

High Resolution Sparse Voxel DAGs

Prateek Chandan Anurag Shirolkar

Department of Computer Science
IIT Bombay

CS 775

Outline

- 1 Abstract
- 2 Introduction
- 3 Previous Work
 - Sparse Voxel Octrees
 - Common subtree merging
 - Shadows
 - Ambient Occlusion
- 4 Reducing a Sparse Voxel Tree to a DAG
- 5 Ray Tracing a Sparse Voxel DAG
- 6 Evaluation
 - Reduction Speed
 - Memory Consumption
 - Traversal Speed
- 7 Conclusion
- 8 Future Work

Abstract

- A binary voxel grid can be represented more efficiently by using a Directed Acyclic Graph(DAG) than Sparse Voxel Octree(SVO)
- Along with efficient encoding of empty regions , DAG allows efficient encoding of identical regions as nodes are allowed to share pointers
- This paper presents a bottom-up algorithm to reduce a SVO to DAG
- The memory consumption of DAG is significantly smaller and it would fit in memory where SVO won't
- DAG requires no decompression and can be traversed easily
- This paper demonstrates this by ray tracing hard and soft shadows , ambient occlusion , and primary rays in extremely high resolution DAG at speed at par or even faster than voxel and triangle GPU ray tracing.

Introduction

- There is increased interest in evaluating secondary rays for effects as reflection, shadows and indirect illumination. Due to the incoherence, a secondary scene representation is required with strict memory budget
- This acceleration structure has to be resident on RAM for efficient access.
- However this is problematic when triangle meshes are augmented with displacement maps as they become infeasibly large to millions of polygons
- Recently Sparse Voxel Octree(SVO) has been used as a secondary scene representation, since they can be efficiently ray traced in real time.
- At high resolution, SVO is still too much memory expensive, therefore it's application has been limited

Introduction

- The data structure **DAG** allows for extremely high resolution enabling us to improve image quality, decrease discretization artifacts, explore high frequency effects like sharp shadows in large scenes.

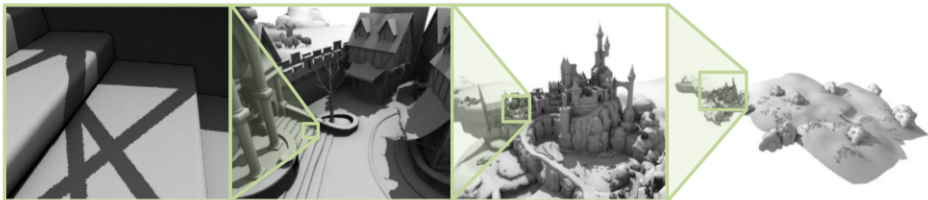


Figure 1: The EPIC CITADEL scene voxelized to a $128K^3$ resolution and stored as a Sparse Voxel DAG.

Introduction

- The paper presents an efficient technique for reducing the size of SVO by merging common subtrees and transforming to a DAG.
- This can dramatically reduce memory demands in real world scenes and scales even better with increased resolutions.
- This is based on assumption that the original DAG contains a large amount of redundant subtrees
- This paper implements and evaluates algorithms for calculating hard and soft shadows and ambient occlusion by Ray tracing in DAG

Outline

- 1 Abstract
- 2 Introduction
- 3 **Previous Work**
 - Sparse Voxel Octrees
 - Common subtree merging
 - Shadows
 - Ambient Occlusion
- 4 Reducing a Sparse Voxel Tree to a DAG
- 5 Ray Tracing a Sparse Voxel DAG
- 6 Evaluation
 - Reduction Speed
 - Memory Consumption
 - Traversal Speed
- 7 Conclusion
- 8 Future Work

Sparse Voxel Octrees

- An octree is a spatial data structure, which subdivides each node into 8 equally large subnodes
- Sparse Voxel Octree has most of the nodes empty and its base primitive is a voxel.
- *Efficient Sparse Voxel Octree* - avoid building the octree to its maximal depth by providing each cubical voxel with a contour

Outline

- 1 Abstract
- 2 Introduction
- 3 **Previous Work**
 - Sparse Voxel Octrees
 - **Common subtree merging**
 - Shadows
 - Ambient Occlusion
- 4 Reducing a Sparse Voxel Tree to a DAG
- 5 Ray Tracing a Sparse Voxel DAG
- 6 Evaluation
 - Reduction Speed
 - Memory Consumption
 - Traversal Speed
- 7 Conclusion
- 8 Future Work

Common subtree merging

- Work by Webber and Dillencourt [1989], where quadtrees are compressed by merging common subtree
- Shows a significant decrease (up to 10) in the number of nodes required to represent a 512 512 binary image
- Parker et al. [2003] extends this to three dimensions

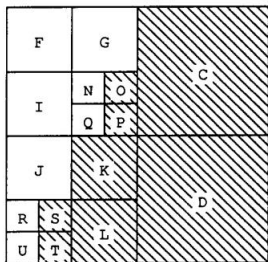


Figure 2. Normal quadtree for raster image.

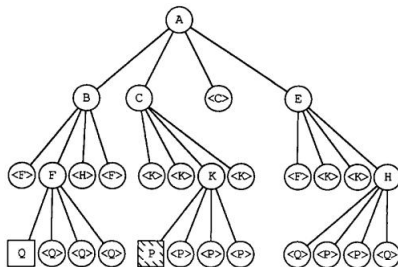


Figure 4. Pointer representation of CSM-quadtree for raster image.

Figure 2:

Outline

- 1 Abstract
- 2 Introduction
- 3 **Previous Work**
 - Sparse Voxel Octrees
 - Common subtree merging
 - **Shadows**
 - Ambient Occlusion
- 4 Reducing a Sparse Voxel Tree to a DAG
- 5 Ray Tracing a Sparse Voxel DAG
- 6 Evaluation
 - Reduction Speed
 - Memory Consumption
 - Traversal Speed
- 7 Conclusion
- 8 Future Work

- *Image based methods* - occluders are somehow recorded as seen from light source in an image, which is later used to determine light visibility while rendering.
- *geometry based methods* - idea of generating shadow volumes that enclose shadowed space. Extended to Soft Shadow Volumes in case of area lights.
- Due to recent advances in real-time ray tracing performance, casting a few shadow rays per pixel is now within reach for interactive applications

Outline

- 1 Abstract
- 2 Introduction
- 3 **Previous Work**
 - Sparse Voxel Octrees
 - Common subtree merging
 - Shadows
 - **Ambient Occlusion**
- 4 Reducing a Sparse Voxel Tree to a DAG
- 5 Ray Tracing a Sparse Voxel DAG
- 6 Evaluation
 - Reduction Speed
 - Memory Consumption
 - Traversal Speed
- 7 Conclusion
- 8 Future Work

Ambient Occlusion

- Technique to determine exposure of each point to ambient light.
- *Screen Space Ambient Occlusion* - Scaled down version of AO where only scene depth buffer is used to determine the occlusion.
- In an alternative approach a volume is generated per primitive with occlusion being computed at runtime
- *Fast Ray-Cast Ambient Occlusion* - occlusion volume is limited by a hemisphere around the receiver point.

Reducing a Sparse Voxel Tree to a DAG

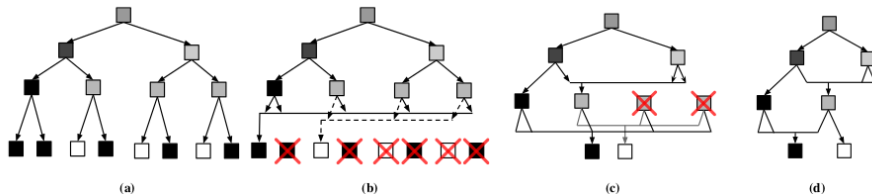


Figure 2: Reducing a sparse voxel tree, illustrated using a binary tree, instead of an octree, for clarity.

- The original tree.
- Non-unique leaves are reduced.
- Now, there are non-unique nodes in the level above the leaves. These are reduced, creating non-unique nodes in the level above this.
- The process proceeds until we obtain our final directed acyclic graph.

Terminology

- Consider an N^3 voxel grid as a scene representation where each cell can be represented by a bit : **0** if empty and **1** if it contains geometry
- A sparse voxel octree recursively divided L times gives us a hierarchical representation of an N^3 voxel grid, where $N = 2^L$ and L is called as *max level*.
- A node is implemented with a *childmask*, a bitmask of 8 bits where bit i tells us if bit i contains a geometry, and a pointer to first of non-empty children.
- Child nodes are stored consecutively in memory
- The unique traversal *path*, from root to a specific node in SVO, defines the voxel without storing spatial information explicitly in the node

The sparse voxel DAG

- To Transform an SVO to a DAG we will use a bottom up approach.
- The leaf nodes are defined by uniquely by childmask of 8-bit, so there can be atmost $2^8 = 256$ unique leaf nodes
- Steps of Reduction:
 - ① Merge identical leaves with same bitmask
 - ② Proceed to level above and merge nodes with *Identical childmask* and *Identical pointers*
 - ③ Iteratively perform this step by merging larger and larger subtrees
- The smallest possible DAG is guranteed to be found in L iterations
- This is also applicable for non-uniform octrees and partially reduced DAG's
- This allows us to conveniently build DAGs from SVOs too large to reside in RAM or on disk.

Implementation Details - DAG Construction

- While building a dag of level $L > 10$ we first construct top $L-10$ levels by triangle/cube intersection in a top down fashion
- For each leaf we construct the subtree till maximum depth
- The subtree is reduced to DAG and then we proceed to next leaf
- After all sub-DAG's get have been constructed, we have top-level SVO with many DAG
- We apply reduction one last time to produce one final DAG
- The result is same as if we processed complete SVO but the memory consumption while construction is considerably low

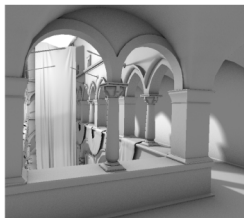
Implementation Details - Merging Nodes

- We sort the nodes of a level using a *256-bit* (8 child * 32-bit pointer) key and pointer as value
- Identical nodes become adjacent and can be compacted to a single nodes
- We also maintain a table of indirection, for each node we know the pointer to compacted node
- To improve performance we store subtrees of resolution 4^3 of last two rows into a 64-bit integer

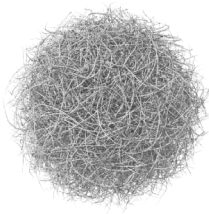
Ray Tracing a Sparse Voxel DAG

- The paper implements a GPU-based Raytracer in CUDA that efficiently traverses the scene representation
- The raytracer is primarily used for visibility queries to evaluate hard shadows, soft shadows, and ambient occlusion
- Ambient occlusion is usually approximated by casting a number of rays over the hemisphere and averaging the occlusion of each ray

Evaluation



(a) CRYSPONZA. *Standard game scene.*
(279k triangles)



(b) HAIRBALL. *Highly irregular scene.*
(2.8M triangles)



(c) LUCY. *Laser-scanned model.*
(28M triangles)



(d) SANMIGUEL. *Common GI-benchmark.*
(10.5M triangles)

Figure 3: Scenes used in the experiments.

Outline

- 1 Abstract
- 2 Introduction
- 3 Previous Work
 - Sparse Voxel Octrees
 - Common subtree merging
 - Shadows
 - Ambient Occlusion
- 4 Reducing a Sparse Voxel Tree to a DAG
- 5 Ray Tracing a Sparse Voxel DAG
- 6 Evaluation**
 - **Reduction Speed**
 - Memory Consumption
 - Traversal Speed
- 7 Conclusion
- 8 Future Work

Reduction Speed

scene	512 ³	1K ³	2K ³	4K ³	8K ³
Cryspenza	8.4ms	33ms	0.30s	1.0s	4.5s
	1.9ms	5.5ms	0.05s	0.14s	0.6s
EpicCitadel	1.5ms	5.0ms	0.05s	0.20s	0.80s
	0.7ms	1.2ms	0.006s	0.024s	0.10s
SanMiguel	2.6ms	8.3ms	0.06s	0.23s	0.95s
	1.3ms	1.7ms	0.008s	0.03s	0.14s
Hairball	43ms	202ms	2.19s	9.7s	40.6s
	5.1ms	29ms	0.23s	1.0s	4.5s
Lucy	2.6ms	8.4ms	0.08s	0.31s	1.3s
	0.7ms	1.6ms	0.009s	0.04s	0.14s

Figure 3: Time taken to reduce the SVOs to DAGs. The first row is the total time and the second row is the part of that spent on sorting.

- Crassin et al. [2012] SVO - CRYSPONZA at resolution 512³ - 7.34ms
- Laine and Karras [2010a] ESVO - HAIRBALL at resolution 1K³ - 628s.

Outline

- 1 Abstract
- 2 Introduction
- 3 Previous Work
 - Sparse Voxel Octrees
 - Common subtree merging
 - Shadows
 - Ambient Occlusion
- 4 Reducing a Sparse Voxel Tree to a DAG
- 5 Ray Tracing a Sparse Voxel DAG
- 6 Evaluation**
 - Reduction Speed
 - **Memory Consumption**
 - Traversal Speed
- 7 Conclusion
- 8 Future Work

Memory Consumption

		Total number of nodes in millions						Memory consumption in MB						bit/vox
scene		2K ³	4K ³	8K ³	16K ³	32K ³	64K ³	2K ³	4K ³	8K ³	16K ³	32K ³	64K ³	
Cryspenza	DAG	1	1	2	4	9	23	4	11	27	71	184	476	0.08
	ESVO	5	12	32	94	226	521	38	88	243	715	1 721	3 970	-
	SVO	12	51	205	822	3 290	13 169	12	48	195	784	3 138	12 559	2.07
EpicCitadel	DAG	1	1	1	3	7	18	3	7	19	53	142	371	0.65
	ESVO	1	3	7	17	38	81	7	20	53	124	287	613	-
	SVO	2	6	21	85	340	1 364	2	5	20	81	324	1 301	2.29
SanMiguel	DAG	1	1	2	5	13	34	3	10	31	93	270	742	0.45
	ESVO	4	14	57	229	922	3 698	26	107	433	1 747	7 030	28 212	-
	SVO	4	14	56	224	903	3 628	4	13	53	214	861	3 460	2.08
Hairball	DAG	5	15	44	115	-	-	117	339	996	2 629	-	-	2.24
	ESVO	53	224	924	3 649	-	-	401	1 709	7 049	27 837	-	-	-
	SVO	45	188	781	3 191	-	-	43	180	745	3 044	-	-	2.59
Lucy	DAG	1	1	2	6	16	46	3	10	32	110	348	983	1.54
	ESVO	2	8	26	76	160	255	15	56	198	576	1 219	1 941	-
	SVO	2	7	27	108	430	1 720	2	7	26	103	410	1 640	2.57

Figure 4: Comparison of the sparse voxel DAG, ESVO and SVO. Cases where the DAG has the smallest memory consumption are highlighted in green

Outline

- 1 Abstract
- 2 Introduction
- 3 Previous Work
 - Sparse Voxel Octrees
 - Common subtree merging
 - Shadows
 - Ambient Occlusion
- 4 Reducing a Sparse Voxel Tree to a DAG
- 5 Ray Tracing a Sparse Voxel DAG
- 6 Evaluation**
 - Reduction Speed
 - Memory Consumption
 - Traversal Speed**
- 7 Conclusion
- 8 Future Work

Traversal speed - Primary Rays

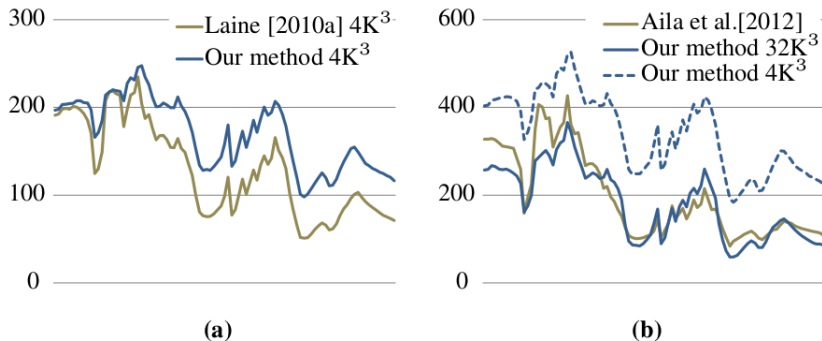


Figure 6: Traversal speeds in MRays/second for primary rays in a fly-through of SANMIGUEL. a) Our DAG and the ESVO on the GTX480. Both use the beam optimization. b) Our DAG vs. triangle ray tracing on the GTX680.

Traversal speed - Shadow Rays

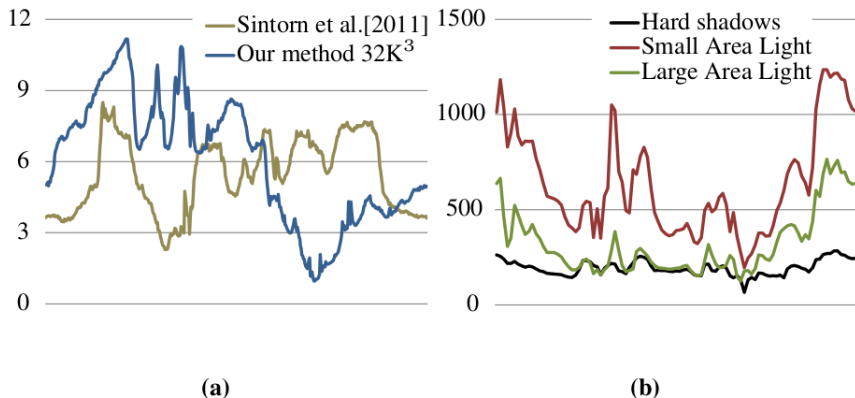
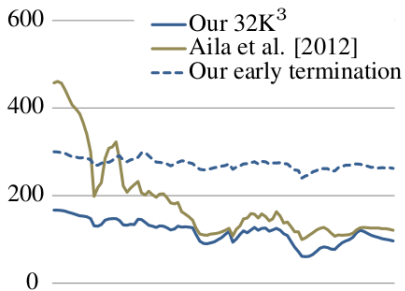


Figure 7: Comparison of traversal speed for shadow rays in a fly-through of the CITADEL scene. a) Comparing frame times (in ms) b) Performance in MRays/sec for our method with varying light source sizes.

Traversal speed - Ambient Occlusion



(a)



(b)

Figure 8: *a) Comparison of traversal speed for AO rays (MRays/sec), in a fly through of SANMIGUEL. b) Comparing image quality for 64 AO rays with and without early termination of rays. Rays are terminated at a cone-angle of 9 degrees (0.07sr)*

Conclusion

We have shown

- that sparse voxel DAGs, an evolution of sparse voxel octrees, allow for an efficient encoding of identical regions of space
- An algorithm to construct smallest possible DAG for the given Sparse Voxel Octree
- that this construction is possible without having to store the entire SVO in memory
- increase in node size is quickly amortized by reduction in node count.
- Despite smaller memory footprint, our Data structure can be efficiently ray traced.

Future Work

- The memory layout of the DAG used in this paper could potentially be improved
- We could add material representation residing in a separate tree or DAG which would be used on finding intersection of ray and the geometry.
- Future work could introduce cycles in the graph to fake unlimited resolution

Thanks
Questions?