Parallel Rendering
with Equalizer

Prof. Dr. Renato Pajarola
Outline

- Motivation
- Parallel Rendering
- Multipipe System
- Equalizer
Massive Data Visualization

- Processing and rendering requirements exceeding available computing resources
  - Memory
    - data size larger than available physical main memory
  - CPU/GPU
    - bandwidth limited to process data interactively
  - Display
    - too many elements to see and display
Why is large a problem?

• Moore’s law
  + Computing power doubles every 18 months
    • improvement of hardware will cope with any conceivable data sets in foreseeable future
  
• But…
  - Data is generated by same hardware
    • same growth can be expected
  - Interactivity
Parallel Hardware

- Exploit parallel computing and rendering resources
  - parallel cluster computer
  - multi-GPU acceleration
  - high-speed interconnect
Application Environments

- Display Walls
- Virtual Reality
- Remote Rendering
- Parallel Rendering
Display Walls

- Group collaboration
- Better data understanding
- One or more displays per computer
- High resolution: 10-100 MPixels
Virtual Reality

• Stereo rendering, head tracking
• Immersive displays with high frame rates
• CAVEs with up to two computers per wall with passive stereo
Remote Rendering

- Centralize data, software and hardware
- Combined with scalable rendering
- Avoids copying of HPC result data
- Simplifies administration
Scalable Rendering

- Render massive data sets interactively
- Exploit multiple graphics cards (GPUs) and processors (CPUs) per display
- Different algorithms for parallelization
Outline

- Motivation
- Parallel Rendering
- Multipipe System
- Equalizer
Rendering Task Decomposition

• Single frame decomposition
  • sort-first: screen-space decomposition
  • sort-middle: only practical on GPU
  • sort-last: database decomposition

• Entire frame decomposition
  • D Plex: time-multiplex
  • Eye: stereo passes
1. Transform geometry into screen space
2. Rasterize primitives into fragments
3. Process fragments into pixels
2D/Sort-First

- Scales fillrate/fragment processing
- Scales geometry with efficient view frustum culling
- Parallel overhead due to primitive overlap limits scalability
DB/Sort-Last

- Scales all aspects of rendering pipeline
- Application needs to be adapted to render subrange of data
- Recomposition relatively expensive
Eye/Stereo

- Stereo rendering
- Excellent load balancing
- Limited by number of eye views
Dplex/Time-Multiplex

- Good scalability and load balancing
- Increased latency may not be acceptable
Conclusion

- No ‘magic bullet’
- 2D is ideal for less than eight pipes
- Use Eye if running in stereo
- DB scales well
- Combine modes

<table>
<thead>
<tr>
<th></th>
<th>2D</th>
<th>DB</th>
<th>Dplex</th>
<th>Eye</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fillrate</td>
<td>++</td>
<td>+</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Vertex Processing</td>
<td>0</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>0</td>
<td>++</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Load Balancing</td>
<td>0</td>
<td>+</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Latency</td>
<td>++</td>
<td>++</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>Re-assembly</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
Equalizer Multilevel Compounds

- Compounds allow any combination of modes
- Combine different algorithm to address and balance bottlenecks
- Example: use DB to fit data on GPU, then use 2D to scale further
Parallel Compositing

- Compositing cost grows linearly for DB
- Parallelize compositing
- Flexible configuration
- Constant per-node cost
- Details in EGPGV’07 paper
Outline

- Motivation
- Parallel Rendering
- Multipipe System
- Equalizer
Parallel Applications

- Single pipe application
  - traditional application and rendering model

- Multipipe application
  - multiple instances of software run on different nodes and interact
Single Pipe Rendering

- Typical rendering loop
- Stages may not be well separated
Multipipe Rendering

- Equalizer separates rendering and application
- Instantiate rendering multiple times
- Synchronize parallel execution
Runtime Scalability

- Parallel execution of the application’s rendering code
- One thread per graphics card, one process per node
- Decomposition of rendering for one view
Asynchronous Execution

• A rendering thread (channel) can start rendering the next frame early
• hides imbalance in load distribution
• only visible channels belonging to the same view are synchronized
• Greatly improves scalability on bigger clusters
Outline

• Motivation
• Parallel Rendering
• Multipipe System
• Equalizer
Equalizer Concepts

“GLUT for multi-GPU systems and visualization clusters”

• Task-driven: init, exit, clear, draw, (readback, assemble)
• Resource-based: Node, Pipe, Window, Channel
Parallel rendering applications are written against a client library which abstracts the interface to the execution environment.

- **Library and API**
  - Minimally invasive programming approach
  - Abstracts multi-processing, synchronization and data transport
  - Supports distributed rendering and performs frame compositing
Resource-based

- Hierarchical resource description: Node → Pipe → Window → Channel
  - Node is a single computer in the cluster
  - Pipe is a graphics card and rendering thread
  - Window is an OpenGL drawable
  - Channel is a viewport within a window
- Resource usage: compound tree
Compound Trees

- Description of resource usage and parallel task distribution
- Easy specification via text configuration files
Holobench

Node

Pipe

Config

compound eye [ LEFT RIGHT ]

wall { ... }

swapbarrier {}

channel "bottom"

Channel name "front" viewport {...}

viewport {...}

Channel name "bottom" viewport {...}

viewport {...}

Window viewport {...}

Pipe

Channel name "front" viewport {...}

viewport {...}
Scalability

David 1mm x4

225M triangle model
Scalability

512^3 voxel model
Open Source

- LGPL license
- Open standard for scalable graphics
- Clusters and shared memory system supported
- More on www.equalizergraphics.com