Equalizer

VizSIG Meeting, Woking, October 2006

Stefan Eilemann

http://www.equalizergraphics.com/documents/Woking06.pdf

Outline

- High performance visualisation (HPV)
- Equalizer
  - Programming interface
  - Resource management
  - Future components

HPV

- Transparent and semi-transparent solutions
- Programming interfaces
  - Scene graphs
  - Generic middleware

HPV Transparent Solutions

- Chromium, ModViz VGP, OMP
- Operate on OpenGL command stream
- Programming extensions for improved performance and scalability (semi-transparent)
- HPC analogy: auto-parallelising compilers

Scene Graph API's

- ScaleViz, Vega Prime, VTK, OpenSG
- Impose overall programming model and data structure
- Best for developing new applications
- HPC analogy: CFD codes

Generic HPV Middleware

- Cavelib, VRJuggler, MPK
- Limited to HPV-critical areas of the code
- Best for porting existing applications
- HPC analogy: MPI, PVM

Equalizer

A Programming Interface

and

Resource Management System

for

Scalable Graphics Applications

Equalizer Programming Interface

Applications are written against a *client library* which abstracts the interface to the execution environment

- Minimally invasive programming approach
- Abstracts multi-processing, synchronisation and data transport
- Supports distributed rendering and performs frame compositing

Equalizer Programming Interface

C++ classes which correspond to graphic entities, e.g.:

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

Equalizer Programming Interface

Application subclasses and overrides methods, e.g.:

- **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

Default methods implement typical use case

Resource Management System

Applications are deployed by a *server* which balances the resource usage across the system

- Centralises the setup for all applications
- Configures application and deploys render clients
- Dynamic load-balancing of the cluster resources

Resource Management System

Server configuration:

```
server
{
    config // 1-n times, currently only the first one is used by the server
    {
        <resources>  // What is being used (next slide)
        <compounds>  // How it is being used (slide after next)
    }
}
```

Resource Management System

Resource configuration:

```
node // 1-n times
{
    pipe // 1-n times
    {
        display  unsigned     // X11 display or ignored
        screen   unsigned     // X11 screen/CGL display/graphics adapter
        window // 1-n times
        {
            viewport [ viewport ] // wrt pipe, default full screen
            channel // 1-n times
            {
                name     string
                viewport [ viewport ] // wrt window, default full window
            }
        }
    }
}
```

Resource Management System

Resource utilisation:

```
compound // 1-n times
{
    channel  string   // where the compound tasks are executed
    task     [ CLEAR DRAW ASSEMBLE READBACK ] // tasks to execute
    viewport [ viewport ]            // wrt parent compound, sort-first
    range    [ float float ]         // DB-range for sort-last
    eye      [ CYCLOP LEFT RIGHT ]   // monoscopic or stereo view
    wall|projection                  // frustum description
    {...}                            // typically at root compound

    <child-compounds>

    swapBarrier { name string } // same barriername = sync swap buffers
    outputFrame { name string }
    inputFrame  { name string } // name corresponding to an output frame
}
```

Decomposition Modes

- DB / sort-last (range)
- 2D / sort-first (viewport)
- Eye / stereo (eye)
- DPlex (period, phase)
- Any combination thereof

Recomposition Modes

- Combination of task and frames allows virtually any recomposition mode, e.g.:
  - 2D tile gathering
  - binary swap
  - direct send

SSI and Clusters

- Supercomputers are just tightly integrated clusters
- Equalizer runs on both architectures
- Execution model is the same
- SSI allows additional optimisations and simplifications
- Stand-alone SSI version planned

Use Cases

Display Walls
Scalable Rendering
Remote Rendering
Virtual Reality
Multipipe Workstations

Equalizer Future

- Transparent Layer: virtual OpenGL screen
- Scene Graph: “transparent scalability”
- Equalizer: Scalable rendering engine

Transparent Layer

- “Enabler” for visualisation clusters
- Legacy and non-critical applications
- System load balancing between transparent and HPV applications
- Single point of configuration
- Performance and compatibility as today

Distributed Scene Graph

- Uses Equalizer for HPV
- Small effort for application developer
- Single point of configuration
- Candidates: OpenSceneGraph, Coin

Remote Visualisation

- Leverages knowledge of the application
  - Frames are often available in main memory
  - Additional frame-transport optimisations
- Loadbalancing of multiple applications on one visualisation cluster

Project Status

- API and resource server are usable
- Transparent layer, remote visualisation and distributed scene graph depend on demand and sponsoring

Demos

- 1-window
- 3-pipe
- 5-channel.cave

Last Words

- LGPL license
- Open standard for scalable graphics
- User-driven development
- Alpha version available on: www.equalizergraphics.com
- Consulting and support available
- Get in touch