Distributed Virtual Reality Computation

Jeff Russell
Introduction

• VR is useful for:
  • Engineering and data visualization
  • Interactive exhibits
  • Entertainment

• Problems arise with rendering; VR displays typically require a very large pixel count
  • 1600x1200 display is 1.92 MPixels
  • Six walled projection display would be at least 11.5 MPixels to fill
  • A single LCD wall with 12 displays would be more than 23 MPixels
  • Three such walls would be 69 MPixels
  • Using stereo? All numbers are *doubled*

• A single computer really can only fill around 2 MPixels effectively
The Classic Approach

- A single multiprocessor shared memory machine could do the job
  - Silicon Graphics Inc. famous for making these among other things
    - SGI Onyx4 has anywhere from 2 to 64 CPU's, 4-128 GB of memory, up to 32 graphics outputs
  - These are pretty good for VR applications, but:
    - Not really upgradeable; only option is to add more CPUs or rendering outputs, which actually doesn’t really help performance in general
    - Extremely costly (computer hardware is not a good investment anyways)
The Cluster Approach

- A desktop computer really drives one display just fine
- What if we just used a bunch of em?
  - Greatly reduces upgradability and costs concerns, since no specialized hardware is needed
  - Problems arise however with communication latency
    - Interaction with the system needs to be real time (20+ fps)
    - Display refreshes should be synchronized
Dividing the Work (1/4)

- A typical VR application needs to:
  - Receive and send input to/from peripherals
  - Run animation or physics simulation
  - Manipulate, transform, and generate geometry
  - Render to display(s)
- Which tasks can be distributed and keep inter-node communication very low?
Dividing the Work (2/4)

• Receive and send input to/from peripherals:
  - Usually pretty light processing, plus physical limitations probably mean the devices are hooked up to only one node

• Run animation or physics simulation:
  - Can be very CPU intensive, but is often quite difficult to parallelize

• Manipulate, transform, and generate geometry:
  - Can also be CPU intensive, might be parallelizable but generally involves lots of data

• Render to display(s):
  - Perfect!! Only exceptions would be full screen convolutions like blur that cross display borders
Dividing the Work (3/4)

- General Solution:
  - Divide render work across nodes evenly
  - Duplicate physics, animation, and geometry computations across nodes
  - Transfer input from “input node” to all others
  - Synchronize from “master node”
Dividing the Work (4/4)

- Some caveats with these clusters:
  - Load balancing is nonexistent due to synchronization (performance is limited by slowest node!)
  - Lack of shared memory makes life hard; if a tough non-graphics simulation has to be run then it may actually be better to incur the latency penalties than to do it on one CPU
  - Synchronization and distributing the display work can be bothersome to set up for each app and system
    - Tools exist to handle this automatically; VRJuggler is one developed and used at ISU [vrjuggler.org]
Sidenote: the GPU (1/2)

- Realtime graphics stopped using general purpose CPUs for rendering pretty much entirely in the late 90's

- Now done entirely on GPUs (Graphics Processing Unit), which is generally present in the form of a single specialized chip with its own memory space on a removable board (easily upgraded!)

- Works by accepting vertex and texture data from the CPU and main memory, then processing these data in parallel and posting the results to the display

- In addition to generally impressive graphics performance, has the added benefit of almost entirely freeing the CPU from rendering tasks, leaving it free to do other things while rendering occurs
Sidenote: the GPU (2/2)

- These chips are SIMD in a big way; each contains 2-6 vertex pipelines, and as many as 16 or 32 pixel pipelines all of which can be concurrently busy.

- Only data type is 128bit vector of 4 floats; has native instructions for geometry operations like cross product, dot product, matrix multiply etc.

- WAY better than a CPU at graphics (A typical fast CPU can theoretically attain approx. 10 GFlops, a modern GPU can reach more than 200 GFlops). Other optimizations allow GPUs to fill billions of pixels per second.

- But drastically limited in terms of functionality because of all the assumptions made for graphics.

- New area of high perf computing is making these things work for general purpose computations by tricking them [gpgpu.org]
Conclusion

- Immersive interactive VR is possible with a variety of solutions
  - Small clusters of desktop PCs with GPUs are by far the most cost effective, and offer excellent scaling with display counts
  - Large shared memory systems are really more convenient to program if you have all the money in the world
- The power and low cost of GPUs has allowed realtime rendering to leave the workspace and enter everyday life (PC video cards, game consoles, etc.)
- VR systems can now be built for tens of thousands of dollars out of commodity hardware, rather than spending hundreds of thousands or millions on a huge computer that will be out of date in 4 years
Questions?