Chromium for Cluster Rendering

Brian Paul

Tungsten Graphics, Inc.

Session Outline (1)

Part 1: Introduction and architecture

- What is Chromium?
- What can I do with it?
- How does it work?
- Simple examples
- Chromium system components and organization
- Q&A

Session Outline (2)

Part 2: Parallel rendering

- Sort-first rendering with Cr
- Sort-last rendering with Cr
- Performance and rendering issues
- Parallel rendering programming:
  - Synchronization
  - Input event propagation
- Q&A

Session Outline (3)

Part 3: Chromium in practice

- Production environment issues
- Ease of use
- Configuration
- Troubleshooting
- Sci-Vis applications
- Future developments
- Q&A

Part 1: Introduction and Architecture

What is Chromium?

Chromium is a special implementation of OpenGL for parallel/cluster rendering:

- Looks like an ordinary OpenGL library to applications
- Has special extensions / functions for parallel rendering

It runs on most popular flavors of Unix and Windows.

Open-source project on SourceForge.

Initially developed by Greg Humphreys, et. al. at Stanford University as a follow-on to WireGL. Subsequent work by Tungsten Graphics, Red Hat and others.

Chromium Principles

Take advantage of commodity hardware

- Low cost
- Fast product cycles

Flexibility:

- Chromium provides the building blocks
- You assemble them as needed
- Try not to impose a particular rendering architecture

Stream Processing:

- A natural extension of the graphics pipeline
- Allows for parallelism

What can I do with Chromium?

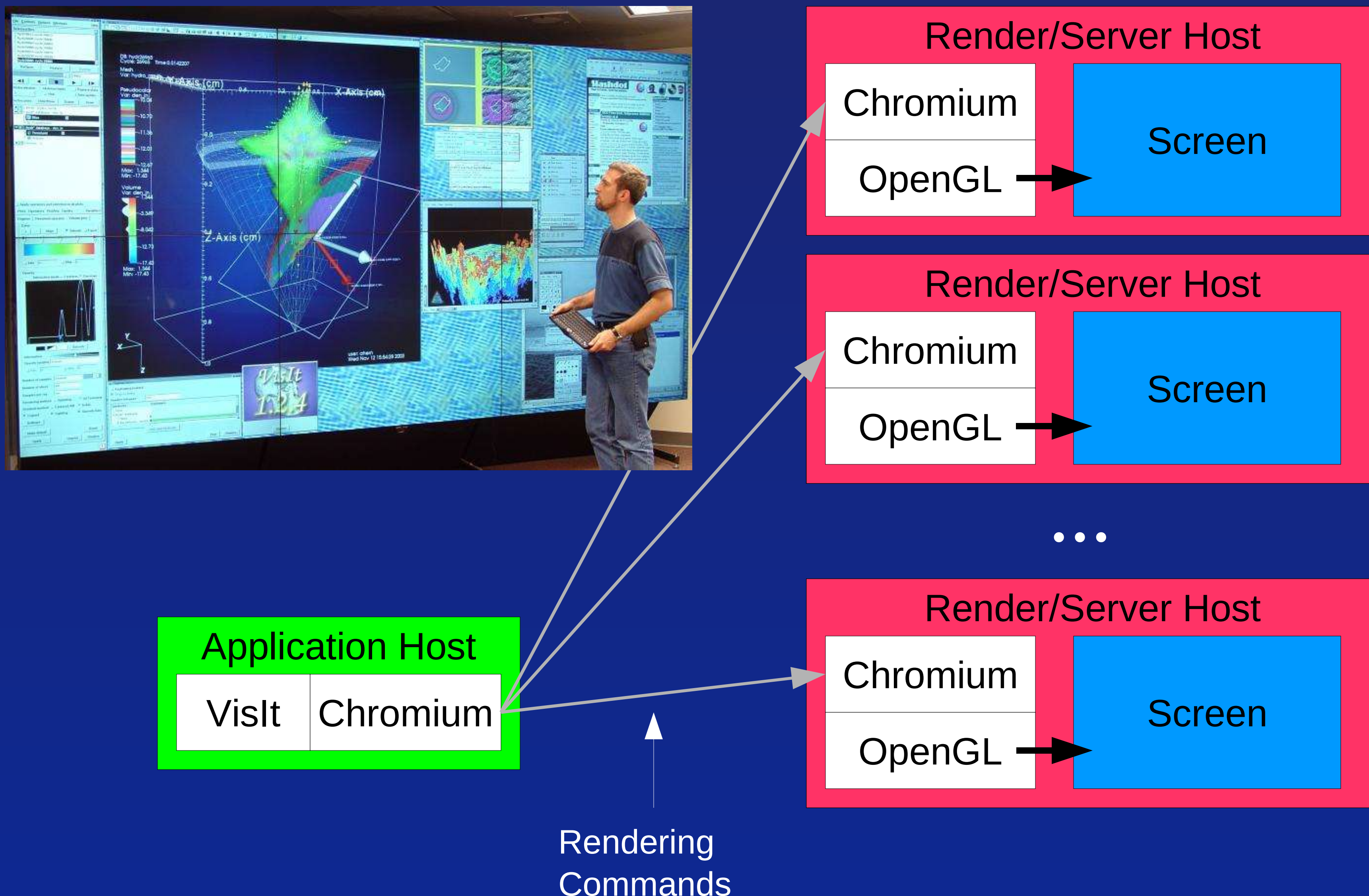
Three basic catagories:

- Run unmodified OpenGL applications on large, multi-screen, mural display walls. Or, on high-res displays like the IBM T221.
- Parallel rendering: both sort-first (image tiling) and sort-last (Z or alpha-based image compositing).
- OpenGL command stream filtering (manipulate OpenGL commands on their way from the application to the OpenGL renderer).

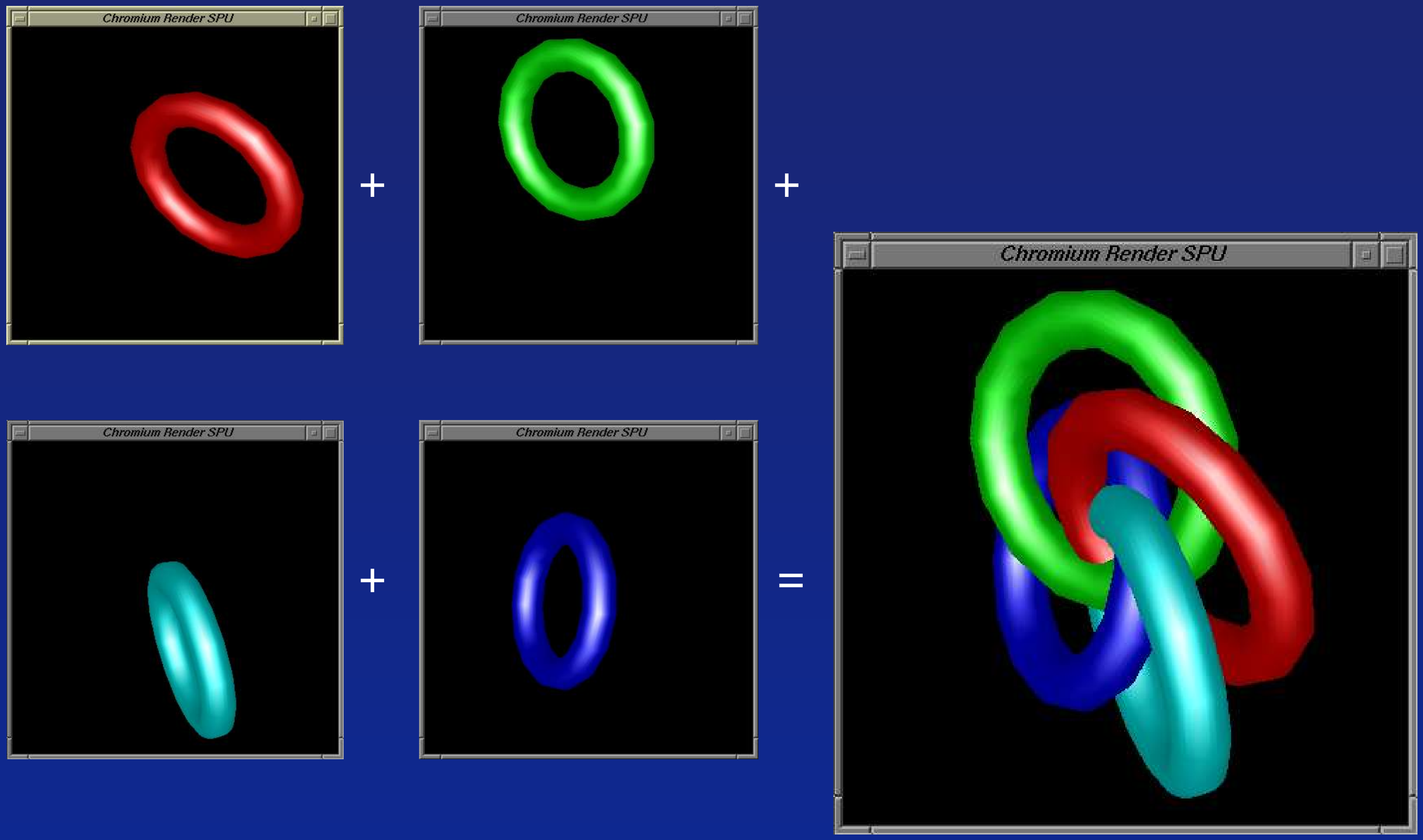
How does Chromium work?

- Intercept the application's OpenGL calls.
- Send the GL commands through a graph (DAG) of processing nodes.
- A processing node may split/distribute the GL command stream, merge several incoming streams, render the stream, or modify the stream.
- Chromium components are building blocks that can be put together in many ways. The configuration is described by a Python program.
- Some similarity to AVS, OpenDX, Khoros and other data-flow systems, but lower-level.

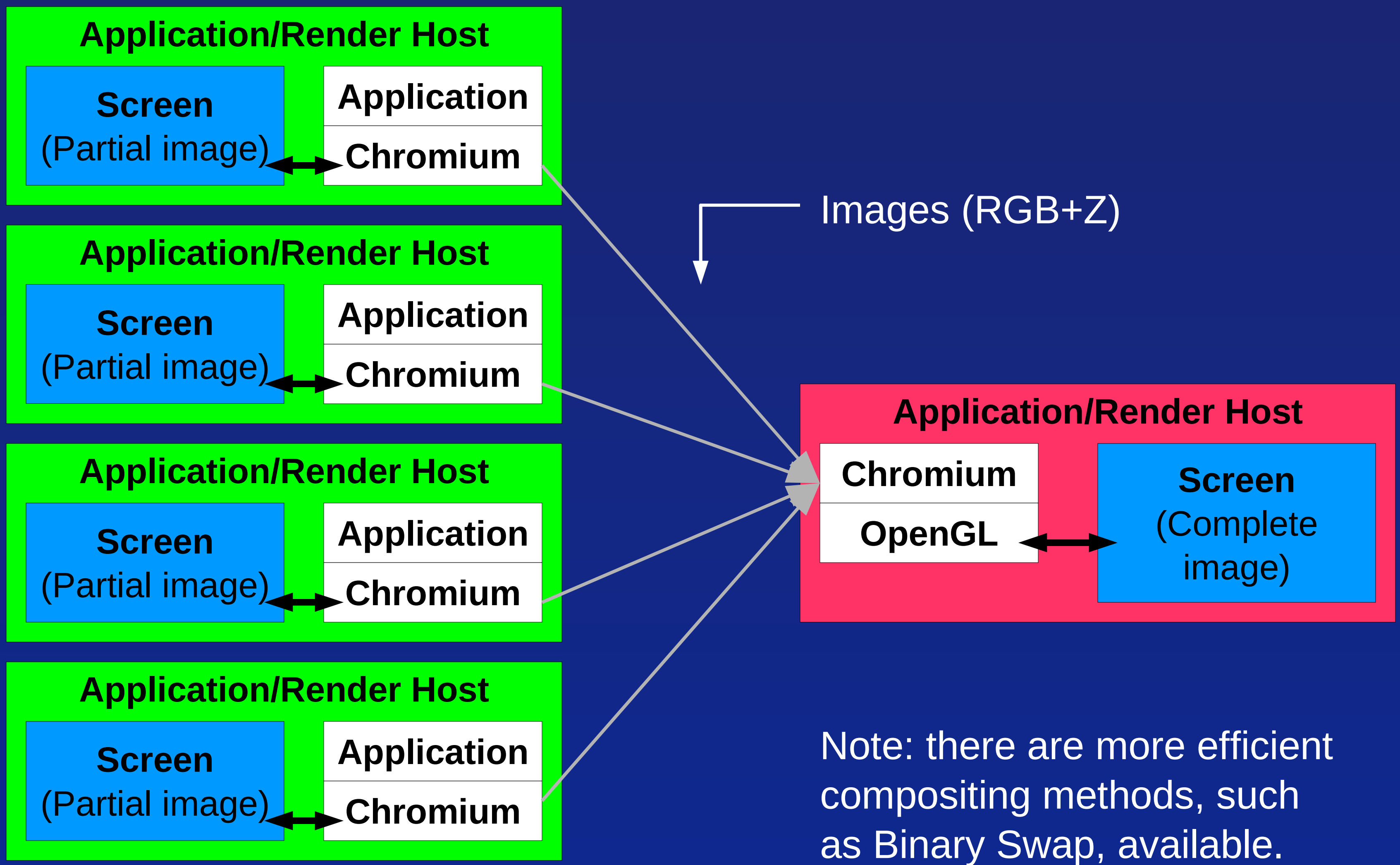
Example 1: Tiled Mural Display



Example 2: Parallel Compositing (1)

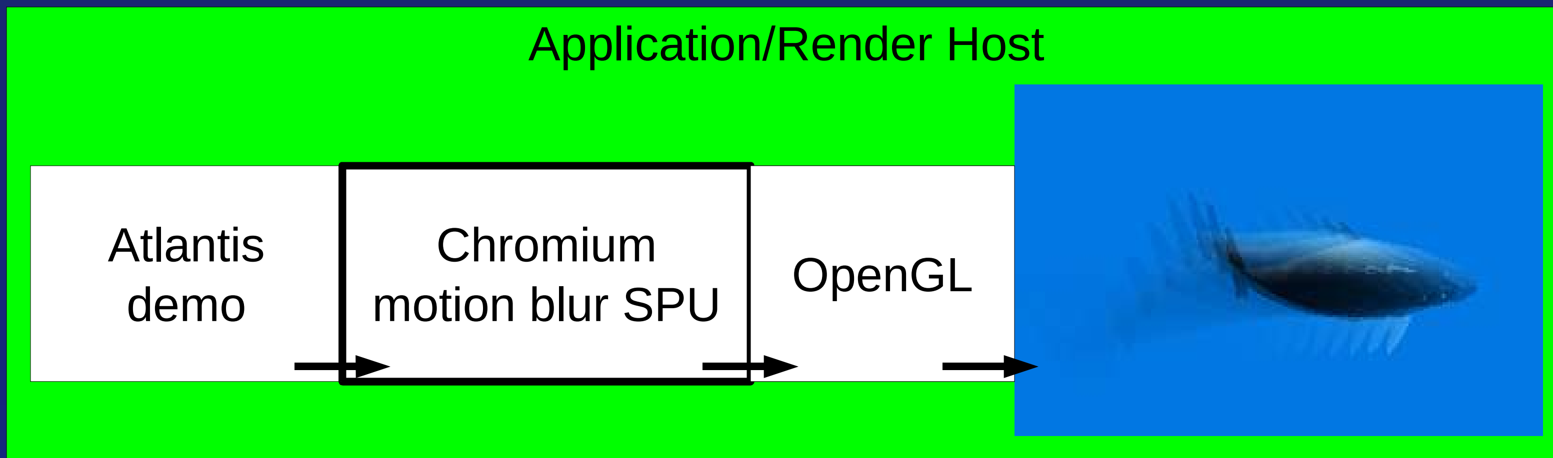


Example 2: Parallel Compositing (2)



Note: there are more efficient compositing methods, such as Binary Swap, available.

Example 3: OpenGL Command Filter



A Stream Processing Unit (SPU) intercepts and modifies the OpenGL commands on their way to the graphics hardware.

In this case, SwapBuffers() is intercepted to do some accumulation buffer operations.

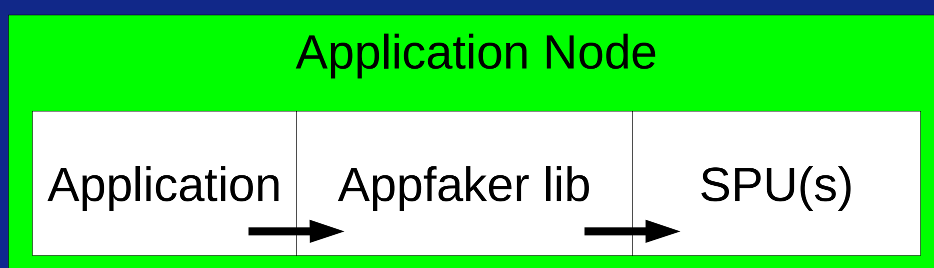
Chromium System Components (1)

Each node in the cluster will either act as a Chromium **application node** or **network (server) node**.

Both application nodes and network nodes host a chain of **Stream Processing Units** (SPUs).

Application Node

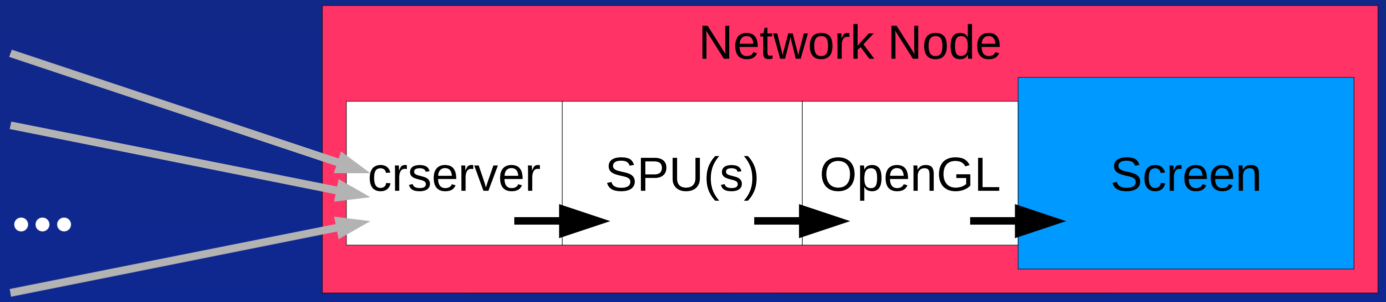
- Hosts the application. The “appfaker” library intercepts OpenGL commands and passes them to the first SPU in its SPU chain.
- A “Source” of rendering commands.



Chromium System Components (2)

Network (aka server) Node

- Hosts a “crserver” process which receives commands from a(n) upstream node(s) and passes them to the first SPU in its SPU chain.
- A “Sink” for rendering commands.
- The SPU chain typically ends with a “Render SPU” which passes the commands to OpenGL for rendering.
- When there are several incoming streams, the network node serializes them, doing context switching as needed.
- We can constrain the serialization to implement synchronization.



Chromium System Components (3)

Stream Processing Units (SPUs)

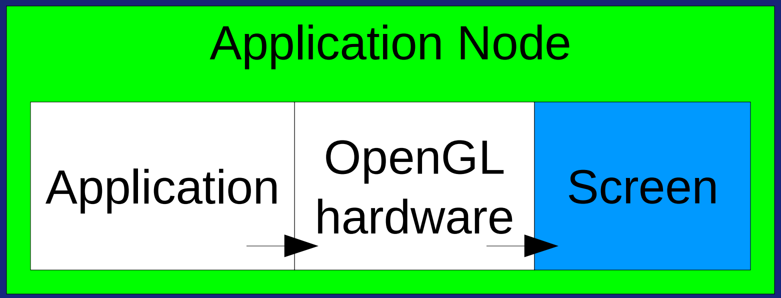
- Manipulates OpenGL commands, or renders them, or packs them into buffers for transmission to network nodes, etc.
- Implemented as dynamic shared libraries.
- About 20 different SPUs in Chromium today.

Mothership

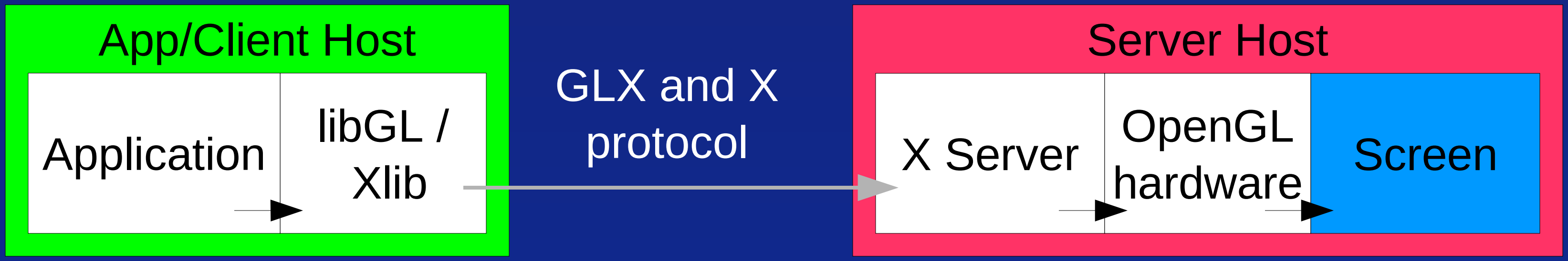
- A Python program/process responsible for system configuration.
- Describes the arrangement of application/network nodes, the SPU chain on each node and the configuration options for everything.
- Chromium nodes and SPUs talk to the mothership to configure themselves and learn about the other components.
- The Python program/script may be hand-written or generated by a graphical configuration tool.
- The mothership can fire-up all the other components.

Ordinary OpenGL Operation

First, let's look at how an ordinary OpenGL application works:

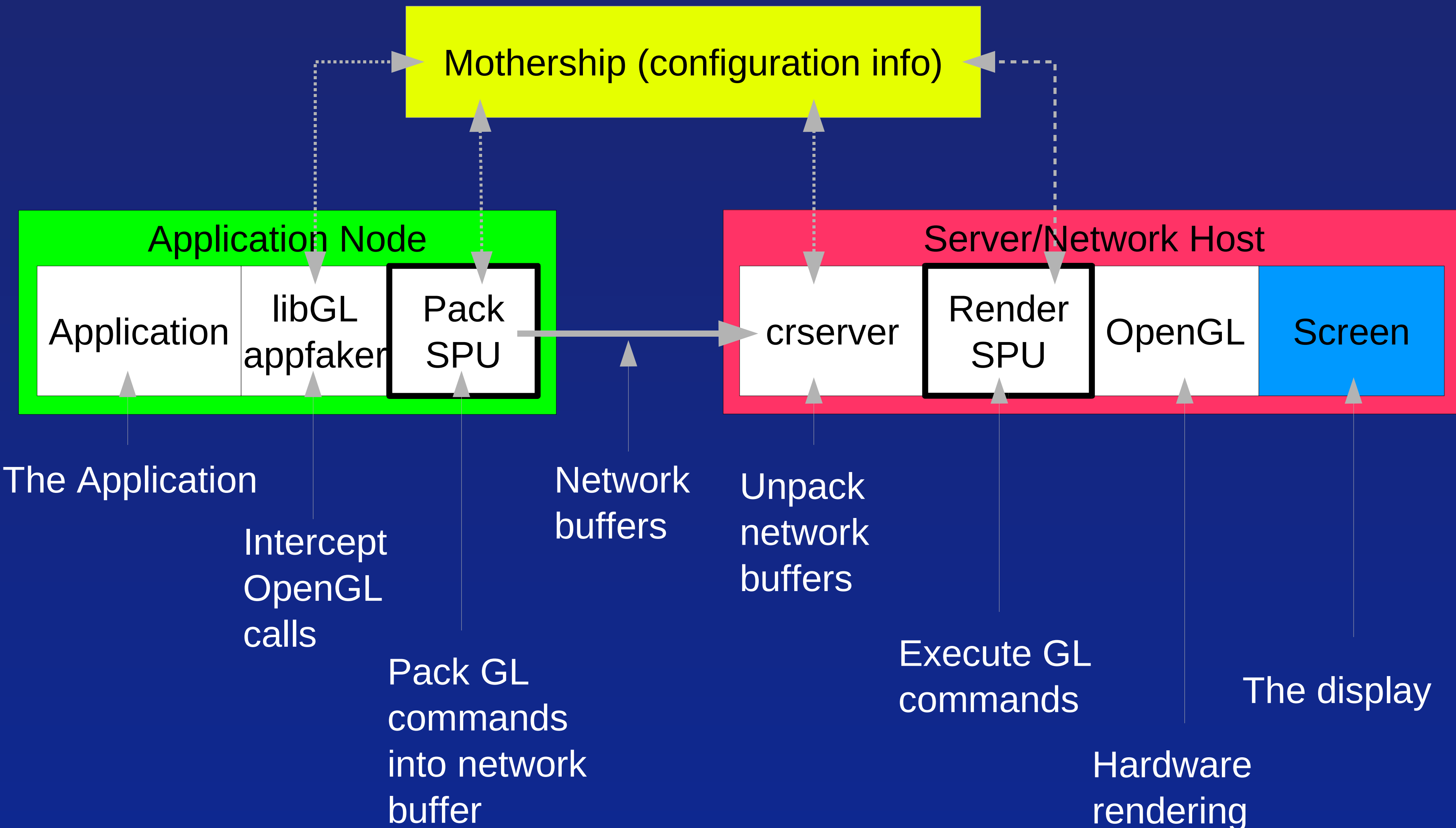


And here's how remote rendering with GLX works:

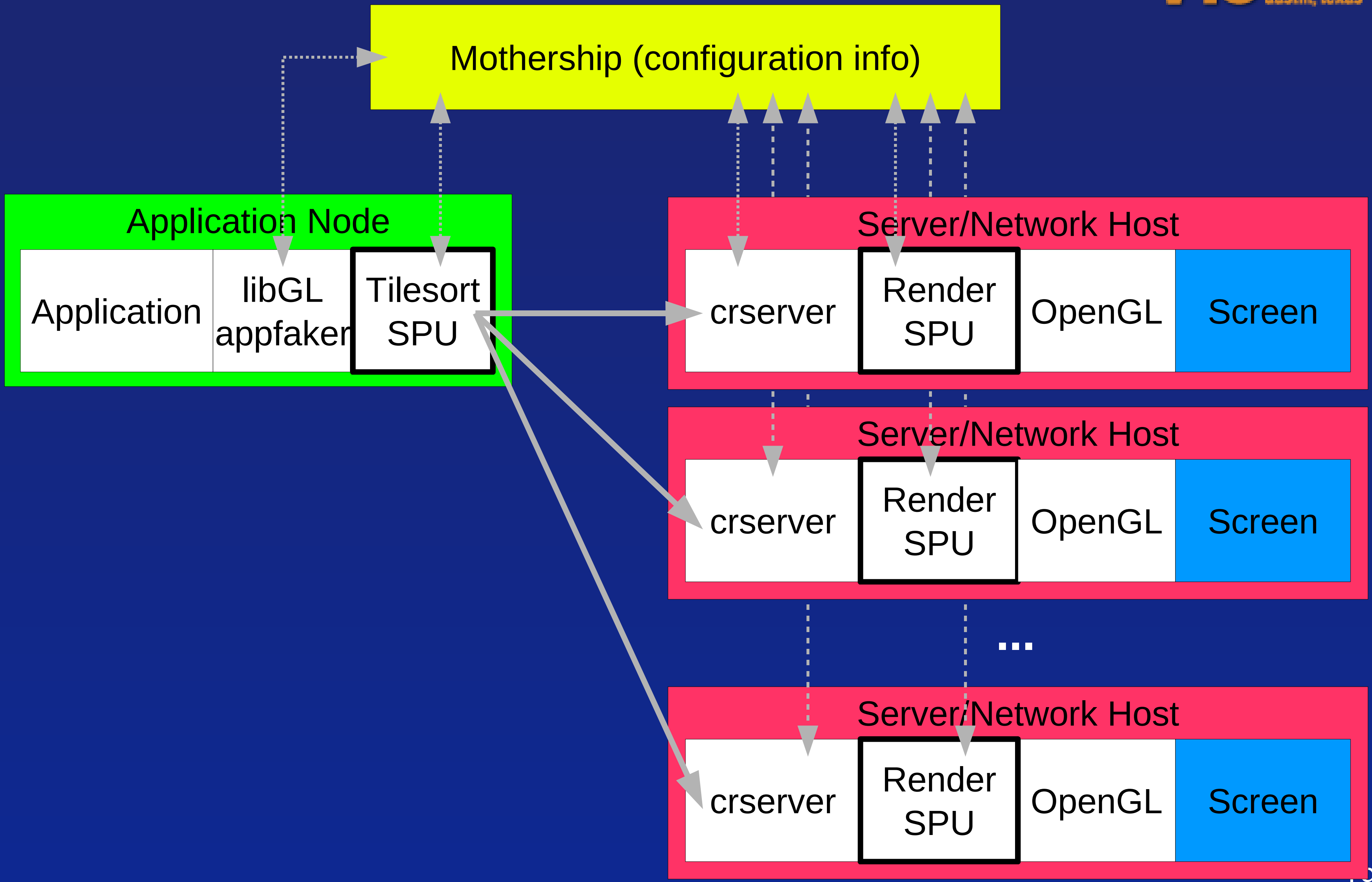


Next, the corresponding Chromium configuration...

Chromium GLX-like Remote Rendering

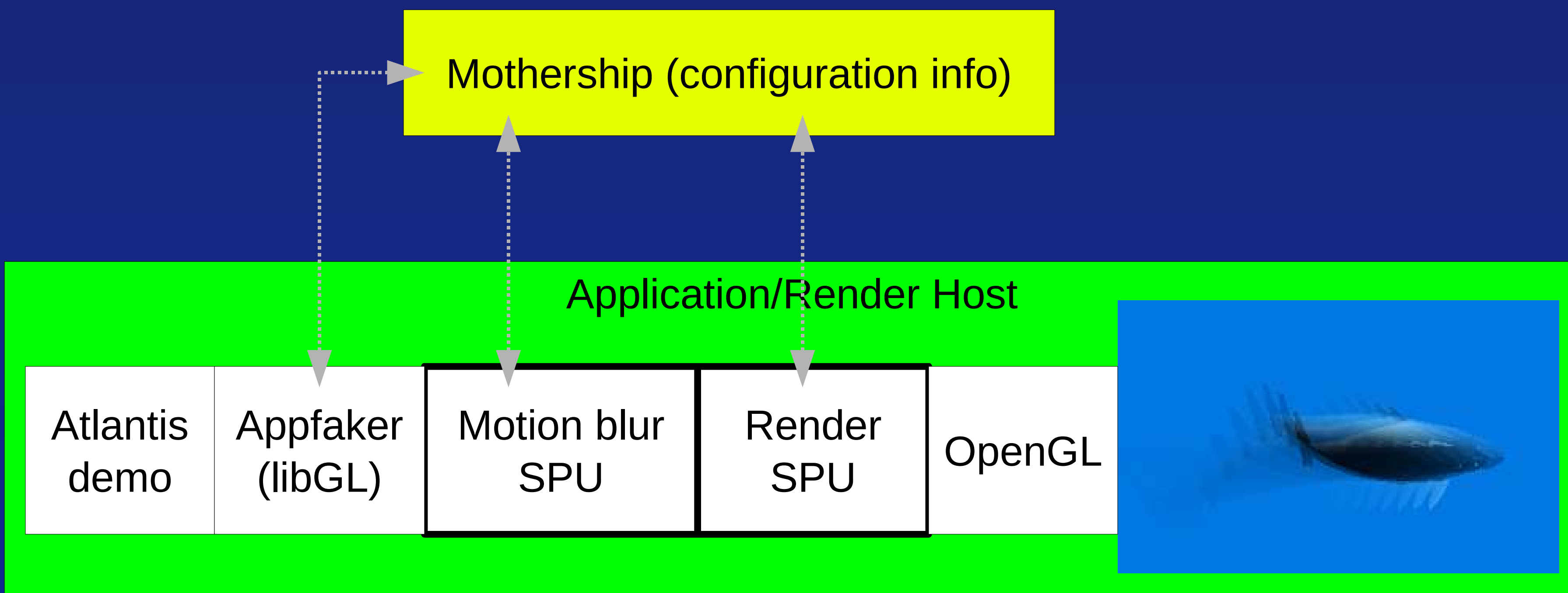


Chromium: Tilesort / Mural Display



Chromium: the Motion Blur Example

The previous motion blur example in full detail:



Available SPUs (1)

Most commonly used:

- Render – most common. Simply pass all commands to the OpenGL library for rendering.
- Pack – pack commands into network buffers and send to a network node (crserver).
- Tilesort – send commands to multiple servers arranged in a multi-screen display wall.
- Readback – for simple sort-last rendering.
- Binary Swap – sort-last rendering with the Binary Swap compositing algorithm.

Available SPUs (2)

Utility / misc SPUs:

- FPS – measure/display framerate
- Save Frame – save rendered images to files
- Print – print commands to stdout or a file and pass them to next SPU in the chain.
- Array – implements vertex array functions
- Feedback – implements selection and feedback features
- Perf – measure/display various performance figures
- Dist Texture – distributes textures to the tilesort rendering nodes for caching / faster loading.

Available SPUs (3)

Stylized Rendering SPUS:

- Hiddenline – outline polygons or hidden line rendering
- Motionblur – uses accumulation buffer to simulate motion blur effect
- Wet – an underwater/wavy effect
- Exploded architectural views - Niederauer, Houston, Agrawala, Humphreys: “Non-Invasive Interactive Visualization of Architectural Environments”, SIGGRAPH 2003.

SPU Inheritance (1)

SPU classes are object oriented.
New SPUs are derived from base class SPUs:

- Error SPU: all functions (methods) default to emitting an error message. Use as a base class when the new SPU must implement all GL functions.

- Passthrough – all functions are simply passed through to the next SPU in the chain. Use as a base class for SPUs which only need to reimplement a few OpenGL functions.

SPU Inheritance (2)

Readback SPU is an interesting example:

- Derived from the Render SPU
- Most incoming commands are simply rendered, just like the Render SPU.
- But SwapBuffers() is special: read the frame buffer with glReadPixels and send the image to the next SPU in chain with glDrawPixels.
- Used to do sort-last image compositing.

A script in the Cr sources is used to generate skeleton code for new SPUs – pretty simple.

Mothership Configuration

The mothership is driven by a Python program/script.

With Python code:

- Create SPUs, application nodes and server nodes
- Set configuration options for SPUs and nodes
- Describe how the components are “wired” together
- Auto-start processes, set network parameters, etc.

An example...

Configuration Code Example (1)

```
# Import mothership functions
from mothership import *

# Create two SPUs
pack_spu = SPU('pack')
render_spu = SPU('render')
render_spu.Conf('window_geometry', [20, 20, 800, 600])

# Create network node on host “mars”
net_node = CRNetworkNode(”mars”)
net_node.AddSPU(render_spu)

# Create app node on “localhost”
app_node = CRAppliationNode(”localhost”)
app_node.AddSPU(pack_spu)
app_node.StartDir('/home/joe/mydata')
app_node.SetApplication('/usr/local/bin/paraview')
```

Configuration Code Example (2)

```
# “Wire” the Pack SPU to the Network node
pack_spu.AddServer(net_node)

# Create mothership object
cr = CR()

# Tell mothership about the nodes
cr.AddNode(net_node)
cr.AddNode(app_node)

# Set Max network Transmission Unit size (bytes)
cr.MTU(1024 * 1024)

# Let 'er rip
cr.Go()
```

A “real” config file will have command-line parsing, more configuration options, etc.

Other Chromium Features

- Supports OpenGL 1.5 plus popular extensions like GL_ARB_vertex/fragment program.
- Several network interfaces/protocols supported:
  - TCP/IP
  - UDP
  - Myrinet GM
  - Quadrics/Elan
  - Infiniband (w/ SDP)
- Support for stereo rendering, in several ways.
- Support for non-planar displays, such as CAVEs

Any questions so far?