Equalizer Programming Guide

Stefan Eilemann*

INCOMPLETE

Version 0.1, August 19, 2007

Version	Date	Changes
0.1	Aug 19, 2007	outlined the basic concepts

http://www.equalizergraphics.com/documents/Developer/ProgrammingGuide.pdf

*eile@eyescale.ch

Contents

1 Introduction

Equalizer provides a framework for the development of parallel OpenGL applications. Equalizer-based applications can run a single shared-memory system with multiple graphics cards (GPU's) or on a distributed graphics cluster. This Programming Guide introduces the programming interface using the eqPly example shipped with Equalizer.

Any questions related to Equalizer programming and this Programming Guide should be directed to the eq-dev mailing list[1].

2 Getting Started

2.1 Compiling and running eqPly

A prerequisite for this Programming Guide is a working eqPly example. The Quickstart Guide[2] explains how to run it. eqPly can also be executed without a server, which simplifies the development cycle. In this case it will be configured to use one window.

2.2 Equalizer Processes

2.2.1 The Server

An Equalizer server is responsible for managing one visualization system[3]. Currently it is only useful for running one application at a time, but it will be extended to support multiple applications concurrently and efficiently on one system. The server controls and potentially launches the application's rendering clients.

2.2.2 The Application

The application connects to a server, which chooses a configuration for the application. It provides a render client, to be launched by the server. The application reacts on events and controls the rendering.

2.2.3 The Render Client

The render client implements the rendering part of an application. It is passive, and receives all its rendering tasks from the server. The tasks are executed by calling the appropriate task methods (see 3.1).

The application might be a rendering client, in which case it can also contribute to the rendering. It can choose not to implement any render client-related code, in which case it is reduced to be the application's 'master' process without any OpenGL windows.

The rendering client can be the same executable as the application, as is the case with eqPly. Real-world applications often implement a separate, light-weight rendering client.

3 The Programming Interface

Equalizer uses a C++ programming interface. The API is minimally invasive, that is, Equalizer imposes only the minimal, natural execution framework upon the

[1] see http://www.equalizergraphics.com/lists.html
[2] http://www.equalizergraphics.com/documents/EqualizerGuide.html
[3] a shared memory system or graphics cluster

application. It does not impose a scene graph or does interfere in any way with the application's rendering code.

3.1 Task Methods

The application subclasses Equalizer objects and overrides virtual functions to implement certain functionality, e.g., the OpenGL rendering code in eq::Channel::frameDraw. These task methods are in concept similar to C function callbacks.

4 The Resource Tree

The rendering resources are represented in a hierarchical tree structure which corresponds to the physical and logical resources found in a 3D rendering environment.

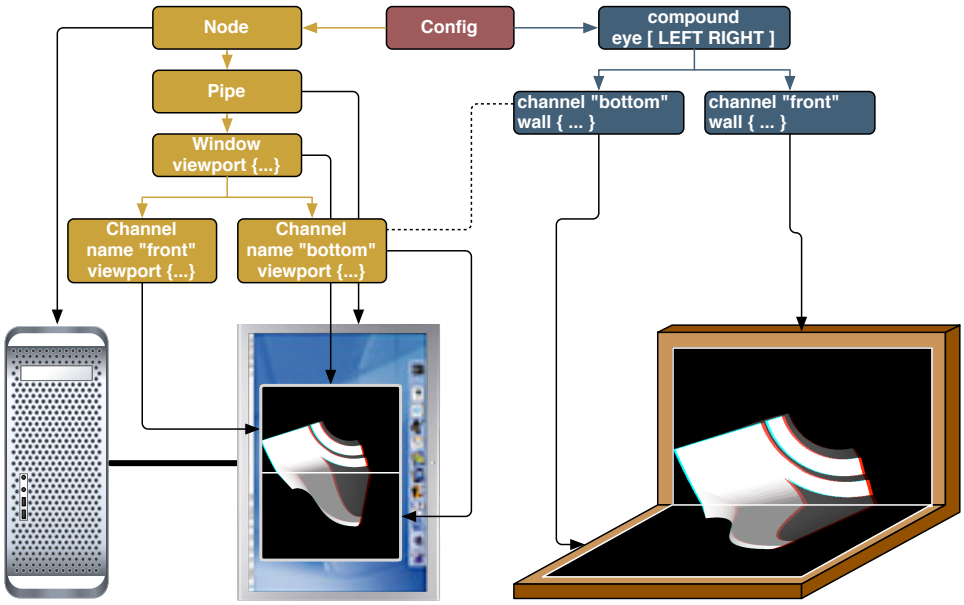

Figure 1: An example configuration

Figure 1 shows one example configuration, running on a single machine (node) using one graphics card (pipe) with one window to render to two output channels connected to a L-shaped projection table. The compound description is only used by the server to compute the rendering tasks. The application is not aware of compounds, and does not need to concern itself with the parallel rendering logics of a configuration.

For testing and development purposes it is possible to use multiple instances for one resource, e.g., to run multiple render client nodes on one computer. For deployment one node and pipe should be used for each computer and graphics card, respectively.

4.1 Configuration

The root of the resource tree is the eq::Config, which represents the current configuration of the application. It currently only holds the local node, not all nodes of the configuration.

4.2 Node

An eq::Node is the representation of a single computer in the system. It is one operating system process of the render client. All node task methods are executed from the main application thread.

4.3 Pipe

The eq::Pipe is the abstraction of a graphics card (GPU). In the current implementation it is also one operating system thread, unless the pipe's thread hint is set to false. All pipe and child window and channel task methods are executed from the pipe thread for threaded pipes or from the main application thread for non-threaded pipes[4].

Further versions of Equalizer might introduce threaded windows, where all window-related task methods are executed in a separate operating system thread.

4.4 Window

An eq::Window is an drawable and OpenGL context. The drawable can be an on-screen window or an off-screen PBuffer or FBO[5].

4.5 Channel

The eq::Channel is the abstraction of an OpenGL viewport within its parent window. It is the entity executing the actual rendering.

[4] see also http://www.equalizergraphics.com/documents/design/nonthreaded.html for non-threaded rendering

[5] off-screen drawables are not yet implemented, but can be created by the application and used with Equalizer