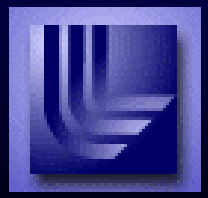
Visualization Environment Issues for Terascale Data

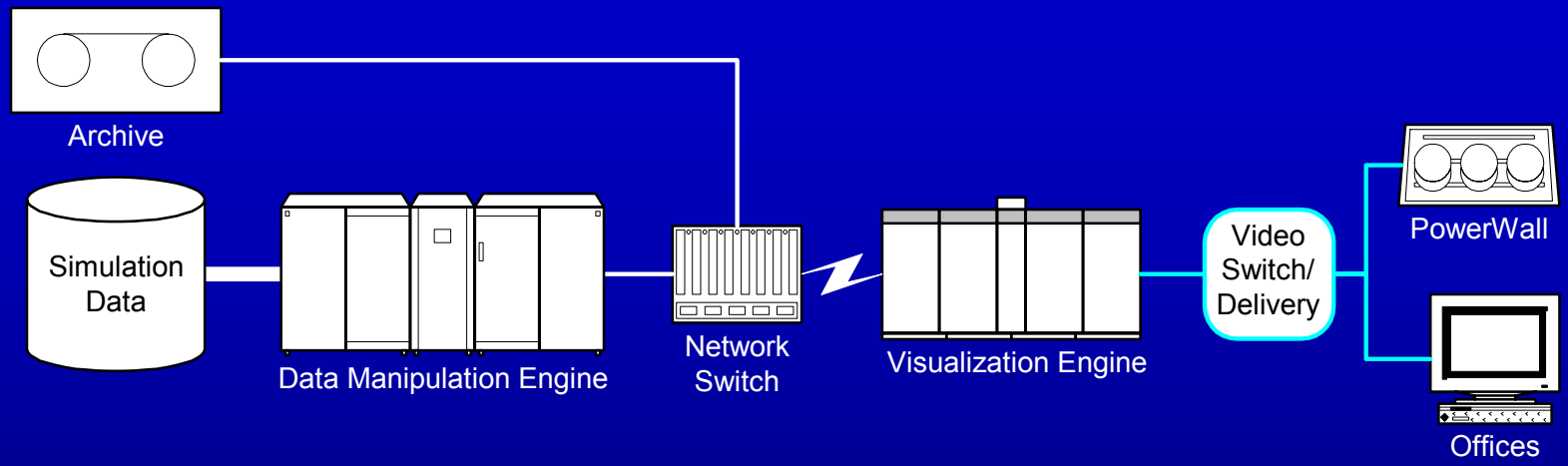
Randall Frank
Lawrence Livermore National Laboratory

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.

The Livermore Model: Visualization



- 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)

Why Distributed Viz Environment?

- **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**
  - In clusters, desktops or displays…

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!)

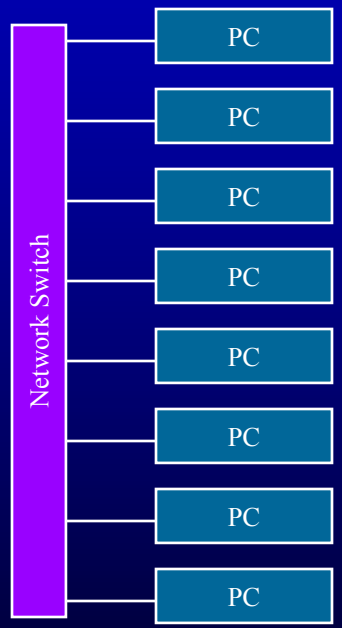
- Real-time imagery
- Interaction devices, human in the loop (latency, prediction issues)

Viz Engine: Compute Nodes

Computational elements upon which the visualization application runs

- Largely the same as the compute platform

- **Distributed nodes**
  - P4, Itanium, Opteron
  - SGI SMP "clusters"
- **Interconnects**
  - Quadrics, Myrinet, GigE
- **I/O systems**
  - Lustre, NFS, PVFS, …
  - Key for effective, production, visualization



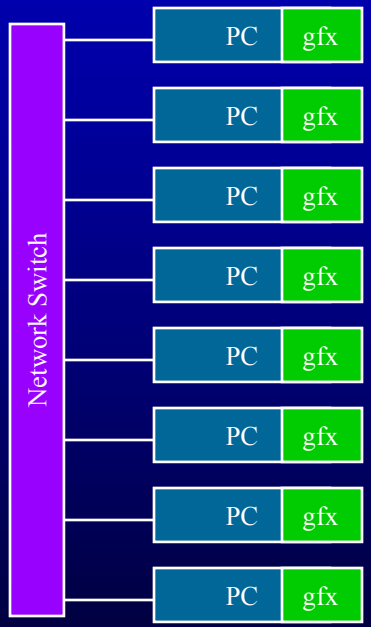
HP Visualization Cluster

Stanford "Chromium" Cluster

Viz Engine: Graphics Options

Add rendering capability to the nodes

- **Software OpenGL & Custom**
  - Mesa, qsplat, *-Ray
- **Hardware**
  - SGI IR pipes
  - COTS graphics
    - AGP or PCI Express slot
    - nVidia, ATI, …
  - Card device drivers
    - OS, Graphics API, etc



ATI FireGL X2

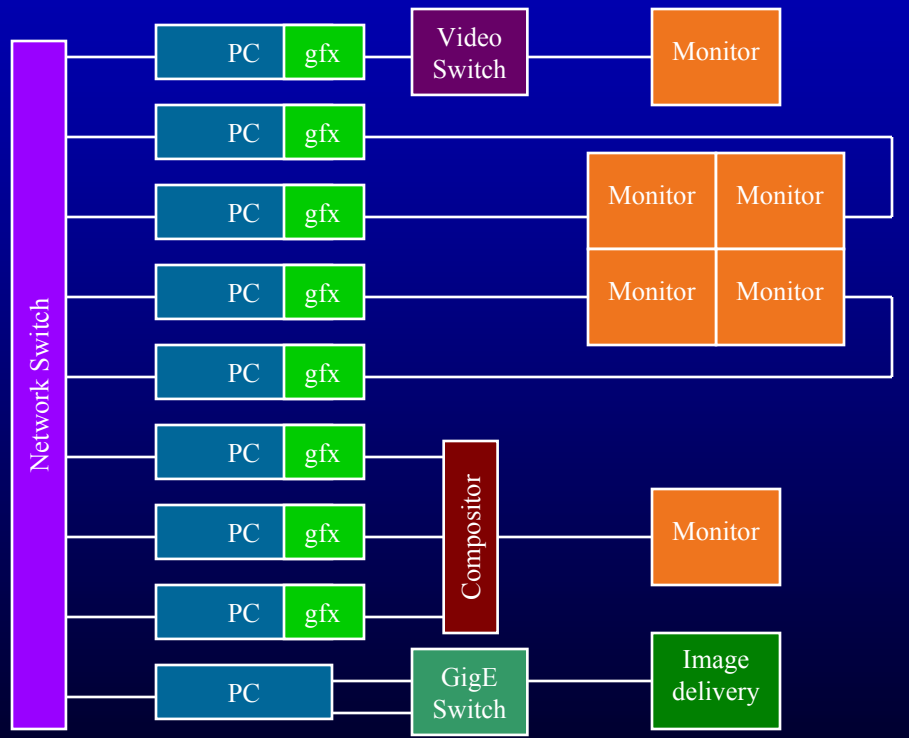
nVidia Quadro FX

Viz Engine: 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)



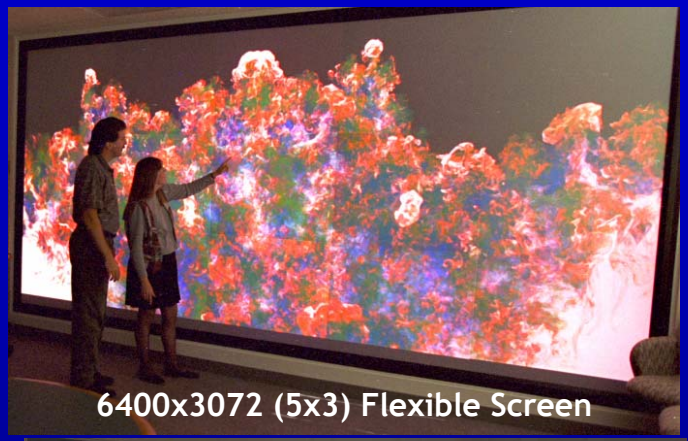
The PC Graphics Revolution…

- A disruptive technology shift
- Raw performance kings
  - $200 GeForce4 vs $80k IR pipe?
- "Interesting" feature kings
  - Creative bandwidth management
  - Fully programmable architectures
- However…
  - Certainly a focus on gaming and DCC markets
  - Relatively crude video support: stereo, sync, etc
  - Can they make it past the conceptual IR ceiling?

Quake III Arena

The Tiled Display PowerWall

- **Many forms…**
  - Stereo, Cubes, Front/Back projection, Non-planar, Edge-blended, CAVEs
- **Multiple uses**
  - Collaborative environments, Theaters, Enhanced desktop/interactive use
- **Increased pixel counts**
  - Matching higher fidelity data (2D vs 3D)
- **Driving a PowerWall**
  - Extreme I/O requirements
    - 2x2 needed 300MB/s
  - Synchronization (at a distance?)
  - Data flow and data staging
  - Requires output scalable image generation via aggregation



6400x3072 (5x3) Flexible Screen

5120x2048 (4x2) Glass Screen

3840x2048 (3x2) Cubes

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/UT)
- sv6/sv7 (HP)/SGI compositor
  - DVI based tiling/compositing
- Sepia (HP/Compaq)
  - Custom compositing (FPGA + NIC)
  - Dedicated network (ServerNet II & IB)
- Scalable Graphics Engine (IBM)
  - Remote framebuffer, gigE/UDP distance solution

Image composition software

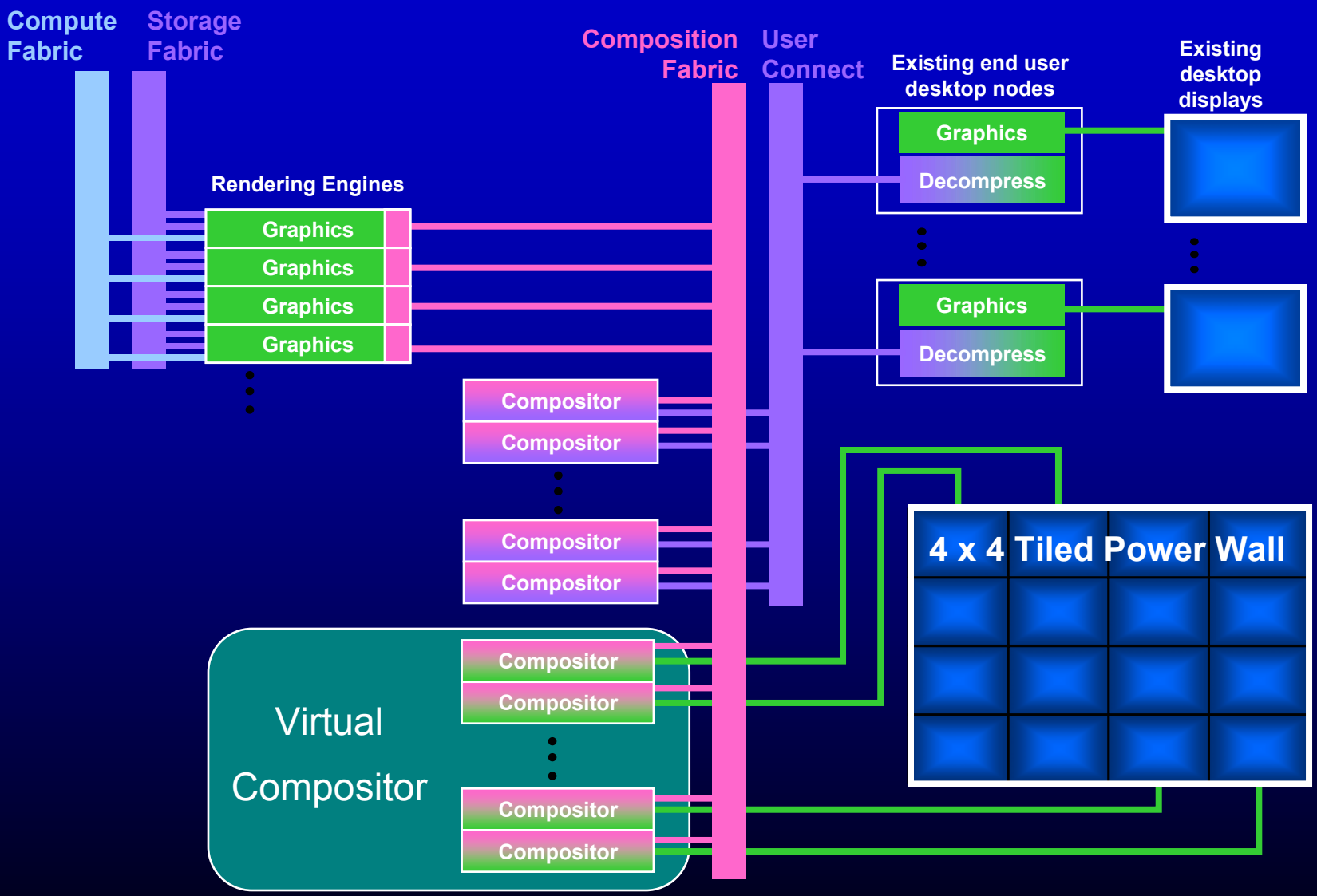
- PICA: Parallel Image Compositing API
- ICE-T: Integrated image/data manipulation (SNL)

Lightning-2

HP Sepia-2

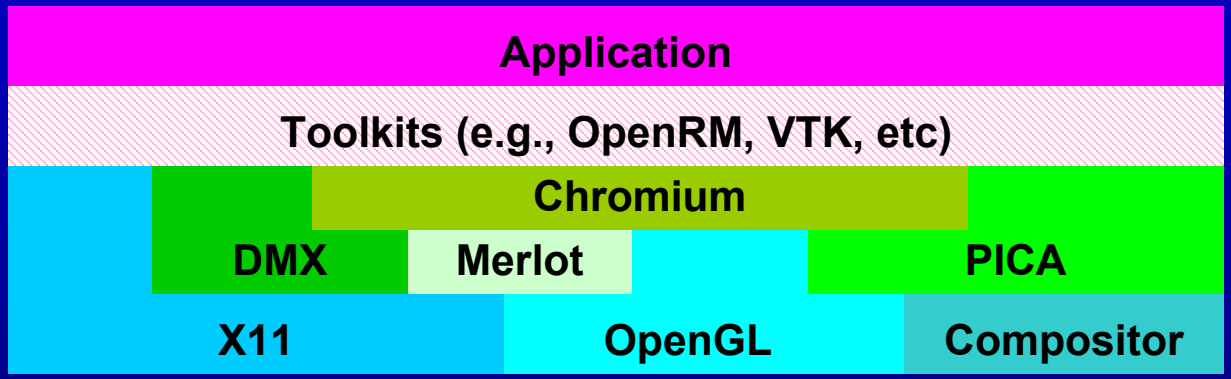
IBM SGE

Idealized Visualization Environment



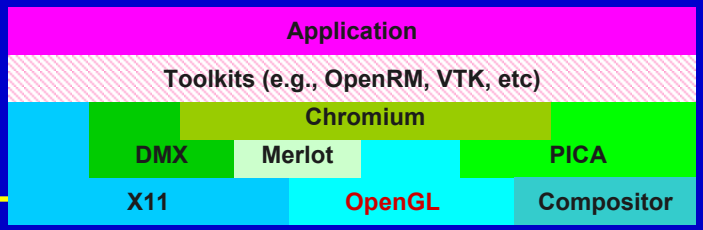
A Distributed Parallel API Stack

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



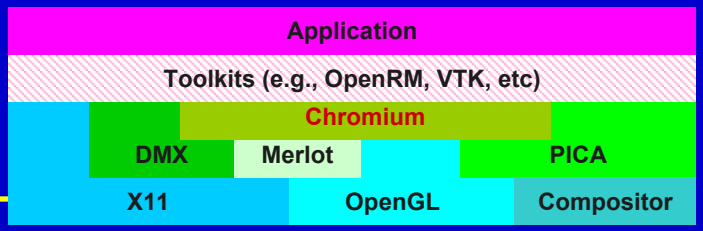
- Applications: VisIt, ParaView, EnSight, Blockbuster, etc…
- Toolkits – Parallel visualization algorithms, scene graphs…
- DMX – Distributed windowing and input devices (X11)
- Chromium – Parallel OpenGL rendering model
- PICA – Parallel image compositing API
- Merlot – Digital image delivery infrastructure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

OpenGL Drivers



- **Some early DirectX/OpenGL concerns**
  - Slow ARB, Lack of innovation, Questionable support
- **Mostly addressed**
  - Welcome to the world of extensions and competition…
- **Linux/COTS OpenGL drivers are looking good**
  - 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

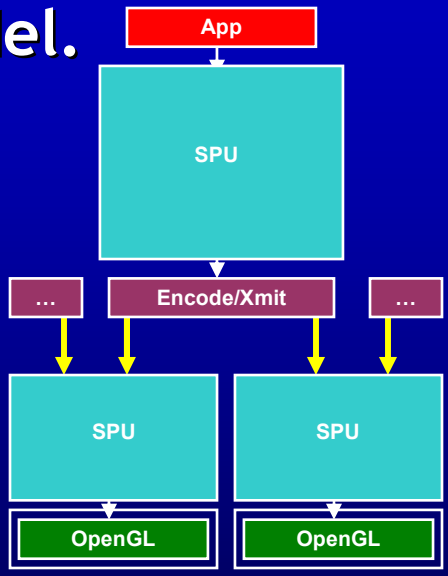
Distributed GL: Chromium



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

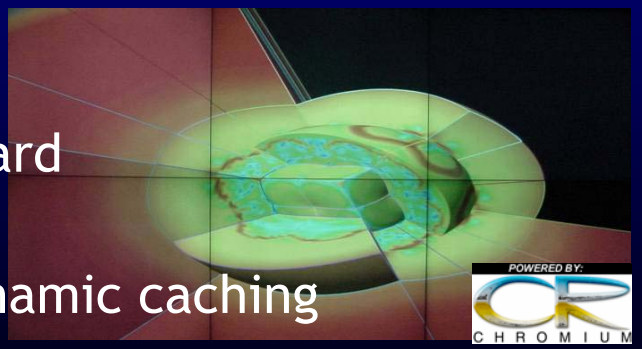
The Stream Processing Unit (SPU)

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

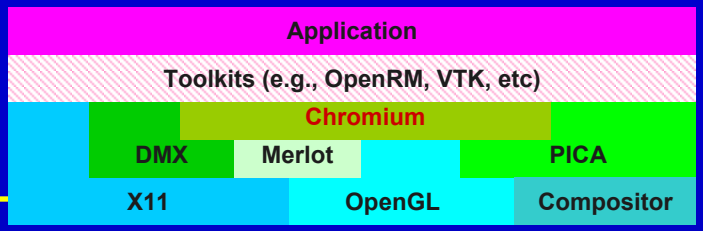


Development:

- chromium.sourceforge.net
- RedHat/Tungsten Graphics ASCI PathForward
- Stanford, University of Virginia
- Stereo, Fragment/Vertex pgms, CRUT, dynamic caching



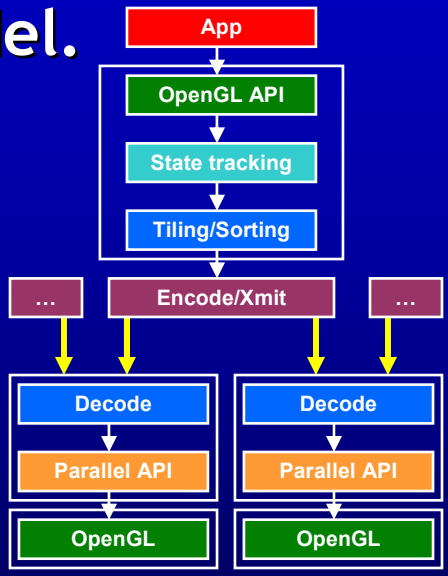
Distributed GL: Chromium



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

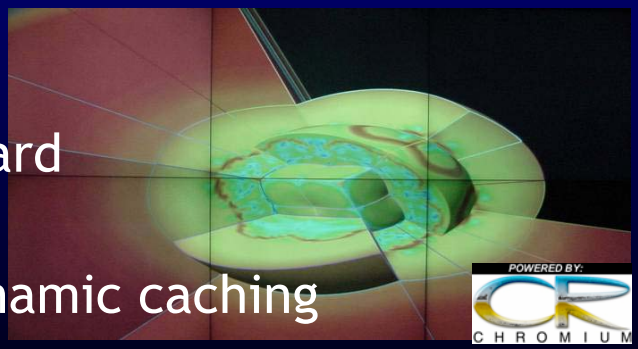
The Stream Processing Unit (SPU)

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

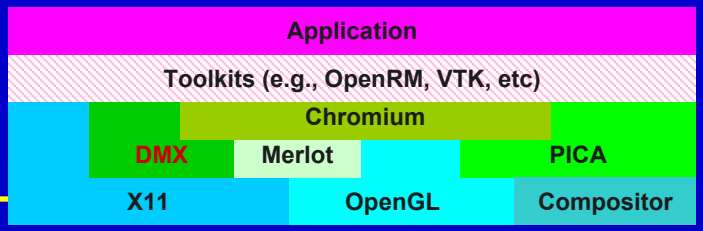


Development:

- chromium.sourceforge.net
- RedHat/Tungsten Graphics ASCI PathForward
- Stanford, University of Virginia
- Stereo, Fragment/Vertex pgms, CRUT, dynamic caching

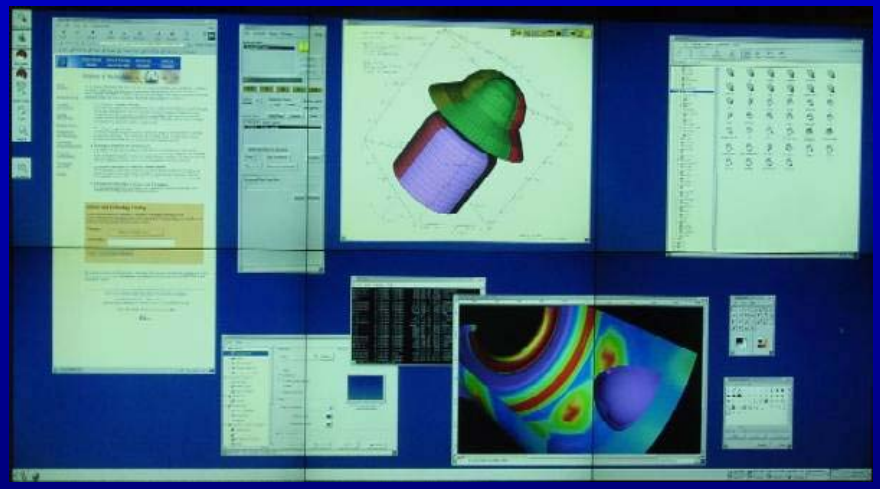


Parallel X11 Server: DMX



Distributed multi-headed X server: DMX

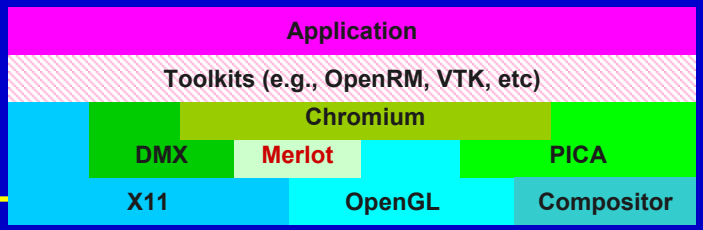
- **Aggregates X11 servers**
  - "Server of servers" for X11
  - Single X server interface
- **Accelerated graphics**
  - 2D via accelerated X server
    - Common extensions as well
  - Back-side APIs for direct, local X11 server access
  - OpenGL via ProxyGL/GLX (from SGI) or via Chromium SPU



Development: dmx.sourceforge.net

- RedHat ASCI PathForward contract
- Integrated with XFree86

Remote Delivery: Merlot

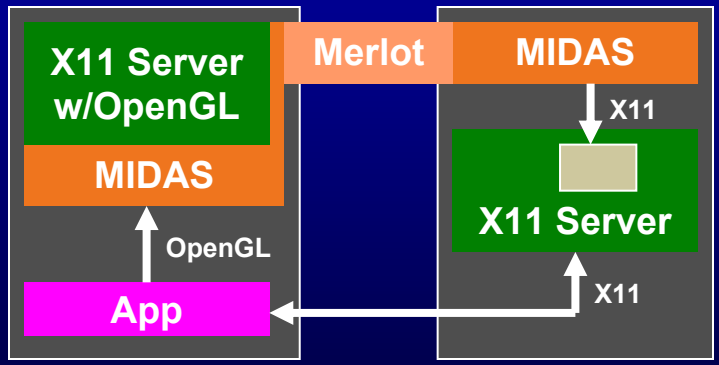


Merlot is a framework for digital image delivery

- Transport layer abstraction, Codec interfaces, Device transparency

MIDAS: Merlot Image Delivery Application Service

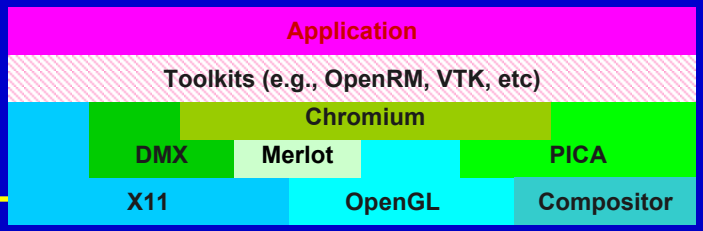
- Indirect OpenGL rendering services for X11 environment
- Indirect window management
- Image stream transport



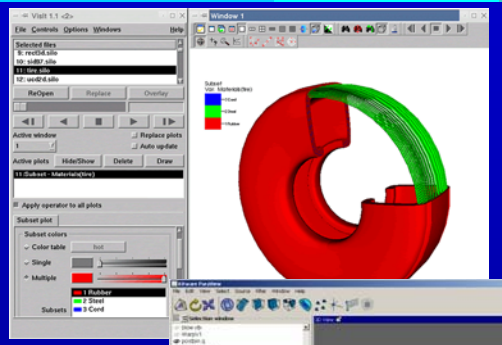
Development:

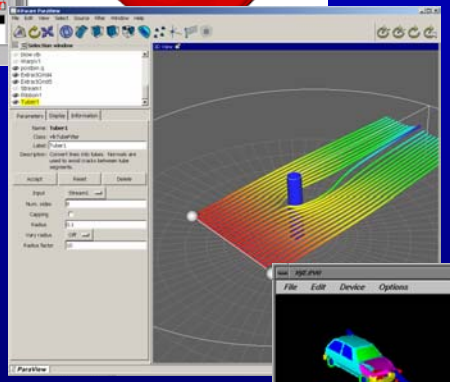
- To be released as OpenSource this month (SourceForge?)
- More apps and experimental hardware support

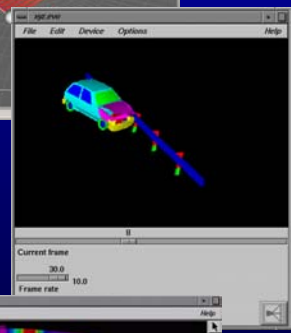
Applications

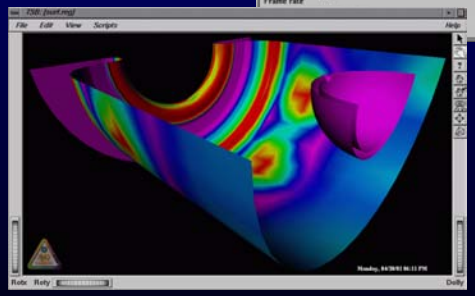


- **Full-featured viz**
  - VisIt: www.llnl.gov/visit
    - VTK, client-server model
  - ParaView: www.paraview.org
    - Parallel VTK viz tool
- **Specialty applications**
  - Blockbuster: blockbuster.sourceforge.net
    - Scalable animations, DMX aware
  - TeraScale Browser/Xmovie/MIDAS…
    - www.llnl.gov/icc/sdd/img/viz.shtml





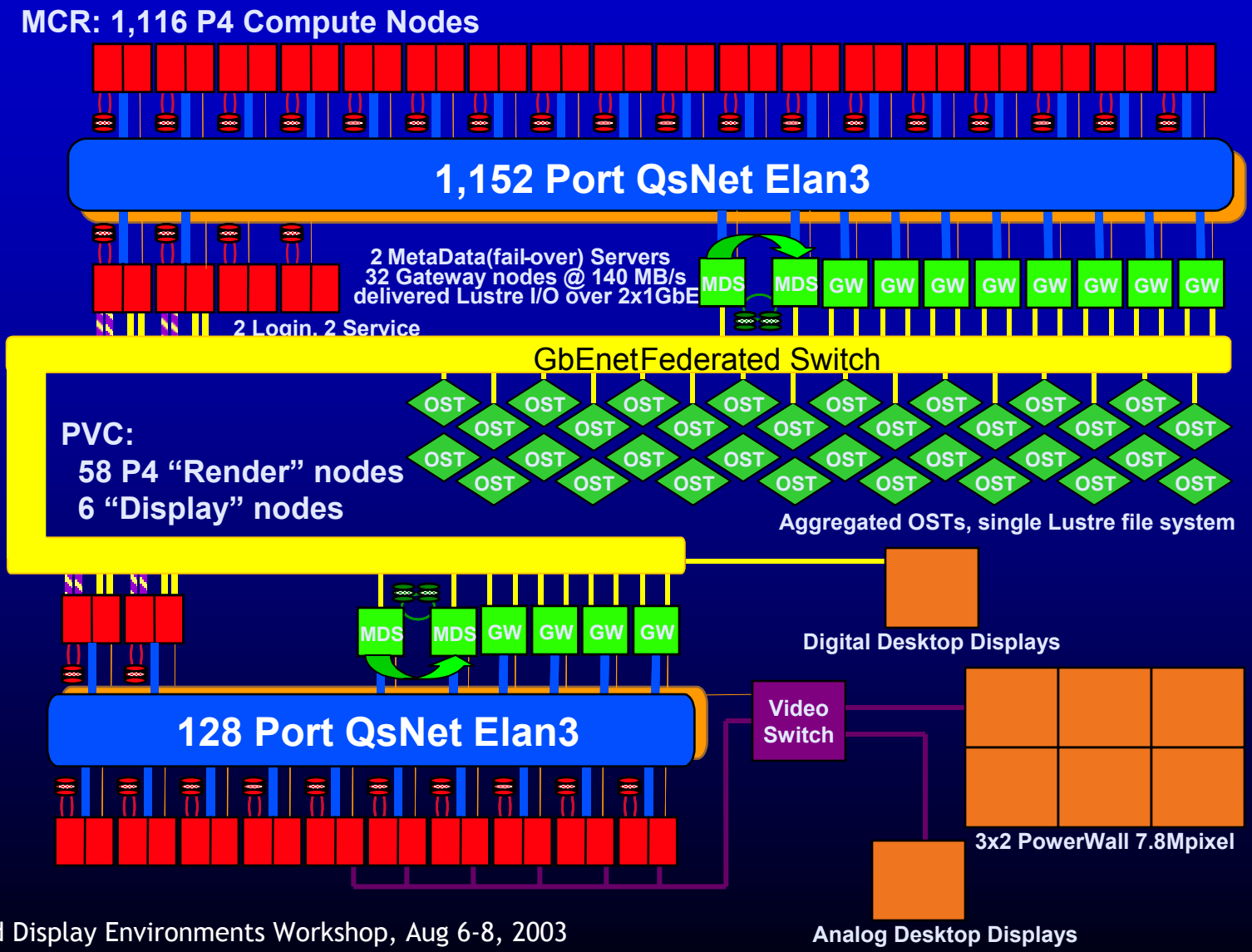




Deployed Viz Environment: ASCI White

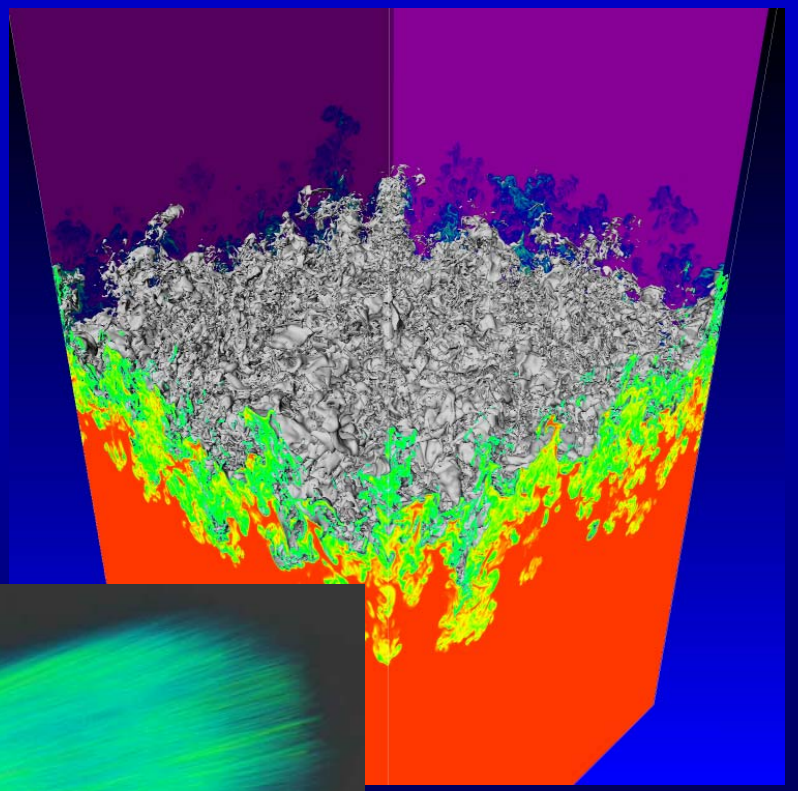
- **Data manipulation**
  - 512p on White (32 nodes)
  - 96p SGI Onyx3
- **Networking**
  - Multi-gigE, Jumbo frames
- **Rendering engines**
  - 40p and 64p SGI Onyx2, 10 and 16 IR2 pipes
- **Video delivery**
  - Lightwave switch/modems
  - ~20 Desktops (1280x1024, 1920x1200, 2560x2048)
  - PowerWalls (5x3: 19.6Mpixel, 4x2: 10Mpixel)

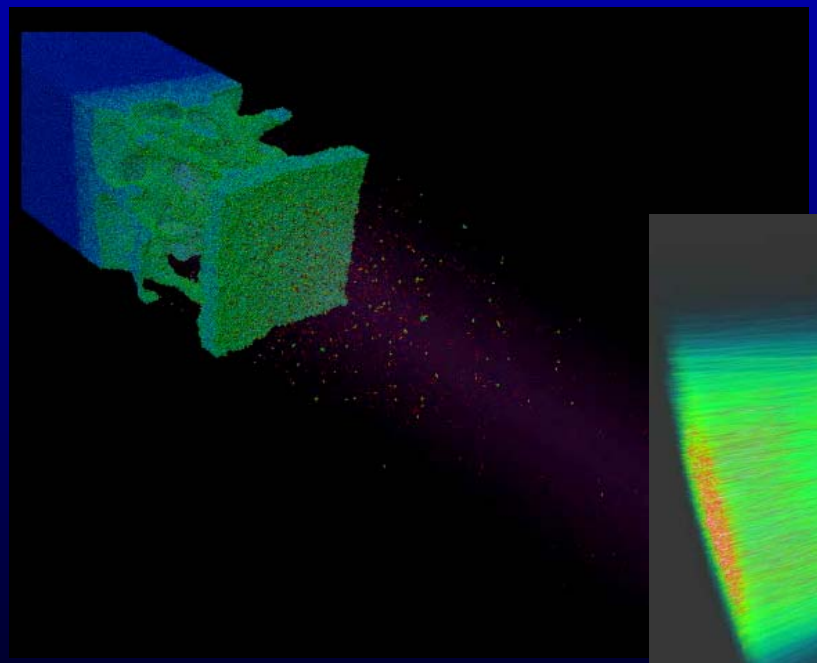
Deployed Viz Environment: MCR & PVC

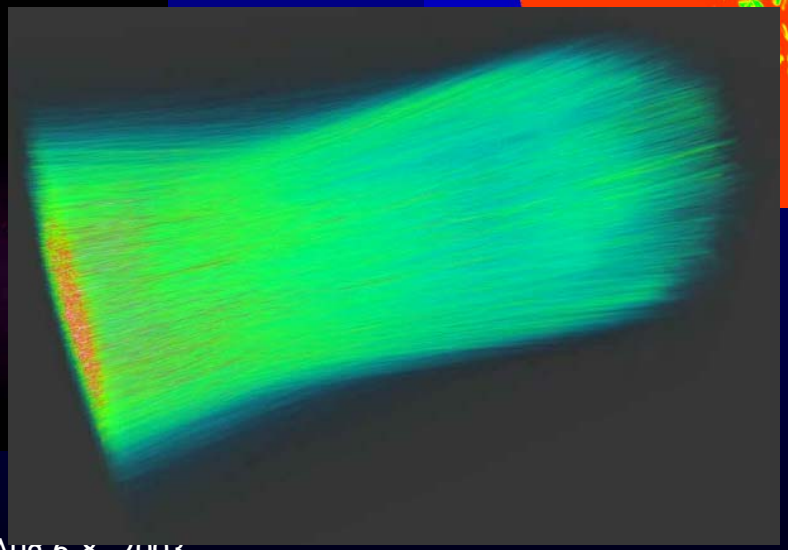


Examples: Systems in use Today…

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

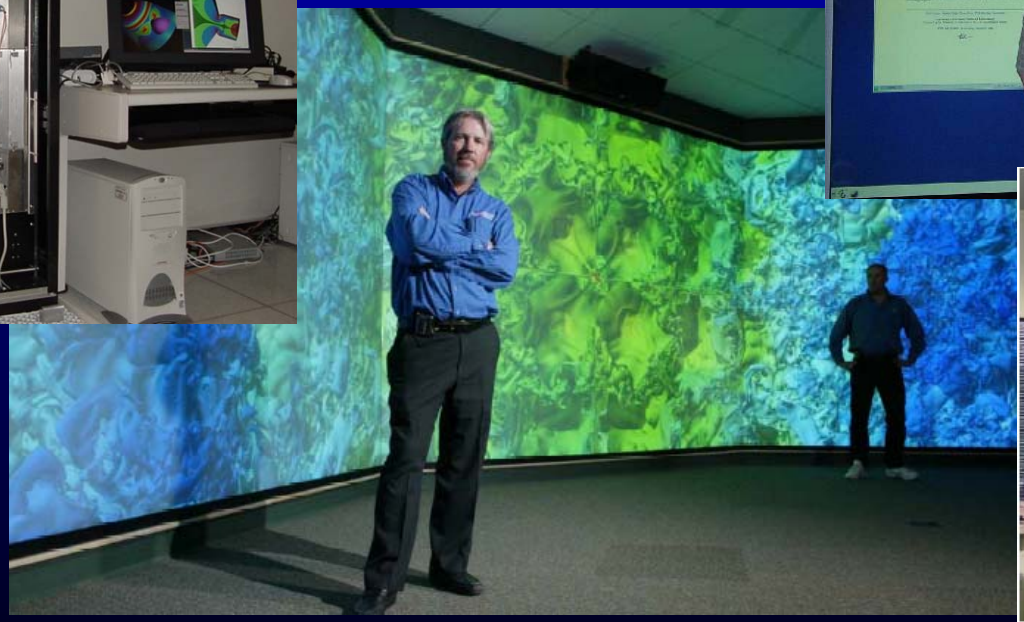


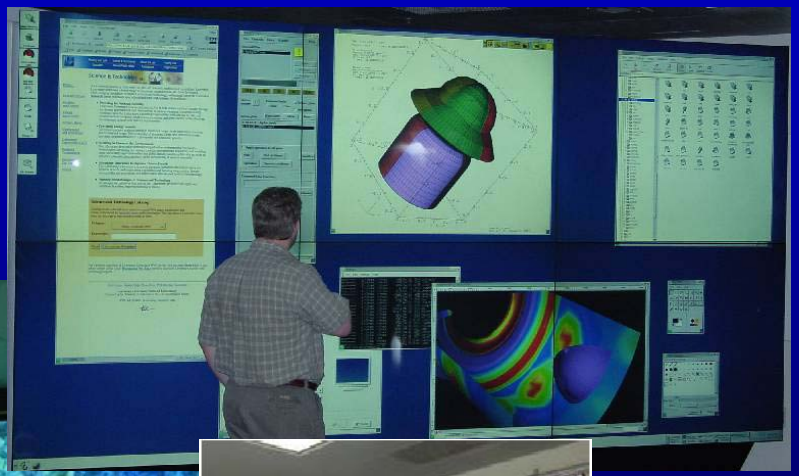




Examples: Systems in use Today…

- SGE: digitally driven displays
- SNL 62Mpixel 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

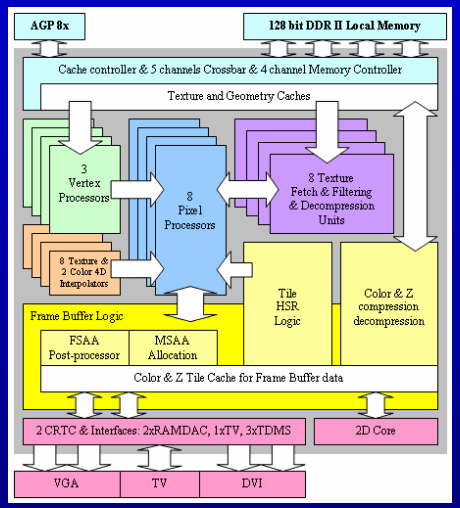
The arena is very dynamic

- 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)?
- Big data and failing algorithms
  - Scaling and representational issues
- The extreme FLOP approach…

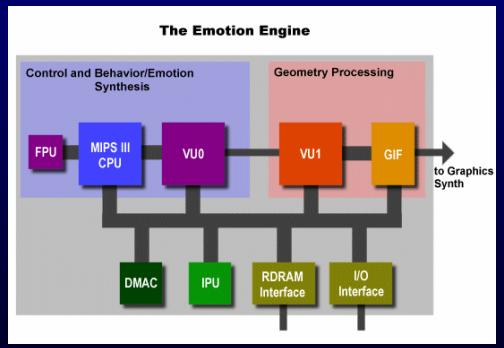
The NV30 and the Sony Playstation 3

Are graphics trends a glimpse of the future?

- **The nVidia NV30 Architecture**
  - 256MB+ RAM, 96 32bit IEEE FP units @ 500Mhz
  - "Assembly language" for custom operations
  - Streaming internal infrastructure
- **The PlayStation3 (patent application)**
  - Core component is a cell
    - 1 "PowerPC" CPU + 8 APUs ("vectorial" processors)
    - 4GHz, 128K RAM, 256GFLOP/cell
    - Building block for multimedia framework
  - Multiple cells
    - Four cell architecture (1TFLOP)
    - Central 64MB memory
    - Switched 1024 bit bus, optical links?



nVidia NV30



Sony PS2 "Emotion" Engine

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

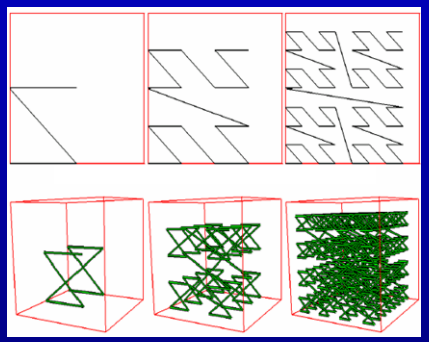
It sounds like a lot of work (it can be), is it worth it?

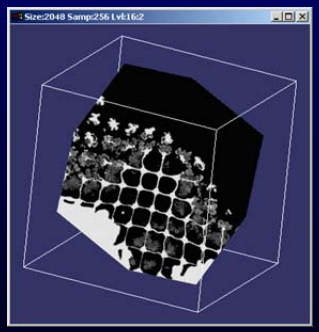
Multiresolution Array Access: VISUS

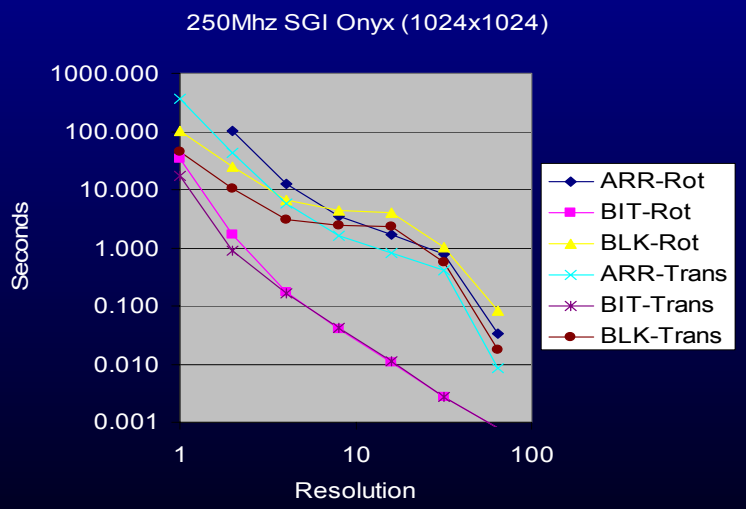
Arbitrary, multi-resolution array data access

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

Auspices: UCRL-PRES-154718

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.