iRun: Interactive Rendering of Large Unstructured Grids

Huy T. Vo†, Steven P. Callahan†, Nathan Smith†, Cláudio T. Silva†, William Martin‡, David Owen‡, and David Weinstein‡

†Scientific Computing and Imaging Institute, University of Utah
‡Visual Influence
Motivation

- Large-scale simulations produce a lot of data
- Interactive visualization techniques not keeping up
- Walkthrough systems exist for polygonal data, why not volumes?

HAVS [Callahan et al. '05]

iWalk [Correa et al. '02]
Interactive Rendering of Large Unstructured Grids
  • Out-of-core volume traversal
  • Active set management with speculative prefetching
  • Level-of-detail interaction
  • Distributed rendering
Objective

- Interactive Walkthrough
  - Maintain responsiveness
  - Memory Insensitive
  - Low vs. High quality
  - Only render pertinent data
- Distributed Rendering
  - Use multiple machines/cores
  - Improve image quality
  - Increase display size
Issues

- Retrieval from storage
  - Out-of-core data structures
    - [Samet ’90]
  - Out-of-core algorithms
    - [Chiang et al. ’98, Farias and Silva ’01]
    - [El-Sana and Chiang ’00, Cignoni et al. ’04]

- Processing in main memory
  - Walk-through systems
    - [Clark ’76, Funkhouser et al. ’92, Aliaga et al. ’99]
    - [Varadhan and Manocha ’92, Correa et al. ’02]
  - Visibility
    - [El-Sana et al. ’01, Correa et al. ’03, Cohen-Or et al. ’03]
Issues

- Hardware-Assisted Rendering
  - Projected Tetrahedra
    - [Shirley and Tuchman '90]
  - Ray Casting
    - [Weller et al. '03, Bernardon et al. '05]
  - Hardware Assisted Visibility Sorting
    - [Callahan et al. '05, Callahan et al. '05, Callahan et al. '06]

- Display
  - Parallel GPU rendering
    - [Humphreys et al. '02]
  - Display wall rendering
    - [Moreland and Thompson '03]
Background

- Hardware Assisted Visibility Sorting (HAVS)
  - Sort in object space and image space

[Callahan et al. 2005]
http://havs.sourceforge.net and vtk/ParaView
Background

- Dynamic Level-of-Detail

[Callahan et al. '05]
http://havs.sourceforge.net and vtk/ParaView
Background

- Progressive Volume Rendering

3% 0.01 sec
33% 7 sec
66% 18 sec
100% 34 sec

[Callahan et al. ‘06]
Overview

User Interface 
Octree Traversal 
Geometry Cache 
Renderer
Preprocessing

- Memory-insensitive unique triangle extraction
  - Write triangle indices to file
  - Perform external sort
  - Extract unique entries
Preprocessing

- Out-of-core octree
  - The octree is stored on disk as a directory structure
  - Each node contains vertices and triangle indices
  - Vertices are accessed globally during insertion

```
0/
  1_0/  1_1/  1_2/  1_3/  1_4/  1_5/  1_6/  1_7/
  1_0/
    1_0_0/  1_0_1/  1_0_2/ ...
  1_0_0/
    data.vtk
...```
Out-of-core octree
- Add triangles one by one into octree
  - Use triangle-box intersections
  - Replicate triangles that span nodes
  - When node reaches capacity, split and redistribute triangles
Preprocessing

- Out-of-core octree
  - Populate parent nodes with internal geometry
    - Use area-based level-of-detail
    - Replicate triangles as they are added
Preprocessing

- Out-of-core octree
  - Populate parent nodes with boundary geometry
    - Simplify boundaries (e.g., 5%)
    - Insert into every node that is not a leaf node
  - Cleanup octree
    - Insert referenced vertices into each octree node
    - Clip triangles to node boundary
Why clip?

- Avoid compositing issues on node boundaries
Geometry Cache

- For each new camera position
  - Frustum culling for visible nodes
  - Level-of-detail culling for interaction
  - Geometry fetching
Geometry Cache

- Level-of-detail culling using priority functions, $P(\text{Camera}, \text{Node})$
  - Breadth-first search
    - $P_{\text{BFS}}(C, N) = <l, d>$
      - $l$ = depth of $N$
      - $d$ = distance of bounding box of $N$ to camera
  - Area
    - $P_{\text{area}}(C, N) = A$
      - $A$ = projected area of bounding box of $N$ on screen
Geometry Cache

- Geometry Fetching
  - Uses a separate thread from rendering
  - Moves geometry from disk to geometry cache
  - If cache is full, replaces least recently used
  - Performs speculative prefetching
Each node rendered separately in front to back order

- Largest common parent
Distributed Rendering

Diagram:
- Camera Server
- Geometry
- Render Server
- Display
- Geometry
- Camera Info
- Thumbnails
Results
Results

- Preprocessing
  - SF1 dataset
    - Input: 14 M tetrahedra, 28 M triangles (515 MB)
    - Output: 63 M triangles (1425 MB)
    - ~2.8X
    - 37 min
  - Bullet dataset
    - Input: 36 M tetrahedra, 62 M triangles (1303 MB)
    - Output: 118 M triangles (2804 MB)
    - ~2.1X
    - 1 hour 10 min
Results

- Geometry cache

![Graph showing displayed nodes and prefetching nodes over frames.](image)
Discussion

- Limitations
  - Vertices are in-core during preprocessing
  - Preprocessing output size
  - Transfer function design

- VTK
  - VTK is not thread safe!
  - Data structures not optimized for rendering

- iRun vs. iWalk
  - Level-of-detail instead of occlusion culling
  - Visibility sorting
  - Compositing
Conclusion

- Handles very large data sets
- Renders with a budget
- Scalable to multiple machines/displays

Future Work:
- Render other data types (i.e., hexahedra, mixed)
- Automatic level-of-detail technique selection
- Transfer function design for large data
- Extend for isosurfacing
Acknowledgments

Datasets
- Shepherd (SNL)
- MacLeod (Utah)
- Neely and Batina (NASA)
- O’Hallaran and Shewchuck (CMU)
- Ma (UC Davis)

Funding
- Army Research Office
- Department of Energy
- IBM
- Sandia National Laboratories
- Lawrence Livermore National Laboratory
- University of Utah