Introduction to OpenGL

Ioannis Tsiombikas

1 Introduction

This article's purpose is to familiarize the reader with OpenGL. The idea is not to provide a comprehensive coverage of the whole thing, nor examine its history and evolution. Rather, the idea is to explain the philosophy of OpenGL, and how to use it for most common tasks. Finally, this article will not attempt to present the fundamendal principles of computer graphics from scratch. The reader is supposed to understand the operation of a rasterization pipeline, transformations, and lighting, although reminders will be included for the more obscure and easily forgotten details.

1.1 What is OpenGL?

But first, an explanation is in order about the "nature" of OpenGL. The name stands for "Open Graphics Language," however it's certainly not a programming language. In reality it's actually a "library," (i.e. `libGL.so`). But a library is an actual piece of code, and the name "OpenGL" doesn't really refer to any particular piece of code, rather to a "specification" of the programming interface of such a library. So, it's more commonly called an "API" (Application Programming Interface), and that's probably the best way to call it.

1.2 OpenGL Design Philosophy

OpenGL is a state machine. Allmost every function's role is to modify or query various aspects of the "current state." This is very fundamendal in understanding how OpenGL works. Let's consider for example the simple operation of clearing the entire framebuffer to a single color: first the "clear color" state must be set, and then the "clear" function must be called in order to perform the action. This theme of setting the appropriate state and performing the action based on the current state, is carried along the whole API, so it's a very orthogonal and predictable design.

1.3 OpenGL – Window System Integration

In most cases, OpenGL is used in conjunction with a window system. OpenGL is a pure graphics API, it doesn't contain any provisions for creating windows, or handling events. This is a good thing, because otherwise it would be tied with a specific window system, however it also means that usually a full OpenGL program must be able to handle the interaction with the window system and the task of connecting OpenGL to it.

The details of these window-system specific tasks are beyond the scope of this article. In a few words, an OpenGL "context" is created, which keeps the whole OpenGL state. This context then can be attached to a "window" or other "drawable," in order for OpenGL to be able to draw on it.

The seperation into contexts implies that a program may create multiple contexts attached to different drawables, and use them in turn. At any given time instance, there is only one active OpenGL context per process. The active context can be set by the application at any time, and all further OpenGL calls apply to that context. The ability to share OpenGL resources between contexts is also usually provided.

Also note, that due to the fact that OpenGL operation relies on global state, only one thread at a time can issue OpenGL calls within a process.

1.4 OpenGL Client/Server Architecture

OpenGL was originally designed to be used with the "X Window System" on Silicon Graphics UNIX workstations. And like the X Window System, OpenGL is also designed in a client-server architecture. The OpenGL server is usually attached to a display device, while the client which is essentially the application program using the OpenGL library, may be on the same machine as the server, or it may be on the other side of the world. In fact, an OpenGL application doesn't need to be modified or even recompiled to run remotely, the location of the OpenGL server is transparent to the application. Of course this concept of network transparency is not supported when using OpenGL in conjunction with a window system that doesn't support the concept of network transparency itself.

2 OpenGL Operation

Drawing in OpenGL is performed with rasterization of various "primitives." These primitives include: *points*, *lines*, and *polygons*.
to be continued...