OpenGL Background

Scope

- What Is OpenGL?
  - Evolution
  - Directx vs OpenGL
- How Does OpenGL Work?
  - Generic Implementations
  - Hardware Implementations
  - The Pipeline

Early History: IFIPS & GKS

- IFIPS (International Federation for Information Processing Societies)(1973) formed two committees to come up with a standard graphics API
    - Graphical Kernel System (GKS) -2D
    - Core - Both 2D and 3D
    - GKS adopted as IS0 and later ANSI standard (1980s)
- GKS not easily extended to 3D (GKS-3D)
    - Far behind hardware development

PHIGS & X

- Programmers Hierarchical Graphics System (PHIGS)
  - Arose from CAD community
  - Database model with retained graphics (structures)
- X Window System
  - DEC/MIT effort
  - Client-server architecture with graphics

SGI and GL

- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982)
- To access the system, application programmers used a library called GL
- With GL, it was relatively simple to program three dimensional interactive applications

OpenGL

- The success of GL lead to OpenGL (1992), a platform-independent API that was
  - Easy to use
  - Close enough to the hardware to get excellent performance
  - Focus on rendering
  - Omitted windowing and input to avoid window system dependencies

Directx

- The first version of DirectX was released in September 1995 as the Windows Games SDK.
    - It was the Win32 replacement for the DCI and WinG APIs for Windows 3.1
- Allowed all versions of Microsoft Windows, starting with Windows 95, to incorporate high-performance multimedia
- DirectX 2.0 became a component of Windows itself with the releases of Windows 95 OSR2 and Windows NT 4.0 in mid-1996.
- Current Version - Directx 11 (For Windows 8)

What Is OpenGL?

- OpenGL is strictly defined as “a software interface to graphics hardware.” In essence, it is a 3D graphics and modeling library that is highly portable and very fast

- OpenGL is not a programming language like C or C++. It is more like the C runtime library, which provides some prepackaged functionality

- OpenGL is intended for use with computer hardware that is designed and optimized for the display and manipulation of 3D graphics

OpenGL vs Directx

- Motivation:
  - OpenGL is designed to be a 3D accelerated hardware rendering system that may be emulated in software. Expects the implementation of OpenGL to manage hardware resources.
  - Direct3D is designed to virtualize 3D hardware interface, expects the application to manage hardware resources
- Design:
  - OpenGL is a much more general purpose 3D API, so it provides features that aren't necessarily exclusive towards any particular kind of user.
  - DirectX was an API designed for low-level, high-performance hardware access for the purpose of game development.
- Implementation :
  - OpenGL drivers consequently more complex to implement that Directx Drivers. However, The two APIs provide nearly the same level of functionality
- Usage:
  - OpenGL- professional graphics market: computer animated movies, and scientific visualisation
  - Directx - Games

How Does OpenGL Work?

- OpenGL is a procedural rather than a descriptive graphics API.
- Instead of describing the scene and how it should appear, the programmer actually prescribes the steps necessary to achieve a certain appearance or effect.
- These “steps” involve calls to the many OpenGL commands.
- These commands are used to draw graphics primitives such as points, lines, and polygons in three dimensions.
- In addition, OpenGL supports lighting and shading, texture mapping, blending, transparency, animation, and many other special effects and capabilities.

How Does OpenGL Work?

- OpenGL does not include any functions for window management, user interaction, or file I/O.

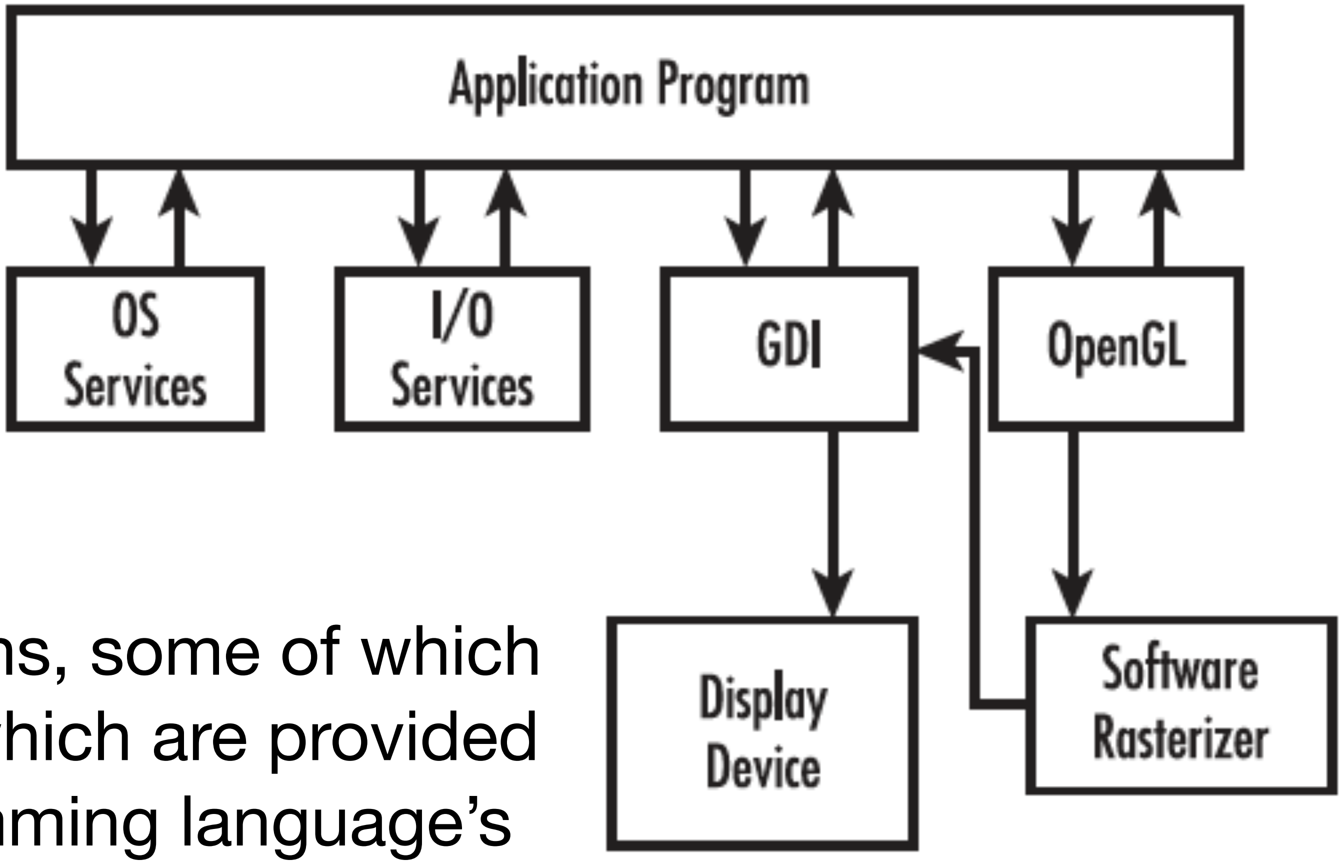
- Each host environment (such as Mac OS X or Microsoft Windows) has its own functions for this purpose and is responsible for implementing some means of handing over to OpenGL the drawing control of a window.

- There is no “OpenGL file format” for models or virtual environments. Programmers construct these environments to suit their own high-level needs and then carefully program them using the lower-level OpenGL commands.

Generic vs Hardware Implementations

- A generic implementation is a software implementation.
- Hardware implementations are created for a specific hardware device, such as a graphics card or game console.
- A generic implementation can technically run just about anywhere as long as the system can display the generated graphics image.
- A software implementation of OpenGL takes graphics requests from an application and constructs (rasterizes) a color image of the 3D graphics.
- http://www.mesa3d.org/

Generic Implementations



- The typical program calls many functions, some of which the programmer creates and some of which are provided by the operating system or the programming language’s runtime library.
- Windows applications wanting to create output onscreen usually call a Windows API called the graphics device interface (GDI).
- The GDI contains methods that allow you to write text in a window, draw simple 2D lines etc.

Generic Implementations

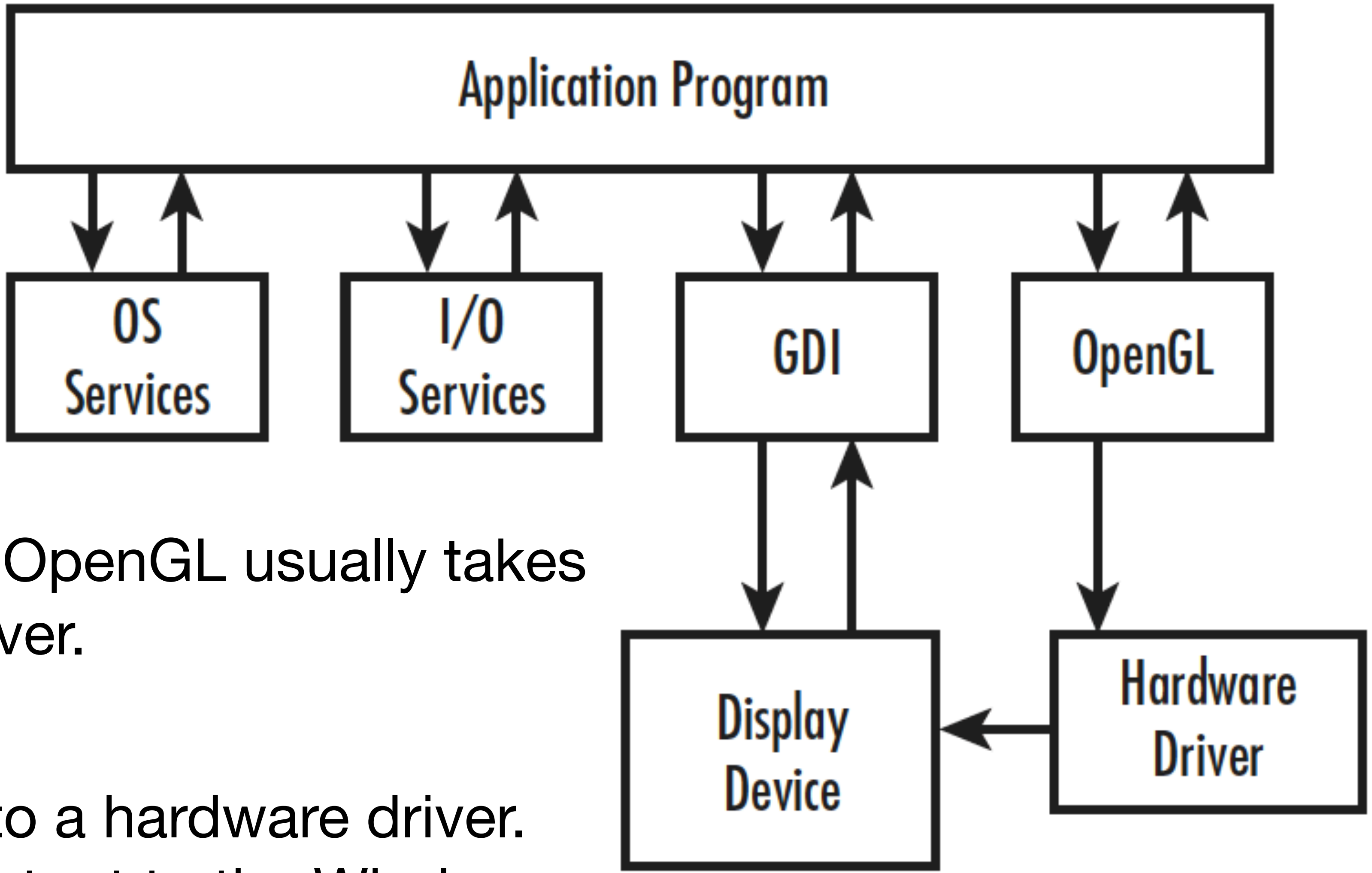
- Microsoft has shipped its software implementation with every version of Windows NT since version 3.5 and Windows 95 (Service Release 2 and later). Windows 2000 and XP also contain support for a generic implementation of OpenGL.

- During the height of the so-called “API Wars,” SGI released a software implementation of OpenGL for Windows that greatly outperformed Microsoft’s implementation.

- MESA 3D is another “unofficial” OpenGL software implementation that is widely supported in the open-source community.

- Mesa 3D is not an OpenGL license, so it is an “OpenGL work-alike” rather than an official implementation

Hardware Implementations



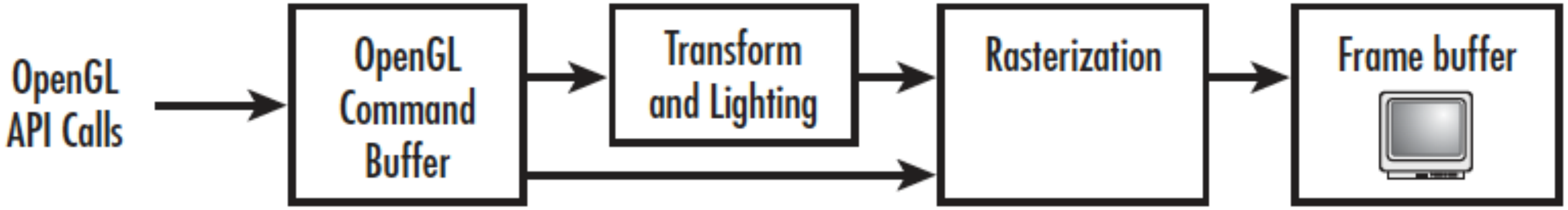
- A hardware implementation of OpenGL usually takes the form of a graphics card driver.

- OpenGL API calls are passed to a hardware driver. This driver does not pass its output to the Windows GDI for display; the driver interfaces directly with the graphics display hardware.

- A hardware implementation is often referred to as an accelerated implementation because hardware-assisted 3D graphics usually far outperform software-only implementations.

The Pipeline

- The word pipeline is used to describe a process that can take two or more distinct stages or steps.
- As an application makes OpenGL API function calls, the commands are placed in a command buffer.
- This buffer eventually fills with commands, vertex data, texture data, and so on.
- When the buffer is flushed, either programmatically or by the driver's design, the commands and data are passed to the next stage in the pipeline.

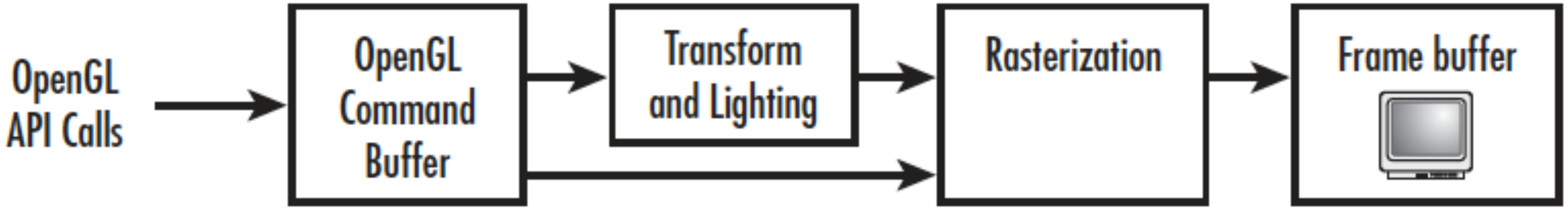


The Pipeline

- “Transform and lighting” to be a mathematically intensive stage where points used to describe an object’s geometry are recalculated for the given object’s location and orientation.

- The rasterizer actually creates the color image from the geometric, color, and texture data.

- The image is then placed in the frame buffer. The frame buffer is the memory of the graphics display device, which means the image is displayed on your screen.



The Pipeline

- Early OpenGL hardware accelerators were nothing more than fast rasterizers.. The host system's CPU did transform and lighting in a software implementation of that portion of the pipeline.
- Higher-end (more expensive) accelerators had transform and lighting on the graphics accelerator -> higher performance.
- Even most low-end consumer hardware today has the transform and lighting stage in hardware.
- The net effect of this arrangement is that higher detailed models and more complex graphics are possible at real-time rendering rates on inexpensive consumer hardware.
- Games and applications developers can capitalize on this effect, yielding far more detailed and visually rich environments.