

## Scalability

Equalizer implements a wide range of algorithms to parallelize the rendering of large data sets. Multiple graphic cards, processors and computers can be combined to render a single view. The Equalizer server distributes the rendering task across the available resources (decomposition) and assembles the results on the final view (recomposition).

For the task decomposition, Equalizer currently supports sort-first (2D), sort-last (DB) and stereo (Eye) compounds. Time-multiplex (DPlex) is planned.

Equalizer supports virtually any recomposition algorithm, for example binary swap or direct send for sort-last rendering.

LGPL license

Open standard for scalable graphics

Clusters and shared memory systems

Collaborations welcome

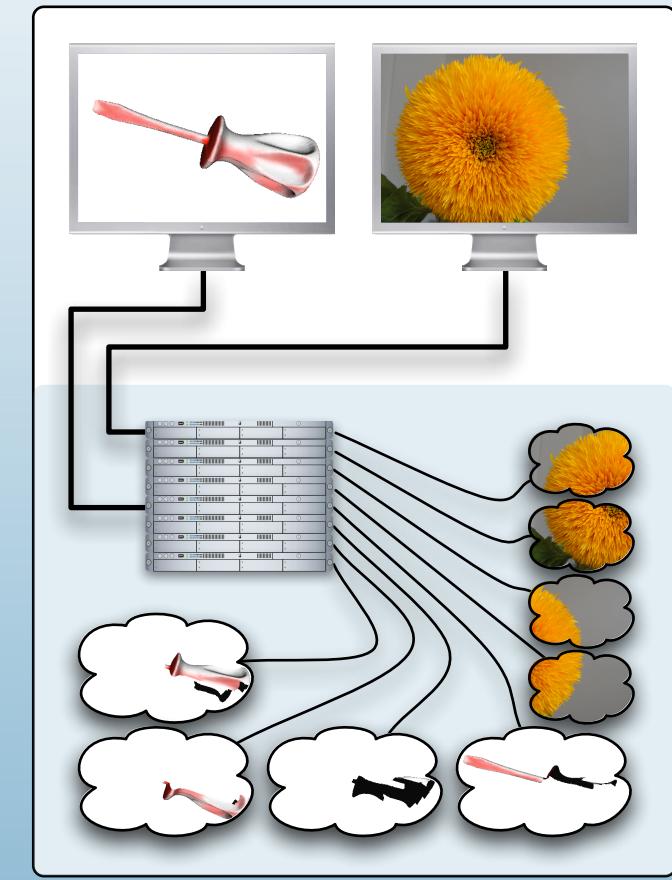
User-driven development

Consulting and support available

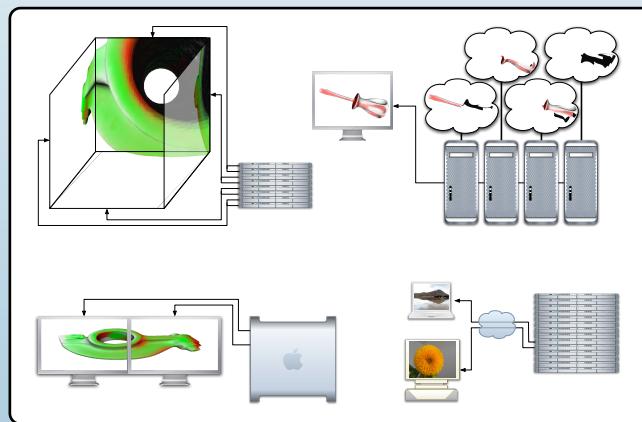
<http://www.equalizergraphics.com>  
[contact@equalizergraphics.com](mailto:contact@equalizergraphics.com)  
+41 76 33 77 247

# Equalizer

## Scalable Rendering



Equalizer is an open source programming interface and resource management system for scalable graphics applications. An Equalizer application can run unmodified on any visualization system, from a singlepipe workstation to large scale graphics clusters and shared memory visualization systems. Equalizer is build from ground up as a parallel, scalable programming interface solving the common problems for any multipipe application.



## Use Cases

Equalizer abstracts the configuration from the application code. This allows the same application to be deployed in different ways, for example:

**Display Walls** are driven by Equalizer by running a local OpenGL execution thread for each display.

**Immersive Installations** use passive or active stereo rendering with head tracking, which is supported by Equalizer.

**Multipipe Workstations** are an affordable way to scale the rendering performance and display size.

**Remote Visualization Clusters** are an cost-efficient way of centralizing large scale data.

**Scalable Rendering** is used to parallelize the rendering of a single view across multiple graphic cards and processors.

## Programming Interface

Equalizer uses a callback-driven interface. Applications provide their rendering callbacks, which are called by the Equalizer framework according to the current configuration and system load. Process and thread creation, task synchronization and network connections are externalized from the application and handled transparently by the Equalizer framework.

The multi-pipe programming interface follows the natural execution model of any multi-pipe application. The application needs to separate the rendering routines from the core application to make them distributable. Equalizer abstracts common graphic entities into C++ classes, for example:

- **Node** - a single computer in the cluster
- **Pipe** - a graphics card and rendering thread
- **Window** - an OpenGL drawable
- **Channel** - a viewport within a window

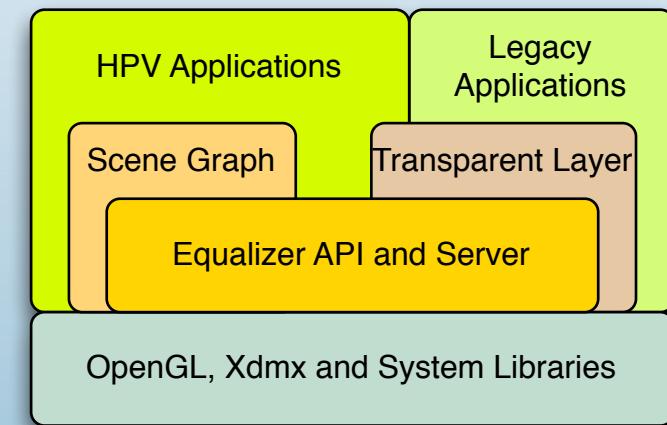
The application subclasses these entities, and overrides methods to provide the application's rendering code. Each method corresponds to a task, and for all of them Equalizer provides a default method which implements the typical use case. This allows a quick start for the developer, which can then replace the defaults gradually with his own code. Some of the most important methods are:

- **Channel::draw** - render using the provided frustum, viewport and range
- **Window::init** - init OpenGL drawable and state
- **Pipe::startFrame** - update frame-specific data
- **Node::init** - initialise per node application data

Equalizer provides an API to build network-distributed applications. In contrast to other cluster API's, such as MPI or PVM, this API is designed for the development of interactive applications. Certain characteristics of visualization software, e.g. the frame-driven rendering, are exploited for better performance and an easier to use programming interface.

## Resource Management

Equalizer uses a configuration server to optimally allocate and balance the available resources on the visualization system. The server is configured using a hierarchically structured configuration file to describe the available resources, and the combination of the resources for rendering. Equalizer applications can also run without a server, in which case they will automatically configure themselves for the local workstation. Possible environments for Equalizer are centralized visualization clusters, multipipe shared memory visualization computers or single- and multipipe workstations.



## Transparent OpenGL Layer

The transparent layer will enable the execution of unmodified OpenGL applications. It allows a seamless integration of unmodified and scalable applications on the same system.

## Distributed Scene Graphs

Equalizer will be integrated with popular scene graphs, such as Coin3D or OpenSceneGraph. Applications using these scene graphs can then easily implement parallel, scalable applications, and profit from the current and future feature set of Equalizer.