Large Data Visualization

Hank Childs, University of Oregon
Announcements (1/2)

• Grading:
  – 6A, 6B, 7A graded
  – To do: 5, 7b (both 510 only)

• Current denominators:
  – 410: 65 points
  – 510: 85 points

• Feedback: fantastic job!
  – put them on your resumes...
Announcements (2/2)

• Final:
  – **location**, Fri Dec 11\(^{th}\), 10:15am
  – 5 mins each for self-defined
    • PPT, demo
    • just PPT
    • just demo
  – all should attend (even pre-defined folks)

• Schedule:
  – today: unstructured grids + FAQ on VTK
  – Weds: large data vis
  – Fri: Lagrangian flow (research)
Why simulation?

- Simulations are sometimes more cost effective than experiments

But: simulations are only valuable if they are accurate

And: to achieve accuracy, simulations often require high resolution meshes
What are the challenges with large data visualization?

- Two grand challenges:
  - Handling the scale of the data
  - Reducing the complexity of the data
How can we gain insight from increasingly complex data?

512^3 portion of turbulent flow data set

Data source: P.K. Yeung and Diego Donzins
How can we gain insight from increasingly complex data?

1024^3 portion of turbulent flow data set

Data source: P.K. Yeung and Diego Donzins
How can we gain insight from increasingly complex data?

2048^3 portion of turbulent flow data set

Data source: P.K. Yeung and Diego Donzins
How can we gain insight from increasingly complex data?

4096^3 portion of turbulent flow data set

Data source: P.K. Yeung and Diego Donzins
How can we gain insight from increasingly complex data?

- We developed a statistical approach for identifying and understanding “worms” in large-scale turbulent data.

- Techniques deal with both scale and complexity.

The best way to reduce the complexity is often application specific.

Why supercomputers?

- More mesh resolution
  - more memory and computation
  - supercomputers

- Very expensive supercomputers can still be cost effective.

The data sets resulting from these simulations are large and require special visualization techniques.
Defining “big data” for visualization

Big data: data that is too big to be processed in its entirety all at one time because it exceeds the available memory.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>In its entirety</td>
<td>Data subsetting / multi-resolution</td>
</tr>
<tr>
<td>All at one time</td>
<td>Streaming (e.g. out of core)</td>
</tr>
<tr>
<td>Exceeds available memory</td>
<td>Parallelism</td>
</tr>
</tbody>
</table>
Different Types of Parallelism

Distributed-memory parallelism

Shared-memory parallelism

Mostly focusing on distributed-memory parallelism
Parallelization: Overview

• It turns out many, many visualization operations follow “scatter, gather”
  – Scatter the pieces over processors
  – Perform operation with no coordination
    • “embarrassingly parallel”
  – Gather the results later (often with rendering)

• This is a very scalable approach

• Very little work is needed to achieve parallelism in this context
Parallelization & Data Flow Networks

• Every processor instantiates an identical data flow network

• The data flow networks are only differentiated by the portion of the larger data set they work on

Visualization reusing nodes from the supercomputer that generated the data

PE = Processing Element (a compute node)
Data Parallel Pipelines

- Duplicate pipelines run independently on different partitions of data.
Data Parallel Pipelines

• Duplicate pipelines run independently on different partitions of data.

Slide courtesy of Ken Moreland, Sandia Lab
Data Parallel Pipelines

• Some operations will work regardless.
  – Example: Clipping.
Data Parallel Pipelines

• Some operations will work regardless.
  – Example: Clipping.
Data Parallel Pipelines

• Some operations will work regardless.
  – Example: Clipping.

Slide courtesy of Ken Moreland, Sandia Lab
Data Parallel Pipelines

• Some operations will have problems.
  – Example: External Faces
Data Parallel Pipelines

- Some operations will have problems.
  - Example: External Faces
Data Parallel Pipelines

- Ghost cells can solve most of these problems.
Data Parallel Pipelines

- Ghost cells can solve most of these problems.

But some algorithms really have problems ... will mention those later.

Slide courtesy of Ken Moreland, Sandia Lab
(All parallel rendering slides courtesy Ken Moreland, Sandia Lab)
The Graphics Pipeline

Points | Lines | Polygons

Rendering Hardware

Geometric Processing
Translation
Lighting
Clipping

Rasterization
Polygon Filling
Interpolation
Texture Application
Hidden Surface Removal

Frame Buffer
Display
Parallel Graphics Pipelines
Sort First Parallel Rendering
Sort Middle Parallel Rendering

[Diagram showing a sorting network with G and R inputs and a resulting network output]
Sort Last Parallel Rendering

[Diagram showing a process with multiple steps involving G and R components leading to a sorting network and an output image.]
Sort-First Bottleneck

Polygon Sorter

Polygon Sorter

Polygon Sorter

Polygon Sorter

Network

Renderer

Renderer

Renderer

Renderer

Network

Renderer

Renderer

Renderer

Network
Sort-Last Bottleneck

Renderer → Renderer → Renderer → Composition Network → Output
Parallel rendering

- So which is best?....

- Vis: images normally $\sim 1$M pixels, data normally really big

- So: sort last
Outline

• Overview
• Parallel Rendering
• Easy-to-Parallelize Algorithms
• Data Flow Networks and Smart Techniques
Parallel Isosurfacing

• Divide pieces over processors
• Each processor applies the isosurfacing algorithm to its pieces
  – There is no coordination required between pieces!!
    • This is called “embarrassingly parallel”
• Each processor now contains a part of the large surface
• ➔ parallel rendering
Parallel Isosurfacing … is it really that easy?

• What can go wrong:
  – You have cell centered data & algorithm works on node-centered data.
  – So must interpolate to nodes
  – And end up with inconsistency at boundaries
  – And thus broken a isosurface

Ghost data solves this problem.
Parallel efficiency + isosurfacing

What are the barriers to parallel efficiency?

- Parallel reads
- Variation in isosurface calculation time
- Parallel rendering
Parallelization similar to isosurfacing

- All the algorithms we discussed as being based on “Marching Cubes”
  - Slicing
  - Interval volumes
  - Box
  - Clip
  - Slicing by non-planes
- Threshold
- and many more...
- BUT: volume rendering, streamlines, and more are much harder....
Parallelizing Particle Advection

- Very difficult to parallelize
- Particle path is data-dependent
  - You don’t know where the particle will go
  - ... so you don’t know which data to load
- In my opinion: hardest problem to efficiently parallelize in visualization
- Two axes:
  - Is the data big or small?
  - Do you have lots of particles or few?
Small data, small number of particles

• Serial processing!
Large data, small number of particles

• Still serial processing!
Small data, large number of seeds

This scheme is referred to as parallelizing-over-particles.

Parallelizes very well, but the key is that the data is small enough that it can fit in memory.

Can also work on large data sets, but can really suffer with redundant I/O.

File

Simulation code

Visualization network

Read

Advect

Render

Render

Render
Large data, large number of particles

This scheme is referred to as parallelizing-over-data.

What if all particles are in P0 and never leave?
Parallelization with big data & lots of seed points & bad distribution

- Two extremes:
  - Partition data over processors and pass particles amongst processors
    - Parallel inefficiency!
  - Partition seed points over processors and process necessary data for advection
    - Redundant I/O!

See “Pugmire, Childs, Garth, Ahern, Weber @ SC09”

<table>
<thead>
<tr>
<th>Parallelizing Over</th>
<th>I/O</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Good</td>
<td>Bad</td>
</tr>
<tr>
<td>Particles</td>
<td>Bad</td>
<td>Good</td>
</tr>
</tbody>
</table>