The Splitted Grid - An acceleration structure for ray tracing

Bachelor thesis

Marc Aurel Kastner

supervised by: Pablo Bauszat

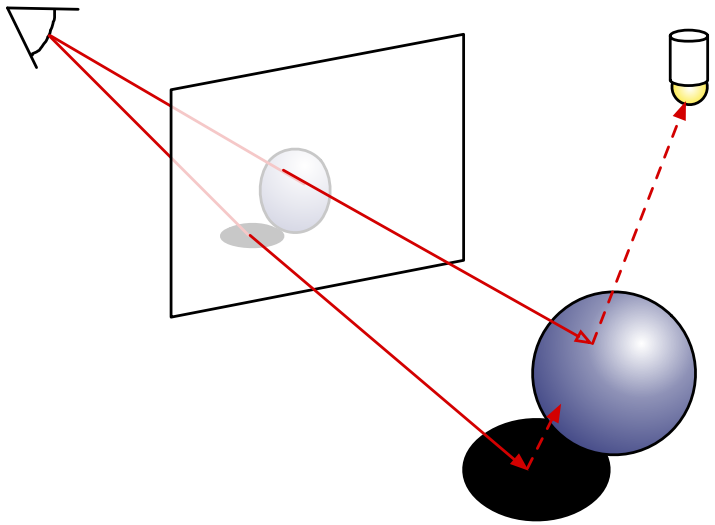
Computer Graphics Lab
TU Braunschweig

2013-04-08

Outline

Ray tracing

- Calculating image based on tracing $n * m$ rays through a pinhole camera
- Ray intersection with geometry in scene, determining the color at hit point
- More complex lighting and shadow models:
  - Whitted Style: simple way, lacks more complex light phenomena, by Whitted [Whi80]
  - Pathtracing: has global illumination, by Kajiya [Kaj86]



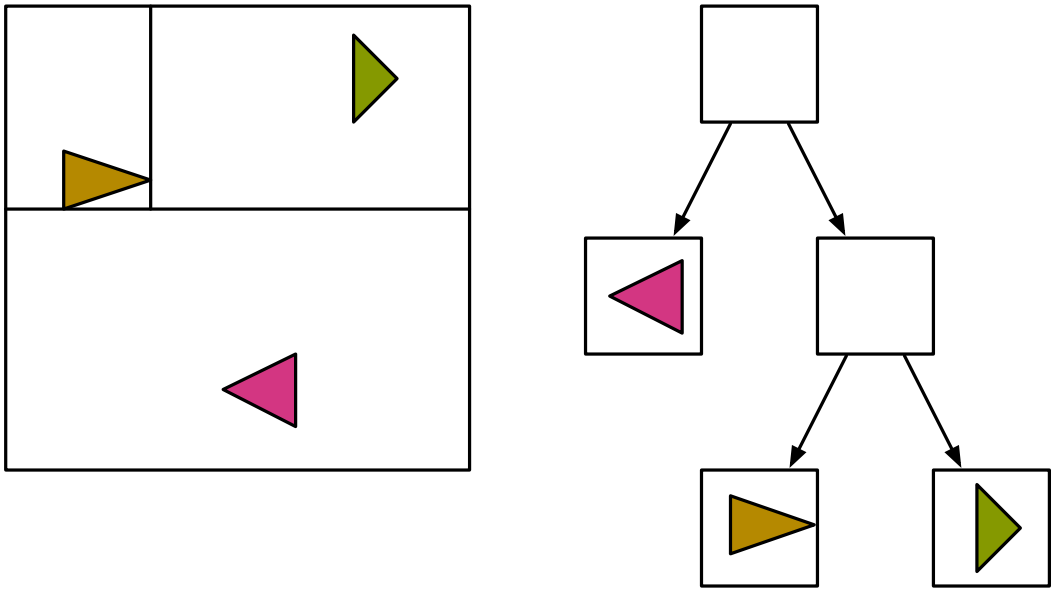
Acceleration

Different concepts on speeding up ray tracing:

- Using Whitted Style ray tracing
- Hardware:
  - newer hardware, more power
  - GPUs instead of CPUs
  - multi-threaded computing
- Reducing the number of ray-geometry intersection tests

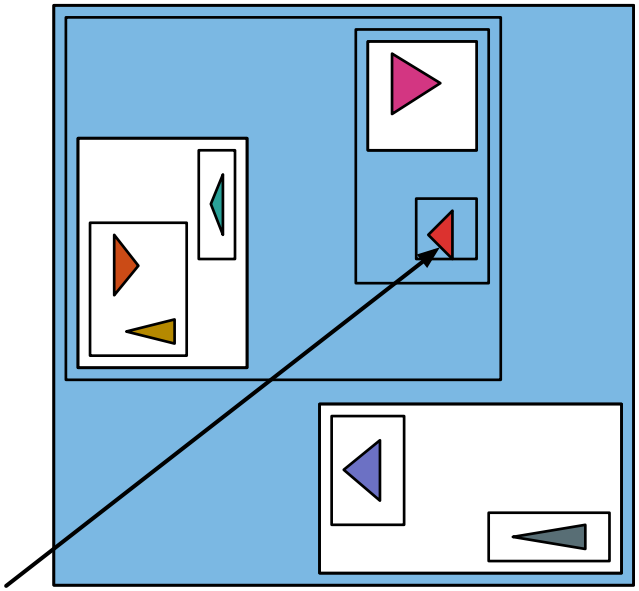
Acceleration structures

- Idea
  - pre-processing the scene
  - generating a data structure out of objects in scene
  - when rendering, traverse the data structure
  - **Result**: Reduction of ray-intersection tests
- Common approaches: BVH, kD-Tree, Grid



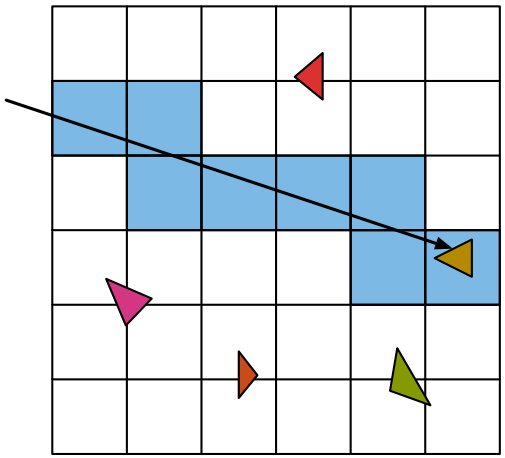
BVH and kD-Tree

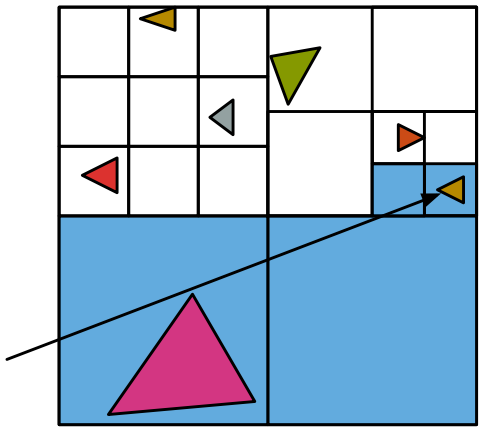
- kD-Tree
  - by Bentley [Ben75]
  - a binary tree, splitting one axis into two childs on every iteration
- Bounding Volume Hierarchy (BVH)
  - by Rubin et al. [RW80] and Kay et al. [KK86]
  - puts bounding volumes of objects in a tree
- Both BVH and kD-Tree are often used in raytracers



Grid

- Uniform Grid
  - by Fujimoto et al. [FTI86]
  - divides 3d space in $n * m * p$ voxels of same size
  - **Advantages**: fast construction and traversal speeds
  - **Disadvantages**: problems with teapot-in-a-stadium scenarios, high memory usage
- Nested Grids
  - using Uniform Grids hierarchically
  - still problems with uneven sized primitives and duplicated





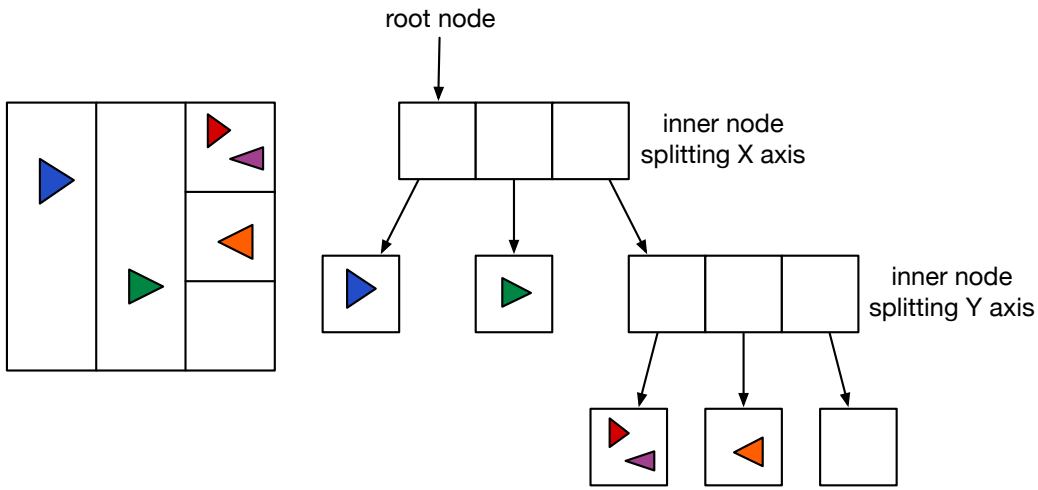
Outline

Why Splitted Grid?

- Grids have fast construction and traversal speed, but very high memory usages and severe problems in some scenes
- Hierarchical Grid approaches do not solve them sufficiently
- Trying out another approach of a hierarchical grid variant

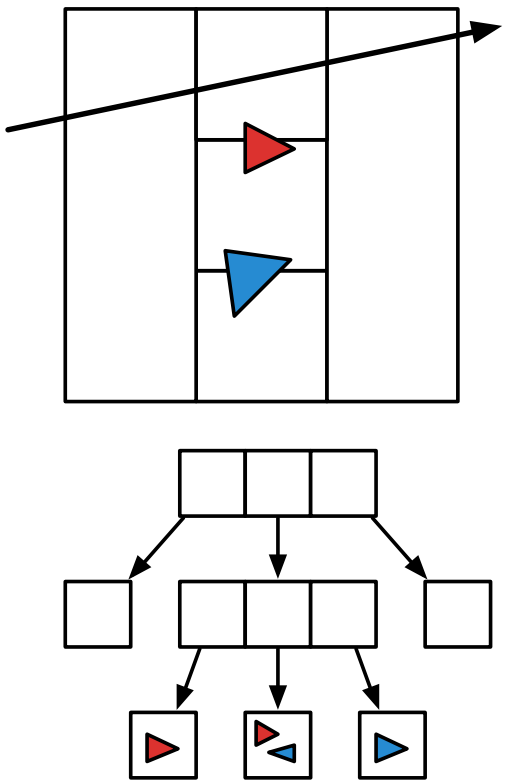
The Splitted Grid

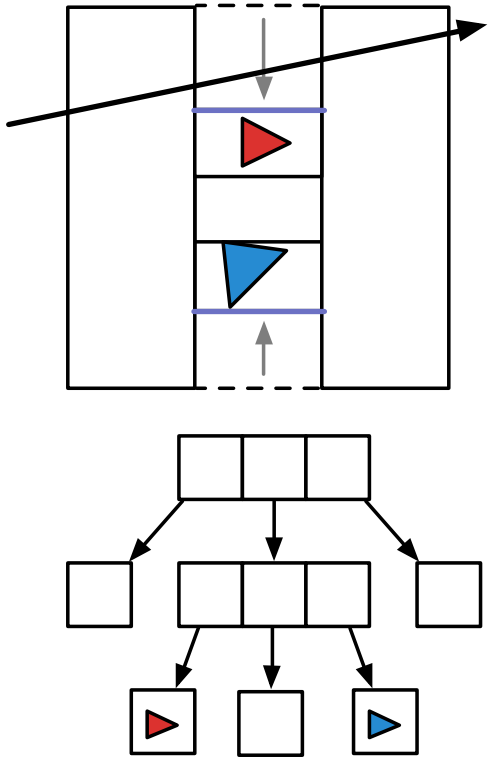
- A Splitted Grid *inner node*
  - a 1-dimensional Grid
  - splitting the space in *n* boxes of same volume *on one axis*
  - can be nested recursively
  - each one can have different resolution and different splitting axis
- When no further room splitting is needed, a *leaf node* is created



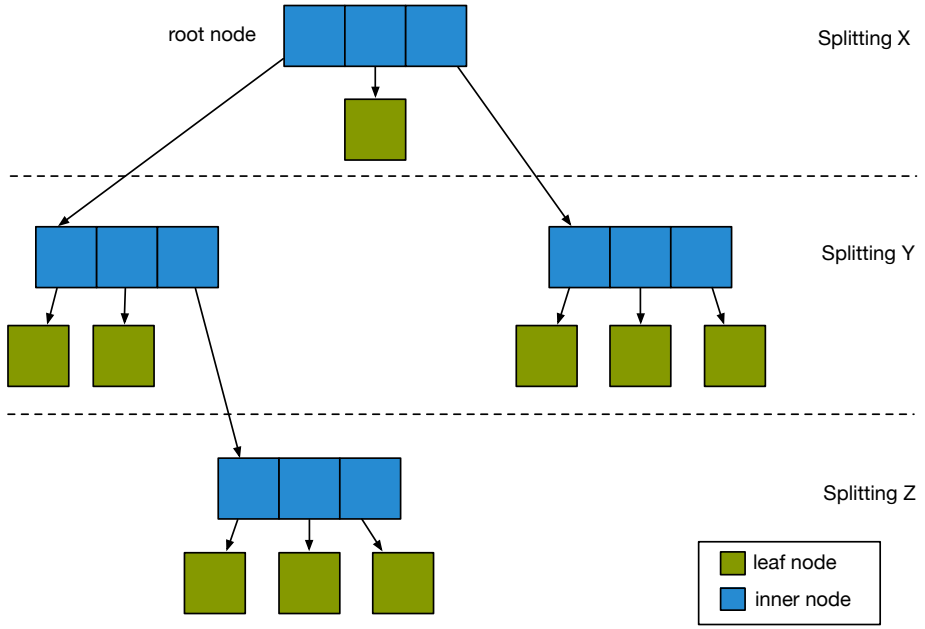
Adding Bounding planes

- Decreasing empty space
  - by limiting the bounding box of a node to the area with primitives inside
- Two planes are saved for each inner node
- Leaf nodes do not use bounding planes



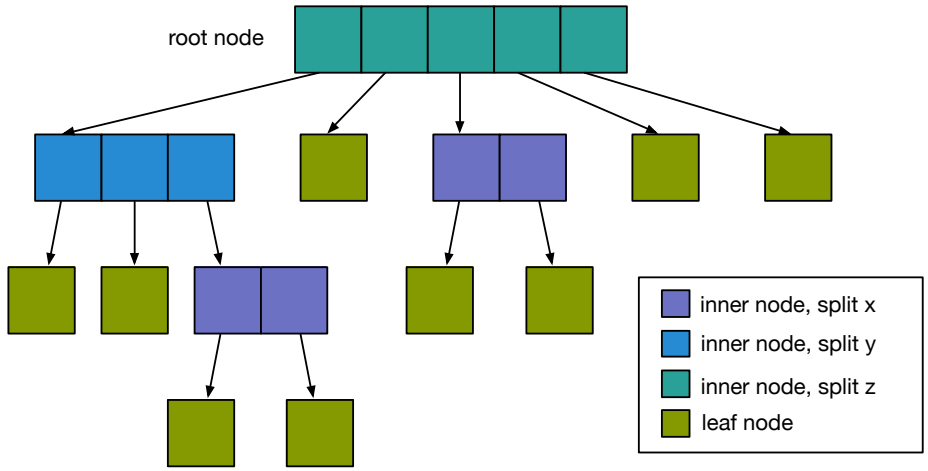


Uniform construction algorithm



- Uniform variant
  - simplest approach
  - same resolution on every inner node
  - chooses splitting axes in a round robin manner
- **Advantage**: construction time is usually fast
- **Disadvantage**: the tree does not fit the scene very well, often leading to high memory usage

SAH construction algorithm



- Choosing parameters with a surface area heuristic
- With Cost estimate function:

$$Cost(c) = r * T + I * \frac{1}{A} * (\sum_{i=1}^{r} A_i * p_i)$$

$$min(Cost(c)), \forall c \in C$$

C: a set of all resolution and splitting axis combinations, r: resolution, A: area of the parent bounding box, A_i: area of childrens bounding box, p_i: the amount of primitives in the childrens node, T and I: cost estimates for traversal and intersection

Memory Layout

- **Out of the nodes**:
  - Array with nodes
  - Array with primitive indices
  - Array with bounding planes
- **Inner Node** data:
  - Leaf flag and splitting axis
  - Offset to children nodes in nodes array
  - Resolution
  - Offset to bounding planes in bounding planes array
- **Leaf Node** data:
  - Leaf flag
  - Amount of primitives
  - Offset to primitive in primitives array
- For both node types, two integers are enough to save this data for reasonably sized scenes

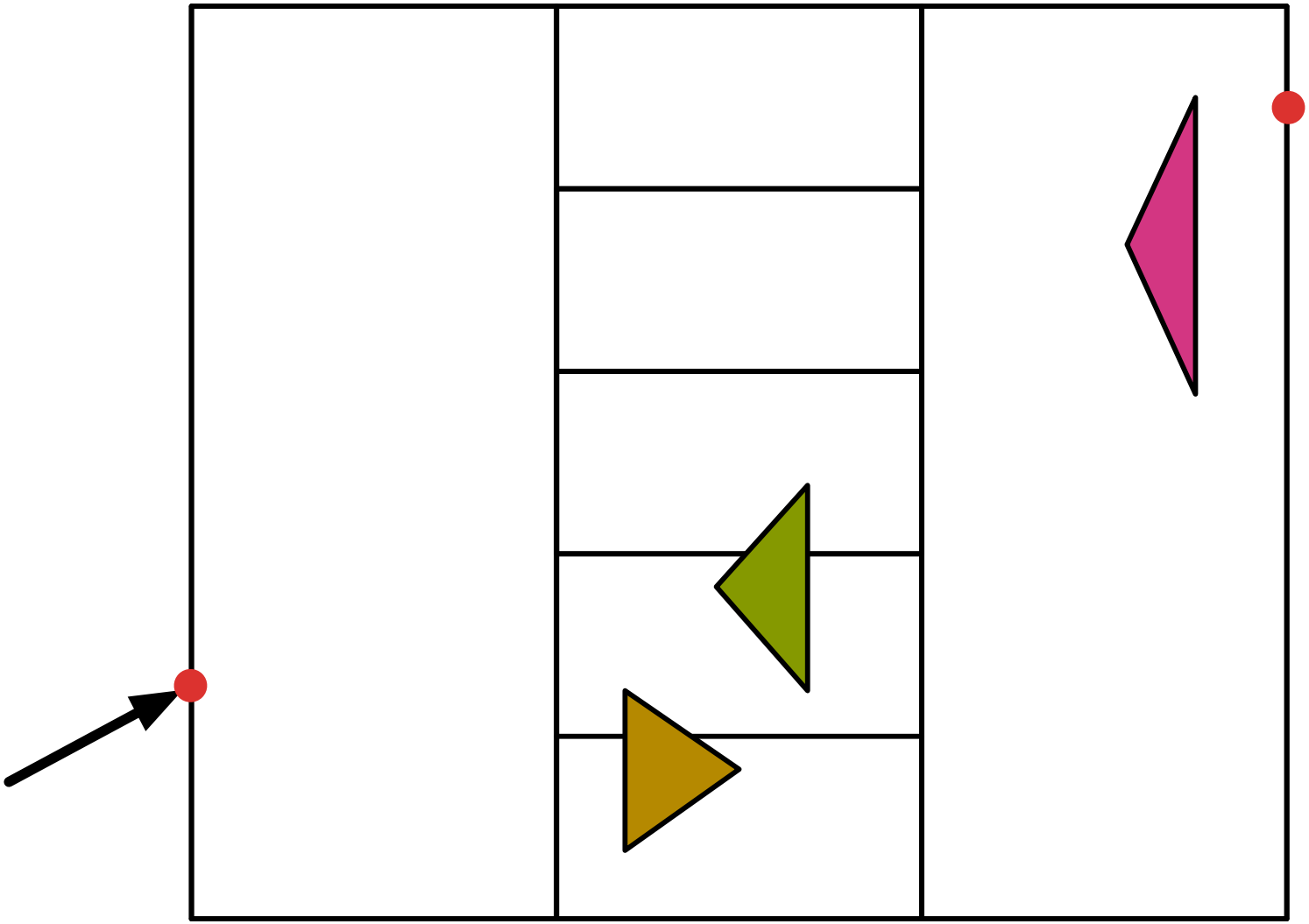
Traversal algorithm I

- Going through tree recursively
- Similar to standard traversal algorithms for hierarchical structures like kD/BVH
- Root node needs a full bounding box-ray intersection tests, then bounding planes clipping is used
- Having the $t_{Near/Far}$ of the parent node, an entry and exit node can be calculated for traversing only child nodes the ray will hit

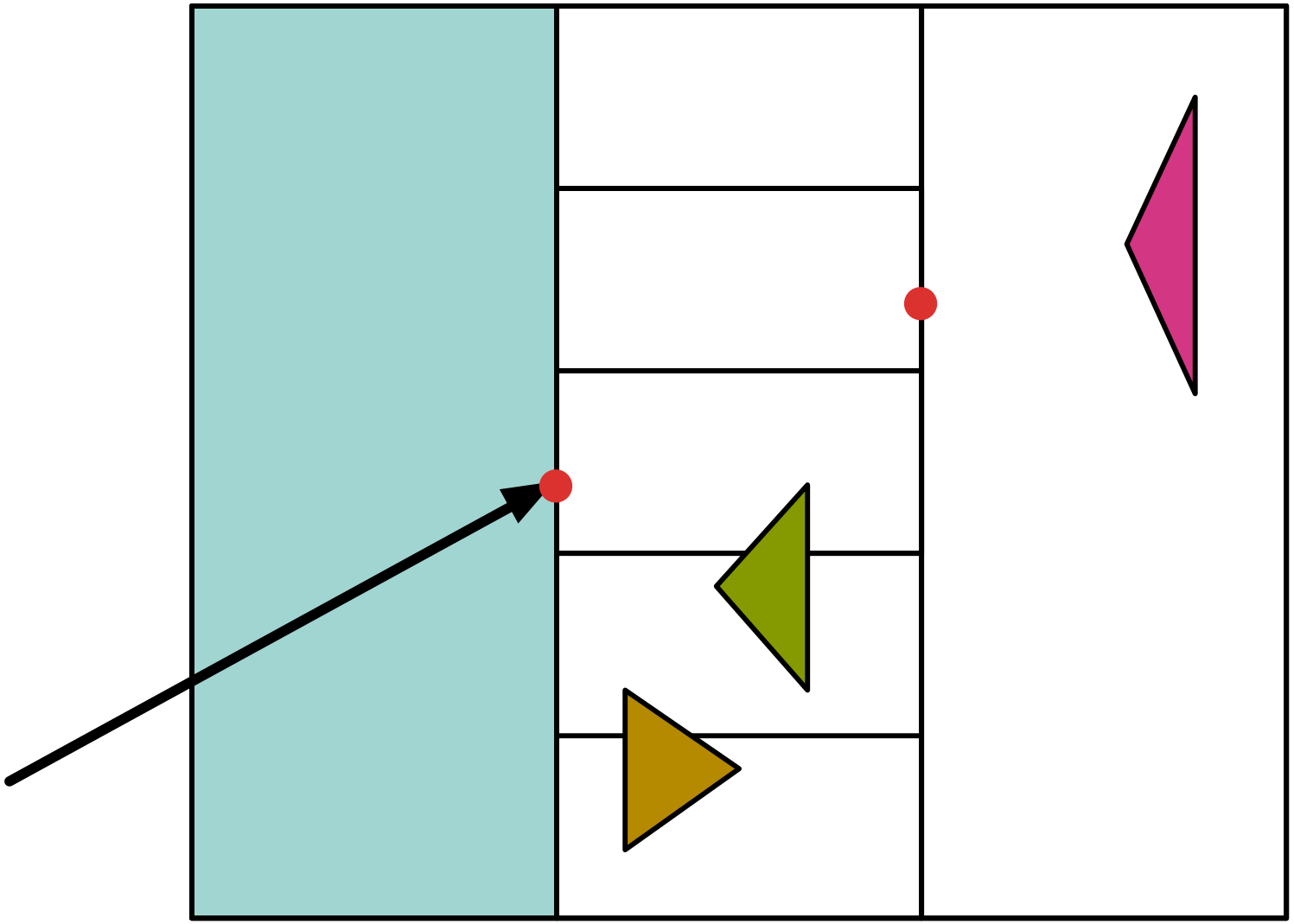
$$node_{entry/exit} = \frac{(o_{Ray} + d_{Ray} * t_{Near/Far}) - plane}{size_{child}}$$

- An early out is possible if $t_{Hit} < t_{Near}$

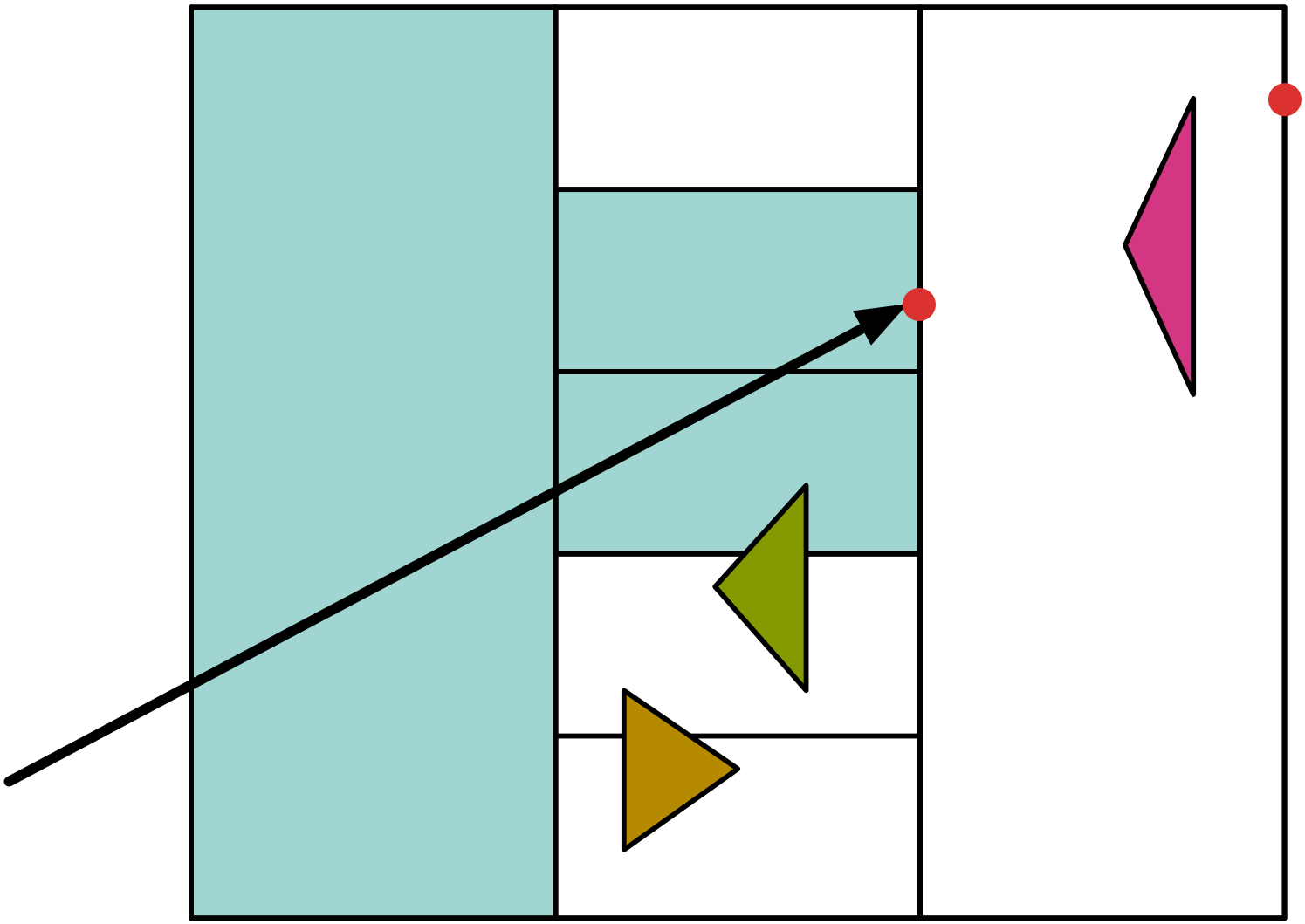
Traversal algorithm II



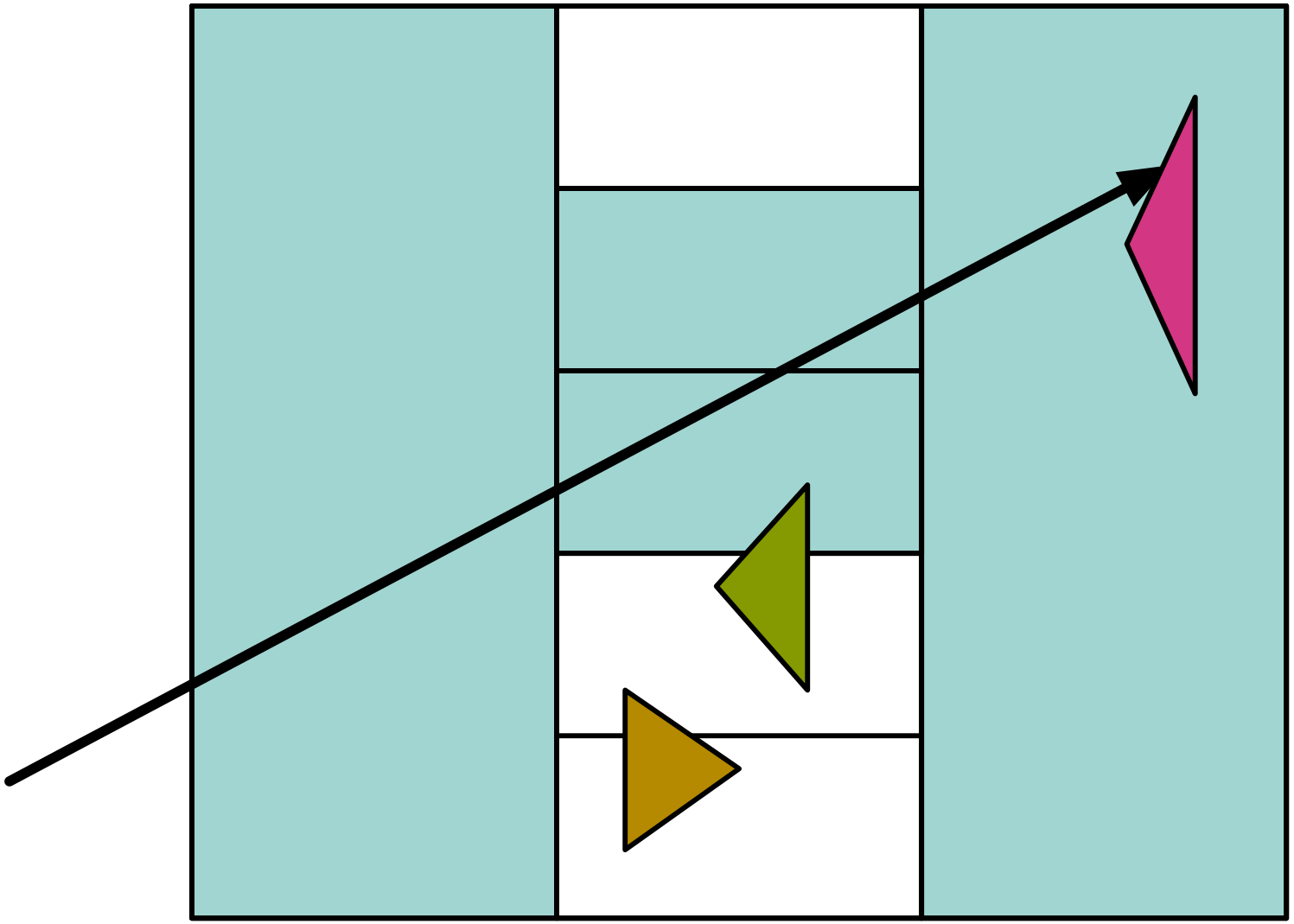
Traversal algorithm II



Traversal algorithm II

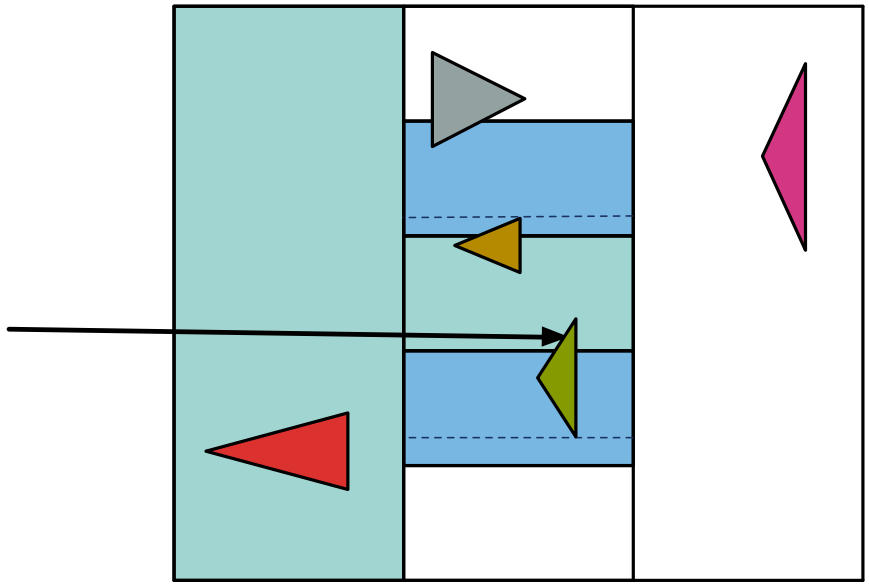


Traversal algorithm II



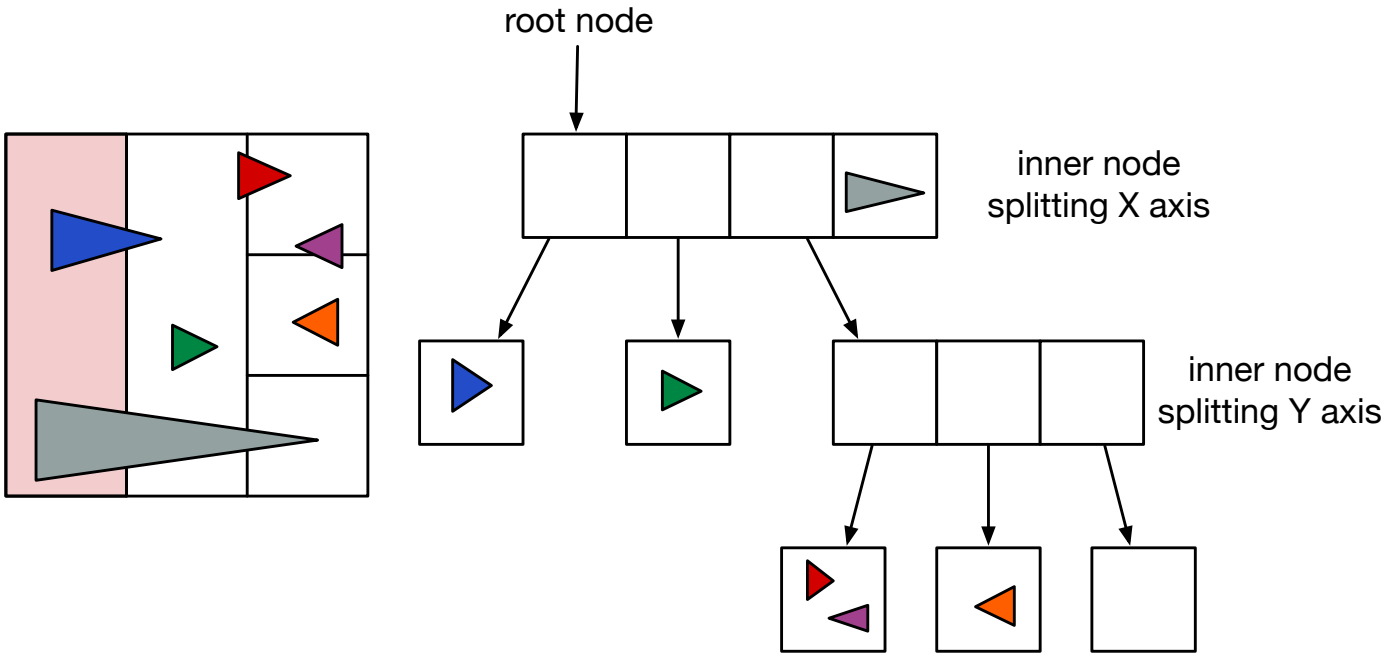
Loose Splitted Grid I

- Based on Splitted Grid
  - but only saves primitives in nodes, where the primitives mid point lies
  - **Result**: no duplicated references and therefore lower memory usage
  - for correct traversing, the overlapping primitives span *loose bounding planes*
  - two additional floats per inner node for these planes



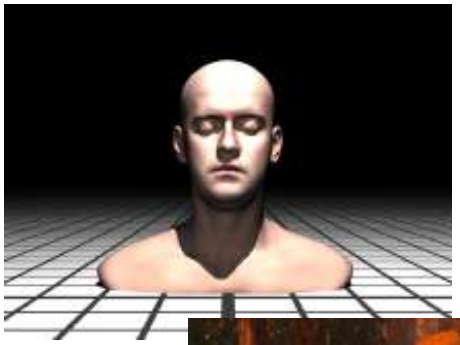
Loose Splitted Grid II

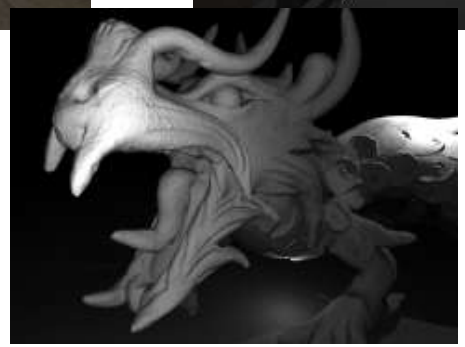
- To avoid big primitives enlarging the loose bounding planes too much, big primitives are saved inside an inner node



Outline

Tested scenes



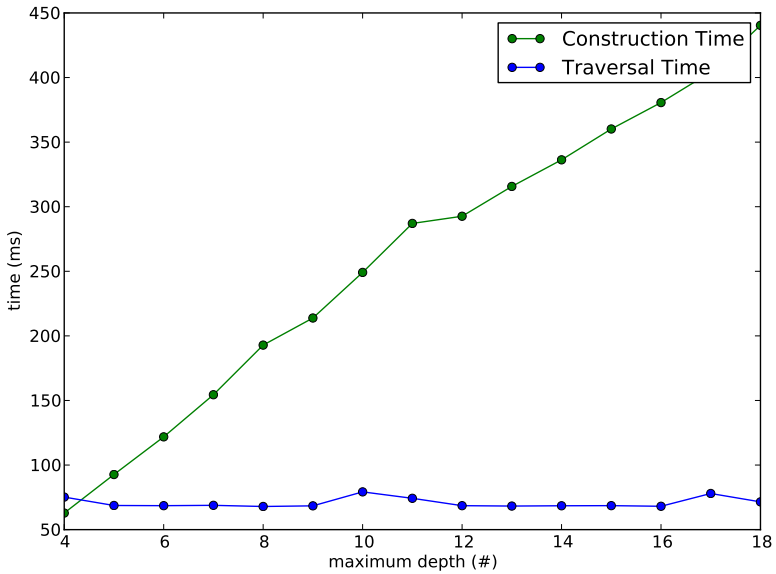


Name	Category	Primitives	Light sources
Head	low resoluted object	17,684	2
Sponza	architectural	279,163	2
Sibenik	architectural	76,521	2
Fairy	outdoor	172,669	2
Dragon	high resoluted object	7.9 mio	3

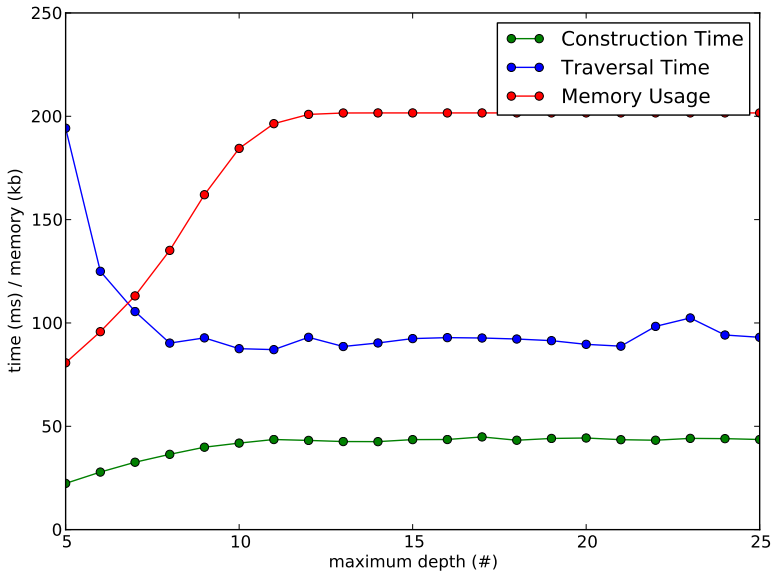
Implementations are in C++ in Pablo's ray tracing framework JUNO. Analysis done on an Intel Core i7-3720QM with 2.60 GHz per core and 16 GB DDR3-RAM.

Evaluating parameters

Different depths in SG



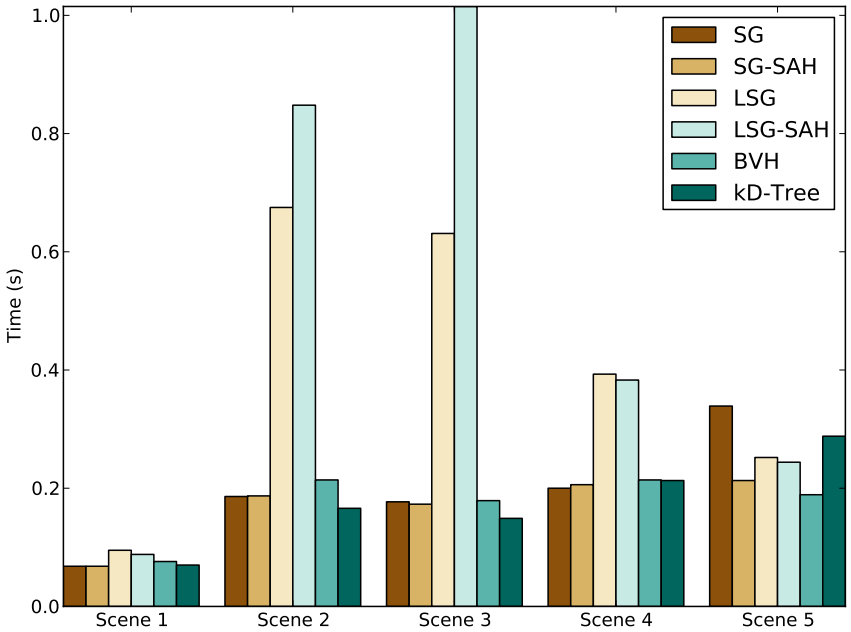
Different depths in LSG



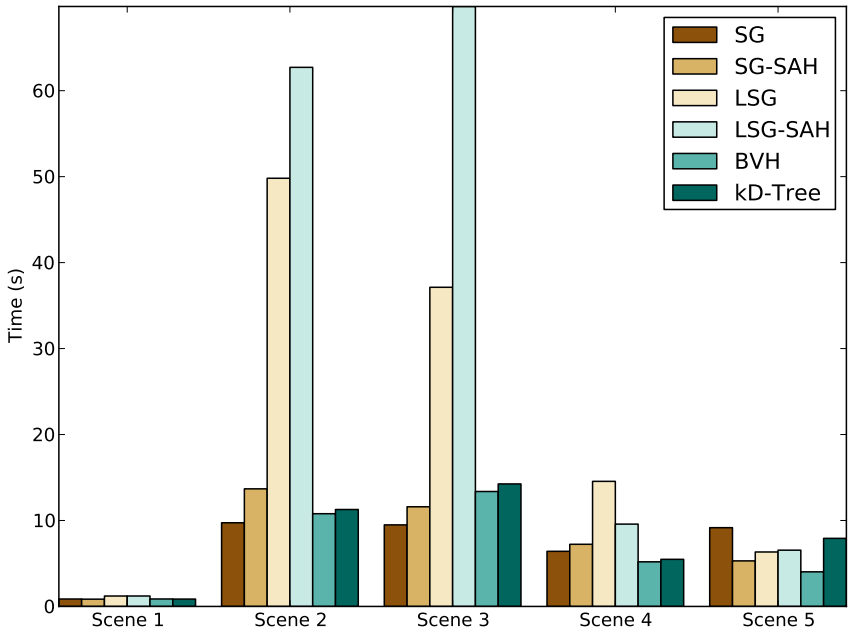
- **Splitted Grid**:
  - High depth results in a very big tree and lots of duplicated references
  - Lower depth and higher resolution works better
- **Loose Splitted Grid**:
  - Due to no duplicated reference, much higher depths can be chosen

Comparison with other approaches I

Raycasting time



Pathtracing time

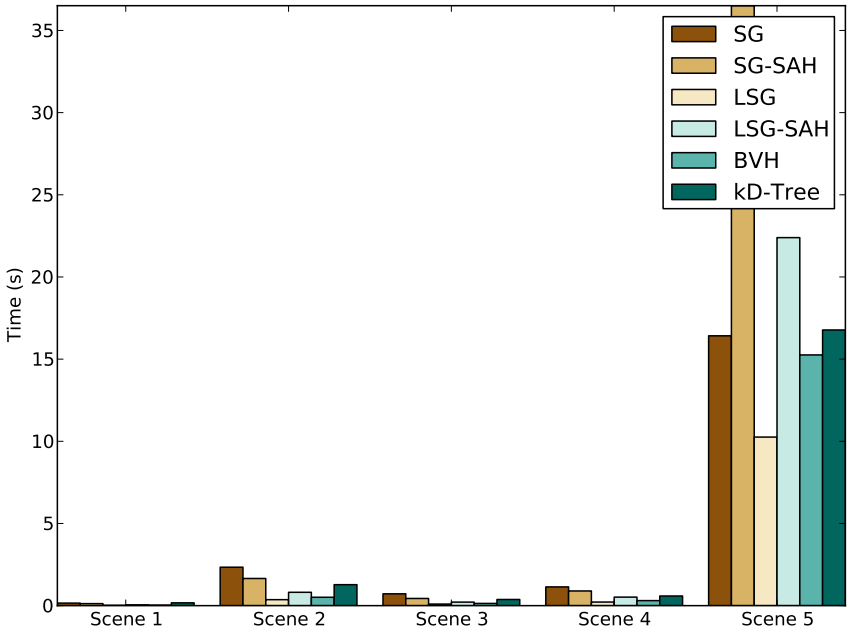


- **Raycasting**:
  - SG most of the times between BVH/kD
  - LSG up to 5 times slower than BVH/kD in TIAS scenes, 1.5 times in other scenes
- **Pathtracing**: similar results

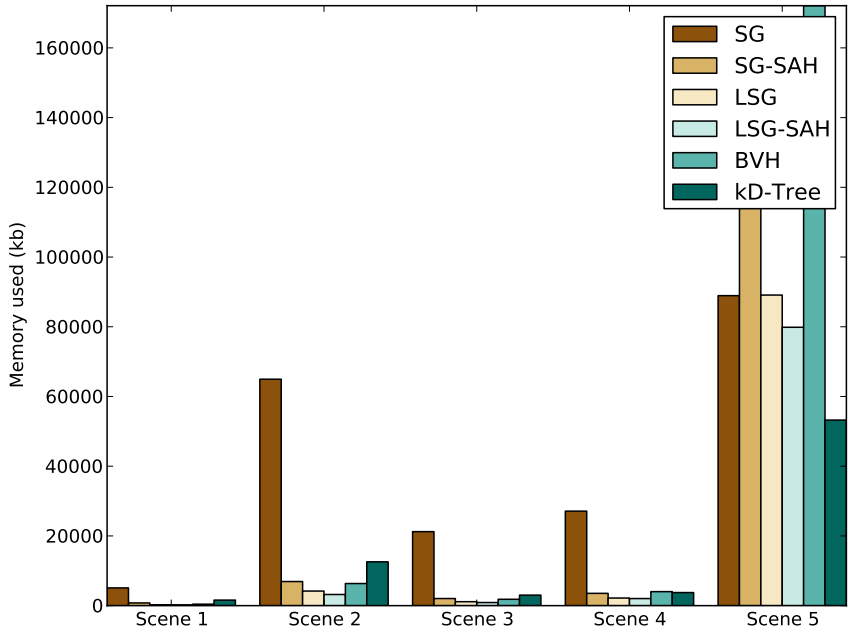
Based on 1024x768, 1 sample per pixel (raycasting), 4 sample per pixel (pathtracing), 4 bounces

Comparison with other approaches II

Construction time



Memory usage



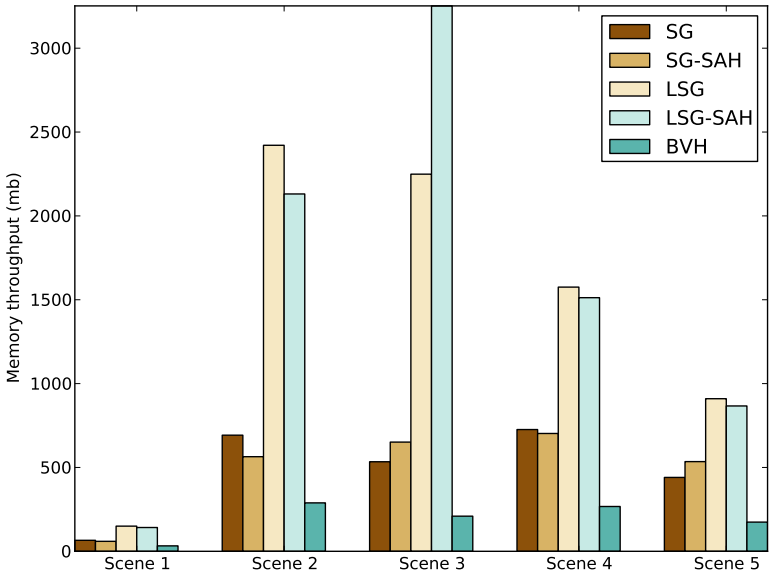
- **Construction**:
  - SG commonly 2 to 3 times slower than kD
  - Uniform LSG up to 50% faster than BVH/kD, but SAH is 2 times slower
- **Memory**:
  - Uniform SG up to 9 times higher memory usage than BVH due to bad fitting to scene, SAH on same level as BVH
  - LSG memory usage the half of BVH in average

Analysis of previous results

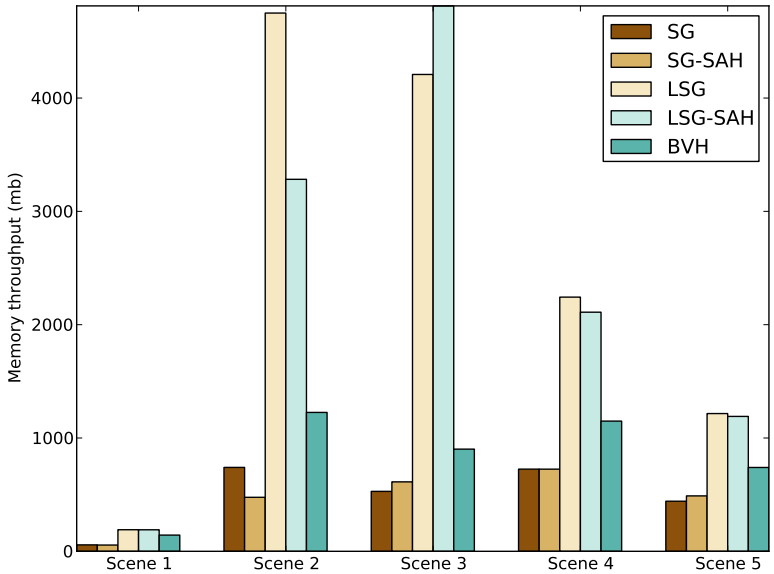
- **Splitted Grid Uniform**:
  - On par with competitors, but high memory usage
  - Does not fit very well to scene, due to the default resolution in higher depths
  - A lot of duplicated references
- **Splitted Grid SAH**:
  - Better memory usage with similar traversal times
  - Very high construction time in scene 5, due to more complexity in SAH construction
- **Loose Splitted Grid**:
  - Both construction variants traverse up to 5 times slower than competitors in scenes with teapot-in-a-stadium problem
  - Even small loose bounding planes lead to traversing whole parts of sub-trees

GPU analysis

Stack memory usage



Node memory usage



- **Splitted Grid**:
  - Stack memory throughput is 2 to 3 times higher than BVH
  - Node memory throughput 33% lower compared to BVH
- **Loose Splitted Grid**:
  - Averagely 2 times the stack memory throughput than SG
  - Up to 4 times the node memory throughput
- There are ways to improve stack memory throughput of current traversal implementation

Outline

Conclusion

- **Splitted Grid**:
  - On par with competitors in a handful of tests in terms of traversing speed and memory usage
  - SAH has high complexity in construction, leads to slower construction
  - Node memory throughput seems promising for GPU usage
- Unfortunately, **Loose Splitted Grid** slow traversal speeds compared to competitors, but good memory usage and construction times

Future Work

- Improving Loose Splitted Grid
- Trying out some additional ideas to improve Splitted Grid traversal algorithm
- GPU Implementation of at least the traversal algorithm
- An implicit variant based on Loose Splitted Grid, similar to NMH [EBM12]
- There are plans to prepare a paper about Splitted Grid and future results later this year as part of my Hiwi job at ICG

Thanks!

Thank you for your attention!

References I

Jon Louis Bentley, *Multidimensional binary search trees used for associative searching*, Commun. ACM **18** (1975), no. 9, 509–517.

Martin Eisemann, Pablo Bauszat, and Marcus Magnor, *Implicit object space partitioning: The no-memory BVH*, Tech. Report 16, Computer Graphics Lab, TU Braunschweig, January 2012.

A. Fujimoto, T. Tanaka, and K. Iwata, *Arts: Accelerated ray-tracing system*, Computer Graphics and Applications, IEEE **6** (1986), no. 4, 16–26.

James T. Kajiya, *The rendering equation*, Proceedings of the 13th annual conference on Computer graphics and interactive techniques (New York, NY, USA), SIGGRAPH '86, ACM, 1986, pp. 143–150.

References II

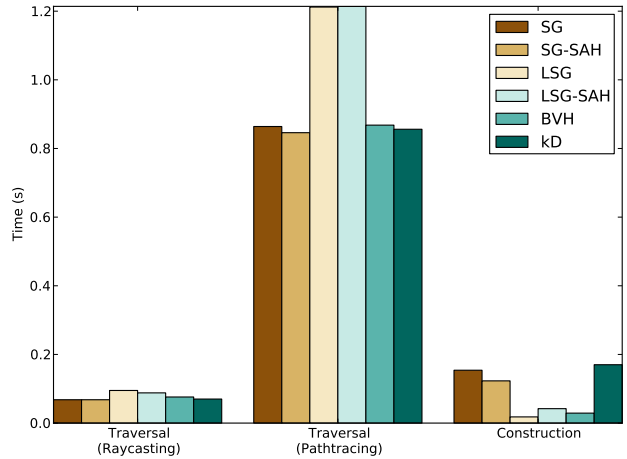
Timothy L. Kay and James T. Kajiya, *Ray tracing complex scenes*, SIGGRAPH Comput. Graph. **20** (1986), no. 4, 269–278.

Steven M. Rubin and Turner Whitted, *A 3-dimensional representation for fast rendering of complex scenes*, SIGGRAPH Comput. Graph. **14** (1980), no. 3, 110–116.

Turner Whitted, *An improved illumination model for shaded display*, Commun. ACM **23** (1980), no. 6, 343–349.

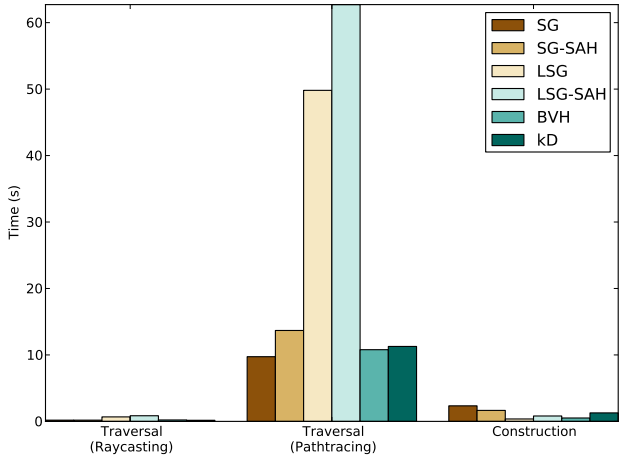
Appendix

Appendix I



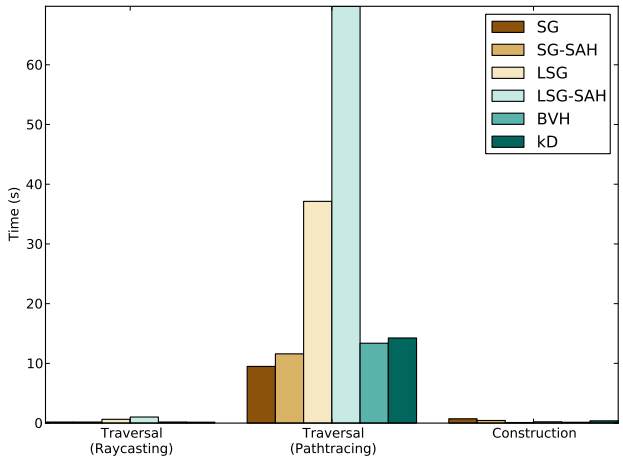
Head	SG	SG-SAH	LSG	LSG-SAH	BVH	KD
trav1	0.068s	0.068s	0.095s	0.088s	0.076s	0.070s
trav2	0.864s	0.846s	1.212s	1.214s	0.868s	0.856s
constr	0.154s	0.123s	0.018s	0.042s	0.029s	0.170s
mem	5086kb	779kb	221kb	201kb	421kb	1592kb
param	r = 8 d = 10	r = 16 d = 6	r = 4 d = 25	r = 4 d = 25	d = 25	d = 25

Appendix II



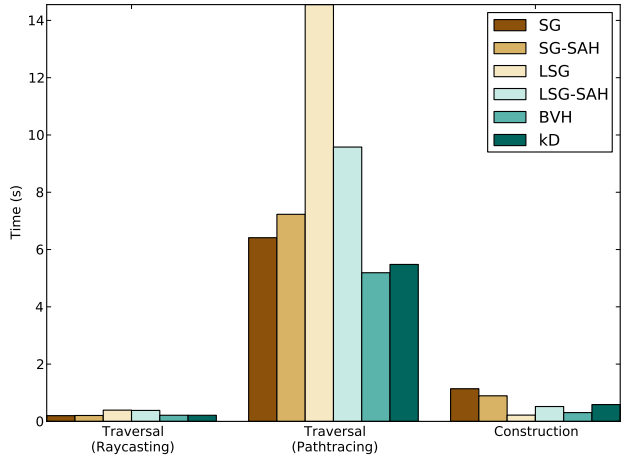
Sponza	SG	SG-SAH	LSG	LSG-SAH	BVH	KD
trav1	0.186s	0.187s	0.675s	0.848s	0.214s	0.166s
trav2	9.734s	13.681s	49.812s	62.715s	10.788s	11.277s
constr	2.339s	1.653s	0.364s	0.812s	0.510s	1.276s
mem	64955kb	6916kb	4183kb	3198kb	6341kb	12568kb
param	r = 8 d = 10	r = 16 d = 6	r = 2 d = 25	r = 4 d = 25	d = 25	d = 25

Appendix III



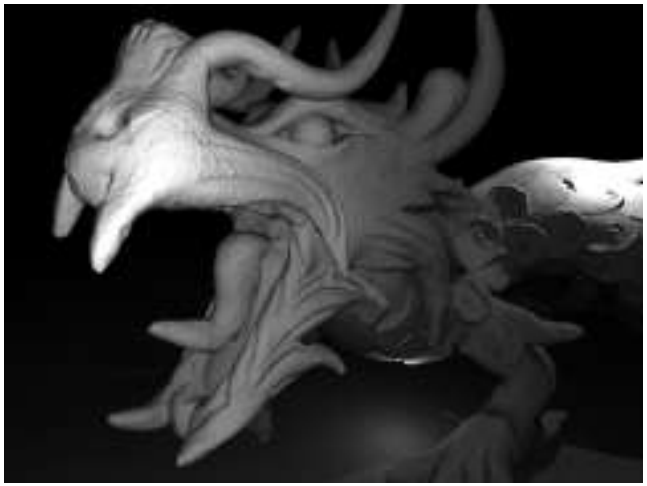
Sibenik	SG	SG-SAH	LSG	LSG-SAH	BVH	KD
trav1	0.177s	0.173s	0.631s	1.015s	0.179s	0.149s
trav2	9.488s	11.594s	37.129s	69.796s	13.372s	14.252s
constr	0.719s	0.437s	0.096s	0.215s	0.135s	0.375s
mem	21229kb	2041kb	1133kb	897kb	1827kb	3018kb
param	r = 8 d = 10	r = 16 d = 6	r = 2 d = 25	r = 4 d = 25	d = 25	d = 25

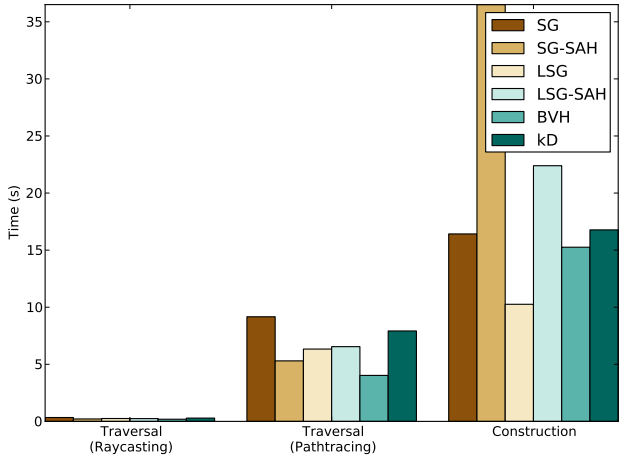
Appendix IV



Fairy	SG	SG-SAH	LSG	LSG-SAH	BVH	KD
trav1	0.200s	0.206s	0.393s	0.383s	0.214s	0.213s
trav2	6.413s	7.230s	14.552s	9.578s	5.190s	5.480s
constr	1.140s	0.893s	0.219s	0.519s	0.307s	0.587s
mem	27107kb	3525kb	2183kb	2031kb	4020kb	3752kb
param	r = 8 d = 10	r = 16 d = 6	r = 4 d = 25	r = 4 d = 25	d = 25	d = 25

Appendix V





Dragon	SG	SG-SAH	LSG	LSG-SAH	BVH	KD
trav1	0.339s	0.213s	0.252s	0.244s	0.189s	0.288s
trav2	9.163s	5.300s	6.335s	6.546s	4.027s	7.920s
constr	16.419s	36.507s	10.257s	22.394s	15.256s	16.774s
mem	88925kb	118660kb	89093kb	79855kb	172102kb	53233kb
param	r = 8 d = 7	r = 16 d = 6	r = 4 d = 25	r = 4 d = 25	d = 25	d = 25

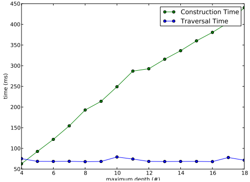
Appendix VI

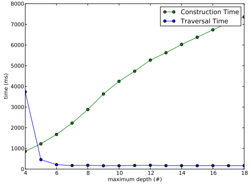
	SG	SG-SAH	LSG	LSG-SAH
Head				
sa	2,682	2,582	2,309	2,115
pit	1,256,137	1,554,753	3,674,168	4,083,506
tin	2,450,626	2,396,075	4,850,796	4,903,404
tln	85,934	52,751	701,910	374,400
Sponza				
sa	2,502,832,128	1,884,039,552	3,608,101,120	2,301,579,008
pit	7,148,465	16,816,158	55,767,103	137,306,763
tin	30,844,931	18,765,122	121,796,021	80,391,290
tln	4,511,333	6,165,578	13,622,717	28,324,701
Sibenik				
sa	77,896	58,381	108,004	93,804
pit	7,408,410	11,329,798	56,683,329	151,301,282
tin	21,771,987	25,467,874	106,762,892	116,277,187
tln	4,009,849	3,931,449	17,754,913	49,498,327
Fairy Forest				
sa	5,731	3,436	3,960	4,253
pit	11,334,281	17,943,501	384,159,544	38,377,664
tin	28,164,253	30,136,692	55,789,704	52,270,831
tln	10,566,321	4,612,806	15,070,382	15,234,843
Dragon				
sa	3,910,959	4,867,261	4,805,505	4,596,750
pit	59,099,747	19,836,105	10,710,582	10,512,202
tin	18,462,513	20,300,633	30,761,915	30,392,431
tln	2,550,212	3,198,146	5,524,866	4,071,065

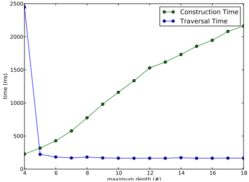
Appendix VII

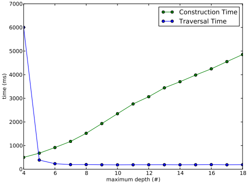
	SG	SG-SAH	LSG	LSG-SAH	BVH
Head					
push	3,716,016	3,397,572	7,886,493	7,561,195	4,249,241
pop	2,014,812	1,799,482	5,228,711	4,835,030	4,249,241
stack mem	65.58mb	59.48mb	150.09mb	141.86mb	32.42mb
node mem	56.75mb	55.24mb	190.40mb	189.91mb	143.01mb
Sponza					
push	40,116,784	30,600,991	141,342,441	112,668,664	37,851,327
pop	20,391,289	18,697,816	70,213,106	73,521,058	37,851,327
stack mem	692.46mb	564.18mb	2421.06mb	2130.77mb	288.78mb
node mem	740.40mb	476.54mb	4750.08mb	3282.79mb	1225.62mb
Sibenik					
push	30,364,763	34,311,111	131,615,155	168,300,563	27,459,098
pop	16,295,460	22,591,815	64,894,307	115,881,741	27,459,098
stack mem	533.98mb	651.20mb	2248.87mb	3252.21mb	209.50mb
node mem	528.91mb	612.91mb	4208.14mb	4813.26mb	901.68mb
Fairy Forest					
push	41,276,918	37,782,311	81,520,006	79,526,630	35,022,499
pop	22,164,962	23,613,104	56,139,486	52,620,765	35,022,499
stack mem	726.03mb	702.61mb	1575.39mb	1512.31mb	267.20mb
node mem	725.24mb	724.97mb	2243.19mb	2110.21mb	1148.96mb
Dragon					
push	24,004,047	27,985,104	47,836,258	45,899,883	22,821,350
pop	14,492,154	18,751,708	31,671,835	29,835,165	22,821,350
stack mem	440.55mb	534.87mb	909.90mb	866.72mb	174.11mb
node mem	442.03mb	489.04mb	1215.63mb	1190.44mb	739.84mb

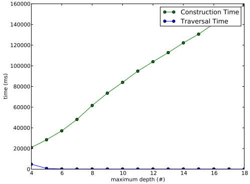
Appendix VIII





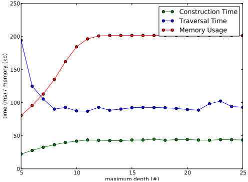


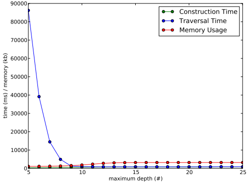


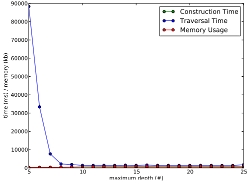


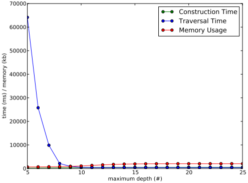
Scene	mem *depth=4*	mem *depth=11*	mem *depth=18*
Head	273kb	2276kb	4109kb
Sponza	2300kb	37709kb	65835kb
Sibenik	627kb	11237kb	20104kb
Fairy	1240kb	21234kb	44189kb
Dragon	37336kb	705242kb	>1500000kb

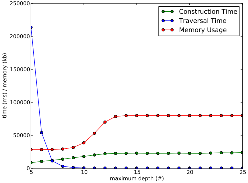
Appendix IX


Construction Time
Traversal Time
Memory Usage
time (ms) / memory (kb)
maximum depth (#)


Construction Time
Traversal Time
Memory Usage
time (ms) / memory (kb)
maximum depth (#)


Construction Time
Traversal Time
Memory Usage
time (ms) / memory (kb)
maximum depth (#)


Construction Time
Traversal Time
Memory Usage
time (ms) / memory (kb)
maximum depth (#)


Construction Time
Traversal Time
Memory Usage
time (ms) / memory (kb)
maximum depth (#)