Parallel Ray Casting of Visible Human on Distributed Memory Architectures

1999. 5. 27

Chandrajit Bajaj Univ. of Texas at Austin, U.S.A. Insung Ihm* and Sanghun Park Sogang Univ., Seoul, Korea

Visualization of the Visible Man









Visible Human Dataset from NLM

- CT, MRI and RGB cryosection images
- Visible Man
 - Axial scans : 1mm interval, over 1870 cross-sections
 - Cross-sectional images : 512 x 512
 - Ex) Frozen CT 512 x 512 x 1871 x 2 bytes = 935.5MB
- Visible Woman
 - Axial scans : 1/3mm interval

The whole data sets amount to 15 GB to 40 GB.

Parallel Ray Casting of Visible Human

- Volume data are often very large.
 - Several hundred MB ~ a few GB
- Huge volume data place considerable demands on computing time and memory space.
- Volume ray-casting provides the highest rendering quality, but is one of the most compute- and memory-intensive algorithms.
- We would like to ray-cast Visible Human on a Cray T3E multiprocessor. So, how do we do that?

Interactive Visualization of Visible Human

⊗Huge volume data place considerable demands on run-time memory space.

©Develop an effective compression technique.

Section: By the section of the se

©Develop an optimization technique for volume rendering.

©Develop an effective distributed volume rendering *method*.

©Develop an effective parallel volume rendering method.

Our Approach

- In most previous rendering algorithm on distributedmemory architectures, volume data is
 - distributed over the local memory spaces in the system, and
 - redistributed between processors through remote memory fetch.
- The cost of communication for 3D volume redistribution is high. → Often a serious factor that degrades performance.
- When the entire volume data is replicated on every processor's local memory, there is no need for redistribution. But ...

Use a proper compression method to
① replicate the entire volume data at each PE in compressed form, and
② ray-cast it without 3D volume redistribution.

Zerobit Encoding: Wavelet-based 3D compression scheme for very large volume data

- Volume data are often very large.
 - Several hundred MB ~ several dozen GB
- Huge volume data place considerable demands on run-time memory space.
- ⊗ Huge volume data need a great deal of computing time for visualization.

Can we interactively visualize a 1~2GB volume dataset on my computer with 128MB of main memory?

Design goals

- High compression ratio
 - Achieve the best compression rate with minimal distortion in the reconstructed images?
 - Often impose some constraints on random access ability
- Fast run-time random access ability
 - The access patterns in interactive applications change in somewhat complicated ways.
- * "Wavelet-Based 3D Compression Scheme for Very Large Volume Data" by I. Ihm and S. Park[3,4] → "Zerobit Encoding" by I. Ihm and S. Park

- With this scheme, we would like to
 - Load the whole very large volume data, say 2GB, into a main memory of moderate size, say 128MB.
 - Allow users feel as if they have a main memory of larger size.
 - Help develop interactive applications for very large volume data on PCs, workstations, and multi-processors with limited memory.

Applications of Zerobit Encoding

(a) Vector Quantization (buddha)

(b) 3% of Coefficients (buddha)

(c) 5% of Coefficients (buddha)

(a) Uncompressed

(b) Compressed (10%)

(d) Vector Quantization (dragon)

(e) 3% of Coefficients (dragon)

(f) 5% of Coefficients (dragon)

(c) Compressed (5%)

(d) Compressed (3%)

Light Field Compression

Solid Texture Compression

Sample Statistics for Zerobit Encoding

- Visible Man Fresh CT
 - 512 x 512 x 1440 x 2 bytes = 720 MBytes
 - 32 x 32 x 90 = 92,160 unit blocks
- SGI Octane Workstation
 - MIPS 195 MHz R10000 CPU
 - 256 MBytes Main Memory

Compression Rate and Fidelity

	3%	5%	7%	10%	15%
Compressed Data Size (MB)	24.6	35.1	45.4	60.5	83.5
Compression Ratio	29.3	20.5	15.6	11.9	8.6

• Error in Voxel Values

	3%	5%	7%	10%	15%
SNR (dB)	22.6	25.9	28.6	32.0	36.4
PSNR (dB)	44.5	47.8	50.4	53.8	58.2

• Error in Normals

	3%	5%	7%	10%	15%
Skin (deg)	15.4	11.4	8.6	6.0	3.9
Bone (deg)	24.7	19.0	15.0	10.8	6.7

Sample Slices

Ray-Cast Images

Random Access Time (512 Slices)

• Zerobit Encoding

							Uncompres
		3%	5%	7%	10%	15%	
Pure Random	(1M times)	3.33	3.49	3.67	3.85	4.16	2.78
Cell-Wise	All	3.88	4.88	5.99	7.52	9.54	18.88
	Skin	2.28	2.91	3.53	4.36	5.55	6.50

• Previous [3,4]

		3%	5%	7%	10%	15%	Uncompressed
Pure Random	(1M times)	6.68	7.06	7.46	7.94	8.69	2.78
Cell-Wise	All	26.55	27.62	28.47	29.63	30.97	18.88
	Skin	9.66	10.19	10.69	11.24	11.94	6.50

Compression-Based Parallel Ray-Casting

• Image partition for tasks

- Image screen is partitioned into tiles of small sizes, forming a task queue.
- Each processor repeatedly gets a task from the queue, and render the corresponding tile.
- Load balancing is performed dynamically.

Object partition for data

- Volumes are partitioned into 16x16x16 sub-blocks, called *unit blocks*, that are subdivided into 4x4x4 sub-blocks, called *cells*.
- The cell is a basic decoding unit for zerobit encoding.
- Rays are traversed unit block by block for efficiency, decompressing cell by cell.
- Use a min-max octree for effective front-to-back traversal.

Object-Order Aspect of Our Ray-Casting

• For an assigned tile,

- Use an octree to enumerate, in front-to-back order, the unit blocks that intersect with the view volume, and contains voxels of interest.
- For each cell in the enumeration,
 - For each ray segment inside the cell,
 - Render the segment, and accumulate its color and opacity.

Why object-order traversal?

- Block-wise traversal is more appropriate for minimizing the reconstruction costs than ray-by-ray traversal is.
 - In our compression-based scheme, voxels must be decompressed as necessary when they are not in cache.
 - Decompression v.s. remote fetch.
 - It is important to
 - minimize the number of decompression operations, and
 - utilize voxels maximally once decompressed.
- By block-wise traversal, data coherence in object space is easily exploited.

Image-Order Aspect of Our Ray-Casting

- The early ray termination technique is not natural to object-order ray-casting.
- Apply a quadtree-based early ray termination technique:
 - For a tile to be rendered, maintain a quadtree on tile's pixels, dynamically.
 - A node have a value that indicates if its corresponding sub-region is opaque or not.
 - For each interesting unit block, project it onto the tile, and traverse the quadtree to quickly remove opaque regions from consideration.
 - If the root is opaque, stop rendering for the tile.

Experimental Results

- Test volume data
 - Fresh CT of Visible Man
 - Resolution: 512 x 512 x 1440
 - Size: 720 MBytes
 - Compressed data
 - Approximate ratio of wavelet coefficients to be used: 7%
 - Size: 45.43 MBytes
- Classification: skin
- Test tile sizes: 16x16, 32x32, 64x64
- Image resolution: 512 x 1024

Cray T3E-900 at ETRI: Distributed-Memory Parallel Computer

- 136 Processing Elements (PEs): 112 for Applications, 21 for Command, and 3 for OS only
- Each Processing Element (PE) includes
 - a 450 MHz Alpha processor, and
 - 128 Mbytes local memory.
- PEs are connected by a high-bandwidth, low latency bidirectional 3D torus system interconnect network.
- Programming tools
 - PVM
 - MPI

✓ SHMEM Library (Cray Shared Memory Library)

Parallel Ray-Casting Snapshots

Rendering Time

Speedup

29

Communication Time

Load Balancing

31

Conclusion

- An effective compression-based parallel ray-casting scheme for very large volume data on distributed memory architectures.
 - Simple and easy to implement.
 - Attempt to achieve high performance by minimizing costly communication between PEs through data compression.
 - Exploit both object- and image-space coherence.
 - Also very appropriate for PC/Workstation clusters, connected via low bandwidth links.

Future Works

- Further optimization of our codes.
- Development of applications on distributed systems.
- What if the data is so large that its compressed version can not be loaded into a local memory?

