

The Evolution of Distributed Visualization Clusters

Randall Frank

Lawrence Livermore National Laboratory

UCRL-PRES-154718



This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

What Makes Visualization Unique?

Unique I/O requirements

- Access patterns/performance

Generation of graphical primitives

- Graphics computation: primitive extraction/computation
- Dataset decomposition (e.g. slabs vs chunks)

Rendering of primitives

- Aggregation of multiple rendering engines

Video displays

- Routing of digital or video tiles to displays (over distance)

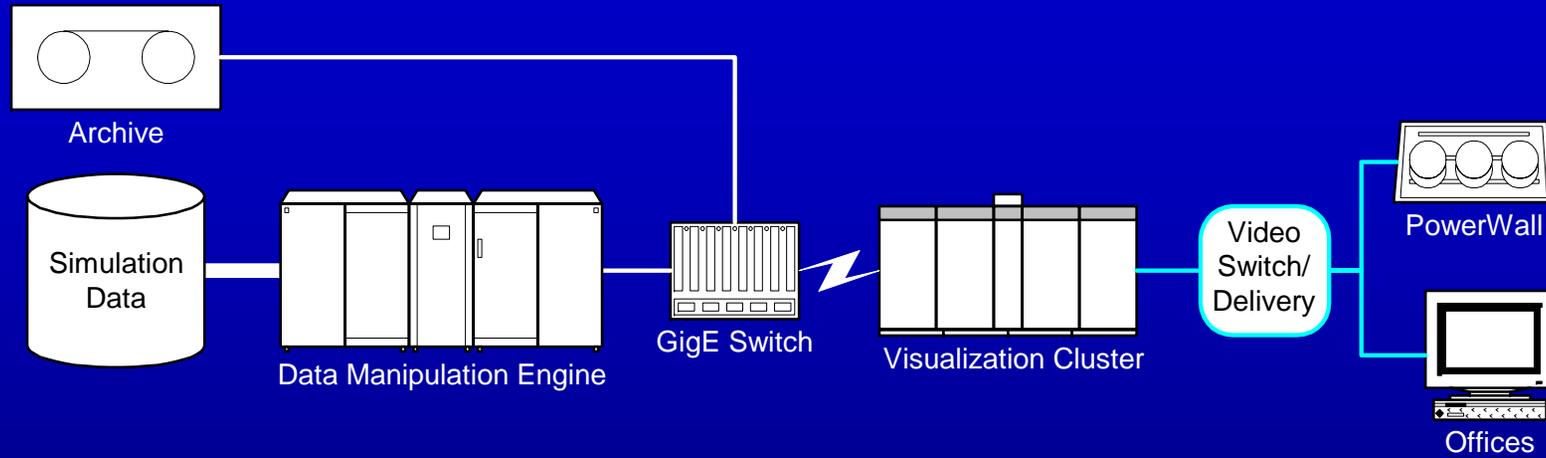
Interactivity (not a render-farm!)

- Real-time imagery
- Interaction devices, human in the loop (latency, prediction issues)

Why Distributed Visualization Clusters?

- The realities of extreme dataset sizes
 - Stored with the compute platform
 - Cannot afford to copy the data
 - Visualization co-resident with compute platform
- Track compute platform trends
 - Distributed infrastructure
 - Commodity hardware trends
- Migration of graphics leadership to the PC

Visualization Environment Architecture

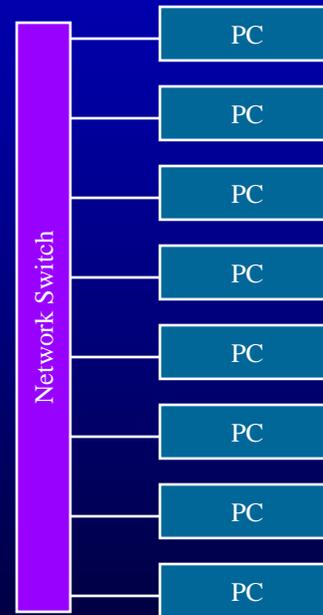


- Raw data on platform disks/archive systems
- Data manipulation engine (direct access to raw data)
- Networking (data/primitives/images)
- Visualization/rendering engine
- Video and remotely rendered image delivery over distance
- Displays (office/PowerWalls)

Cluster Architecture: Compute Nodes

Computational elements upon which the visualization application runs

- Largely the same as the compute cluster
- **Distributed nodes**
 - P4, Itanium, Opteron
 - Chipsets I8[5-7]0, E7505
- **Interconnects**
 - Quadrics, Myrinet, GigE
- **I/O systems**
 - Lustre, NFS, PVFS, ...



HP Visualization Cluster



Stanford "Chromium" Cluster

rjf, 5/28

Cluster Architecture: Graphics Options

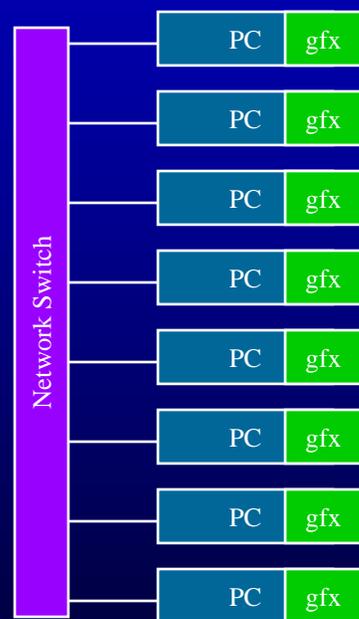
Add rendering capability to the nodes

- Software OpenGL & Custom

- Mesa, qsplat, *-Ray

- Hardware

- AGP or PCI Express slot
- COTS graphics
 - nVidia, ATI, ...
- Card device drivers
 - OS, Graphics API, etc



ATI FireGL X2



nVidia Quadro FX

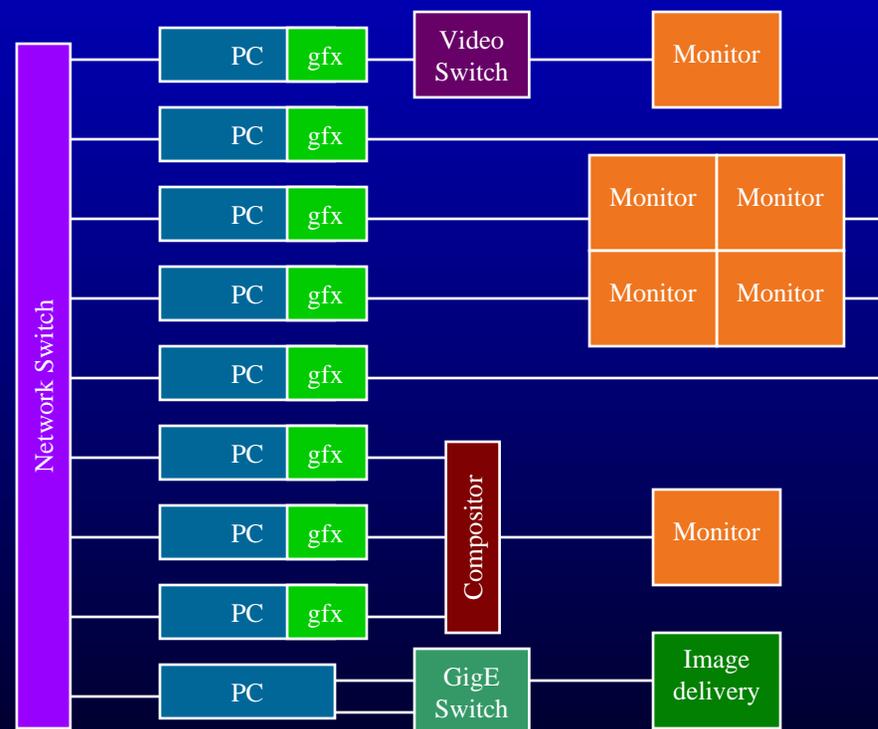
Cluster Architecture: Displays

Connect rendering capability to displays

- Desktop(s)
- Tiled displays
 - PowerWalls, IBM T221
- Video switching/routing
- Remote image delivery
 - Analog, Digital
- Compositing hardware
 - HP Sepia-2 & sv6
 - IBM SGE
 - Stanford Lightning2
 - SGI DVI Compositor



IBM T221 (3840x2400)



A Little History (~6 years ago?)

We have been doing large data visualization since we started generating datasets...

- Software parallel rendering dominated
 - Visualization done right on the compute nodes
 - Usually w/Mesa (OpenGL work-alike) or ray-casting
 - Lots of parallel image compositor software
 - Little interactively, mostly movies

Then along came Quake and 3D PC graphics...

The Birth of Modern COTS graphics

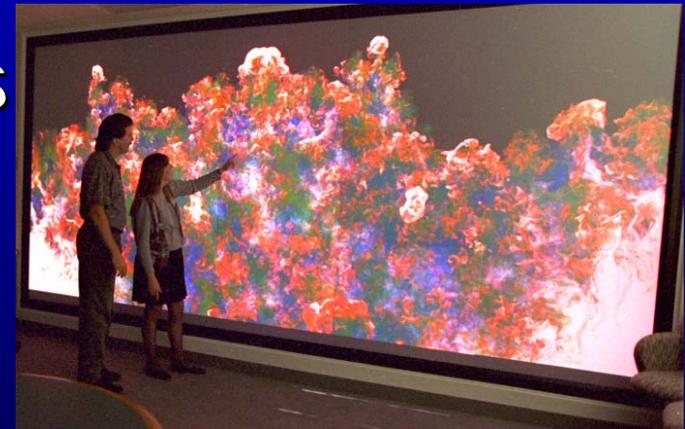
- 3D games were the driver
 - Windows/DirectX was the target
 - John Carmack saves OpenGL?
 - “Huge” Markets
 - Cost/performance becomes key
 - No viable Linux gaming market (despite attempts)
- Cards rivaled raw workstation performance
 - \$300 GeForce3 == \$80k SGI IR pipe?
 - Today they rival in capability as well...
- **Why fill a cluster with graphics cards?**



Quake III Arena

The Birth of the PowerWall

- The '94 PowerWall (UMN, Paul Woodward)
 - Tiled displays of abutted display units
 - Extreme I/O requirements (2x2 needed 300MB/s)
- Collaborative environments
- Increased pixel counts
 - Matching higher fidelity data
 - Ignoring 3D vs 2D issue
- Requires output scalable image generation
 - With COTS gfx cards, image aggregation is required



LLNL 6400x3072 (5x3) PowerWall

Image Aggregation Solutions

Image “compositing”: Take the (digital) outputs of multiple graphics cards and combine them to form a single image.

Multiple goals/dimensions of scaling via aggregation

- Output scaling: Large pixel counts (PowerWalls)
- Data scaling: High polygon/fill rates/data decomposition
- Interaction/Virtual reality: High frame rates
- Image quality: Anti-aliasing, data extremes

Hardware acceleration is natural

- Efficient access to rendered imagery
- Provide for image “fragment” transport
- Flexible, pipelined “merging” operations

Solutions balance speed, scale...

- Image input/transport solutions
- Application transparency
- Parallel rendering models



HP sv6

Examples: Compositing Systems

Image composition hardware

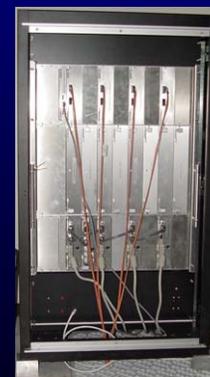
- **Lightning-2/MetaBuffer (Stanford & Intel/UTexas)**
 - DVI based tiling/compositing
- **Sepia (HP/Compaq)**
 - Custom compositing (FPGA + NIC)
 - Dedicated network (ServerNet II & IB)
- **Scalable Graphics Engine (IBM)**
 - "Tiled" framebuffer
 - gigE/UDP based, distance solution



Lightning-2

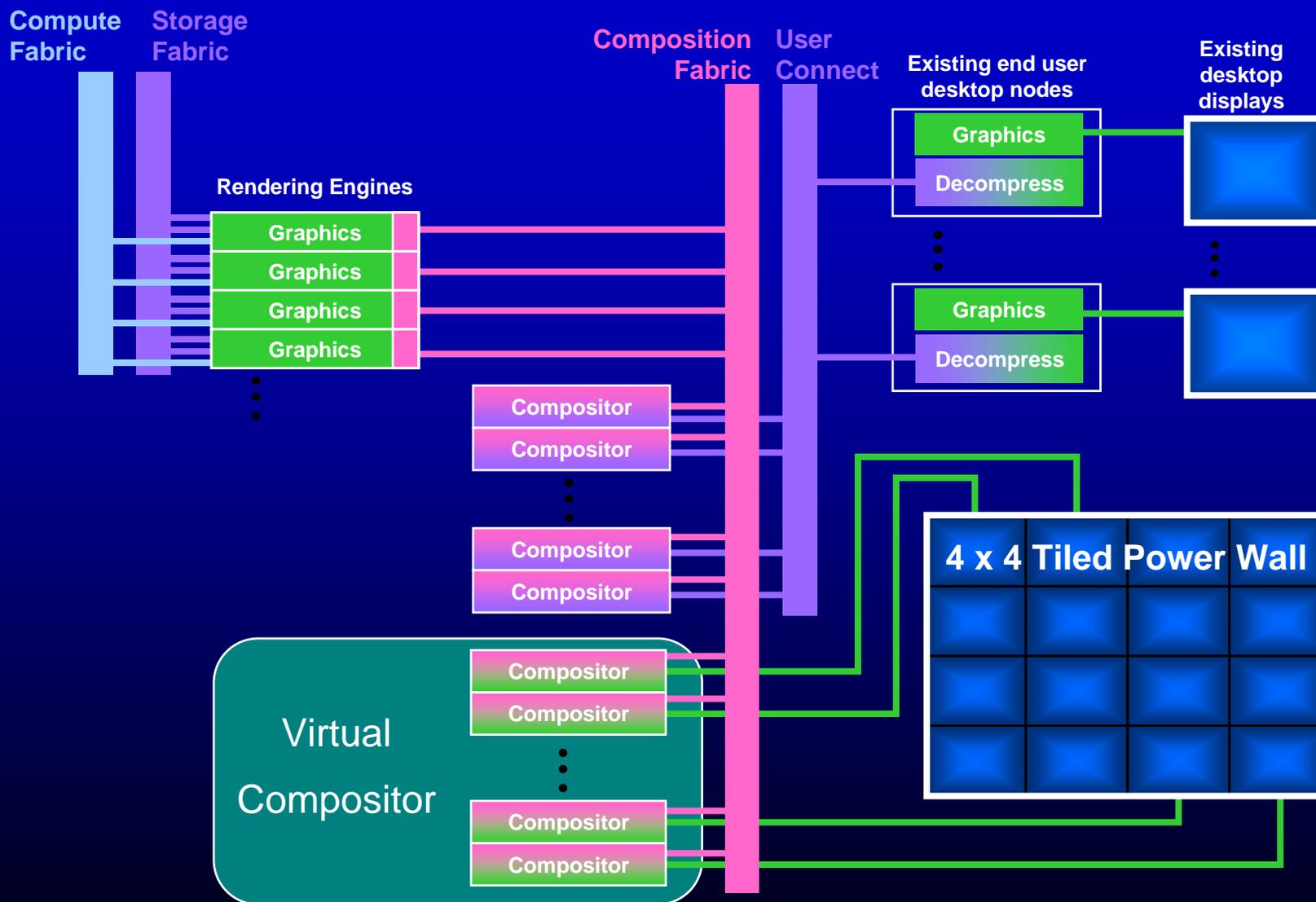


HP Sepia-2



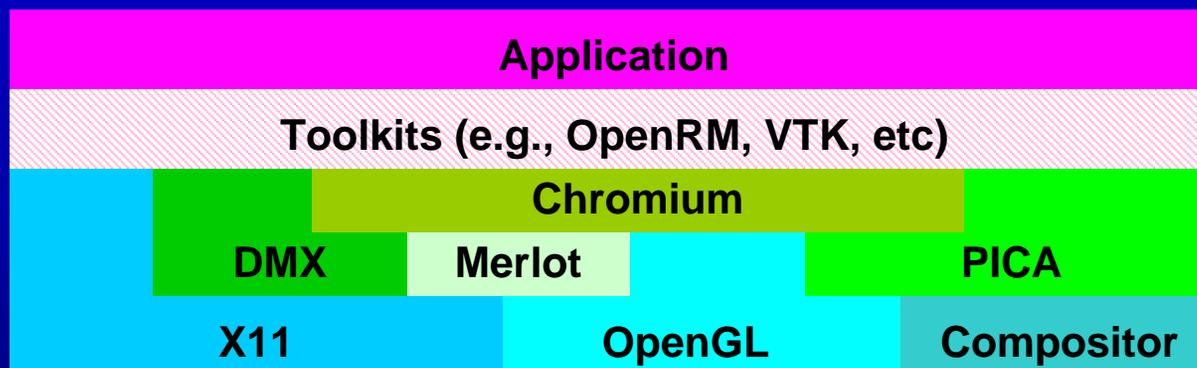
IBM SGE

Idealized Visualization Environment



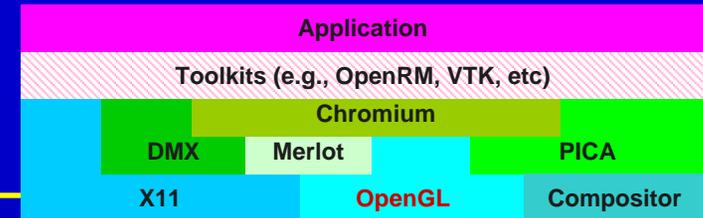
A Distributed Parallel API Stack

Goal: Provide integrated, distributed parallel services for viz apps.
Encourage new apps, increase portability & device transparency.



- Applications: VisIt, ParaView, EnSight, Blockbuster, etc...
- Toolkits - Parallel visualization algorithms, scene graphs...
- DMX - Distributed windowing and input devices (X11)
- Chromium - Parallel OpenGL rendering model
- PICA - Parallel image compositing API
- Merlot - Digital image delivery infrastructure
- Core "vendor" services - X11/OpenGL/compositors/NICs

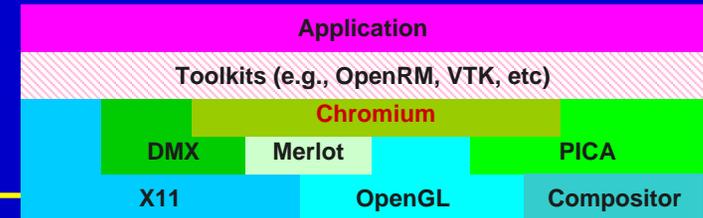
Linux OpenGL Drivers



ASCI funded the bootstrapping of Linux OpenGL driver development under PathForward. Today drivers are largely vendor supplied to support digital content creation apps.

- Most initial concerns have been addressed
 - Viability of OpenGL, ongoing support, non-game features...
- Current status of Linux OpenGL drivers:
 - Solid support from nVidia and ATI
 - Drivers support recent ARB extensions
 - Vertex & fragment programs, ARB_vertex_buffer_object, etc
 - Complete buffer support ("float" pbuffers, multi-head, stereo, etc)
 - Excellent performance, rivaling Windows, but features can lag
 - Some interactions with kernels & other drivers (e.g. Elan)

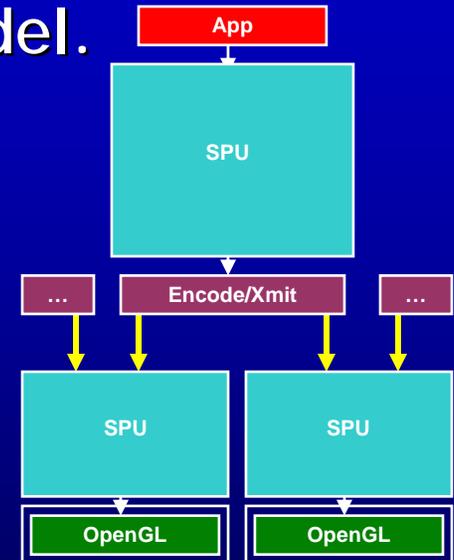
Distributed GL: Chromium



Distributed OpenGL rendering pipeline. Provides a parallel OpenGL interface for an N to M rendering infrastructure based on a graphics stream processing model.

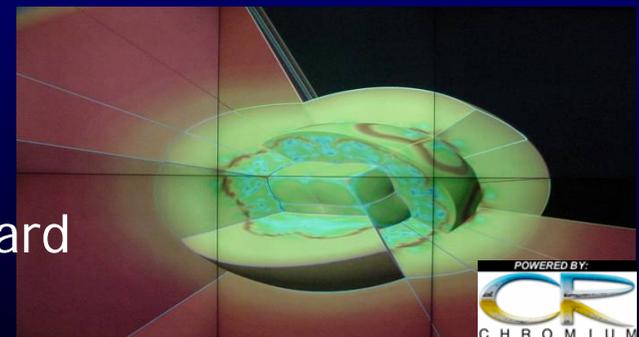
The Stream Processing Unit (SPU)

- "Filter" view OpenGL
- SPU interface is the OpenGL API
 - Render, modify, absorb...
- Allows direct OpenGL rendering
- Supports SPU inheritance
- Application "translucent"

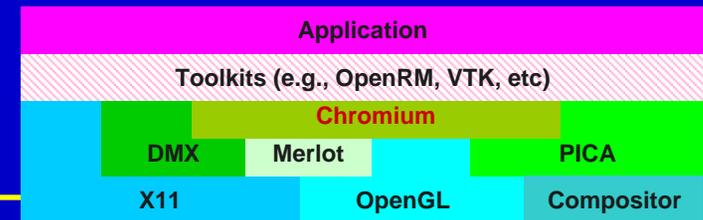


Development:

- OpenGL 1.3 & 1.4 & Extensions
- <http://chromium.sourceforge.net>
- RedHat/Tungsten Graphics ASCI PathForward
- Stanford, University of Virginia



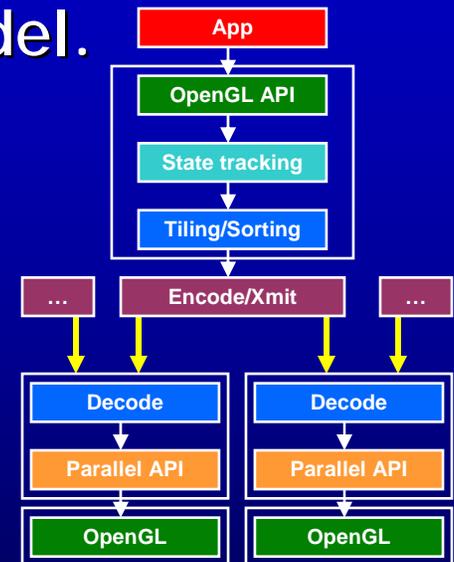
Distributed GL: Chromium



Distributed OpenGL rendering pipeline. Provides a parallel OpenGL interface for an N to M rendering infrastructure based on a graphics stream processing model.

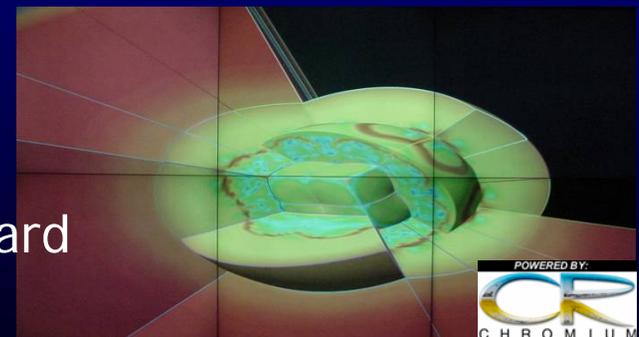
The Stream Processing Unit (SPU)

- “Filter” view OpenGL
- SPU interface is the OpenGL API
 - Render, modify, absorb...
- Allows direct OpenGL rendering
- Supports SPU inheritance
- Application “translucent”

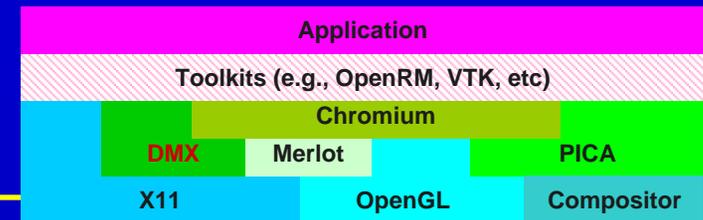


Development:

- OpenGL 1.3 & 1.4 & Extensions
- <http://chromium.sourceforge.net>
- RedHat/Tungsten Graphics ASCI PathForward
- Stanford, University of Virginia

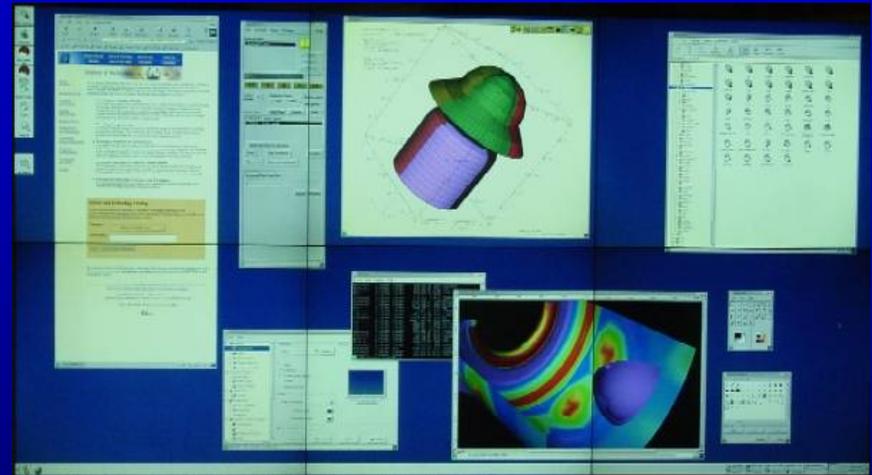


Parallel X11 Server: DMX



Distributed multi-headed X server: DMX

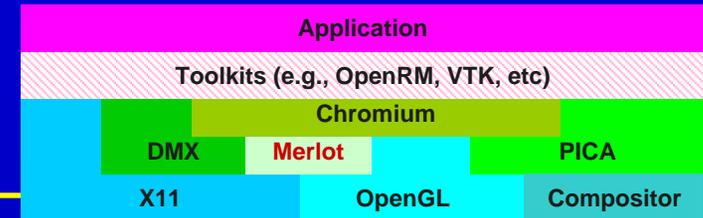
- **Aggregates X11 servers**
 - “Server of servers” for X11
 - Single X server interface
- **Accelerated graphics**
 - 2D via accelerated X server
 - Common extensions as well
 - Back-side APIs for direct, local X11 server access
 - OpenGL via ProxyGL/GLX (from SGI) or via Chromium SPU



Development: <http://dmx.sourceforge.net>

- RedHat ASCI PathForward contract
- Integrated with XFree86

Remote Delivery: Merlot



Merlot is a framework for digital image delivery

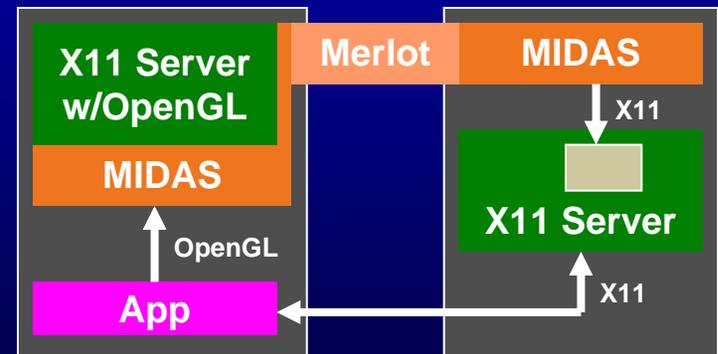
- Transport layer abstraction, Codec interfaces, Device transparency

MIDAS: Merlot Image Delivery Application Service

- Indirect OpenGL rendering services for X11 environment
- Indirect window management
- Image stream transport

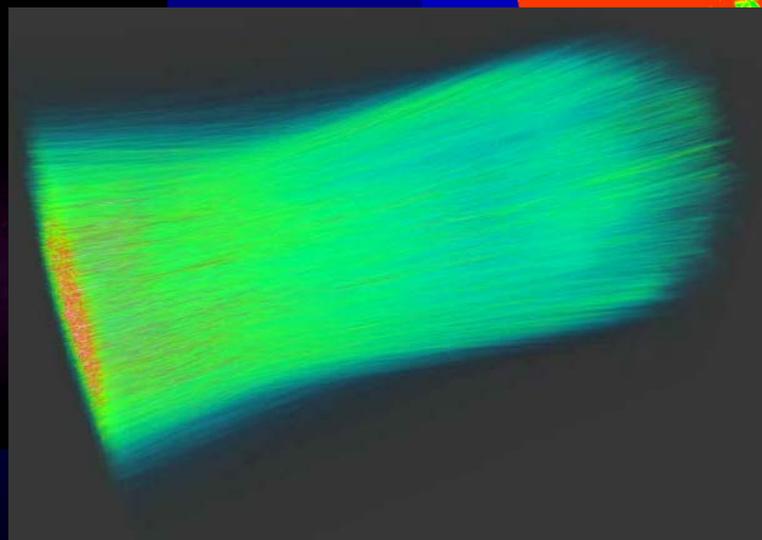
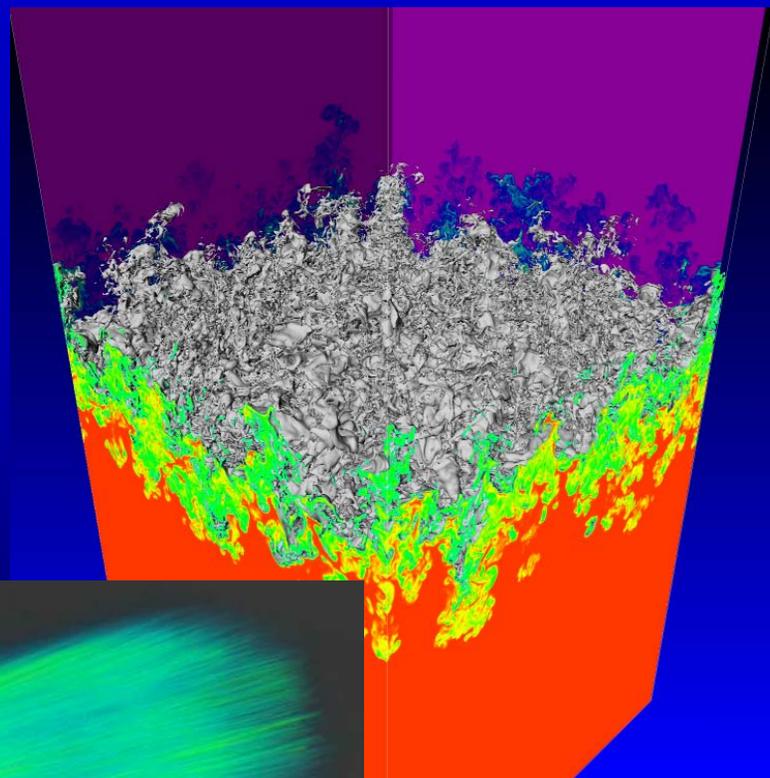
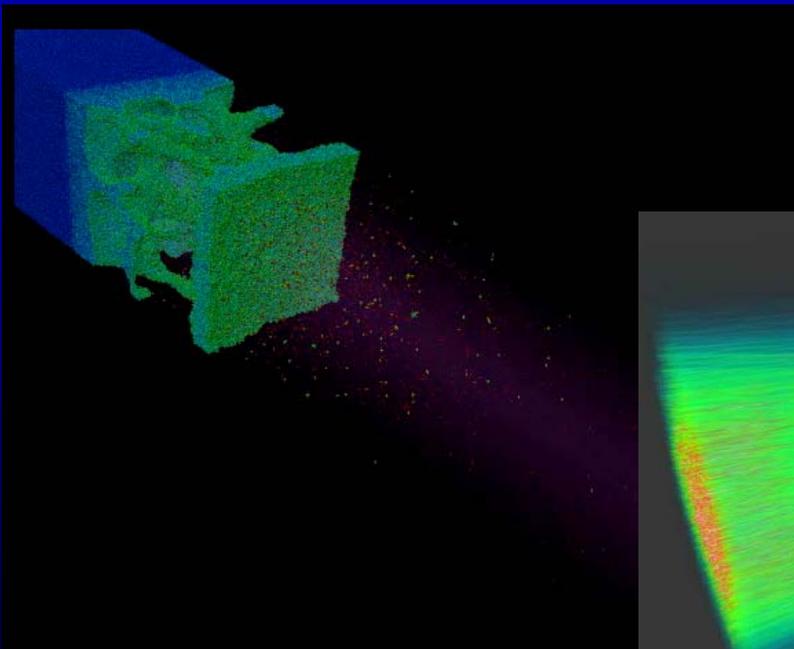
Development:

- To be released as OpenSource this month (SourceForge?)
- More apps and experimental hardware support



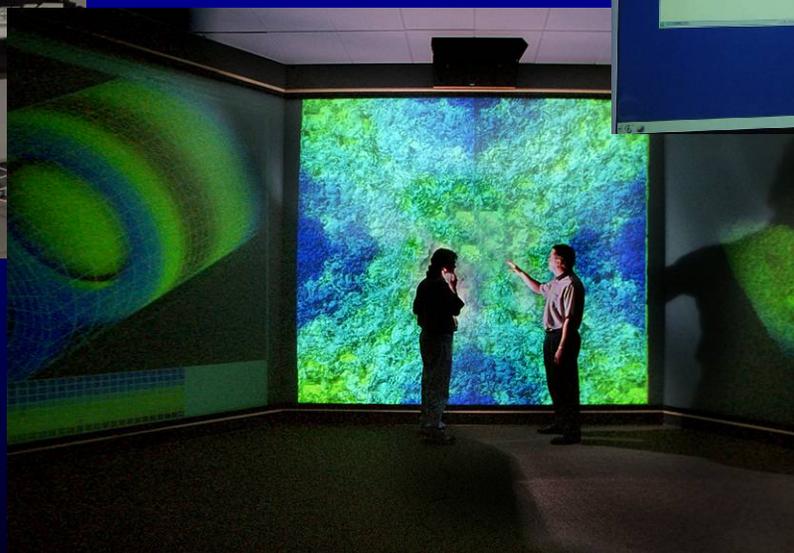
Examples: Systems in use Today...

- MCR+PVC science runs
 - Custom renderers
 - Hardware rendering



Examples: Systems in use Today...

- SGE: digitally driven displays
- SNL 64Mpixel wall surfaces
- DMX: desktop integration



Current Issues

- **Data access**
 - Cluster-wide parallel I/O systems can be fragile
 - Often not optimized for access patterns (e.g. random reads)
- **The impedance mismatch problem**
 - Smaller number of nodes generally for visualization
 - Improper data decomposition
- **Scheduling complexity**
 - Co-scheduling of multiple clusters
 - Combinations of parallel clients, servers, services and displays
- **Visualization/rendering algorithms issues**
 - Extreme dataset sizes and ranges
 - Complexity of visuals from large datasets

The Road Ahead

The COTS graphics cluster arena is dynamic

- Longevity challenge: software and hardware
 - New graphics bus (PCI Express)
 - Changes in video technologies
 - DVI to 10gigE (TeraBurst)
 - Next generation graphics cards
 - New rendering abstractions (are polygons dead?)
 - » How to address current card bottlenecks (e.g. setup)?
- The extreme FLOP approach
 - The NV30 and the Playstation 3

The Streaming Programming model

Streaming exposes concurrency and latency at the system level as part of the programming target

Data moves through the system: exposed concurrency

- Avoid global communication: prefer implicit models (e.g. Cr)

Memory model: exposed latency/bandwidth

- Scalable, must support very small footprints
- Distributed, implicit flow between each operation

A working model:

- Computational elements + caching and bandwidth constraints
- External "oracle" for system characterization and realization

Goals:

- Optimally trade off computation for critical bandwidth
- Leverage traditionally "hidden" programmable elements

Streaming Impacts on Software Design

How does one target this model?

- Integrated data structure and algorithm design
 - Run-time targets
 - Algorithm remapping
- “Intent” expressive and architecture aware
 - Abstract run-time compiled languages
- Memory design is key
 - “Small” memory models, out-of-core design
 - “Cache oblivious” data flow

Implementations

- New languages: Cg, Brook, DSP-C, Stream-C
- Hidden beneath layers of API: OpenGL, Chromium, Lustre

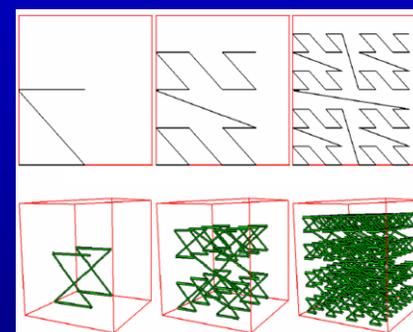
It sounds like a lot of work (it can be), is it worth it?

Multiresolution Array Access: VISUS

Arbitrary, multi-resolution array data access

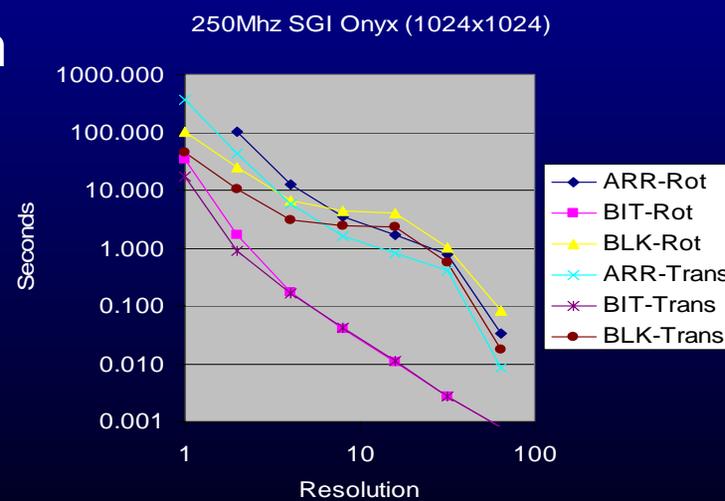
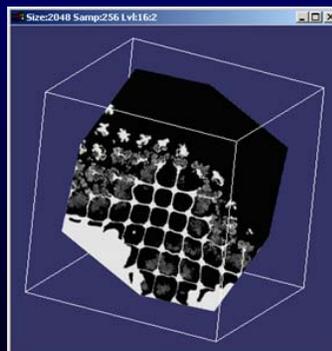
Integrated algorithm/data structure design

- Data “reordering” on a spacefilling curve
 - In-place transformation
 - Reordering is a simple bit manipulation
- Cache oblivious
 - Arbitrary blocking is supported
- Coupled asynchronous query system
 - Parallel rendering and queries
 - RAM used as a cache



Example

- Slicing 8B cells
- 15MB RAM



Next Generation Streaming Cluster...

- **Computation and memory caches everywhere**
 - NICs, Drive controllers, Switches, TFLOP GPUs
 - Add PCI Express and the GPU effectively becomes a DSP chip
 - Utilizing them may require a disruptive programming shift
- **Modified visualization algorithms**
 - Cache oblivious: local, at the expense of computation
 - Non-graphical algorithms & a move away from polygon primitives
 - Need to address data scaling and representation issues
- **New languages with higher levels of abstractions**
 - Run-time "realization", dynamic compilation and scheduling
 - Glue languages: "shader" languages, graphics APIs themselves

Abstract

The recent revolution in the commodity graphics cards coupled with advances in Linux and commodity clustering have enabled a new type of system, the Distributed Visualization Cluster. These systems have demonstrated unprecedented capabilities for both scientific visualization and even computation. In conjunction with tiled display wall technologies and scalable software infrastructures, they have shown to be capable of handling the largest visualization tasks at hand. Increasing vendor support and new graphics capabilities combine to expand the future vision for these systems well beyond their current capabilities. This talk will explore the evolution of these systems and their present forms and limitations. It will also discuss some current and future directions being explored by researchers and industry and how they might shape visualization enabled, Petaflop-class clusters on the horizon.

Auspices: UCRL-PRES-154718

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

