediump, lowp*
- range/precision:

Qualifier	Floating Point Range	Floating Point Magnitude Range	Floating Point Precision	Integer Range
<b>highp</b>	$(-2^{62}, 2^{62})$	$(2^{-62}, 2^{62})$	Relative: $2^{-16}$	$(-2^{16}, 2^{16})$
<b>mediump</b>	$(-2^{14}, 2^{14})$	$(2^{-14}, 2^{14})$	Relative: $2^{-10}$	$(-2^{10}, 2^{10})$
<b>lowp</b>	$(-2, 2)$	$(2^{-8}, 2)$	Absolute: $2^{-8}$	$(-2^8, 2^8)$

# OpenGL ES 3.x

---

- **OpenGL ES 3.x is fully compatible with OpenGL 4.3**
- **OpenGL ES 3.0**
  - Multiple render targets
  - Occlusion queries, transform feedback, instanced rendering
  - ETC2/EA texture compression
  - A new ESSL with 32-bit integers and floats
  - Textures: NPOT, floating-point, 3D, 2D array
  - Swizzle, LOD, mip level clamps, seamless cube maps, and sampler objects

# OpenGL ES 3.x

---

- **OpenGL ES 3.1**

- Computer shaders
- Indirect draw commands

- **OpenGL ES 3.2**

- Geometry and tessellation shaders
- Floating-point render targets
- ASTC (adaptive scalable texture compression)
- Enhanced blending and handling of multiple color attachments
- Advanced texture targets: texture buffers, multisample 2D array, cubemap arrays
- Debug and robustness for security

- **EGL (Khronos Native Platform Graphics Interface)**

- Interface specification between Khronos rendering APIs (e.g., OpenGL and OpenGL ES) and the underlying native windowing platform.
- Handles graphics context management, surface/buffer binding, rendering synchronization, and mixed-mode 2D and 3D rendering.
- Prior to EGL 1.2, it was OpenGL ES Native Platform Graphics Interface.

- **Similar interfaces**

- WGL: for windows
- CGL (Core OpenGL): for OS X
- GLX: for X11
- WSI (Window System Interface) : for Vulkan

# Other APIs

---

- **WebGL**

- Javascript implementation of OpenGL ES 2.0
- Supported on most modern browsers (e.g., Chrome, FireFox, Safari, Opera)

- **Vulkan**

- The next-generation graphics API: the successor of OpenGL 4
- Designed for high-performance graphics and computing with less driver overhead
- Useful for low-level graphics developers