Definitions and Goals (1)

• **Scalable rendering definition**
  For an increase in work quanta, an equivalent increase in rendering resources maintains constant rendering time.

• **The Goal: Aggregation**
  The basic scalable rendering goal is to effectively and efficiently aggregate various system components to address increasing data driven demands.
Definitions and Goals (2)

- Visualization scalability along two axis
  - Datasets
    - Increases in cell count, etc.
  - Displays
    - Higher pixel counts
    - Multiple/tiled displays

- LLNL ASCI White (8192 CPUs, 13TF)
- 469M Triangle Isosurface (2048x2048x1920x273 Grid)
- LLNL PowerWall (6400x3072)

IBM T221 ‘Bertha’ (3840x2400)
What makes rendering unique?

Generation of graphical primitives
- Graphics computation: primitive extraction/computation
- Multiple rendering engines (both in number and type)

Video displays
- Routing of video tiles
- An aggregation of multiple rendering engines

Interactivity (not a render-farm!)
- Real-time imagery
- Interaction devices, human in the loop (prediction issues)

Unique I/O requirements
- Access patterns/performance
Systems Architecture

What does a scalable rendering system look like?

One or more systems/machines providing:

- Computational capacity (extract primitives)
- Graphical rendering capacity (draw primitives)
- Display capacity (display images)

Layered, integrated software:

- Application (domain/problem specific)
- Toolkits, Data models, etc
- Rendering (Mesa, OpenGL, DirectX, Qsplat, etc)
System Architecture: Compute Nodes

Computational elements upon which the visualization application runs

- May run the computation itself!
- Single system image (SSI)
  - SGI Onyx
- Distributed clusters
  - IBM SP, Myrinet+PCs
- I/O systems
  - “Interconnect”
    - NUMA, Myrinet, GigE, ...
  - Disk subsystems
System Architecture: Graphics Options

Add rendering capability to the nodes

- **Software**
  - Mesa
  - Custom (e.g. qsplat)

- **Hardware**
  - IR pipes/pipelets
  - PC graphics
    - nVidia, Matrox, ...
    - 3Dlabs, HP, ...

---

nVidia GeForce 2
Connect rendering capability to displays

- Desktop(s)
- Tiled displays
  - PowerWalls
  - IBM T221 ("Bertha")
- Video switching
- Video delivery
- Custom hardware
  - IBM SGE
  - Stanford Lightning-2
  - Compaq Sepia-2
  - SGI, HP video compositors
Systems Architecture: Software

Various goals result in very different software solutions...

Must provide an appropriate solution to the aggregation problem.

A balance must be struck between data handling and rendering requirements.
Rendering Aggregation Approaches

- **2D - “screen space”**
  - “Sort-first” rendering model
  - Targets display scalability
  - Supports high frame rates

- **3D - “data space”**
  - “Sort-last” rendering model
  - Targets large data scalability
  - Supports high primitive counts
Aggregation: Sort First

Tiling (2D decomposition in screen space)

Route portions of a final aggregate display to their final destination with no overlap

- Order independent
- Destination determines bandwidth
- Graphics primitives may be moved, replicated or sorted for load balancing
- Simple color data
- Can support local framerate
Aggregation: Sort Last

Compositing (3D decomposition in data space)

3D blocks that are combined using classic graphics operators (e.g. Z-buffering, alpha blending, etc)

- Z, α, stencil enhanced pixels
- Fixed 3D data decompositions (data need not move)
- Bandwidth exceeds that of output display (3D vs 2D)
- Hierarchy trades bandwidth for latency
- Ordering may be critical
- “Generation” at local framerate
Aggregation: Issues

- Multi-pass rendering algorithms
  - “Complete” intermediate image requirements difficult
- Framebuffer readback
  - Performance and availability of graphics APIs
- Rendering resolution issues
  - Dataset domains and (global) Z-buffer resolution
  - Limited pixel resolution (e.g. \( \alpha \)-compositing)
- Compositing Issues
  - Latency and ultimate framerate
  - Ordering/Transparency
- Flexible/scalable software interfaces
  - Data partitioning: The “zoom” problem
  - Anisotropic rendering environments
Examples: Scalable Rendering Systems

Hardware

- Classical SSI and clusters (SGI, IBM, etc)
- "Viz Sim" cluster systems (SGI, HP, Artabel)
  - Limited data (replicated), targets walls/"immersion"

Software

- Integrated application/framework solutions
  - VTK, OpenDX (MPI), EnSight, MeshTV, VisIt
- "Low-level" toolkits
  - libpglc, TNT-Pmesa, WireGL, Chromium
Examples: Sort-last Toolkits

- **TNT-PMesa (Sandia, Lisa Ice, et al)**
  - Composition system integrated into Mesa
  - Details of the scheme hidden behind OpenGL API
  - Linked to software rasterizer (Mesa)
  - Not the SourceForge PMesa project!

- **Libpglc (Sandia, Brian Wylie, et al)**
  - Sort-last composition, several optimized modes
  - New application API/rendering structure
  - Supports hardware rendering
  - 300M Polys/sec @ 64 nodes on a 469M tri isosurface
Examples: WireGL Toolkit (1)

“Drop-in” replacement for OpenGL (Stanford)

- Supports unmodified applications

Targets tiled displays

- Geometry bucketed/sorted
- “Sort-first” routing of all primitives

Diagram:
- Application connected to Virtual Renderer
- Virtual Renderer connected to multiple Renderers

IEEE Visualization 2001
Examples: WireGL Toolkit (2)

Efficient network protocol

- All OpenGL commands encoded onto a stream
- Rendering on remote “servers”
- Context state tracking

Parallel OpenGL API

- Semaphores/Barriers
- “Handled” in the servers

Demonstrated input and output scalability

- Many noted limits, so...
Examples: Chromium Toolkit (1)

WireGL replacement (Stanford)

- Work-in-progress
- Based on WireGL technologies
- Chromium implements WireGL

Addressing limitations

- “Local” node rendering
- Extensibility
- Non-render/tile operations
- Better hardware abstractions
Examples: Chromium Toolkit (1)

WireGL replacement (Stanford)

- Work-in-progress
- Based on WireGL technologies
- Chromium implements WireGL

Addressing limitations

- ‘Local’ rendering (sort-last)
- Extensibility
- Non-render/tile operations
- Better hardware abstractions
Examples: Chromium Toolkit (2)

Distributed rendering pipeline management system

- Hinting/query interfaces (application “translucent”)

Extensibility: The Stream Processing Unit (SPU)

- “Filter” view of an OpenGL implementation
- Allow for direct OpenGL rendering (e.g. sort last)
- True rendering framework for algorithm integration
- Any SPU can render, modify, absorb... the OpenGL API
- Supports SPU inheritance

Improved context handling

- Multiple contexts/application windowed contexts
Examples: Compositing Systems

Custom hardware for image composition

- **Sepia (Compaq)**
  - Dedicated network (ServerNet II)
  - Custom compositing (FPGA on NIC)

- **Lightning-2/MetaBuffer (Stanford & Intel/UTexas)**
  - DVI based tiling/compositing

- **Scalable Graphics Engine (IBM)**
  - “Tiled” framebuffer
  - gigE/UDP based
Present and Future...

Basic and partial solutions available today...

- Turnkey tiled display and cluster aware applications
- Integrated data and rendering ‘toolkits’

Still many research and development topics

- Improved compositing (HW and SW solutions)
- Data representations/decompositions
- Alternate visualization approaches
  - Multi-resolution/progressive techniques
  - New primitives/rendering methods
Some Library/Application Links

Mesa: mesa3d.sourceforge.net
TNT-PMesa: www.cs.sandia.gov/VIS/pmesa.html
WireGL: graphics.stanford.edu/software/wiregl
Chromium: chromium.sourceforge.net
VTK: www.kitware.com
OpenDX: www.opendx.org
EnSight: www.ceintl.com
MeshTV: www.llnl.gov/meshtv