Improving Memory Space Efficiency of Kd-tree for Real-time Ray Tracing

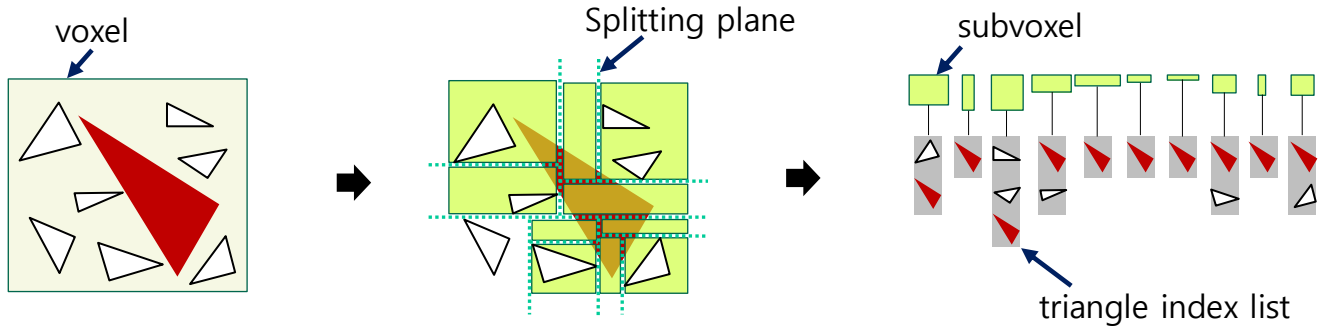
Byeongjun Choi, Byungjoon Chang, Insung Ihm

Department of Computer Science and Engineering
Sogang University, Korea

Motivation

- **Kd-tree for ray tracing**
  - Known as one of the most effective structures for accelerating ray-object intersection calculations
  - Constructed in a top-down manner by recursive subdividing the scene's voxel space using axis-aligned splitting planes

- **An inherent problem of the kd-tree construction algorithm**
  - When a triangle in a voxel penetrates a splitting plane, it is sorted into both subvoxels redundantly.
  - Inevitably, **the replication of triangles** during kd-tree construction leads to **spatially inefficient tree structures**.



- **Example: Redundant triangles in leaf nodes of kd-trees**
  - **Conference scene with 190,947 triangles**:
    - Built using the SAH-based construction algorithm in [Wald & Havran 06]
    - A substantial amount of *bad triangles* intersected repeatedly with the splitting planes
    - **A 14.41 times increase in the number of triangle references**!

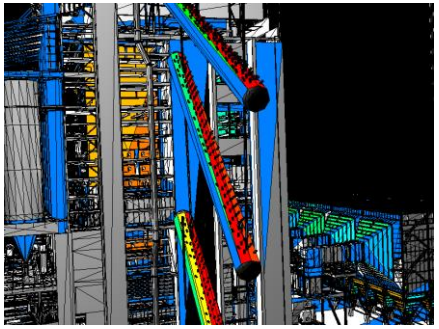
The number of triangles in the frequency interval

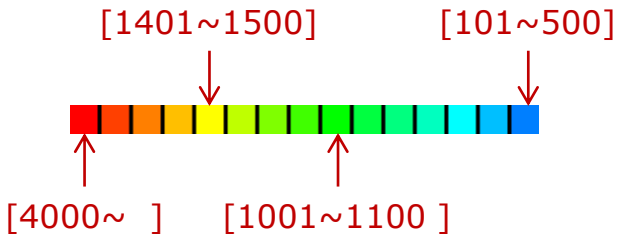
The average length of the longest edges of triangles

interval	Kitchen (101,015)		Conference (190,947)		Soda Hall (2,167,474)		San Miguel (10,500,551)		Power Plant (12,748,510)	
	count	longest	count	longest	count	longest	count	longest	count	longest
10,001 ~ 15,000	0	-	0	-	0	-	2	6.395	0	-
5,001 ~ 10,000	0	-	0	-	0	-	3	4.267	18	139.3
2,001 ~ 5,000	0	-	1	84.77	0	-	43	5.069	150	73.43
1,001 ~ 2,000	1	6.265	10	11.77	36	184.6	147	1.276	451	72.85
501 ~ 1,000	0	-	73	5.029	215	139.0	776	1.020	1,955	63.27
201 ~ 500	3	3.763	591	2.882	3,131	42.66	4,926	0.398	15,619	36.00
101 ~ 200	8	0.995	4,100	2.054	4,468	51.36	15,360	0.196	36,530	32.04
51 ~ 100	50	0.860	6,901	1.808	20,527	34.96	64,113	0.121	85,873	25.38
21 ~ 50	322	0.315	16,143	1.061	112,036	24.24	364,056	0.076	363,373	15.51
11 ~ 20	1,782	0.157	22,758	0.521	249,662	16.86	1,151,922	0.047	690,882	10.94
1 ~ 10	98,849	0.034	140,370	0.353	1,777,399	7.304	8,899,203	0.025	11,553,659	2.625
no. of indices	323,606 (x3.20)		2,752,009 (x14.41)		16,539,116 (x7.63)		77,619,842 (x7.39)		78,550,737 (x6.16)	

The total numbers of indices stored in the leaf nodes

- **Examples of highly redundant triangles in kd-trees**
  - In general, relatively large and/or skinny triangles cause a trouble.





- **Relative sizes of kd-trees with respect to polygon datasets**

	Number of triangles (K)	Number of vertices (K)	Size of geometry (MB)	Size of Kd-tree (MB)
Kitchen	101.0	53.4	**2.79**	**2.94**
Conference	190.9	114.3	**5.67**	**21.06**
Soda Hall	2,167.5	5,489.8	**192.34**	**131.98**
San Miguel	10,500.6	6,093.5	**306.13**	**662.74**
Power Plant	12,748.5	5,731.5	**320.81**	**635.22**

Geometry: stored in indexed face set, **Kd-tree:** built via the algorithm by Wald and Havran(2006)

Our Contributions

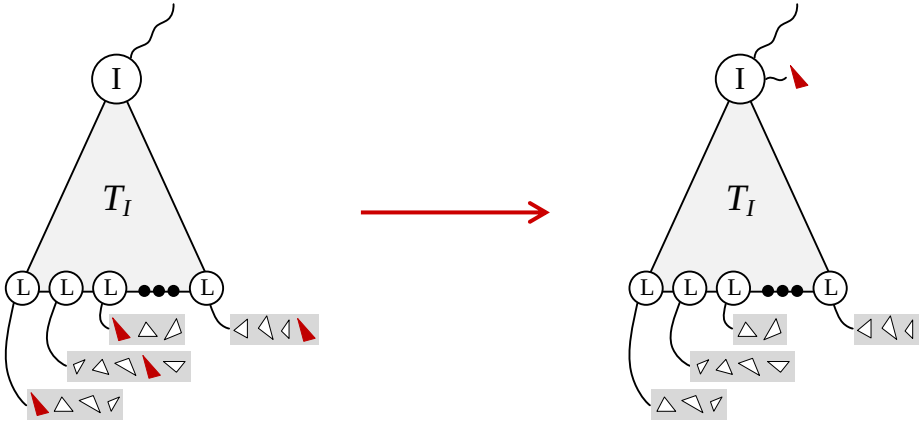
- **A space-efficient kd-tree representation method** that allows the optional storage of a triangle reference in an inner node
  - Reduce memory space by storing indices of excessively duplicated triangles in appropriate inner nodes

- **A kd-tree construction algorithm** based on **two simple cost metrics** that determine when and how to choose triangles for inner nodes
  - Minimize the run-time inefficiency caused by unnecessary early intersection tests during ray tracing

- **A compact unified data layout** to store the new kd-tree structure
  - Particuarly, allow an efficient representation of leaf nodes using two different modes

Our Idea

- **Select and move the triangles that are excessively duplicated in leaf nodes to proper inner nodes**



  - In **the modified kd-tree traversal algorithm**, an additional ray-triangle intersection computation is required each time an inner node containing a triangle reference is visited.
  - The ray-tracing performance **may be degraded** due to possibly unnecessary early calculations for ray-triangle intersections.
  - The triangles must be selected carefully to **avoid wasteful computations as much as possible**.

Two Heuristic Metrics for Triangle Selection

- **Occupancy measure**
  - **Select a triangle if the likelihood that, for a given random ray, the triangle will eventually be tested for a ray-triangle intersection is high.**

A subtree of a kd-tree *T*, rooted in an inner node *I*

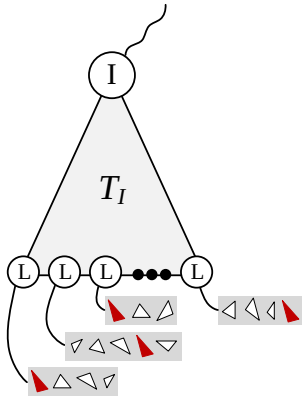
The set of all leaf nodes of T_I passed by t

$$f_{occu}(t, T_I) = \frac{\sum_{L \in \mathcal{L}(t, T_I)} SA(V_L)}{\sum_{L \in \mathcal{L}(\cdot, T_I)} SA(V_L)}$$

A triangle in a leaf node

The surface area of the voxel for leaf node L

The set of all leaf nodes of T_I



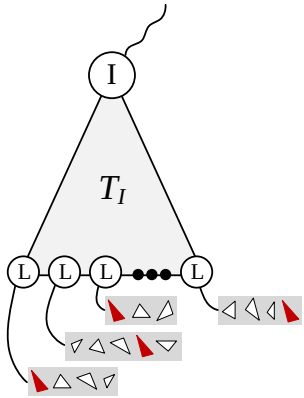
  - **The sum of the geometric probability (numerator) that each leaf node intersecting with the triangle *t* will be hit by a random ray**, normalized between 0 and 1.
    - Estimate the chance that a ray-triangle intersection computation for t occurs during the traversal of T_I.
  - **Store only the triangle with a sufficiently high occupancy measure in the inner node!**

- **Frequency measure**
  - **Cull a triangle from leaf nodes if its effect in terms of data size reduction is sufficiently high.**

The number of leaf nodes of T_I passed by t

$$f_{freq}(t, T_I) = \frac{|\mathcal{L}(t, T_I)|}{|\mathcal{L}(\cdot, T_I)|}.$$

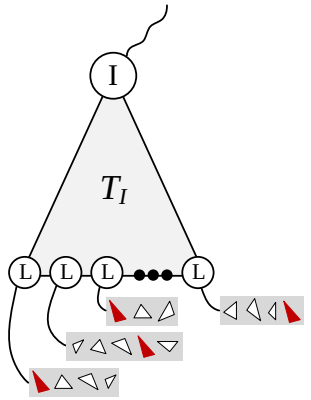
The total number of leaf nodes of T_I



    - **The number of leaf nodes containing a reference to *t* (numerator), normalized between 0 and 1.**
      - Estimate the amount of size reduction of T_I when t is culled from leaf nodes.
  - **Store only the triangle with a sufficiently high frequency measure in the inner node!**

- **Our strategy for selecting a triangle *t* from a subtree T_I**
  ① Enumerate the triangles in T_I with an **occupancy** measure, greater than a given **occupancy threshold**, in a nonincreasing order.
  ② Check each triangle until one with a **frequency** measure, greater than a given **frequency threshold**, is found.

  – **An oracle function for triangle selection** $pick\_triangle(T_I)$
    - A triangle is returned if a suitable one is available.
    - Otherwise, a null value is returned.



The New Kd-tree Construction Algorithm

Algorithm 1 Iterative kd-tree construction

```
1:  Build an SAH-based kd-tree T for T_S;
2:   push(root_node(T));
3:  while ( stack is not empty) do
4:     I = pop();
5:     if ((t = pick_triangle(T_I)) != NULL) then
6:        Build an SAH-based kd-tree T_I* for T(T_I) − {t};
7:        Replace T_I with T_I*, and place t on I;
8:        if (count_triangles(I) == max_t2rn) then
9:           continue; {Skip the subtrees of T_I*.}
10:       end if
11:    end if
12:    if (rchild_node(T_I*) is not a leaf node) then
13:       push(rchild_node(T_I*));
14:    end if
15:    if (lchild_node(T_I*) is not a leaf node) then
16:       push(lchild_node(T_I*));
17:    end if
18: end while
19: return T;
```

Start with a kd-tree T, built for an input scene using a standard construction algorithm.

Examine the next non-leaf node I.

If there exists no eligible triangle in T_I, stop.

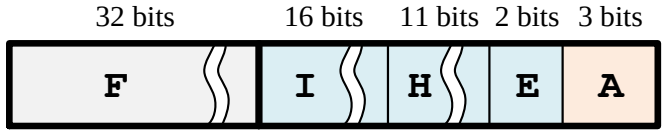
If t is selected from T_I,
1. replace T_I with a new kd-tree T_I^*, built using the standard method, for the set of triangles in T_I except t, and
2. store the index of t in the root node of T_I^*.

Limit the number of triangles in inner nodes from the node I to the root node.

Recursive apply the same process to the subtrees rooted in each child of T_I^*.

Compact Node Layouts for New Kd-trees

- Extend the kd-tree representation proposed in [Wald 04]
- **Inner nodes: type II inner node + T-reference node**
  - Details are given in the paper.
- **Leaf nodes: 2-byte mode leaf node**
  - Added to the conventional 4-byte mode type
    - Frequently, a leaf node contains a few triangles, **stored with high spatial coherence in the triangle array**.
    - Their indices in 4-byte unsigned integer **share the same upper 16 bits**.
    - Allocate an extra array of triangle indices in 2-byte unsigned short **to store the lower 16 bits**.



A: 011, **E**: 01

F: Pointer to the first item in the new array

H: Number of triangles in leaf node

I: Upper 16 bits shared by triangle indices

✓ Among **1,705,325 triangle indices** stored in the leaves for the **Conference** scene,
  - **1,565,148 indices** are stored **in the 2-byte mode**!

Implementation and Experiments

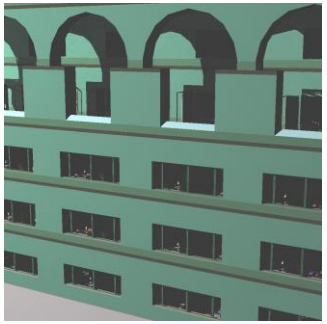
- **Compared kd-tree construction algorithms**
  - **Standard:** the SAH-based algorithm in [Wald and Havran 2006]
  - **Ours:** the presented algorithm

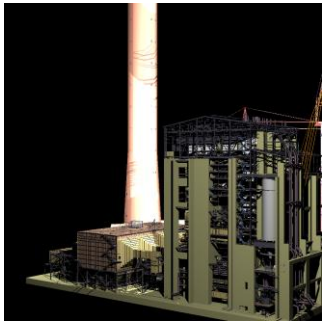
- **Tested processors**
  - **CPU:** Dual 3.46 GHz Intel Xeon (six-core)
    - 1-threaded 4x4 SIMD ray packets
    - 12-threaded 4x4 SIMD ray packets
  - **GPU:** An NVIDIA GeForce GTX 580 (Kitchen, Conference, and Soda Hall) or an NVIDIA Quadro 6000 (San Miguel and Power Plant)
    - CUDA blocks with 4x32 threads

- **Rendering method**
  - Full Whitted-style ray tracing of an 1024x1024 image

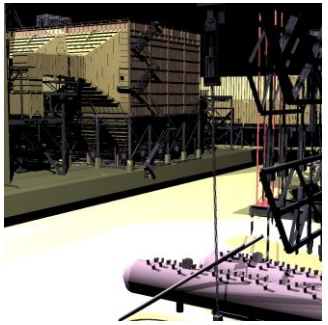
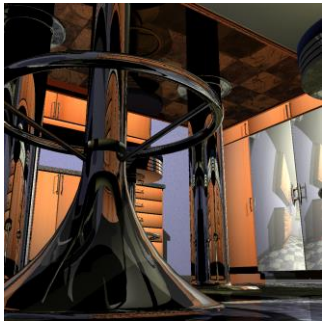
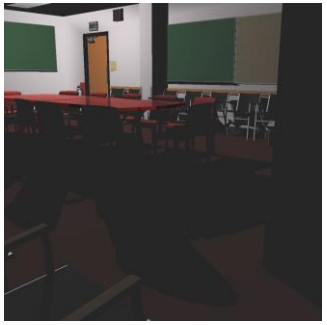
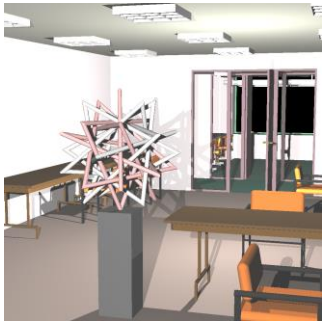
Example Scenes and Tested Camera Views

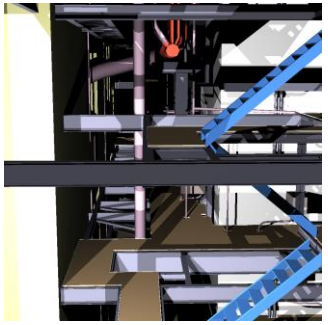








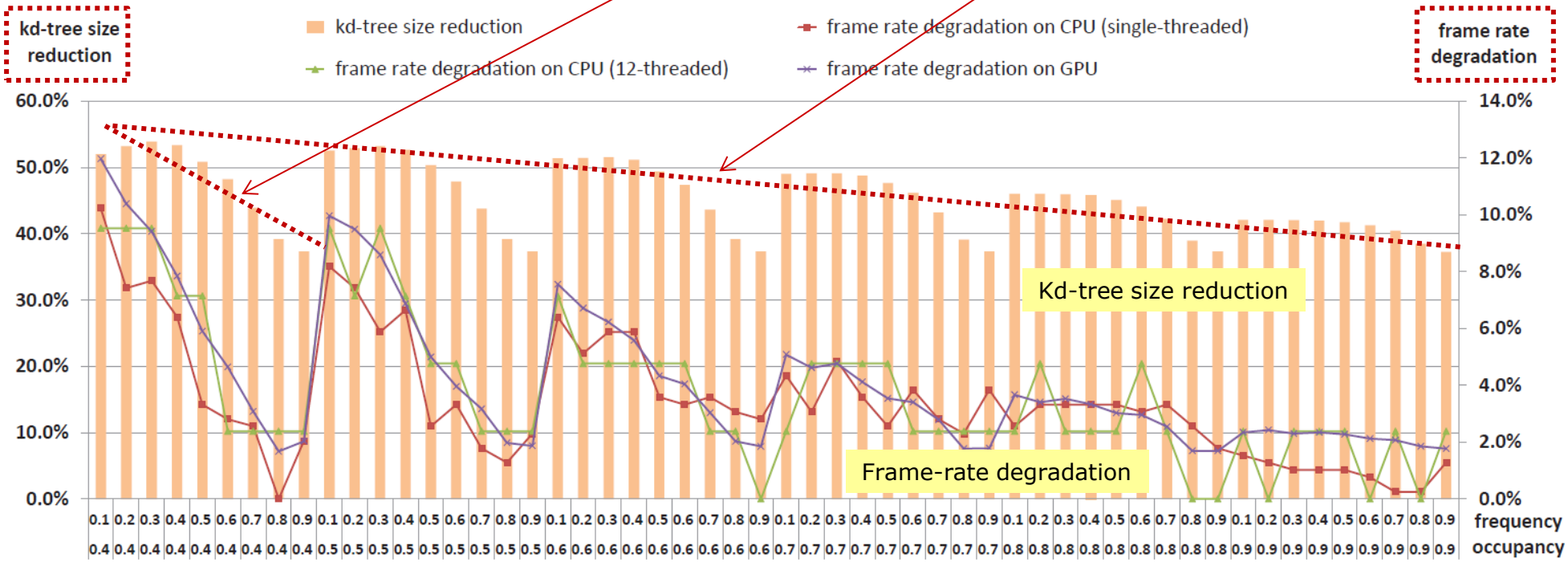








Kitchen (101,015) | **Conference (190,947)** | **Soda Hall (2,167,474)** | **San Miguel (10,500,551)** | **Power Plant (12,748,510)**

Effects of Occupancy and Frequency Thresholds

- In general, **the effect of size reduction** declined as the frequency or occupancy threshold increased.
  - A higher threshold made it hard to pick triangles for inner nodes.
- In general, **the frame-rate degradation** decreased as either threshold increased.
  - A sparse distribution of triangles on the inner nodes reduced unnecessary early intersection tests.



Experimental Results

- Tested two combinations of occupancy and frequency thresholds
  - **Ours(fast):** *fast rendering preferred*
    - (occupancy, frequency) = (0.9, 0.7)
  - **Ours(small):** *small size preferred*
    - (occupancy, frequency) = (0.5, 0.4)

- Measured **kd-tree-size reduction**
  - Number of kd-tree nodes
  - Number of indices to triangles
  - Memory usage

- Measured **frame-rate degradation** observed during ray tracing
  - A single-thread mode on a CPU (CPU1)
  - A 12-thread mode on dual 6-core CPUs (CPU12)
  - A 512-core (or 448-core) GPU (GPU)

Reduction in Number of Kd-tree Nodes

	Method	Inner	T-inner	Leaf	Total
Kitchen (101,015)	Standard	112,100	0	112,101	224,201
	Ours(fast)	87,711	11,923	99,635	199,269
	Ours(small)	50,616	32,859	83,476	166,951
Conference (190,947)	Standard	692,332	0	692,333	1,384,665
	Ours(fast)	356,436	104,548	460,985	921,969
	Ours(small)	197,930	142,872	340,803	681,605
Soda Hall (2,167,474)	Standard	4,514,974	0	4,514,975	9,029,949
	Ours(fast)	2,765,685	600,247	3,365,933	**6,731,865**
	Ours(small)	1,288,647	1,132,261	2,420,909	**4,841,817**
San Miguel (10,500,551)	Standard	24,028,541	0	24,028,542	48,057,083
	Ours(fast)	15,596,361	3,213,304	18,809,666	37,619,331
	Ours(small)	7,915,317	6,186,060	14,101,378	28,202,755
Power Plant (12,748,510)	Standard	21,992,127	0	21,992,128	**43,984,255**
	Ours(fast)	13,770,505	3,030,957	16,801,463	**33,602,925**
	Ours(small)	7,326,642	5,649,934	12,976,577	**25,953,153**

A less restrictive threshold created a kd-tree with a smaller no. of nodes: **(0.9, 0.7)** vs **(0.5, 0.4)**.

The no. of kd-tree nodes decreased markedly, requiring less memory space for storage.

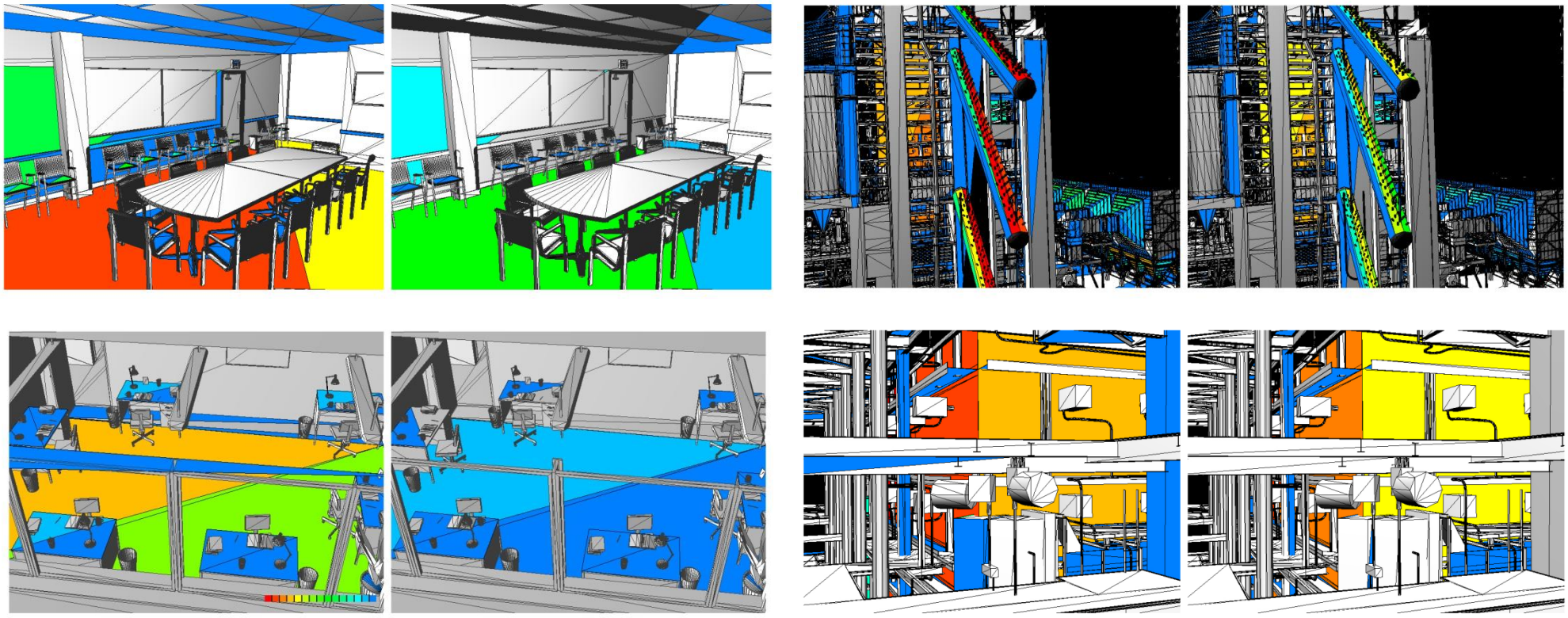
Reduction in # of Indices to Triangles & Memory Usage

	Method	4-byte mode	2-byte mode	Total indices	Memory Usage (MB)
Kitchen (101,015)	Standard	323,606	0	323,606	**2.9**
	Ours(fast)	12,570	265,761	278,331	**2.2 (26.4%)**
	Ours(small)	10,369	227,220	237,589	**2.0 (32.2%)**
Conference (190,947)	Standard	2,752,009	0	2,752,009	**21.1**
	Ours(fast)	140,177	1,565,148	1,705,325	**11.4 (46.1%)**
	Ours(small)	117,488	1,280,786	1,398,274	**9.2 (56.4%)**
Soda Hall (2,167,474)	Standard	16,539,116	0	16,539,116	**132.0**
	Ours(fast)	427,739	11,026,989	11,454,728	**78.6 (40.4%)**
	Ours(small)	326,192	8,230,869	8,557,061	**62.5 (52.6%)**
San Miguel (10,500,551)	Standard	77,619,842	0	77,619,842	**662.7**
	Ours(fast)	15,856,444	42,257,800	58,114,244	**452.6 (31.7%)**
	Ours(small)	13,472,086	33,806,033	47,278,119	**378.2 (42.9%)**
Power Plant (12,748,510)	Standard	**78,550,737**	**0**	78,550,737	**635.2**
	Ours(fast)	**8,984,545**	**46,722,160**	55,706,705	**402.9 (36.6%)**
	Ours(small)	**7,628,722**	**38,600,428**	46,229,150	**343.8 (45.9%)**

Culling redundant triangles from leaf nodes markedly increased spatial coherence between triangles in a leaf node.

Kd-tree size reduction: 26.4%-46.1%(Ours(fast)) and 32.2%-56.4%(Ours(small)).

Problematic Triangles in Kd-Trees: Before and After



	Kitchen (101,015)			Conference (190,947)			Soda Hall (2,167,474)			San Miguel (10,500,551)			Power Plant (12,748,510)		
interval	std.	ours(f)	ours(s)	std.	ours(f)	ours(s)	std.	ours(f)	ours(s)	std.	ours(f)	ours(s)	std.	ours(f)	ours(s)
10001 >> 15000	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
9001 >> 10000	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
5001 >> 9000	0	0	0	0	0	0	0	0	0	3	0	0	16	0	0
4001 >>5000	0	0	0	0	0	0	0	0	0	1	0	0	10	0	0
3001 >> 4000	0	0	0	1	0	0	0	0	0	12	3	0	36	10	0
2001 >> 3000	0	0	0	0	0	0	0	0	3	30	4	0	104	19	6
1501 >> 2000	1	0	0	0	0	0	3	0	16	40	12	12	101	35	14
1401 >> 1500	0	0	0	1	0	0	0	0	4	6	7	6	39	15	3
1301 >> 1400	0	0	0	2	1	0	5	0	3	18	9	10	36	17	3
1201 >> 1300	0	0	0	1	0	0	11	0	6	14	8	7	60	16	8
1101 >> 1200	0	1	0	2	0	0	8	0	3	34	10	9	100	13	13
1001 >> 1100	0	0	0	4	0	0	9	0	2	35	20	13	115	18	14
901 >> 1000	0	0	0	6	0	0	21	0	2	53	28	18	182	31	22
801 >> 900	0	0	0	6	0	0	10	4	6	80	40	13	234	53	34
701 >> 800	0	0	1	9	1	0	17	1	8	111	68	38	282	119	70
601 >> 700	0	0	0	23	1	0	46	5	10	175	83	43	435	236	170
501 >> 600	0	0	0	29	0	2	121	9	17	357	129	103	822	301	297
401 >> 500	0	0	0	35	2	0	254	24	31	570	185	197	1514	614	494
301 >> 400	0	0	0	65	10	4	1194	34	48	1155	377	411	3613	1072	906
201 >> 300	3	1	0	491	50	26	1683	226	130	3201	1129	1216	10492	3055	2121
101 >> 200	8	9	3	4100	816	696	4468	1502	822	15360	5345	7036	36530	19687	13938
91 >> 100	4	2	0	971	420	316	1144	815	197	4798	1411	1914	8734	5212	4380
81 >> 90	3	2	4	1004	653	473	2253	992	266	6702	2077	2441	11360	6654	5811
71 >> 80	9	2	1	1299	963	648	3212	1263	332	10284	3150	3503	15105	8383	7851
61 >> 70	9	4	5	1507	1509	998	5341	2082	414	16218	5341	5008	21029	12672	11087
51 >> 60	25	7	7	2120	1984	1416	8577	3104	989	26111	10108	8315	29645	18255	16688
41 >> 50	33	12	24	3118	2664	2070	16029	6077	2907	45221	21849	15222	50054	30257	28709
31 >> 40	79	38	25	4852	4140	3544	30265	16489	8904	90091	50097	35591	95718	57613	55200
21 >> 30	210	157	134	8173	7525	6646	65742	49084	31222	228744	144304	105418	217601	141950	125239
11 >> 20	1782	1302	1062	22758	16210	16059	249662	184406	138800	1151922	818539	581762	690882	499949	418852
1 >> 10	98849	99478	99749	140370	153998	158049	1777399	1901357	1982332	8899203	9436218	9732245	11553659	11942254	12056580
tri. redun.	x3.20	x2.87	x2.68	x14.41	x9.48	x8.07	x7.63	x5.56	x4.47	x7.39	x5.84	x5.09	x6.16	x4.61	x4.07

Frame-rate Degradation During Ray Tracing

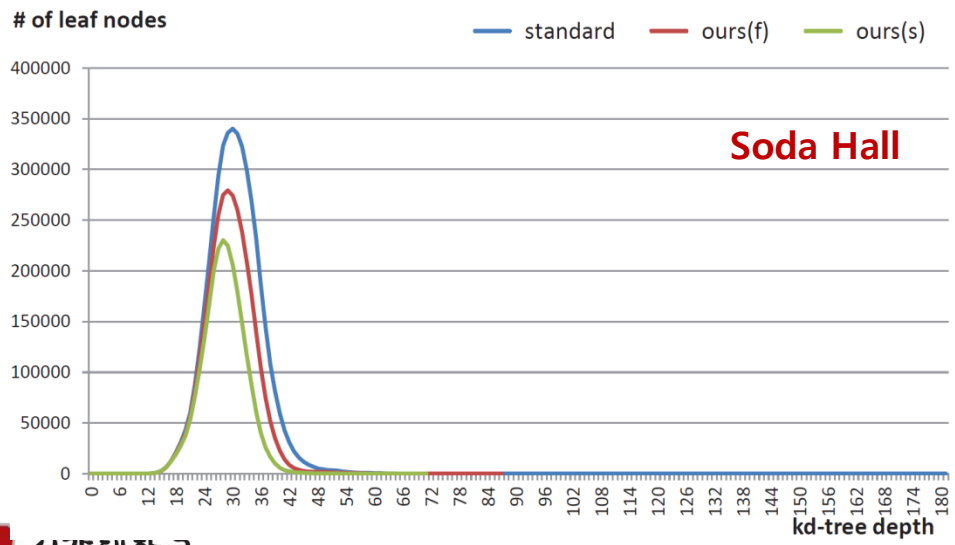
	Method	Camera view 1			Camera view 2			Camera view 3		
		CPU1	CPU12	GPU	CPU1	CPU12	GPU	CPU1	CPU12	GPU
Kitchen (101,015)	Standard	0.867	8.621	16.448	0.959	8.772	20.388	0.709	6.536	11.489
	Ours(fast)	0.854 (1.6%)	8.403 (2.6%)	15.971 (3.0%)	0.949 (1.1%)	8.696 (0.9%)	19.396 (5.1%)	0.700 (1.3%)	6.410 (2.0%)	11.189 (2.7%)
	Ours(small)	0.794 (9.2%)	7.874 (9.5%)	15.009 (9.6%)	0.878 (9.2%)	8.000 (9.6%)	18.242 (11.8%)	0.657 (7.9%)	5.988 (9.2%)	10.545 (9.0%)
Conference (190,947)	Standard	1.852	18.868	24.131	1.689	16.393	22.933	2.024	19.608	26.332
	Ours(fast)	1.828 (1.3%)	18.182 (3.8%)	23.895 (1.0%)	1.634 (3.4%)	15.625 (4.9%)	22.695 (1.0%)	1.961 (3.2%)	18.868 (3.9%)	26.044 (1.1%)
	Ours(small)	1.727 (7.2%)	17.241 (9.4%)	23.083 (4.5%)	1.555 (8.6%)	15.152 (8.2%)	22.054 (4.0%)	1.869 (8.3%)	**17.857 (9.8%)**	25.361 (3.8%)
Soda Hall (2,167,474)	Standard	1.248	11.364	14.638	1.395	12.658	19.975	0.454	4.132	5.123
	Ours(fast)	1.209 (3.2%)	10.989 (3.4%)	14.397 (1.7%)	1.374 (1.5%)	**12.195 (3.8%)**	19.378 (3.1%)	0.442 (2.8%)	3.984 (3.7%)	5.496 (-6.8%)
	Ours(small)	1.135 (10.0%)	10.309 (10.2%)	13.687 (6.9%)	1.276 (9.3%)	11.364 (11.4%)	18.241 (9.5%)	0.430 (5.5%)	3.906 (5.8%)	5.471 (-6.4%)
San Miguel (10,500,551)	Standard	0.102	1.016	2.576*	0.153	1.406	3.321*	0.285	2.584	4.744*
	Ours(fast)	0.097 (5.1%)	0.965 (5.3%)	2.473* (4.2%)	0.149 (2.8%)	1.364 (3.1%)	3.166* (4.9%)	0.277 (3.1%)	2.519 (2.6%)	4.511* (5.2%)
	Ours(small)	**0.091 (12.5%)**	0.907 (12.1%)	2.307* (11.7%)	0.138 (11.0%)	1.297 (8.4%)	3.016* (10.1%)	0.255 (11.8%)	2.387 (8.3%)	4.330* (9.6%)
Power Plant (12,748,510)	Standard	0.421	3.497	11.458*	0.522	4.444	6.842*	1.229	11.111	16.092*
	Ours(fast)	0.424 (-0.7%)	3.401 (2.8%)	11.173* (2.6%)	0.506 (3.2%)	4.202 (5.8%)	6.630* (3.2%)	1.217 (1.0%)	10.753 (3.3%)	**15.058* (6.9%)**
	Ours(small)	0.399 (5.4%)	3.185 (9.8%)	10.460* (9.5%)	0.473 (10.4%)	3.953 (12.4%)	6.198* (10.4%)	1.171 (4.9%)	10.417 (6.7%)	14.376* (11.9%)

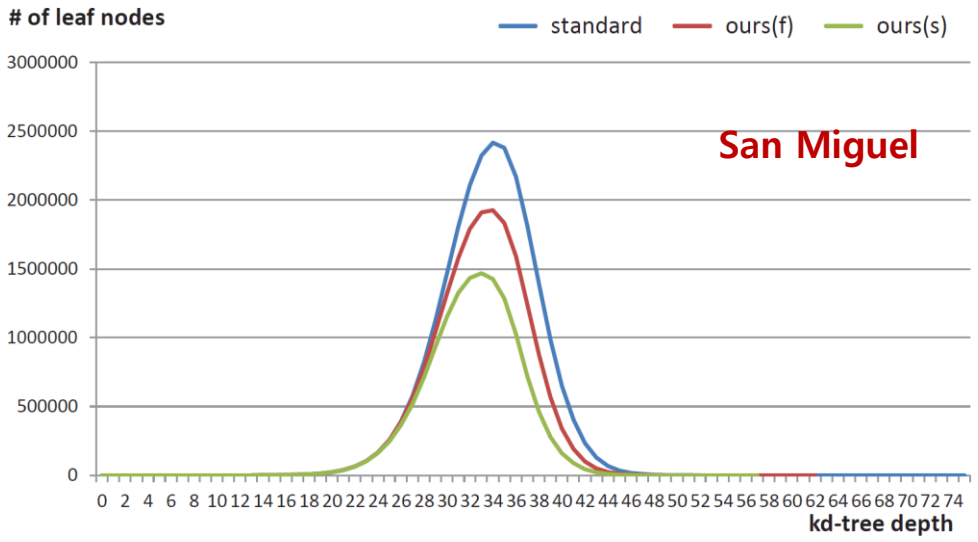
Ours(fast) resulted in less than 3.8%(Soda Hall) to 6.9%(Power Plant) frame-rate degradation.

Ours(small) resulted in less than 9.8%(Conference) to 12.5%(San Miguel) frame-rate degradation.

Frequency Distribution of Leaf Node Depths

- **Culling triangles from leaf nodes contracted the kd-trees**.
  - **The maximum depth** of the kd-tree.
    - Soda Hall: 181(standard) → 87(ours(fast)), 71(ours(small))
    - San Miguel: 75(standard) → 62(ours(fast)), 57(ours(small))
  - **The peak of the distribution** tended to move slightly to the left, where significant frequency reductions were made.
- The shorter structures **decreased the kd-tree traversal cost during ray tracing**.





Conclusion: Pros and Cons

- Presented **a space-efficient kd-tree construction algorithm**

  - **Utilized two effective heuristic measures** for determining when and how to choose triangles for inner nodes.

  - **Significantly reduced the memory requirements** for storing the kd-tree structure for nontrivial scenes.

  - **Effectively suppressed the unavoidable frame-rate degradation** during ray tracing.

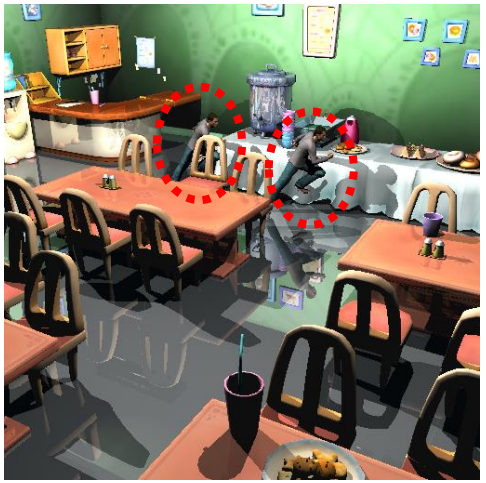
- Our algorithm is **currently limited to static scenes**.

Method	Kitchen	Conference	Soda Hall	San Miguel	Power Plant
Standard	1.4	2.7	27.4	249.2	286.0
Ours(fast)	5.2	8.6	54.4	1,671.5	2,283.6
Ours(small)	5.0	7.0	60.2	1,323.8	1,825.6

Construction time (sec) measured on a 3.1 GHz Intel Xeon 8-core CPU

- Can be useful for large static geometry in a dynamic scene where only a small portion of triangles need to be updated.

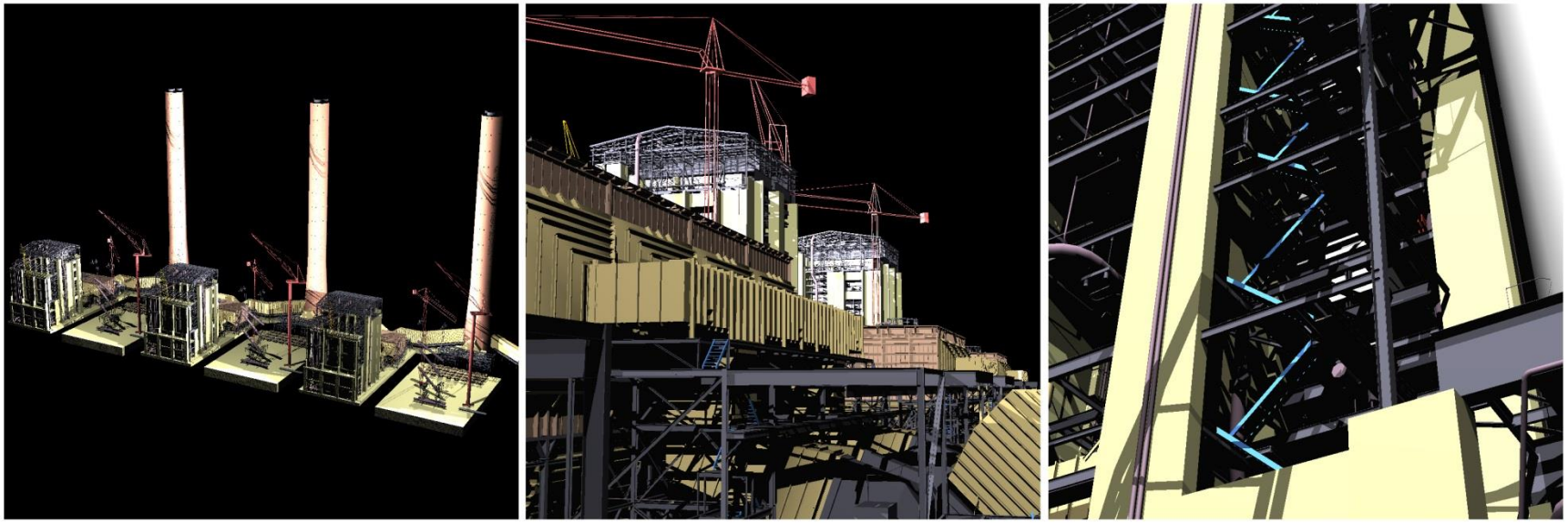
- Need **an effective occupancy measure that does not require a repeated application** of the standard SAH-based construction process.



Conclusion: Application to Very Complex Scenes

- **Similar performance** results were observed for a larger scene with 38.2M triangles.
  - Kd-tree-size reduction: 2,181.9MB to 1,414.1MB(35.2%) / 1,377.2MB(36.9%)
  - Frame-rate degradation: 2.9% / 5.8% on average (the single-thread mode)

- Need **proper quantization and/or geometry compression techniques** to combine with our kd-tree construction algorithm.



☺ Thank you~

http://grmanet.sogang.ac.kr
oipini@sogang.ac.kr