Out-of-Core Simplification and Crack-Free LOD Volume Rendering for Irregular Grids

Zhiyan Du and Yi-Jen Chiang

Polytechnic Institute of New York University, NY, USA

Volume Rendering for Tetrahedral Meshes

- Large datasets pose a big challenge in efficiency → Out-of-core and level-of-detail (LOD) methods are essential
- Existing approaches:
  - GPU-based methods: can only handle small datasets
  - LOD methods: very few (mostly in-core)
  - Out-of-core methods: either do not support LOD or do not completely resolve the crack-free issue (explained in a moment)
- Our Goal: To perform out-of-core simplification and LOD volume rendering efficiently while resolving the crack-free issue completely

Crack-Free Issue & Our Contribution

* In out-of-core setting, we need to use block-based structure: sub-volumes from space decomposition (e.g., octree, kd-tree, etc.)
* Crack-free issue: neighboring sub-volumes at different levels of tree in the selected LOD mesh must have consistent boundaries
* Crack-free LOD for general meshes in out-of-core is very challenging
  - Naïve way: not to simplify the boundary --- low quality, not desirable
  - Only a few methods address this issue --- Not guaranteed that the boundary cells can always be simplified (simplified at future levels)

* **Our new out-of-core algorithm:**
  1. Guarantees to simplify both boundary & interior at each current level
  2. Achieves crack-free LODs with a theoretical guarantee
  3. Supports selective-refinement LODs: highest image quality, much faster speed

Previous Work

- GPU-based methods: see survey [Silva et al 05]
  --- in-core, single resolution
- LOD methods: [Cignoni et al 04] [Callahan et al 05]
  --- in-core
- Out-of-core methods
  * [Farias et al 01] [Chiang et al 01] --- no LOD support
  * Progressive rendering [Callahan et al 06]
  --- preprocessing is in-core, no mesh LODs
  * iRun [Vo et al 07] --- sample-based, no mesh LODs
  * [Sondershaus et al 06] --- mesh LODs, boundary not simplified

Previous Out-of-Core Crack-Free LOD Methods

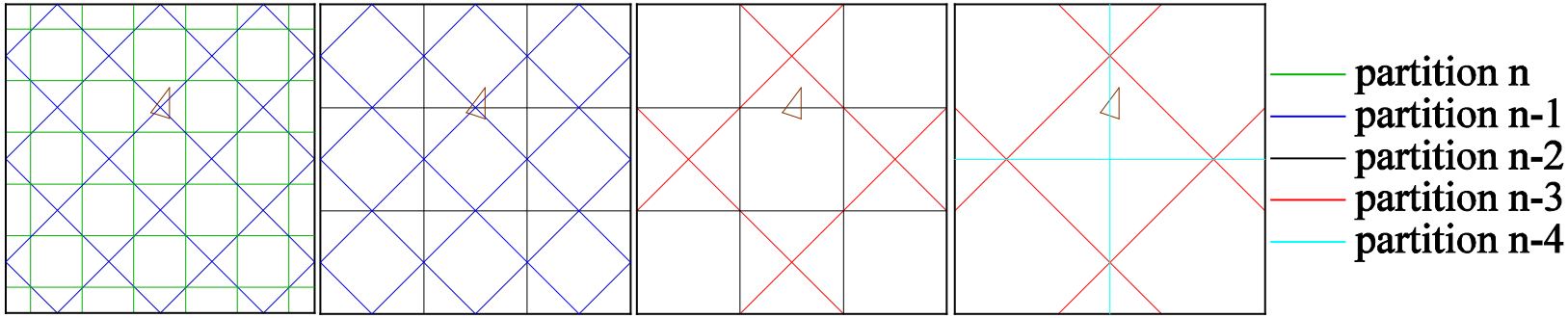
- Triangle Meshes
  * TetraPuzzles [Cignoni et al 04]
  * Quick VDR [Yoon et al 04]
  * Progressive Buffers [Sander & Mitchell 05]
  * Batched Multi Triangulation [Cignoni et al 05]
  --- each has some un-resolved issues (see paper for details)

- Tetrahedral Meshes
  * Segment-Based Tetrahedral Meshing [Sondershaus et al 06] --- based on Batched Multi Triangulation [Cignoni et al 05]

Previous Best Out-of-Core Crack-Free LOD Method(s)

[Cignoni et al 05], [Sondershaus et al 06] (based on Batched Multi Triangulation)

- Use a sequence of coarser space partitions; at each level two consecutive partitions are super-imposed. Only cells lying entirely within a region can be simplified.
- Cannot guarantee that the boundary cells can always be simplified (e.g., the triangle shown below). [They must be simplified at future levels]
- Neighboring nodes in LOD must have level difference at most 1

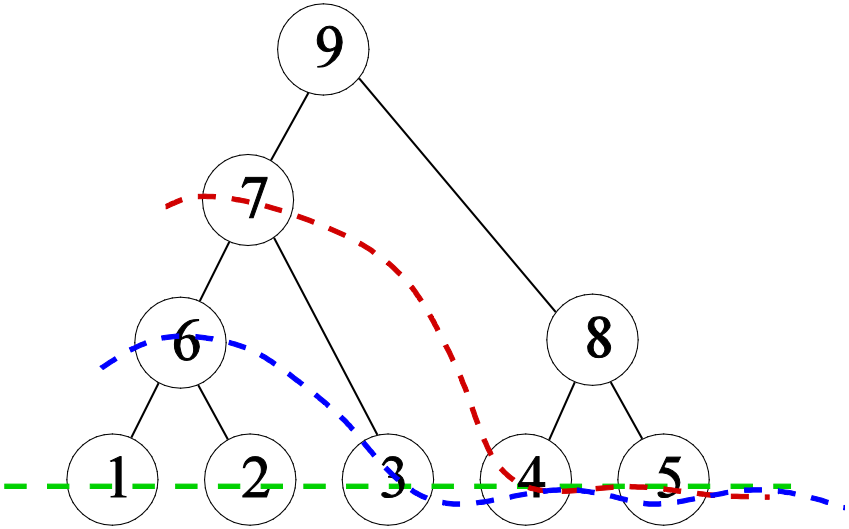


Our New Algorithm

- Out-of-core simplification and crack-free LOD volume rendering
- Smooth & high-quality simplification
  - Guarantees to simplify both boundary & interior at each current level
  - Uses edge collapses with extended quadric error metric [Garland & Zhou 05]
- Achieves crack-free LODs with a theoretical guarantee
- Supports selective-refinement LODs --- highest image quality, much faster speed
- Integrates GPU-based HAVS [Callahan et al 05] as volume rendering engine --- out-of-core HAVS as a by-product

Key Idea 1: Error-Based LOD Cuts

- Merge Tree (only tree skeleton is kept in-core):
  - Spatially partition input mesh into sub-volumes of roughly same # of vertices (meta-cell technique [Chiang et al 98]). These sub-volumes are tree leaves.
  - Simplify and merge neighboring sub-volumes bottom-up recursively.
  - Create & simplify one node at a time. Propagate the boundary updates to its neighbors so that their boundaries are always consistent → One new node creates one new breadth cut (n new nodes <-> n+1 cuts)



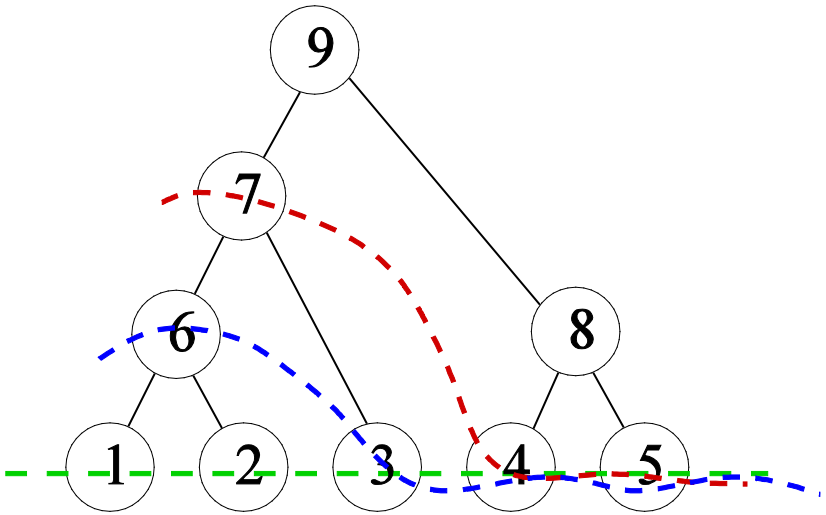
  - Create the new nodes in the order of increasing errors
    → For any query error ε, the LOD cut for ε must be one of the n+1 breadth cuts created before (boundary consistent)

Simplification for Continuous LODs

- For each node (sub-volume), perform 2 phases:
  1. Global simplification: Simplify globally including the interior & boundary (in increasing errors up to ε_l), and propagate updates to neighbors. Set a simplification lower bound ε_l
  2. Internal simplification: Further simplify only the interior (in increasing errors), to get a simplification upper bound ε_u (progressive representation: the reverse linear sequence gives the refinement sequence) --- for continuous LODs
- Resulting mesh: base mesh, with a refinement sequence & error bound range $[\varepsilon_l, \varepsilon_u]$

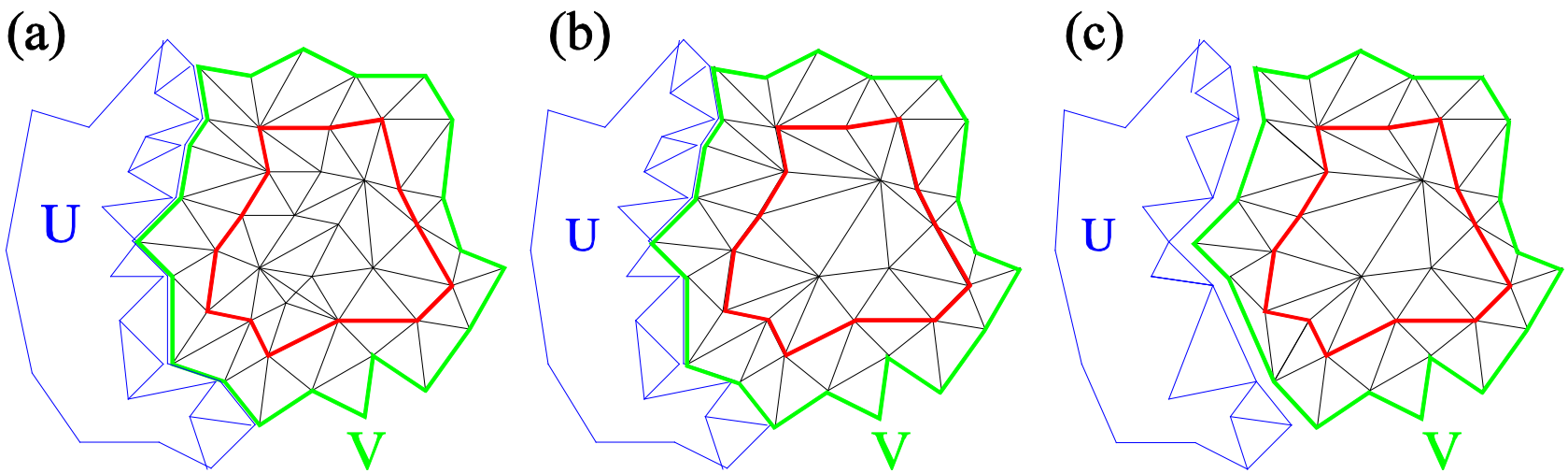
Key Idea 2: Progressive Boundary & Interior with Fire Wall

- Issue: Same sub-volume V in different LODs --- different neighbors with different boundaries → Progressive boundary



- After a node V finishes its global & internal simplifications (base mesh), future neighbor simplifications will propagate updates to further simplify the boundary of V → Collect the updates into a simplification sequence for V
- This requires us to refine interior & simplify boundary of V independently.

Solution: Fire-Wall

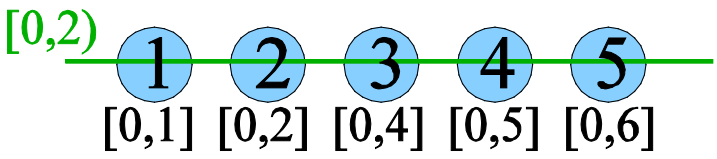


- V is simplified before U.
  (a) After global simplification on V. Define the fire wall (in red) of V.
  (b) After internal simplification on V. The fire wall is made intact.
  (c) Simplifying U updates the boundary of V. The fire wall is intact.
- Not restrictive: First perform global simplification (already simplify V as desired), then identify fire wall and simplify interior.

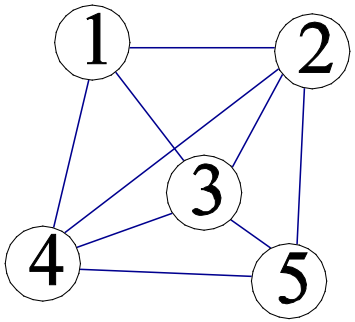
Simplification Algorithm

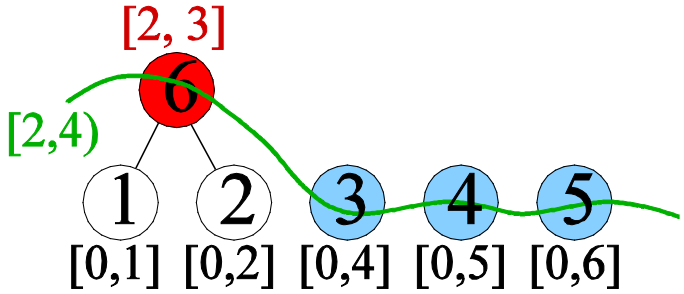
- For each leaf only simplify its interior (up to pre-defined size c) to get error bound [0, ε_u]. Put all leaves to a priority queue Q with error upper bound ε_u being the key.
- Repeat the following recursively:
  1. w ←Extract_Min(Q).
  2. Check the place holder, **pool**, for w's neighbors:
     - If not found: Put w into pool.
     - If found:
       * Merge those neighbors with w into a new node x (also remove these nodes from pool)
       * Set the error lower bound of x: $\varepsilon_l(x) = \varepsilon_u(w)$.
       * Simplify new node x globally up to error lower bound $\varepsilon_l(x)$.
       * Simplify x internally up to size c & get error upper bound $\varepsilon_u(x)$.
       * Put x into Q with key $\varepsilon_u(x)$.

Simplification Example

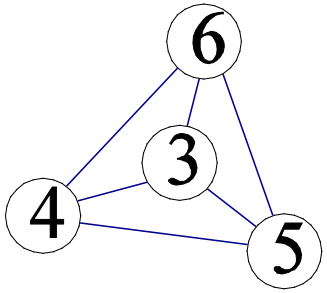


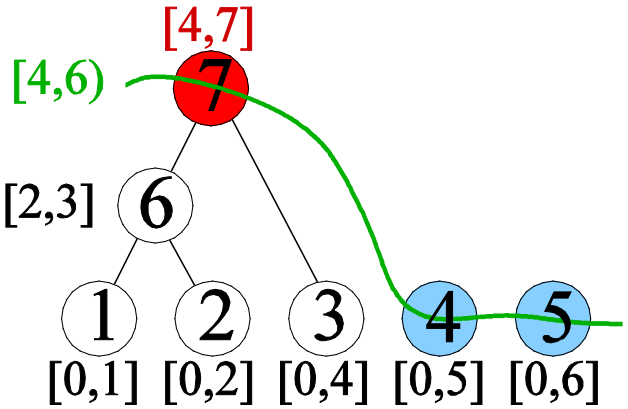
pool=empty
min(priority Q)= 1



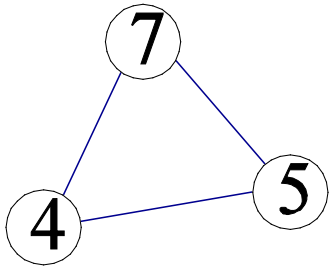


pool={ 1 }
min(priority Q)= 2

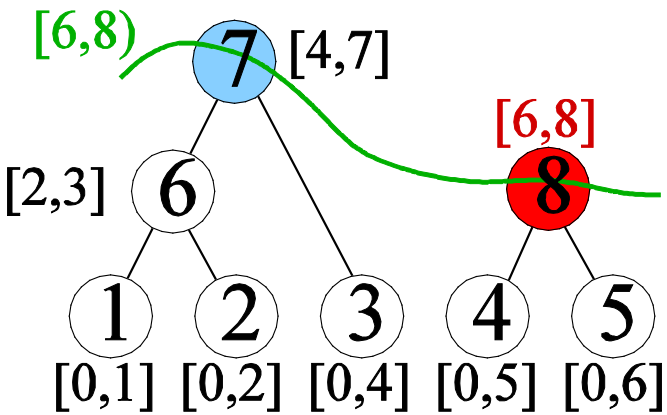




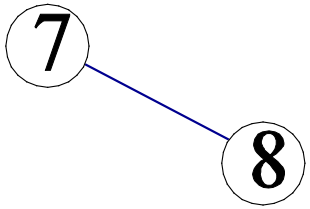
pool=empty
min(priority Q)= 6
pool={ 6 }
min(priority Q)= 3

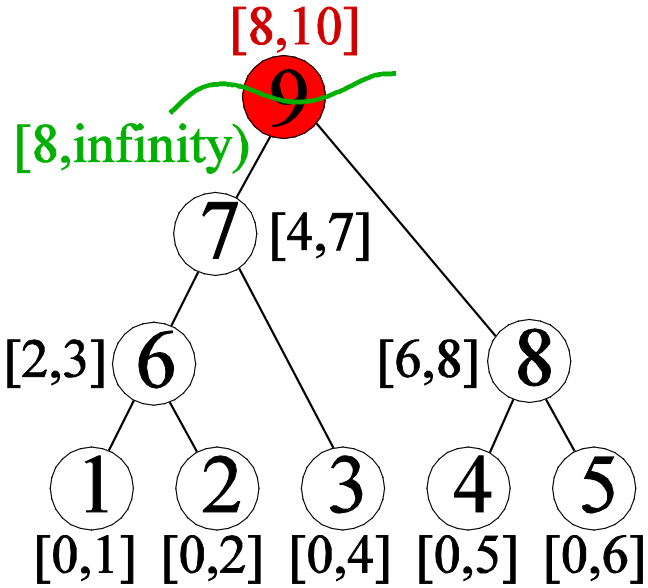


Simplification Example (con'd)



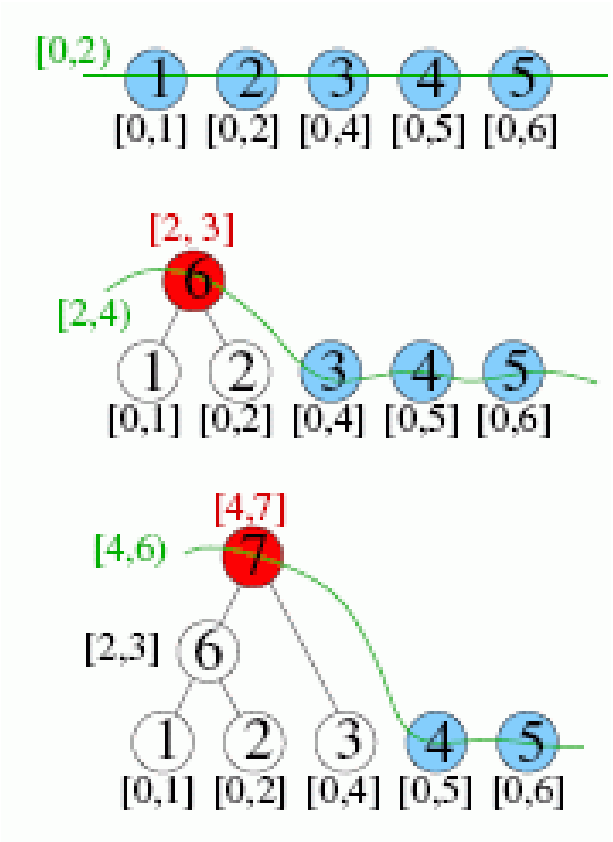
pool=empty
min(priority Q)= 4
pool={ 4 }
min(priority Q)= 5

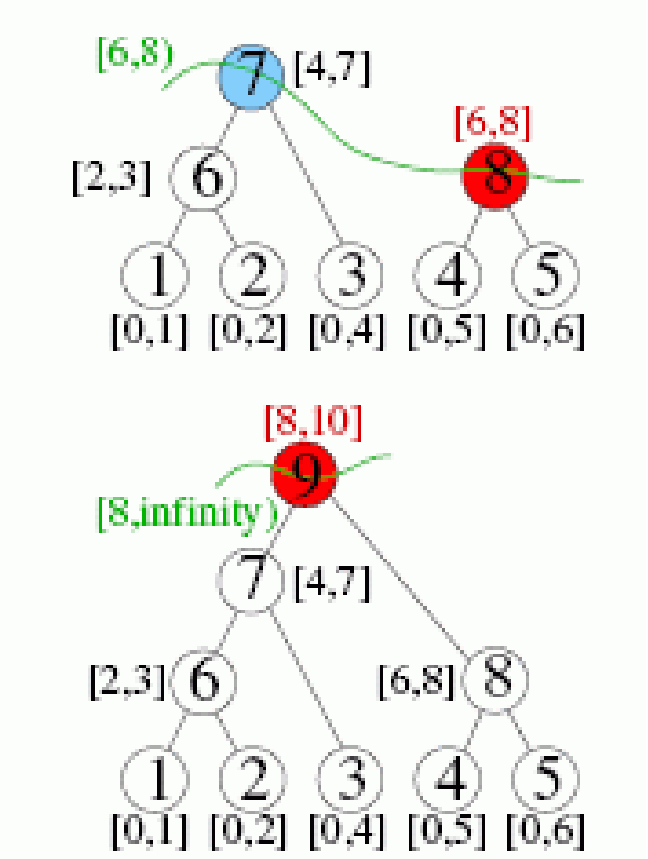




pool=empty
min(priority Q)= 7
pool={ 7 }
min(priority Q)= 8
pool=empty
min(priority Q)= 9
pool= 9
priority Q=empty

Properties of the LOD Cuts





- New nodes 6, 7, 8, 9 are created with ε_l = 2, 4, 6, 8 (increasing), which partition the whole range [0, infinity) into 5 ranges [0, 2), [2,4), [4,6), [6,8), [8, infinity]
- Query against ε_l : For any query error ε, the LOD cut for ε must be one of the 5 breadth cuts created before (already made boundary consistent)

Selecting the LOD Mesh (1): Uniform LOD

Uniform LOD: the LOD mesh satisfying query error bound ε everywhere

- On the Merge Tree, find the LOD cut satisfying ε
- For each node in the cut:
  - Use the simplification sequence to simplify the boundary (up to ε) to get the boundary version of this LOD cut (consistent boundary, crack-free)
  - Use the refinement sequence to refine the interior up to ε

Selecting the LOD Mesh (2): Selective-Refinement LOD

Selective-Refinement LOD: the LOD mesh with the highest LOD cut where the selected sub-volumes satisfy ε and other sub-volumes are just enough detailed to ensure crack-free

- Find the LOD cut satisfying ε
- Select nodes from the cut:
  M1. view-dependent (t%): select the t% nodes closest to the viewer
  M2. scalar-range ([a,b]): select the nodes whose scalar-value range [min, max] intersects [a,b]
- Make the LOD cut go higher (highest possible yet still going thru the selected nodes) (see paper) --- fewer cut nodes
- In the new LOD cut:
  * Make boundary consistent as before
  * Only refine the interior of selected nodes up to ε (for other nodes, use the base-mesh interior without any refinement)

Experiments: Simplification

- **Datasets**: up to 101M cells, 3.9 GB
  [**Remark:** In-core HAVS needs 6-10 times of dataset size for its memory footprint (e.g., 13.1GB footprint for a 2.1GB dataset)]
- **PC**: RAM size 1.5GB, GPU memory 512MB
- **Out-of-core simplification**
  - Tree size up to 82.5 KB
  - Final total size on disk up to 4GB
  - Memory footprint up to 1.2 GB
  - Total time up to 4.5 hours
    (partition: 0.92 hours; simplification: 3.53 hours)
- **Comparison: In-core simplification**
  - For dataset of 148MB, memory footprint was 1.8GB and thrashing occurred (512.5s vs. 295.6s out-of-core time)

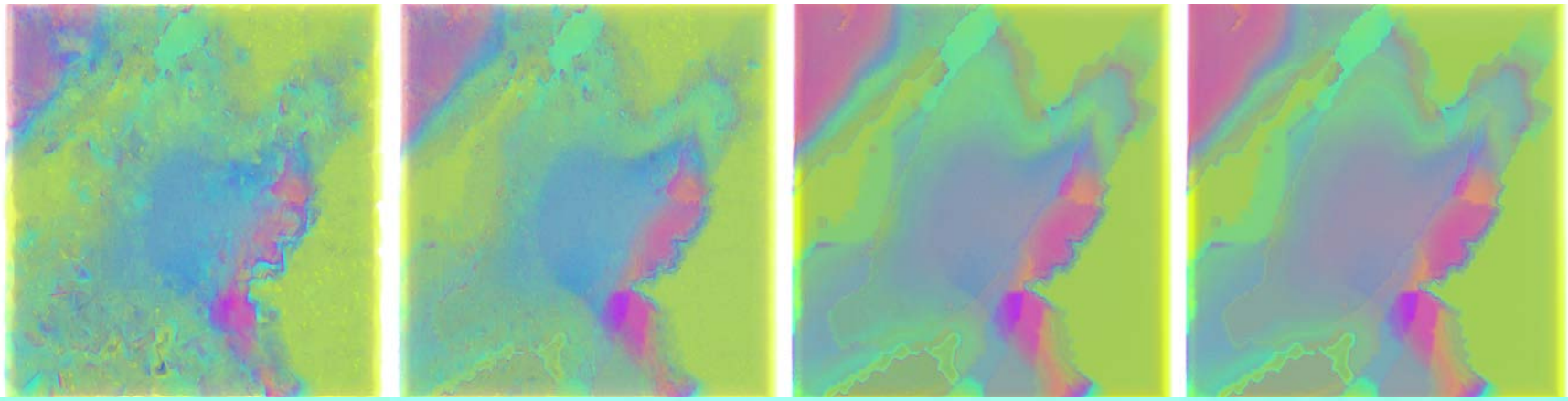
Experiments: Rendering

- Single Resolution
  - In-core HAVS [Callahan et at 05]
    2.1 GB data --- 13.1GB memory footprint, >> 24 hours
  - Out-of-core HAVS
    * 2.1GB data --- 630MB footprint, 6.3 minutes
    * 3.9GB data --- 870 MB footprint, **19.7 minutes**
- **Out-of-Core Uniform LOD**
  - Speed depends on LOD resolution (ratio of extracted vs. total # triangle faces)
  - 3.9 GB data --- 0.07s (**14 fps**) at 0.21% resolution
    [at 100% resolution: 870 MB footprint, **22 minutes**]
- **Out-of-Core Selective-Refinement LOD**
  - Selected parts at full resolution → almost the same image quality as the best one (i.e. full resolution everywhere)
  - 3.9 GB data: Scalar-range (ave. 47.8%): 4.5 minutes
  - 3.9 GB data: View-dependent (ave. 38.56%): **3.82 minutes**

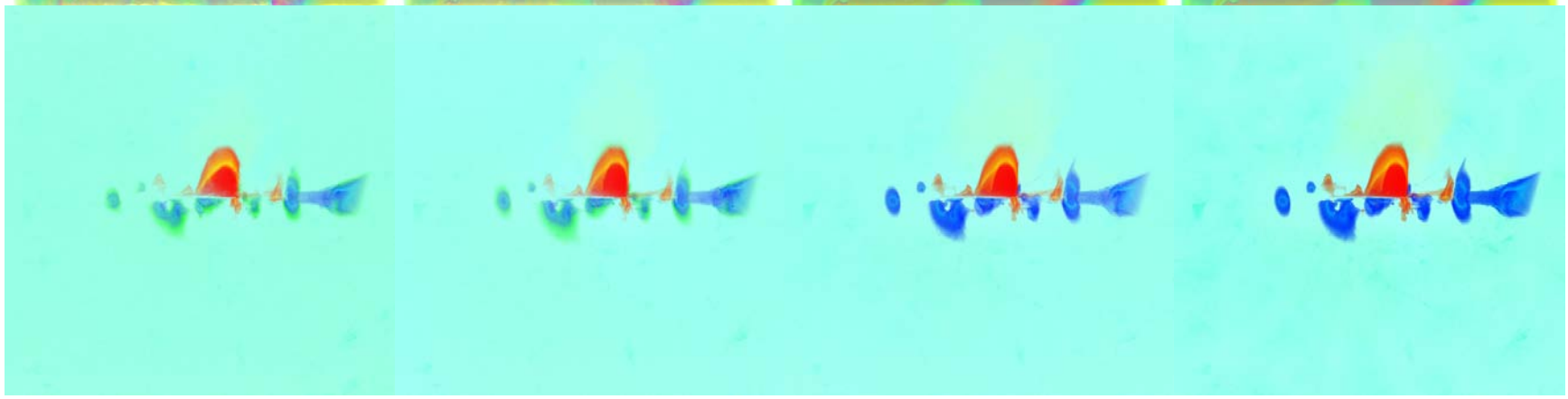
Uniform LOD

LOD Resolution: 0.28% 14.45% 32.8% 100%

2.1GB data



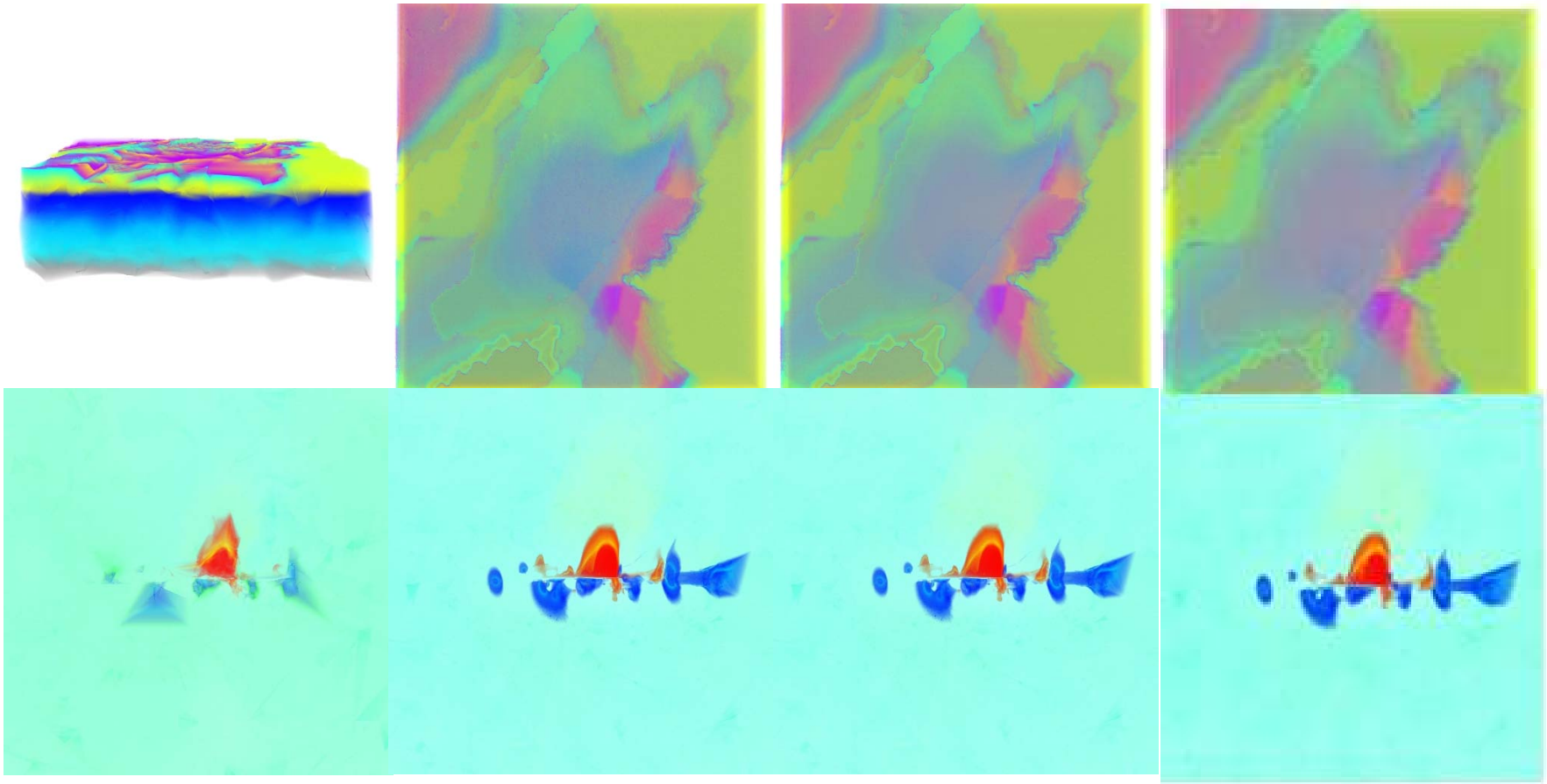
3.9GB data



LOD Resolution: 0.21% 10.27% 25.66% 100%

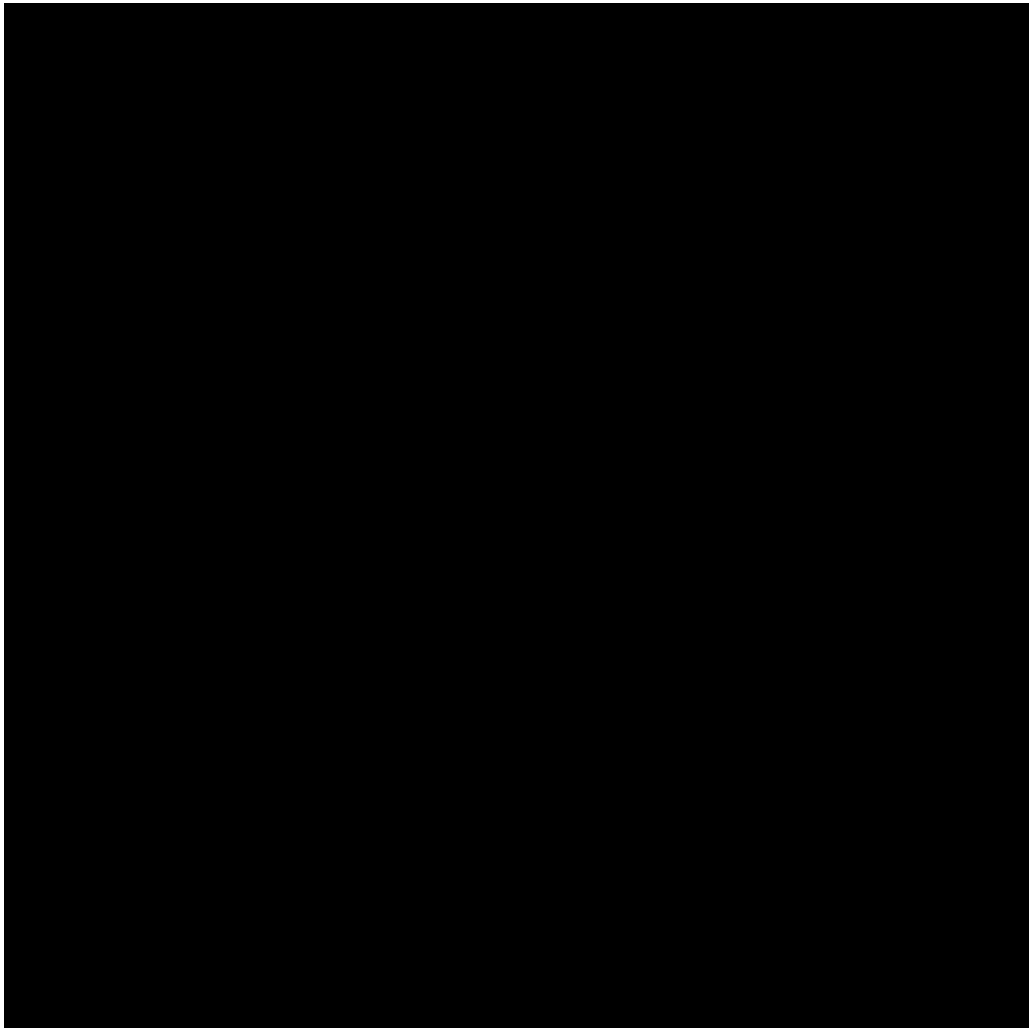
Selective-Refinement LOD

Rough view	view-dependent	scalar-range	original
ave. LOD	58.12%	70.05%	(100%)



ave. LOD	38.56%	47.8%	(100%)

Interaction (Video)



Conclusions

- The first out-of-core algorithm that can always simplify both boundary & interior at each current level, while achieving crack-free LODs with a theoretical guarantee
- Supports selective-refinement LODs
- Achieves significant speed-ups in both simplification & rendering
- Extension: out-of-core LOD isosurface extraction

Acknowledgments

- We thank Claudio Sliva and Steven Callahan for the HAVS code and the test datasets

- NSF grant CCF-0541255, NSF CAREER grant CCF-0093373