So far we have covered ...

- Basic visualization algorithms
- Parallel polygon rendering
- Occlusion culling

They all indirectly or directly help understanding and analyzing large scale data (volumetric or geometric)

But they are really just a small part of the whole problem
Tera-scale Data Visualization
Big Data?

- Big data collection vs. big data objects
- Big data collection: aggregates of many data sets (multi-source, multi-disciplinary, heterogeneous, and maybe distributed)
- Big data objects: single object too large
  - For main memory
  - For local disk
  - Even for remote disk
Big Data Objects

- As a result of large-scale simulations (CFD, weather modeling, structural analysis, etc)
- A sample of problems
  - Data management – data models, data structures, storage, hierarchy etc.
  - Too big for local memory (e.g. 10GB time-varying data)
  - Too big for local disk (e.g. 650GB simulation data)
  - High bandwidth and latency
Possible Solutions

- Write-A-Check Approach

* Buy your own supercomputers

Diagram:
- Simulation
  - Data in big fast memory/disk
  - Visualization Algorithms
  - Geometry
  - Rendering
Possible Solutions (2)

Write-A-Check Approach 2:

- Buy your own high-end Workstation

Simulation
- Big fast disk
- Fast network

Visualization algorithms
- Big fast disk
- Big fast memory

Geometry

Rendering

High-end Workstation (a complete package)
Possible Solutions (3)

- Perhaps a better approach ...

1. Supercomputer
2. Commercial server
3. Lower-end workstation/PC
Data reduction techniques

- Goal: Reduce the memory/disk/network resource requirements
  - Memory Hierarchy
  - Indexing
  - Compression
  - Multiresolution
  - Data mining and feature extractions
A system approaches

- Break the data in pieces
- Retrieve only the relevant pieces
- Demand-driven
- Sparse traversal using index
Break Data in Pieces

- Although O.S supports this long time ago... (VM)

“Flat” File

Bad locality
Break Data in Pieces (2)

- It is better for the application to decide how the data should be subdivided.
- Caution: Don’t be too algorithm specific.
  - You don’t want to have one file layout for each viz algorithm.
- Most of the vis algorithms have similar memory access patterns.
- Issues: fast addressing without bloat the data.

“cubed file”
Demand-Driven Data Fetch

- Virtual Memory typically won’t do a good job
  - do not know what are the necessary blocks
- An application control data retrieval system is needed
  - Fast block retrieval
  - Overlap I/O with Computation
  - Smart pre-fetch
  - ...
Sparse Traversal

- Memory hierarchy approach works the best when the underlying algorithms only need a sparse traversal of the data with high access locality.
- Examples:
  - Isosurface extraction (Marching Cubes Algorithm is not)
  - Particle Tracing (naturally sparse traversal)
  - Volume Rendering
- This requires the algorithms to be somewhat modified – out-of-core visualization algorithms
  - Sparse traversal
  - High data locality
Case Study

Out-of-Core Streamline Visualization on Large Scale Unstructured Meshes

Ueng et al, 1996
A perfect example of sparse traversal

Goal:

- Reduce the memory requirement
- Disk access should be minimized
- Increase the memory access locality
- Interactivity is important
- Deal with unstructured data
The Challenge of Unstructured Data

- Need explicit specification of node positions - files become large
- File layout lacks of spatial coherence -> VM will work even worse
- Cell sizes can vary significantly -> difficult to subdivide evenly
- Visualization algorithms are also hard to design (not out-of-core specific)
Typical File Layout
Sample Unstructured Mesh
Out-of-Core Streamline Viz

- Data preprocessing
  - Data partitioning
  - Data preprocessing
- Run-time streamline computation
  - Scheduling
  - Memory management
Data Partitioning

- Using octree spatial decomposition
- Use the geometry center of cells to subdivide the volume (average of centers)
- Subdivide the octree node until each octane has approximately the same number of cells (note: a cell may be assigned to more than one buffer)
Data Partitioning (2)

- Data partitioning has to be done in an out-of-core manner
- Create eight disk files, read cells into memory incrementally and write to corresponding files
- Examine the file size at each run and subdivide as needed
Data Partitioning (3)

- How big an octane should be?
  - The octane will be the unit to bring into memory each time
  - Small block
    - More redundant cells
    - More frequent disk access (see time increases)
    - High hit rate
    - Faster to bring in
Run-time Algorithm

- Execution scheduling – compute multiple streamlines at a time to improve memory access locality (better than one at a time)
- Memory management – reduce internal fragmentation
Execution Scheduling

- For each streamline, there are three possible states:
  - Wait: no data is available
  - Ready: has data, computation proceeds
  - Finished: done

- Multiple streamlines are considered
Execution Scheduling (2)

- Three queues are used to store the active streamlines.
- All streamlines are put into the wait queue initially.

![Queue Diagram]

Wait Queue → Ready Queue → Finished Queue
Execution Scheduling (3)

- When a streamline steps out of the current block, it is moved from ready queue to the end wait queue
- Another streamline starts
- When the ready queue is empty, then a batch I/O is performed to move in the blocks needed for waiting streamlines
Memory management

- Each octant still has a different size
- Careful memory management is needed to avoid fragmentation
- Use a free space table