High Resolution Sparse Voxel DAGs

Prateek Chandan Anurag Shirolkar

Department of Computer Science
IIT Bombay

CS 775

Outline

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 extremly 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 scondary scene representation is required with strict memory budget
- This acceleration structure has to be resident on RAM for effiecient 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 extremly high resolution enabling us to improve image quality, decrease discretization artifacts, explore high frequency effects like sharp shadows in large scenes.

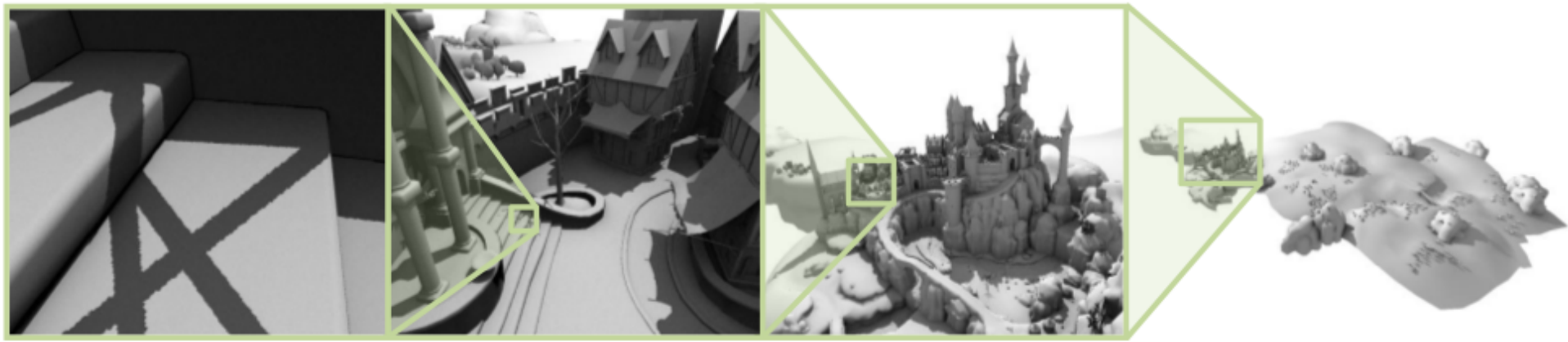


Figure 1: The EPIC CITADEL scene voxelized to a $128K^3$ resolution and stored as a Sparse Voxel DAG.

Introduction

- The paper presents and 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 ambitent occlusion by Ray tracing in DAG

Outline

Sparse Voxel Octrees

Outline

Common subtree merging

- work by Webber and Dillencourt [1989], where quadtrees are compressed by merging common subtree
- show 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

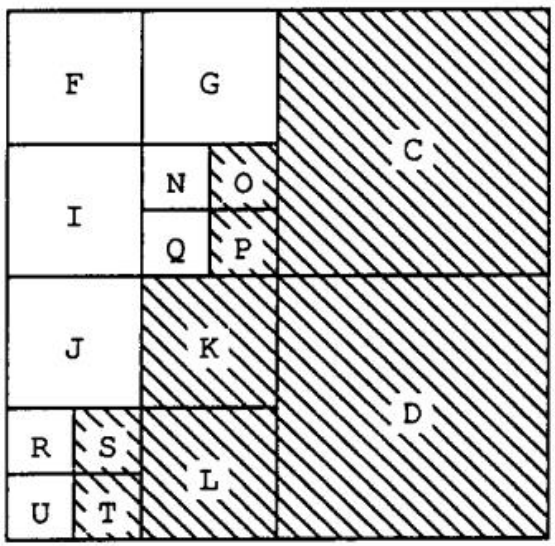


Figure 2. Normal quadtree for raster image.

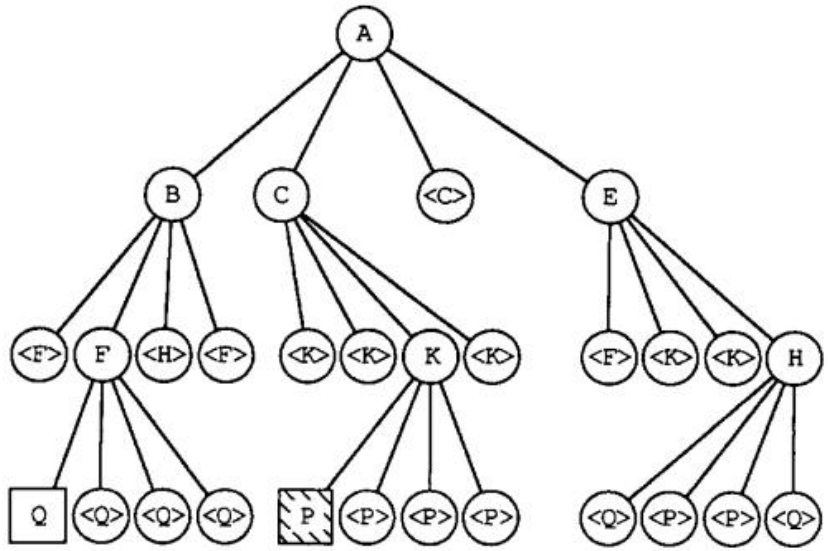


Figure 4. Pointer representation of CSM-quadtree for raster image.

Figure 2:

Outline

Shadows

- *image based methods* originating from the work by Williams [1978]
- *geometry based methods*, based on the idea of generating shadow volumes [Crow 1977]
- third approach - *view-sample mapping* [Aila and Laine 2004; Johnson et al. 2005]
- recent advancements

Outline

Ambient Occlusion

- screen space ambient occlusion
- precomputation of occlusion into texture [Kontkanen and Laine 2005]
- Ambient occlusion volumes [McGuire 2010]
- Fast Ray-Cast Ambient Occlusion [Laine and Karras 2010]

Reducing a Sparse Voxel Tree to a DAG

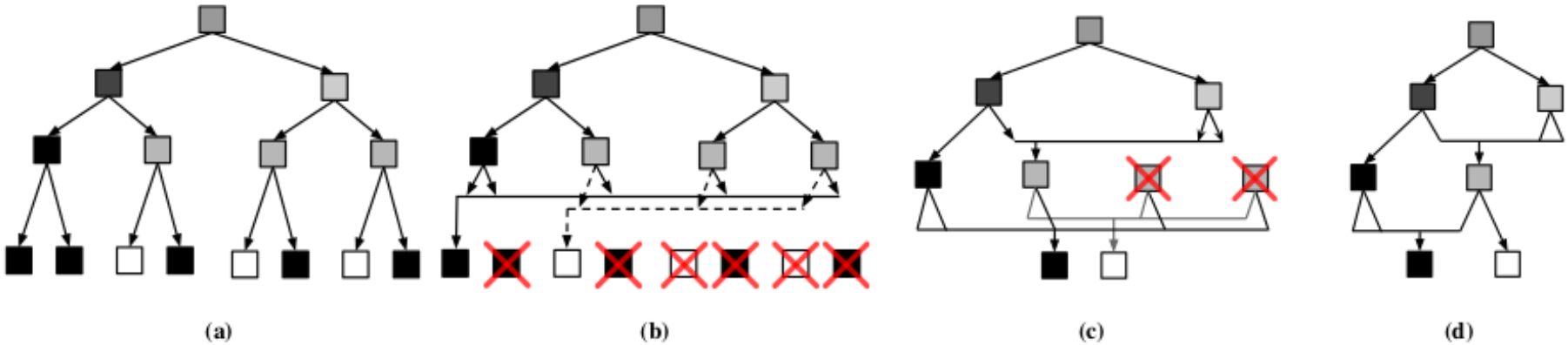


Figure 2:Reducing a sparse voxel tree, illustrated using a binary tree, instead of an octree, for clarity.

a. The original tree.

b. Non-unique leaves are reduced.

c. Now, there are non-unique nodes in the level above the leaves. These are reduced, creating non-unique nodes in the level above this.

d. 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 recursivley divided L time gives us a hierarchical representation of and 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 mege 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 mazimum 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

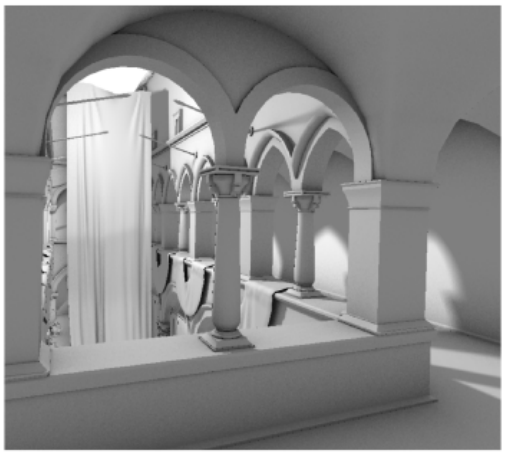
Implementation Details - Merging Nodes

- 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 inderection, for each node we know the pointer to compacted node
- To imporve performance we store subtrees of resolution 4^3 of last two rows into a 64-bit integer

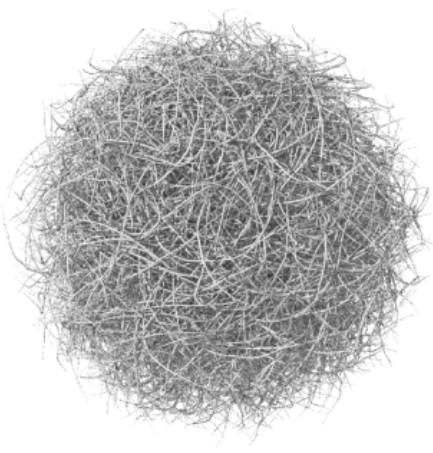
Ray Tracing a Sparse Voxel DAG

- 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

Reduction Speed

scene	512^3	$1K^3$	$2K^3$	$4K^3$	$8K^3$
Crysponza	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 512^3 - 7.34ms
- Laine and Karras [2010a] ESVO - HAIRBALL at resolution $1K^3$ -628s.

Outline

Memory Consumption

		Total number of nodes in millions						Memory consumption in MB						bit/vox
scene		$2K^3$	$4K^3$	$8K^3$	$16K^3$	$32K^3$	$64K^3$	$2K^3$	$4K^3$	$8K^3$	$16K^3$	$32K^3$	$64K^3$	
Crysponza	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

Traversal speed

Conclusion

Block Title

You can also highlight sections of your presentation in a block, with it's own title

Theorem

There are separate environments for theorems, examples, definitions and proofs.

Example

Here is an example of an example block.

Future Work

- The first main message of your talk in one or two lines.
- The second main message of your talk in one or two lines.
- Perhaps a third message, but not more than that.

- Outlook
  - Something you haven't solved.
  - Something else you haven't solved.

Thanks

Questions?