Part 2: Parallel Rendering

Sort-first vs. Sort-last

Parallelism in the graphics pipeline can be characterized in three ways:

- Sort-first – divide the screen into tiles/regions; a full graphics pipeline assigned for each tile. (Cr Tilesort)
- Sort-middle – arbitrarily distribute triangles among geometry units, then send the transformed triangles to the rasterizers depending on screen position.
- Sort-last – arbitrarily distribute triangles among a number of parallel, full graphics pipelines. Each has a private framebuffer. Merge all framebuffers at the end.

Sort-first and Mural Displays

Uses and basic operation

Performance issues

Rendering issues

Tilesort with DMX displays

Tilesort Uses

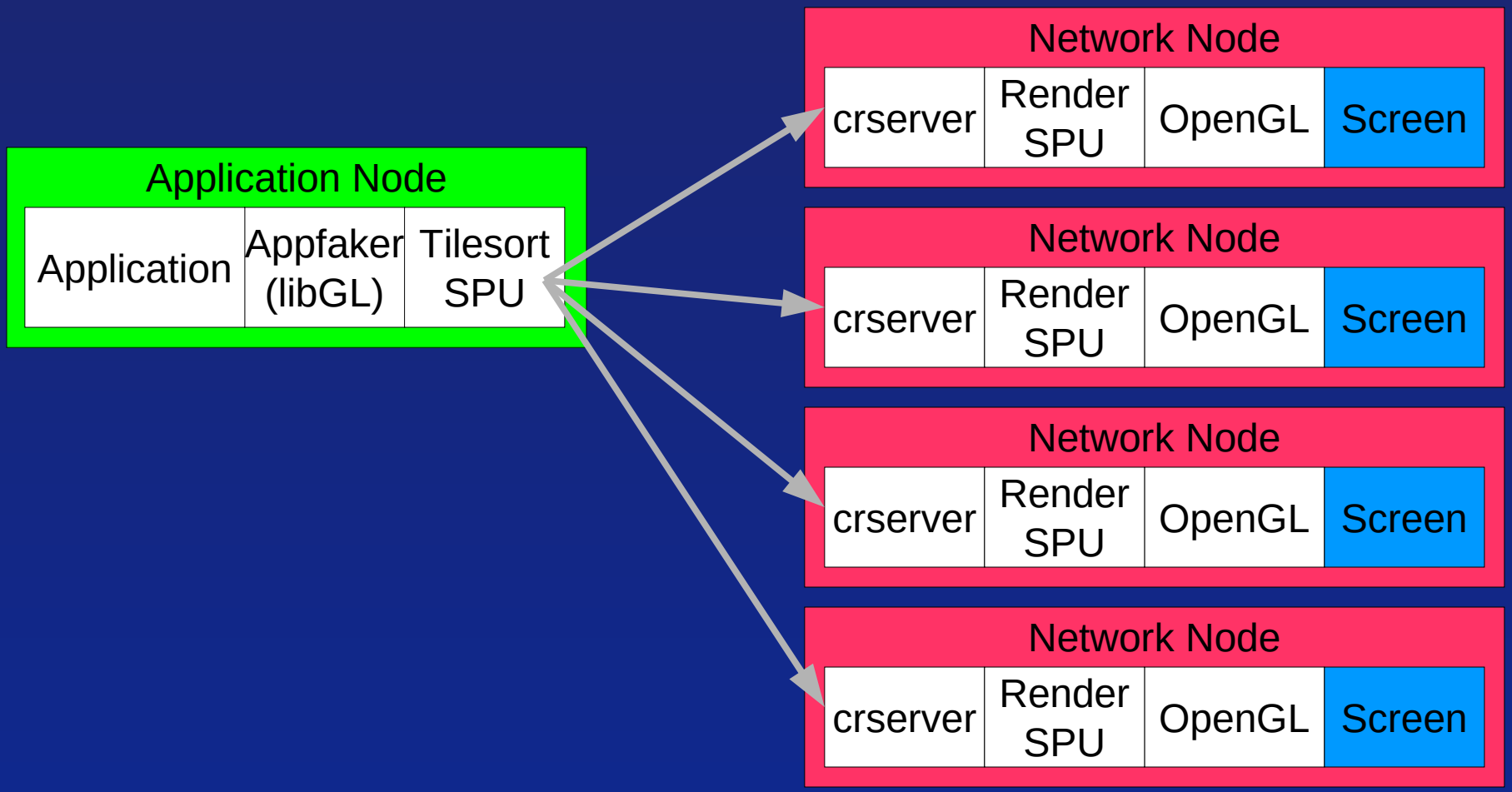
Three basic uses:

- Run unmodified OpenGL applications on multi-screen mural display walls. Good for group presentations and seeing the whole picture with full detail.
- Replicate rendering to two or more screens.
- Overcome fill-rate limitations by dividing screen into smaller tiles and reassembling them at the end. For example, divide a 1280x1024 window into four 640x512 tiles which can be rendered in parallel. (ex: IBM SGE)

Tilesort Basic Operation

- The application node hosts a Tilesort SPU.
- The Tilesort SPU sends OpenGL commands to a set of network nodes (display servers). A 4x3 screen arrangement implies 12 network nodes.
- Each network node renders a tile of the overall display.
- The Tilesort SPU does a few smart things:
  - Only send geometry (rendering commands) to the nodes where it really needs to be rendered.
  - Track OpenGL state changes and only send them to the nodes that need them (like textures).

Tilesort Diagram



Tilesort Configuration Options

Common configuration options include:

- `bucket_mode`: which technique to use for testing bounding boxes against image tiles:
  - **Uniform Grid** – best, all tiles are same size
  - **Non-Uniform Grid** – some difference in tile sizes
  - **Test All Tiles** – just loop over tiles and test them all
  - **Broadcast –** just send all commands to all servers
- `dlist_state_tracking`: should Tilesort SPU track the state changes made by display lists? This incurs a performance penalty.
- `lazy_send_dlists`: should display lists be sent out only as needed, or broadcast immediately?

Tilesort Performance Concerns (1)

The tilesort SPU has a lot of work to do:

- Track OpenGL state changes and compute state differences.
- Compute bounding boxes for glBegin/glEnd primitives.
- Bucket/sort geometry – determine which network/rendering nodes need to draw each primitive (bounding box tests).
- Pack OpenGL commands into buffers for transmission to network/rendering nodes.

Plus, the application node hosts the application itself.

So, you need a fairly speedy host system.

Tilesort Performance Concerns (2)

Network: Need a very fast network between the Tilesort node and rendering nodes. Consider 1Gbit/s Ethernet at 50% efficiency:

0.5Gbits/s ~= 62.5MB/s

If 24 bytes/vertex, then:

62.5 MB/s ***/*** 24 bytes/vertex = 2.6M vertices/sec

2.6M vertices / second isn't too hot!

And that's just one Tilesort->Render link! Suppose we need to send the geometry to 4 nodes. :(

Suppose we have a network that's 10 times faster, that still may not be sufficient to handle a large triangle workload.

Tilesort Performance Concerns (3)

To reduce network bandwidth requirements, store geometry on the servers:

- Display Lists
- Vertex Buffer Objects

But, consider memory usage: Textures and display lists and VBOs must potentially be replicated on all rendering nodes. May need lots of RAM/VRAM on all rendering nodes.

Also, initial transfer of display lists and VBOs to the server can take some time.

Tilesort Performance Concerns (4)

Miscellaneous performance issues:

- glCopyPixels – must read back image from all network nodes with glReadPixels, then do glDrawPixels.
- glCopyTexImage – Tilesort SPU must read back image data from servers, then send back with glTexImage.
- If the application depends on state changes inside display lists, the Tilesort SPU has to execute lists locally too in order to properly track the GL state changes.
- Specify bounding boxes for geometry whenever possible (by calling a Cr OpenGL extension function).

Tilesort Performance Concerns (5)

Future work for improving performance:

- Broadcast/Multicast – to send geometry buffers to several servers in parallel. More feasible with some types of network hardware than others.
- Asynchronous (threaded) networking in Tilesort SPU – try to overlap tilesort computation and network sends.
- Asynchronous networking in crserver – try to overlap server-side rendering with network receives.

Tilesort Rendering Issues (1)

The tilesort SPU often tricks the application into rendering a larger image than it realizes For example, the application's window may be 900x700 pixels but the mural display is 5000x3000. Chromium automatically scales up the rendering.

Possible issues:

- glDrawPixels/glBitmap scaling
- glReadPixels and glCopyPixels size
- Line width, point size (should be scaled up)

Tilesort Rendering Issues (2)

glRasterPos is a sticky bit. For example, if the glRasterPos command depends on lighting to set the current raster color, it'll fail.

Fragment programs which depend on the fragment position (`fragment.position`) will not get a mural-relative pixel coordinate but a screen-relative coordinate.

Display lists are troublesome. Consider glCopyTexImage in a display list. Must be executed “client-side” by the Tilesort SPU!

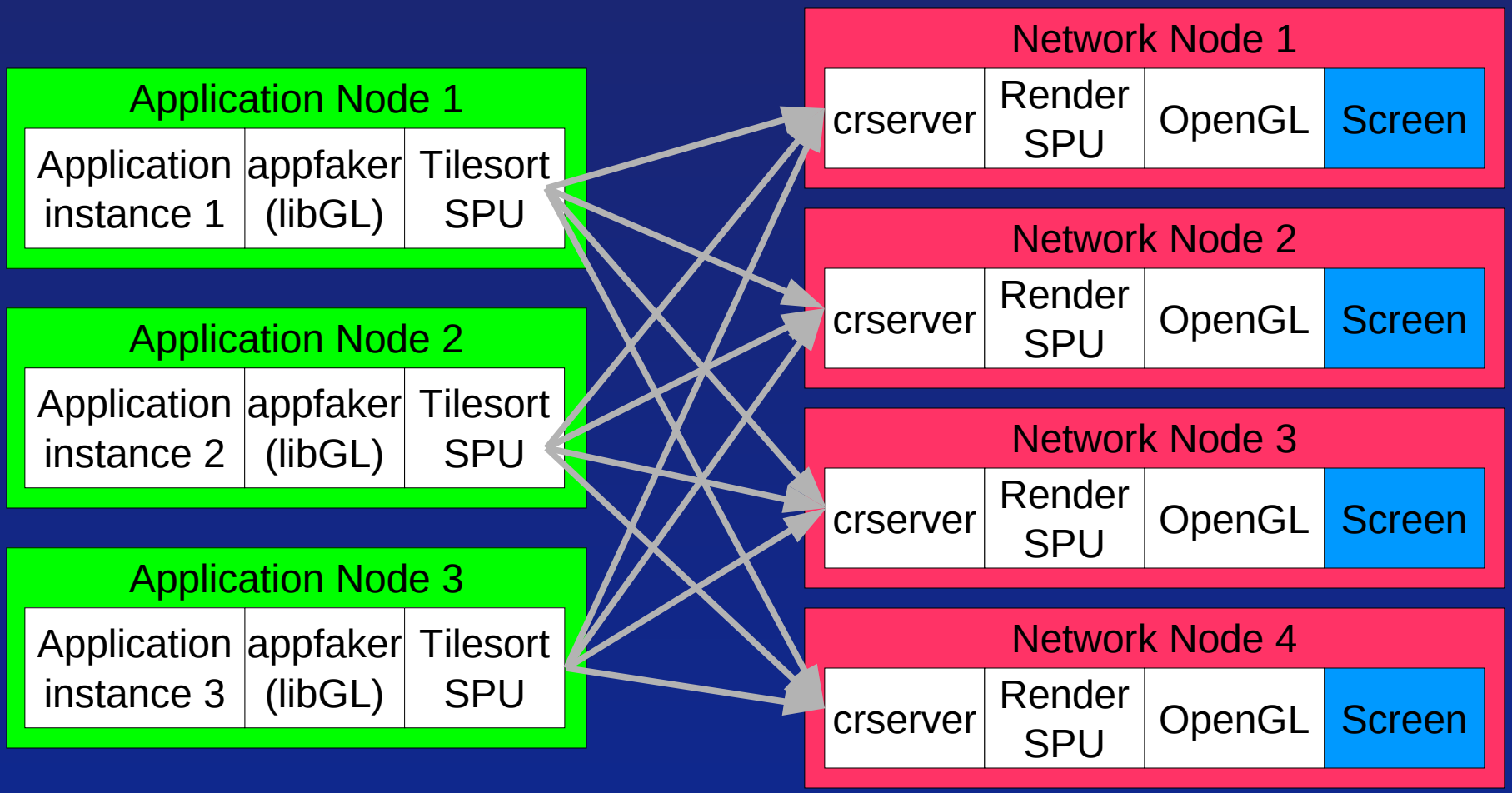
Tilesort Miscellaneous

Parallel applications can use tilesort too:

'Many-to-many node arrangement' – N application/Tilesort nodes can be connected to M network/rendering nodes.

Requires explicitly coded parallelism in the application with synchronization, etc. More on that later.

Parallel Tilesort Diagram



Sort-last rendering with Chromium

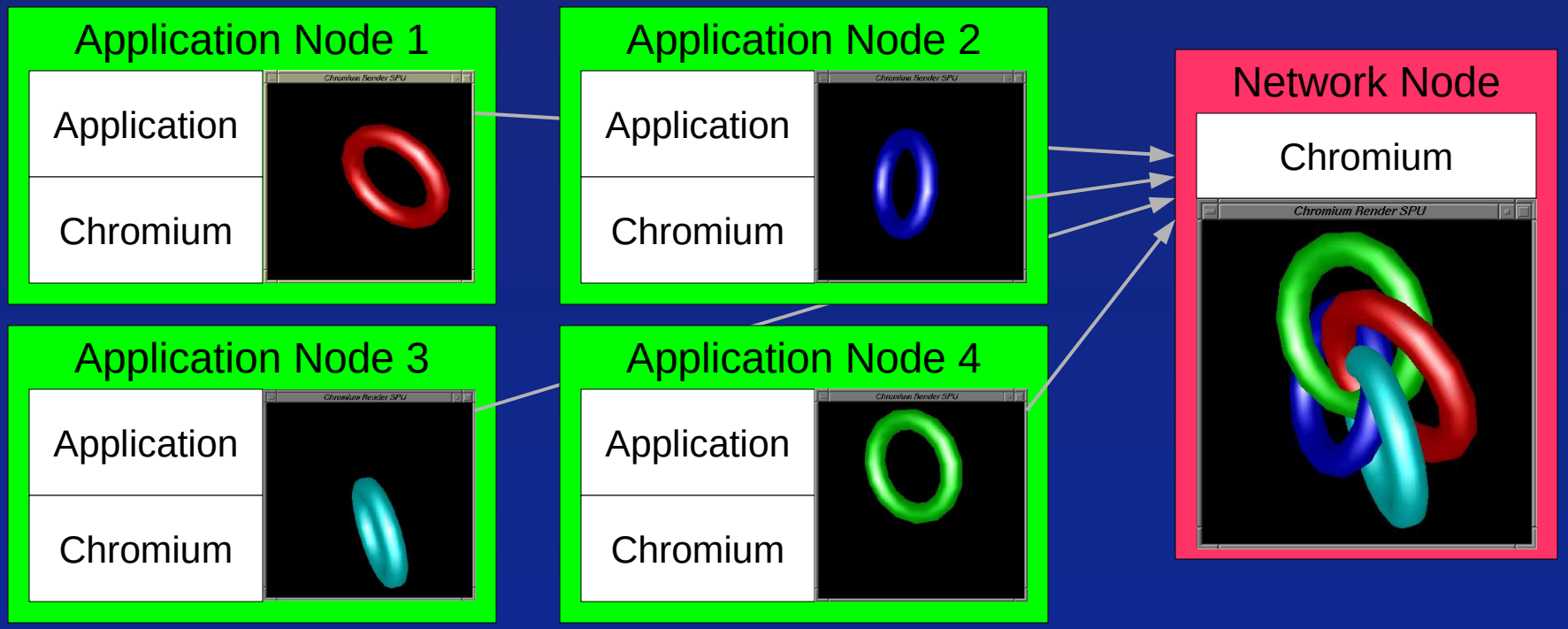
Basic concepts and properties

Sort-last performance concerns

Rendering issues

Sort-last Rendering Concepts

Basically, N instances of a parallel application each render 1/N of the total scene, each producing a partial rendering. The partial renderings are combined (either with Z testing or alpha blending) to produce the final image.



Sort-last Properties

Nice properties:

- Images are sent over the network, instead of buffers of OpenGL commands. So:
- Network load is constant and independent of the scene complexity (ignoring image compression).
- The application instance and rendering hardware reside on the same host. Having the app close to the 3D hardware is a good thing. Vertex rate not limited by the network, for example.
- Parallelizing the application and partitioning the dataset is fairly straight-forward.
- Compositing can be quickly done with special dedicated hardware.

Typical Sort-last Applications

Volume rendering

- Each application instance renders a “brick” or “slab” of the overall volume.
- Partial renderings are combined (back-to-front or front-to-back) with alpha blending.
- Use Semaphores to arrange the ordering.

“Triangle Soup” rendering

- Lots and lots of triangles to render.
- Composite partial images with Z-buffer testing.
- Rendering time is substantial, and longer than compositing time.

Sort-last in Chromium

Three relevant SPUs:

- Readback SPU – many-to-one compositing tree. Simple, but doesn't scale very well.

- Binary Swap SPU – a better algorithm: instead of doing all compositing on the final destination node, do compositing on the rendering nodes. Organize nodes into pairs, swap ½ the image between pairs, composite, collect the results.

- Zpix SPU – image compression SPU (zlib and a simple RLE-like scheme)

Sort-last Performance (1)

Hot spots:

- Getting images out of framebuffer (glReadPixels)
- Network capacity
- Image composition

Sort-last Performance (2)

Pixel Readback:

- Need to pull color and Z values out of the graphics card so we can send it downstream (or to a Binary Swap peer).
- glReadPixels is often not very fast, but gradually improving
- Use the right GL datatype and image format
- Use bounding boxes to avoid reading whole framebuffer
- Dynamic LOD: render smaller images
- Try out OpenGL Pixel Buffer Objects
- Looking forward to PCI express
- Perhaps avoid readback altogether with special hardware compositors (ex: DVI-based)

Sort-last Performance (3)

Networking: again consider 1Gbit/sec Ethernet at 50% efficiency. That's about 62.5 MB/sec.

Let's use the Readback SPU and suppose our window is 1024 x 1024 pixels, 7 bytes/pixel (RGB + Z32). That's 7MB / frame.

62.5 MB/sec / 7 MB/frame = 8.9 frames/sec.

Again, that's only one renderer -> compositor link!

If 8 rendering nodes, about 1 frame/second - not too great!

Possible Improvements:

- Image compression
- Dynamic LOD / Image scaling
- Special hardware compositors (again, perhaps DVI based)

Sort-last Performance (4)

Image compositing:

OpenGL:

- Alpha blending is simple (could apply multitexture)
- Z-compositing with glDrawPixels (two passes with stencil test)
- Z-compositing with multitexture and fragment programs

Better algorithms:

- Binary Swap (SPU)
- SLIC, ICE-T?

Software implementations:

- Combine RLE/API decompression with compositing.

Special compositing hardware...

Hardware Image Compositors

There have been various hardware-based image compositors over the years:

- PixelFlow – UNC Z-compositor (circa '92).
- HP Sepia – Image acquisition via DVI. Alpha and Z-based composition.
- Lightning-2 – Stanford/Intel effort. Image tile routing / reassembly and Z-compositing.
- IBM SGE – Image tile reassembly and buffering.
- SGI InfinitePerformance / SGC.

So, there continues to be alternatives to software compositing. Chromium SPUs can encapsulate these.

Sort-last Rendering Issues (1)

- If using Z-compositing, everything must be rendered with Z values. Otherwise, compositing results are undefined.
  So, never call `glDisable(GL_DEPTH_TEST)`.

- If you need to draw some 2D text/overlays, use `glDepthRange(0, 0)` to force the primitive's fragment's to Z=0.

- Currently, no provision for mixing Z-compositing with alpha compositing. Just one or the other.

- 24-bit Z buffer may not have sufficient precision.

Sort-last Rendering Issues (2)

Sort-last rendering can also be complicated by:

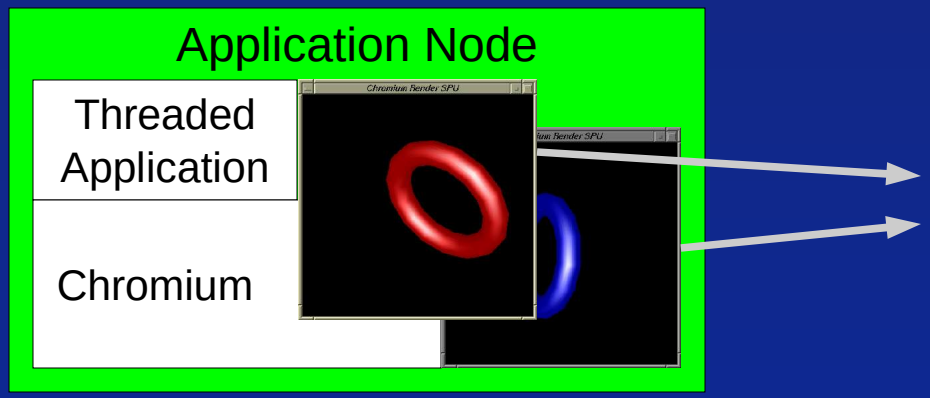
- Ordering constraints: such as semi-transparent surfaces needing to be rendered and composited in a particular order.
- glReadPixels – does the caller want its local, partial rendering or the server's complete rendering?
- glCopyTexImage – similar issue

Threading

A multiprocessor may host a parallel application which emits several parallel OpenGL streams.

Chromium is thread-safe.

Sort-last: if the host has several graphics pipes, one process might create several threads, one per pipe.



Synchronization for Parallel Rendering

To facilitate parallel rendering needs, Chromium supports two basic, well-known synchronization primitives: Barriers and Semaphores.

See Igehy, Stoll and Hanrahan's “The Design of a Parallel Graphics Interface” from SIGGRAPH '98.

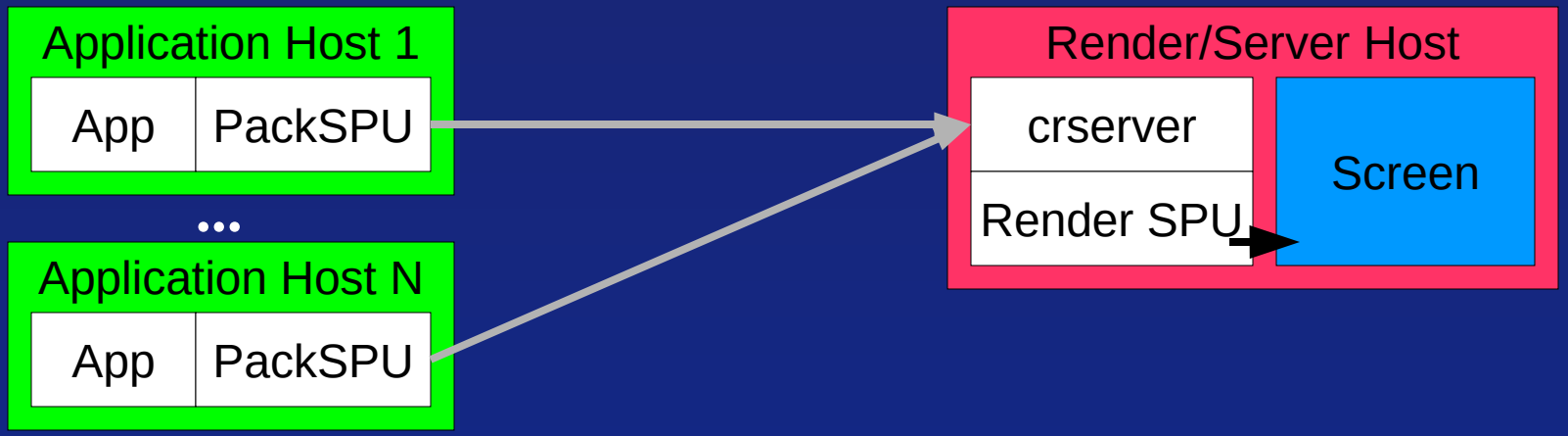
Implemented as an OpenGL extension named GL_CR_synchronization:

```
glBarrierCreateCR(GLuint name, GLuint size);
glBarrierExecCR(GLuint name); // execute barrier

glSemaphoreCreateCR(GLuint name, GLuint initVal);
glSemaphorePCR(GLuint name); // wait
glSemaphoreVCR(GLuint name); // signal
```

Barrier Usage

Suppose N parallel application instances are rendering to one server:



- We want to clear the framebuffer once (not N times).
- We want the N apps to render their stuff.
- Finally, issue one SwapBuffers at end of frame.

Need to synchronize between those three steps. Inserting a couple glBarrierExecCR() calls does the trick.

Semaphore Usage

Again, suppose N parallel application instances are rendering to one server.

If we want to impose an ordering on rendering (ex: back-to-front ordered blending) we can use a semaphore.

App instance N waits until instance N-1 completes its part. When N-1 completes its part, it signals.

Barrier and Semaphore Implementation

The semaphores and barriers are really implemented in the crserver (network node), not in the application nodes!

When a server's incoming stream is blocked on a semaphore or barrier, the server simply doesn't read/execute anymore commands from that stream until it's unblocked.

Meanwhile, the application node can perhaps make some forward progress doing something else.

SwapBuffers = End of Frame

When parallel rendering, how do we indicate the beginning and ending of each frame?

Cr uses SwapBuffers() to explicitly indicate end-of-frame, and implicitly indicate the start of next frame.

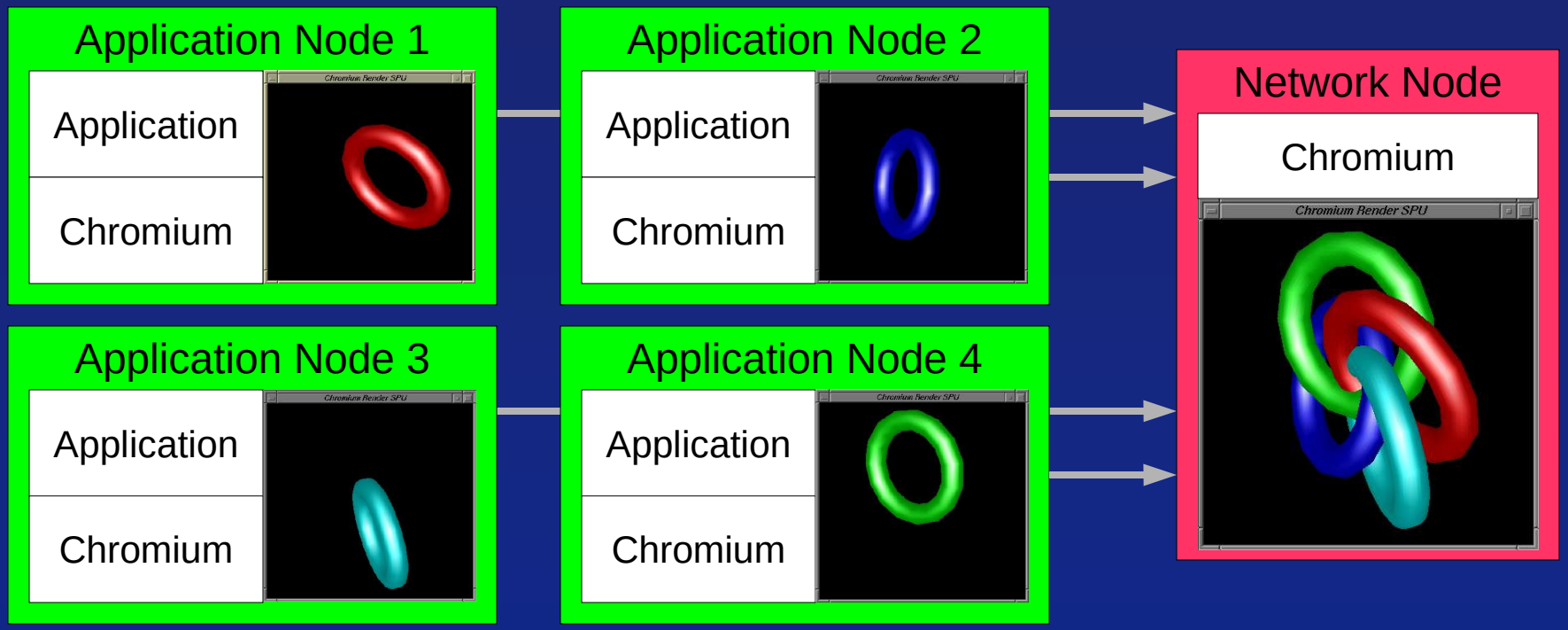
So SwapBuffers means two things:
- End of frame: do image compositing now
- Swap front/back color buffers (conventional thing)

The crSwapBuffers() function takes a flag bit to indicate end-of-frame, but don't swap buffers.

In the future, we may need real Begin/EndFrame() functions.

Input Events and Parallel Rendering

Consider a typical sort-last configuration:



The user is going to want to use the mouse to rotate the scene, etc. in the final rendering window.

How do the mouse events get sent back to the parallel application instances?

CRUT

CRUT = Chromium Utility Toolkit

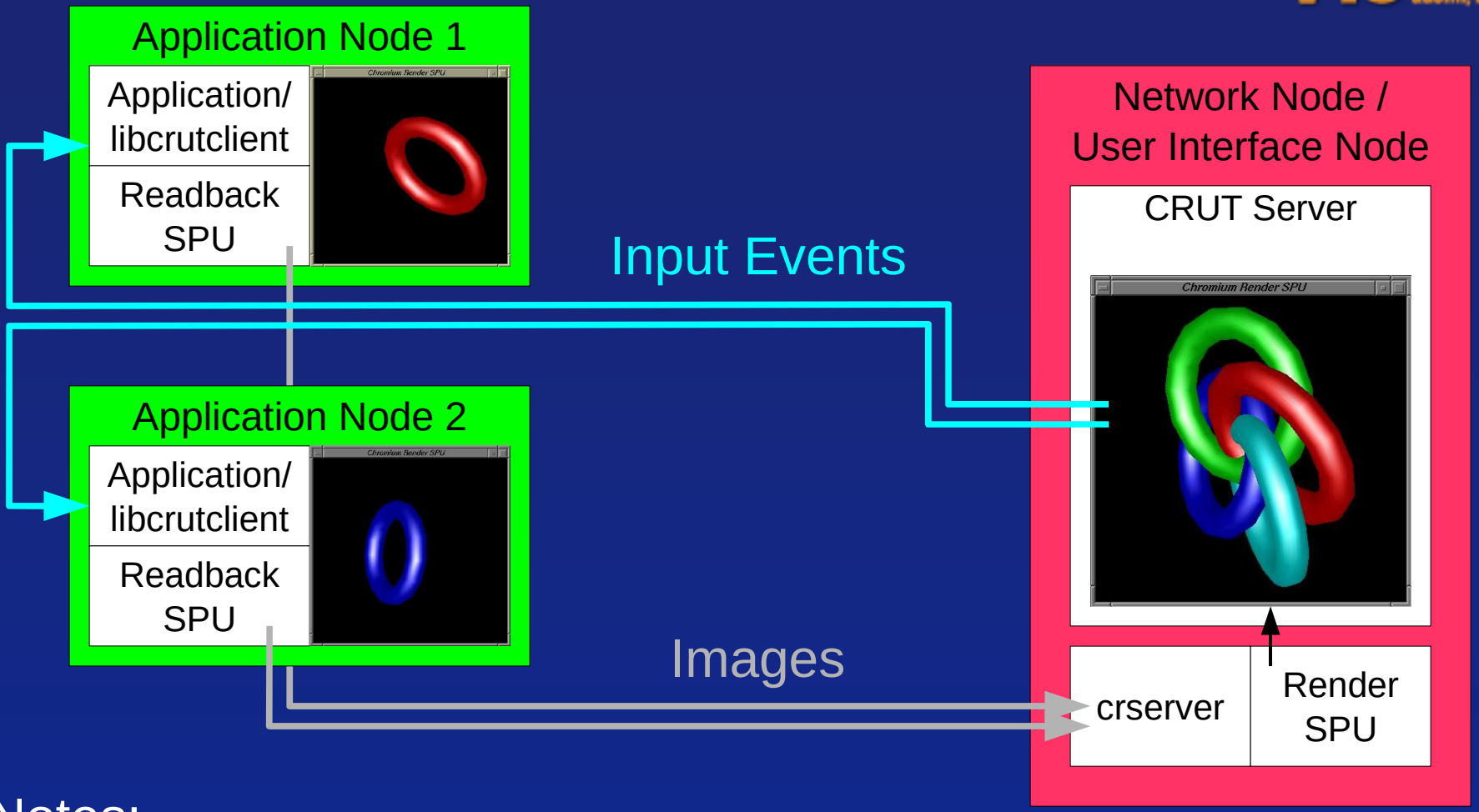
Written by Dale Beermann.

Layers on, and extends, GLUT.

The 'crutserver' program creates a window which displays the final image. Input events in that window are caught by GLUT/CRUT and propagated back to the parallel application instances.

A diagram...

CRUT Diagram



Notes:

- Network node hosts a crutserver AND crserver.
- Render SPU draws into the crutserver's window.

CRUT Info

Chromium includes documentation for CRUT, a sample program and several configuration files.

Note that CRUT, like GLUT, is generally intended for small/simple programs and not full-blown applications.

Parallel sci-vis applications like CEI Ensight/Enliten already have their own infrastructure for dealing with input events and don't need CRUT.

See the 'Raptor' volume rendering program for a complete example of Chromium + CRUT.

Without CRUT

A production, parallel application will probably process input events on the front-end host, then send scene rendering commands to the parallel rendering processes: Rotate model, zoom, set colors, set clipping planes, etc.

Any questions at this point?