The View from Ten Thousand Feet

Outline

• How does Big Data come about?
  - Measurement
  - Simulation

• Why is it a problem?
  - Bandwidth
  - Latency
  - Resolution/Fidelity

• What can we do about it?
  - Parallelism
  - Architectures
  - Algorithms
  - Bigger Displays
Where Does Big Data Come From?

Sources of Big Data

- Measurement
  - Range Scanning
  - Imagery
  - Sensor Networks

- Simulation
  - Scientific computing
  - Light Transport
  - CAD

- Other
  - Databases
  - The Web!
Range Scanning (1D)

Object to be scanned

laser

Range Scanning (1D)

\[ S, L \text{ Fixed} \]
\[ O \text{ Measured} \]
\[ D \text{ Unknown} \]
Range Scanning

- Project a stripe, not a dot
- Use a digital camera, not a 1D PSD
- Move camera + stripe to sample a shell

The Digital Michelangelo Project: 3D Scanning of Large Statues
Levoy et al., SIGGRAPH 2000

Statistics about David

- 480 individually aimed scans
- 2 billion polygons
- 7,000 color images
- 32 gigabytes
- 30 nights of scanning
- 22 people
Scientific Simulation

Billion-Atom Dislocation Dynamics in Copper

*Mark Duchaineau, LLNL*

---

CAD

Powerplant Model: 13 million triangles

*UNC Walkthrough Project*
Inherently Big Things

What’s the Problem?
Resources Are Limited!

- Bandwidth
- Latency
- Operations Per Second
- Time
- Spatial Resolution
- Dynamic Range
- Money

Bandwidth

- 80 Mpolys @ 60 Hz (optimally stripped):
  - Flat shading: 57 GB/sec
  - Colored, lit: 129 GB/sec
- 25 Projector display wall (6400x5120):
  - 24 bit color, 60 Hz: 5.9 GB/sec
- Peak bandwidths:
  - Broadband: 128 KB/sec
  - LAN: 12 MB/sec
  - Cluster: 100-200 MB/sec
  - DVI: 495 MB/sec
  - System Bus: 500-3000 MB/sec
- Achievable bandwidth typically much less
Example: Remote Rendering

- How fast can this possibly run?
- Assume:
  - 13 bytes per triangle (x,y,z + 1 byte opcode)
  - Network runs at N MB/sec
- Peak triangle rate: N/13
  - 100 MB/sec cluster interconnect: 7.7 Mtris/sec
  - 12 MB/sec LAN: 9.2 Ktris/sec

How Bad Is 7.7 (Remote) Mtris/sec?

- Not too bad!
- How fast can you call glVertex3fv?
  - On my laptop, 15M/sec
- Two conditional branches
- 13(+) memory accesses!
- Upwards of 800MB/sec to the memory system
What Can Be Done?
The Graphics Pipeline

Application

Geometry

Rasterization

Display

Compute 3D geometry
Make calls to graphics API

Transform geometry from 3D to 2D

Generate pixels from 2D geometry

Continuously refresh display device

The Graphics Pipeline

Application

Geometry

Rasterization

Display

Application

Command

Geometry

Rasterization

Texture

Fragment

Display
Measuring Performance

- Input Rate
- Triangle Rate
- Pixel Rate
- Texture Memory
- Display Resolution

Parallel Rendering

- Today’s hardware is interface limited
- Geometry work is application-visible
- Rasterization work is not application-visible
- Broadcast communication does not scale
- Temporal locality $\Rightarrow$ spatial load imbalances
Interface Overhead

Compute
Data

Encode
API

CPU
B/W

I/O

GFX
B/W

Decode
API

I/O

Draw
Data

Interface Bottlenecks

Application

Geometry

Rasterization

Display

Application

G G G G G G G

R R R R R R R

D D D D D D D
Solutions to the Interface Limit

- Vertex arrarys
- Retained mode rendering (display lists)
- Compression
- *Parallelize the interface*

Synchronizing Parallel Interfaces

- Immediate mode API’s are *ordered*
  - Framebuffer effects (transparency)
  - Painters algorithm (BSP visibility)
  - State effects (enable lighting)
- Traditional synchronization primitives
  - `glBarrierExec`
  - `glSemaphoreV`
  - `glSemaphoreP`

The Design of a Parallel Graphics Interface
Igehy, Stoll, and Hanrahan, SIGGRAPH 1998
Marching Cubes

MarchParallelOrdered \((M, N, \text{grid}, \text{sema})\)

\[
\text{for (i=0; i<M; i++)}
\]

\[
\text{for (j=(myP+i)\%numP; j<N; j+=numP)}
\]

\[
\text{if (i>0) glSemaphoreP(sema[i-1,j])}
\]

\[
\text{if (j>0) glSemaphoreP(sema[i,j-1])}
\]

\[
\text{ExtractAndRender(grid[i,j])}
\]

\[
\text{if (i<M-1) glSemaphoreV(sema[i,j])}
\]

\[
\text{if (j<N-1) glSemaphoreV(sema[i,j])}
\]
Taxonomy of Parallel Graphics

• Object parallel input
• Image parallel output
• “Sort” between them
  - Communicate work to the correct location on the screen

Examples: Princeton Display Wall, WireGL

A Sorting Classification of Parallel Rendering, Molnar et al.
Taxonomy of Parallel Graphics

- Object parallel input
- Image parallel output
- “Sort” between them
  - Communicate work to the correct location on the screen

Examples: SGI InfiniteReality, UNC PixelPlanes 5

A Sorting Classification of Parallel Rendering, Molnar et al.

---

Taxonomy of Parallel Graphics

- Object parallel input
- Image parallel output
- “Sort” between them
  - Communicate work to the correct location on the screen

Example: UNC PixelFlow, Kubota Denali, SUN SAGE

A Sorting Classification of Parallel Rendering, Molnar et al.
Extended Taxonomy

- Application
- Command
- Geometry
- Rasterization
- Texture
- Fragment
- Display

- Sort-First
- Sort-Middle
- Sort-Last Fragment
- Sort-Last Image Composition

Screen Space Parallelism

- Divide screen into “tiles”
- Assign tiles to separate processors
- Primitives that overlap multiple tiles are rendered *redundantly*
- What makes a good partition?
Screen Space Parallelism

What makes a good screen partition?

Screen Space Parallelism

- Static partitions
  - Balance between load balance and efficiency
  - Minimize length of region boundaries
- Adaptive partitions
  - Simple methods [Roble88]
  - Median-cut [Whelan85]
  - Top-down decomposition [Whitman92]
  - MAHD [Mueller94]
  - Hybrid algorithms [Samanta00]
Image Compositing

- Sort-last (usually) requires image merging
- High bandwidth requirements
- Ordering semantics?

Software Compositing

- Many-to-one communication: bottleneck
Binary Tree Compositing
- Reasonable bandwidth requirements
- Resource under-utilization

Binary Swap Compositing
- Reasonable bandwidth requirements
- Resources fully utilized
Image Compositing Hardware

Lightning-2 = Digital Video Switch from Intel Research
Route partial scanlines from \( n \) inputs to \( m \) outputs
Tile reassembly or depth compositing at full refresh rate

Software Systems

- Iris Performer
  - Efficient geometry/state management
  - Frustum Culling / LOD support
  - Multiple processors / graphics pipes
Mesh Simplification

- Idea: draw less stuff!
- Pixels per polygon $\rightarrow$ polygons per pixel

![Mesh Simplification Diagram](image-url)
Mesh Simplification

- Idea: draw less stuff!
- Pixels per polygon $\rightarrow$ polygons per pixel
Mesh Simplification

- Idea: draw less stuff!
- Pixels per polygon $\rightarrow$ polygons per pixel

Simplification Hierarchies

10,108 polygons
1,383 polygons
474 polygons
46 polygons
Simplification Hierarchies

• Choose an appropriate representation based on screen coverage
• Smooth transitions?

(Courtesy IBM)

Subsampling

• Idea: Not enough time to draw everything
• So don’t bother!
Frameless Rendering

Point Rendering

- Idea: Very dense 3D models need not store connectivity!
- Bounding sphere hierarchy [Rusinkiewicz00]
- Traverse hierarchy to some depth
- Depth $\rightarrow$ projected area and time
QSplat Demo

QSplat: A Multiresolution Point Rendering System for Large Meshes
Rusinkiewicz and Levoy, SIGGRAPH 2000

Using Commodity Technology

- Commodity parts
  - Complete graphics pipeline on a single chip
  - Extremely fast product cycle
- Flexibility
  - Configurable building blocks
- Cost
  - Driven by consumer demand
  - Economies of scale
- Availability
  - Insufficient demand for “big iron” solutions
  - Little or no ongoing innovation in graphics “supercomputers”
Cluster Graphics

- Raw scalability is easy (just add more pipelines)
- Exposing that scalability to an application is hard

Cluster Graphics

- Graphics hardware is indivisible
- Each graphics pipeline managed by a network server
Cluster Graphics

- Flexible number of clients, servers and displays
- Compute limited = more clients
- Graphics limited = more servers
- Interface/network limited = more of both

Stanford/DOE Visualization Cluster

- 32 nodes, each with graphics
- Compaq SP750
  - Dual 800 MHz PIII Xeon
  - i840 logic
  - 256 MB memory
  - 18 GB disk
  - 64-bit 66 MHz PCI
  - AGP-4x

- Graphics
  - 16 NVIDIA Quadro2 Pro
  - 16 NVIDIA GeForce 3

- Network
  - Myrinet (LANai 7 ~ 100 MB/sec)
**WireGL/Chromium**

- Replace system’s OpenGL driver
  - Industry standard API
  - Support existing unmodified applications
- Manipulate streams of API commands
  - Route commands over a network
  - Render commands using graphics hardware
- Allow parallel applications to issue OpenGL
  - Constrain ordering between multiple streams

*WireGL: A Scalable Graphics System for Clusters*
*Humphreys et al., SIGGRAPH 2001*

*Chromium: A Stream Processing Framework for Interactive Rendering on Clusters of Workstations*
*Humphreys et al., SIGGRAPH 2002*

**WireGL: Output Scalability**

*Marching Cubes*

Point-to-point “broadcast” doesn’t scale at all
However, it’s still the commercial solution [SGI Cluster]
WireGL: Peak Rendering Performance

Immediate Mode
- Unstructured triangle strips
- 200 triangles/strip

Data change every frame
Peak observed: 161,000,000 tris/second
Total scene size = 16,000,000 triangles
Total display size at 32 nodes: 2048x1024 (256x256 tiles)

Stream Processing

Streams:
- Ordered sequences of records
- Potentially infinite

Transformations:
- Process only the head element
- Finite local storage
- Can be partitioned across processors to expose parallelism
Why Stream Processing?

- Elegant mechanism for dealing with huge data
  - Explicitly expose and exploit parallelism
  - Hide latency
- State of the art in many fields:
  - Databases [Terry92, Babu01]
  - Telephony [Cortes00]
  - Online Algorithms [Borodin98, O’Callaghan02]
  - Sensor Fusion [Madden01]
  - Media Processing [Halfhill00, Khailany01]
  - Computer Architecture [Rixner98]
  - Graphics Hardware [Owens00, NVIDIA, ATI]

Stream Computing

- Observation: media applications are a poor match for current microprocessor design
- Idea: build hardware to efficiently support streaming computation
- Achieve very high arithmetic density
- Exploit locality of reference
- Exploit parallelism
- “Imagine”: Reconfigurable stream computing
Dynamic Range Compression

- What’s a poor photographer to do?
- Idea: non-global, non-linear tone mapping
Dynamic Range Compression

HDR Compression in 1D

Gradient Domain High Dynamic Range Compression
Fattal, Lischinski and Werman, SIGGRAPH 2002
HDR Compression in 1D

Gradient Domain High Dynamic Range Compression
Fattal, Lischinski and Werman, SIGGRAPH 2002
Volume Rendering

- Directly render \textit{volumetric} data
- Volume = regularly sampled scalar function
- Possibilities:
  - Extract isosurface, render polygons [Lorensen87]
  - Ray tracing [Levoy90]
  - Splatting [Laur91]
  - 2D Texture Mapping [Cabral94]
  - 3D Texture Mapping [Wilson94]
  - Special Hardware [Pfister99]

Marching Cubes (okay, Squares)

Marching Cubes: a high resolution 3D surface reconstruction algorithm
\textit{Lorensen and Cline}, SIGGRAPH 1987
Marching Squares

Marching Cubes: a high resolution 3D surface reconstruction algorithm
Lorensen and Cline, SIGGRAPH 1987

Big Data in Computer Graphics Fall 2002 Lecture 2
Marching Cubes

Visualization

Computed reflectance
Visualization

• Yikes!
• Minimize crossings
• Reveal structure
  - Hierarchy?
  - Symmetry?
  - Cycles?
• Deal with enormity
Example: Internet Links

- Idea: use a *hyperbolic* visualization space
  [Munzner 95,97,98]
Texture Management

- Only part of graphics state that’s big
- Textures need to be accessed by rasterizers
- Caching?
- Multiple rasterizers = replicated textures?

Texture Caching

- Representation of textures critical
- Cache design affects rasterization algorithm
- Memory bandwidth is precious
- Single rasterizer design: [Hakura97]
- Prefetching to hide latency: [Igehy98]
- Multiple rasterizer design: [Igehy99]
Huge Textures

- Satellite images @ 1m resolution = 11 PB
- For very large mipmaps, most pixels are not accessed for any given view
- Idea: only store a partial mipmap pyramid
- Update the partial pyramid on the fly

The Clipmap: A Virtual Mipmap
Tanner, Migdal, and Jones, SIGGRAPH 1998

Huge Textures

- Homogeneous representation
- Textures for overview renderings
- Polygons for up-close viewing
- Recompute mipmap tiles on the fly

Using Texture Mapping with Mipmapping to Render a VLSI Layout
Solomon and Horowitz, DAC 2001
Big Displays

Interaction

- Keyboard/mouse/desktop metaphor?
- New types of interaction techniques:
  - Gesture
  - Voice
  - Pen
- Legacy interaction?
- Collaborative interaction
- Most effective use of space
Yes

No
Projector Alignment: Static
Projector Alignment: Dynamic

UNC Office of the Future Group

Projector Alignment: Dynamic

WireGL extensions for casually aligned displays
*UNC PixelFlex team and Michael Brown, UKY*
Ray Tracing

- Object-order algorithm
- Sensitive to output resolution
- Built-in occlusion culling

Memory Coherence

- Complex scenes can be much bigger than memory
- Efficient rendering \(\rightarrow\) data management!
- Monte Carlo ray tracing has poor coherence
- But ray tracing need not be recursive!
- Idea: Reorder computation for maximum coherence
- Every ray is completely independent

Rendering Complex Scenes with Memory-Coherent Ray Tracing
Pharr, Kolb, Gershbein, and Hanrahan, SIGGRAPH 1997
Memory Coherent Ray Tracing
Interactive Ray Tracing

- Raytracing: more efficient than rasterization
- Highly efficient and parallel
- Deferred shading
- Huge parallel machines [Parker99]
- Clusters of workstations [Wald01]
- GPU-based implementation [Purcell02]

Streaming Ray Tracer

Ray Tracing on Programmable Graphics Hardware
Purcell, Buck, Mark, and Hanrahan, SIGGRAPH 2002
Geometry in Texture Maps!

Uniform Grid

3D Luminance Texture

<table>
<thead>
<tr>
<th>vox0</th>
<th>vox1</th>
<th>vox2</th>
<th>vox3</th>
<th>vox4</th>
<th>vox5</th>
<th>...</th>
<th>voxM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>11</td>
<td>38</td>
<td></td>
<td></td>
<td>...</td>
<td>564</td>
</tr>
</tbody>
</table>

Triangle List

1D Luminance Texture

<table>
<thead>
<tr>
<th>vox0</th>
<th>vox2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>21</td>
<td>216</td>
</tr>
</tbody>
</table>

Triangles

3x 1D RGB Textures

<table>
<thead>
<tr>
<th>tri0</th>
<th>tri1</th>
<th>tri2</th>
<th>tri3</th>
<th>tri4</th>
<th>tri5</th>
<th>triN</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0</td>
<td>v1</td>
<td>v2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xyz</td>
<td>xyz</td>
<td>xyz</td>
<td>xyz</td>
<td>xyz</td>
<td>xyz</td>
<td>xyz</td>
</tr>
<tr>
<td>xyz</td>
<td>xyz</td>
<td>xyz</td>
<td>xyz</td>
<td>xyz</td>
<td>xyz</td>
<td>xyz</td>
</tr>
</tbody>
</table>

Ray Tracing on Programmable Graphics Hardware

*Purcell, Buck, Mark, and Hanrahan, SIGGRAPH 2002*

OpenRT

![OpenRT Image]
OpenRT

OpenRT
OpenRT

High Performance Architectures

- Huge variety of graphics hardware out there
- We’ll look at 12 at the end of the semester
- Stanford graphics architecture course:
  graphics.stanford.edu/courses/cs448a-01-fall/