Out-Of-Core Construction of Sparse Voxel Octrees

Jeroen Baert, Ares Lagae & Philip Dutré

Computer Graphics Group, KU Leuven, Belgium
Voxel-related research

- **Voxel Ray Casting**
  - Gigavoxels (Crassin, 2009-...)
  - Efficient SVO's (Laine, Karras, 2010)

- **Voxel Cone Tracing**
  - Indirect Illumination (Crassin, 2011)

- **Voxel-based Visibility**
  - Voxelized Shadow Volumes (Wyman, 2013, later today!)

- ...
Why voxels?

- Regular structure
- Hierarchical representation in Sparse Voxel Octrees (SVO's)
  - Level of Detail / Filtering
- Generic representation for geometry and appearance
  - In a single data structure
Polygon mesh to SVO

• We want **large**, highly **detailed** SVO scenes

• Where do we find content?

• Let's voxelize massive polygon meshes
  - Majority of current content pipelines is polygon-based
What do we want?

- Algorithm requirements:
  - Need an **out-of-core** method
    - Because polygon mesh & intermediary structures could be >> system memory
  - Data should be streamed in/out
    - from disk / network / other process
  - Ideally: out-of-core as fast as in-core
Pipeline construction (1)

- **Voxelization** step
  - Polygon mesh → Voxel grid
- Followed by **SVO construction** step
  - Voxel grid → Sparse Voxel Octree
Pipeline construction (2)

• Key insight:
  – If voxel grid is **Morton-ordered**
  – SVO construction can be done **out-of-core**
    • Logarithmic in memory usage ~ octree size
    • In a streaming manner
  – So voxelization step should deliver ordered voxels
Pipeline construction (3)

- High-resolution 3D voxel grid may be \(\gg\) system memory
  - So \textbf{partitioning} step (into subgrids) is needed
  - Separate triangle streams for each subgrid
Final Pipeline

- Now, every step in detail ...
Morton order / Z-order

- Linearization of n-dimensional grid
  - Post-order depth-first traversal of $2^n$-tree
- Space-filling curve, Z-shaped
Morton order / Z-order

- Hierarchical in nature
- Cell at position \((x,y)\)
  - Morton code
  - Efficiently computed
  - \((x,y,z) = (5,9,1)\)
    → \((0101,1001,0001)\)
    → 010001000111
    → 1095^{th} cell along Z-curve
Partitioning subprocess

- **Partitioning** (1 linear pass)
  - Into power-of-2 subgrids until it fits in memory
  - Subgrids temporarily stored on disk
  - Subgrids correspond to contiguous range in Morton order
- If we voxelize subgrids in Morton order, output will be Morton-ordered
Voxelization subprocess (1)

- **Voxelize** each subgrid in Morton order
  - **Input**: Subgrid triangle stream
    - Each triangle voxelized **independently**
  - **Output**: Morton codes of non-empty cells
    - Typically, majority of grid is empty
Voxelization subprocess (2)

- We use a simple voxelization method
  - But any method that works one triangle at a time will do

Morton codes of non-empty cells
Out-of-core SVO Construction

- **Input**: Morton-ordered voxel grid
- **Output**: SVO nodes + referenced data
SVO Construction algorithm in 2D

• Required: **queues** of $2^d$ nodes / octree level
  – Ex: $2048^3$ grid → $11 \times 8$ octree nodes-

```
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
Level 0
Level 1
Level 2
```

```
In Memory / On-disk
```

```
Octree representation
```

```
SVO Builder queues
```

Out-Of-Core Construction of Sparse Voxel Octrees
SVO Construction algorithm in 2D

- Read Morton codes 0 → 3 (+ voxel data)
  - Store them in level 2 queue
  - Level 2 queue = full
SVO Construction algorithm in 2D

• Create internal parent node
  - With level 1 Morton code 0
  - Store parent-child relations
  - Write non-empty level 2 nodes to disk+clear level 2
SVO Construction algorithm in 2D

- Read Morton codes 4 → 7 (+ voxel data)
  - Store them in level 2 queue
SVO Construction algorithm in 2D

- Create internal parent node
  - With level 1 Morton code 1
  - Store parent-child relations (there are none)
  - Write non-empty level 2 nodes to disk+clear level 2

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

SVO Builder queues

Octree representation
SVO Construction algorithm in 2D

- Same for Morton codes $8 \rightarrow 11$
SVO Construction algorithm in 2D

- Same for Morton codes $12 \rightarrow 15$
SVO Construction algorithm in 2D

- Now **level 1** is full
  - Create parent node (**root node**)
  - Store parent-child relations
  - Write non-empty **level 1** nodes to disk+clear **level 1**
SVO Construction: optimization

- Lots of processing time for empty nodes
  - Sparseness = typical for high-res voxelized meshes

- Insight for optimization
  - Pushing back $2^d$ empty nodes in a queue at level $n$
  - = Pushing back 1 empty node at level $n-1$
SVO Construction: optimization

- Implementation details in paper
- Optimization exploits sparseness of voxelized meshes
- Speedup: two orders of magnitude
  - Building SVO from grid:
    - David: 471 vs 0.55 seconds
    - San Miguel: 453 vs 1.69 seconds
Results: Tests

• Resolution: $2048^3$

• Memory limits
  - 8 Gb (in-core)
  - 1 Gb (out-of-core)
  - 128 Mb (out-of-core)

• Models
  - David (8.25 M polys)
  - San Marco (7.88 M polys)
  - XYZRGB Dragon (7.2 M polys)
Results: Out-Of-Core performance

- Out-Of-Core method = \(~ as fast as In-Core\)
  - Even when available memory is 1/64
Results: Time breakdown

- Partitioning speedup from skipping empty space
Results: Extremely large models

- $4096^3$ – In-core: 64 Gb
- **Atlas** model
  - 17.42 Gb, 507 M tris
  - < 11 min at 1 Gb
- **St. Matthew** model
  - 13.1 Gb, 372 M tris
  - < 9 min at 1 Gb
Results: SVO Construction

• SVO output stream
  - Good locality of reference
    • Nonempty siblings on same level always stored next to each other
  - Nodes separated from data itself (separation hot/cold data)
    • Using data pointers + offsets as reference
Appearance

- Pipeline: **binary** voxelization
- Extend with **appearance data**?
  - Interpolate vertex attributes (color, normals, tex)
  - Propagate appearance data upwards

- Global data access ↔ Out-Of-Core algorithms
  - Multi-pass approach
Conclusion

- Voxelization and SVO construction algorithm
  - Out-of-core as fast as in-core
  - Support for extremely large meshes

- Future work
  - Combine with GPU method to speed up voxelization
  - Handle global appearance data
Thanks!

- **Source code / binaries** will be available at project page

- **Contact**
  - jeroen.baert @ cs.kuleuven.be
  - @jbaert

- **Acknowledgements**:
  - Jeroen Baert funded by Agency for Innovation by Science and Technology in Flanders (IWT), Ares Lagae is a Postdoctoral Fellow of the Research Foundation – Flanders (FWO), David / Atlas / St. Matthew models : Digital Michelangelo project, San Miguel model : Guillermo M. Leal Llaguno (Evolucien Visual)