Ray Tracing with the BSP Tree

Thiago Ize
Ingo Wald
Steven G. Parker

THE UNIVERSITY OF UTAH

SCI INSTITUTE

Ray Tracing with the ***Real*** BSP Tree

Thiago Ize

Ingo Wald

Steven G. Parker

THE UNIVERSITY OF UTAH

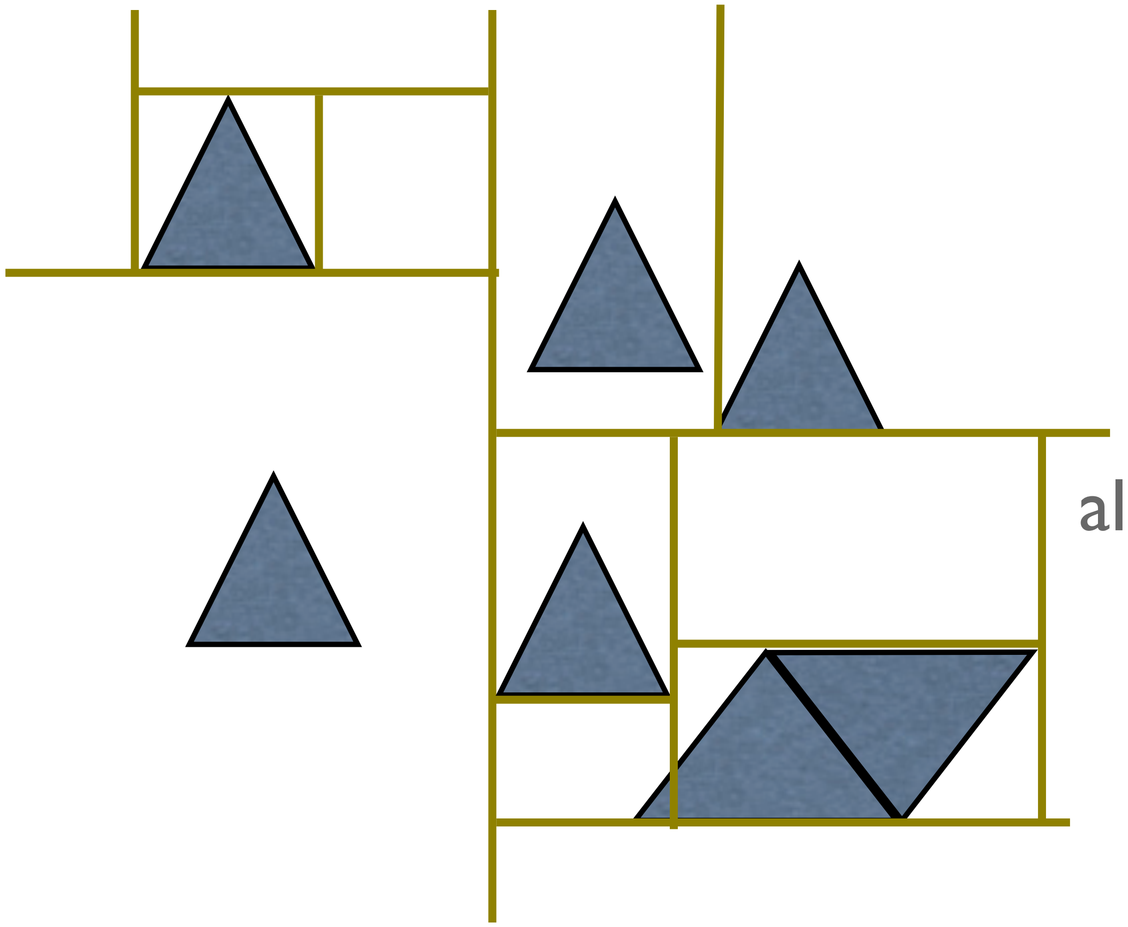
SCI INSTITUTE

History

- General BSPs are commonly assumed to be:
  - Too complicated to build
  - Numerically unstable
  - Slow to traverse
- Nothing published about actually trying them for ray tracing

kd-trees

- BSP with axis-aligned splitting planes
- Fast high quality $O(n \log n)$ SAH build
- Often the fastest acceleration structure for static scenes



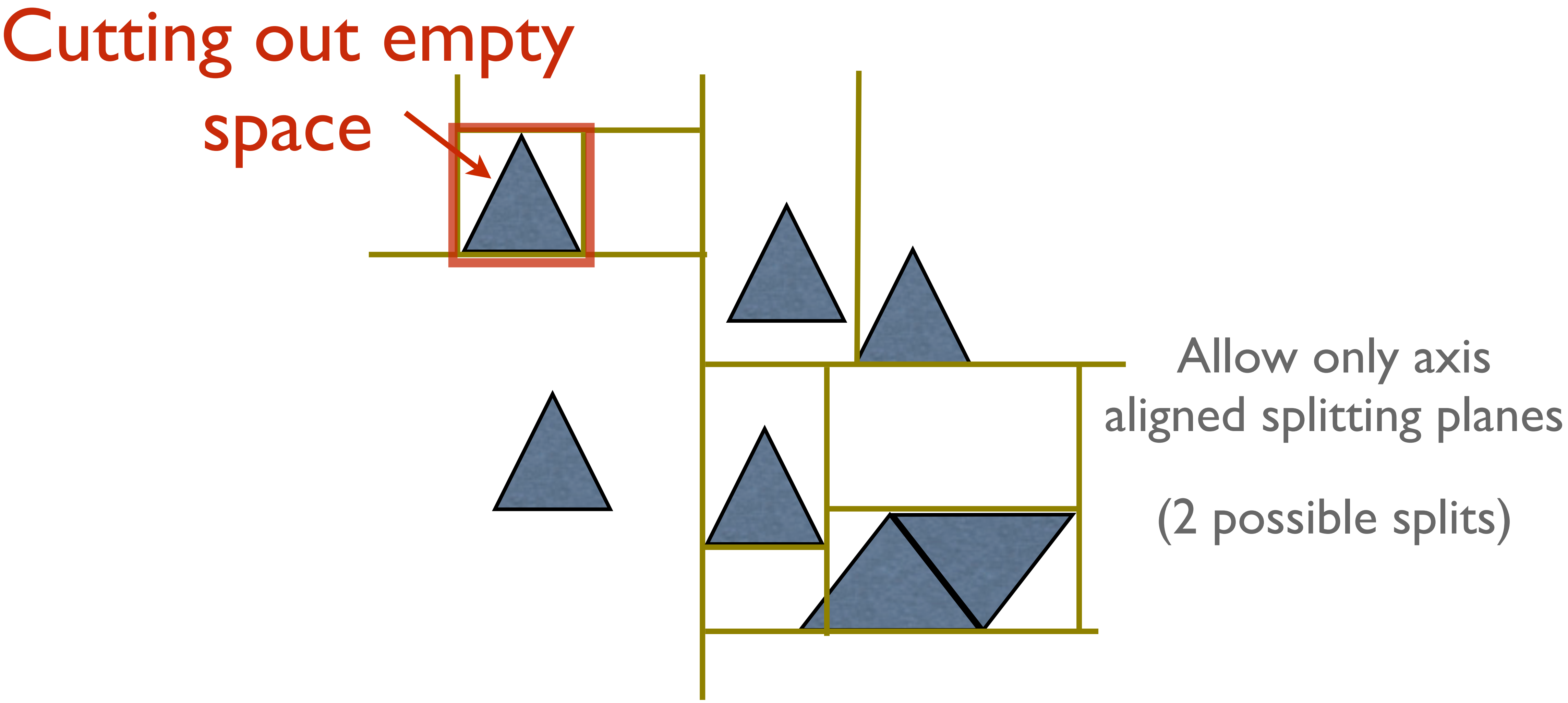
Allow only axis aligned splitting planes

(2 possible splits)

half the node still has empty space

kd-trees

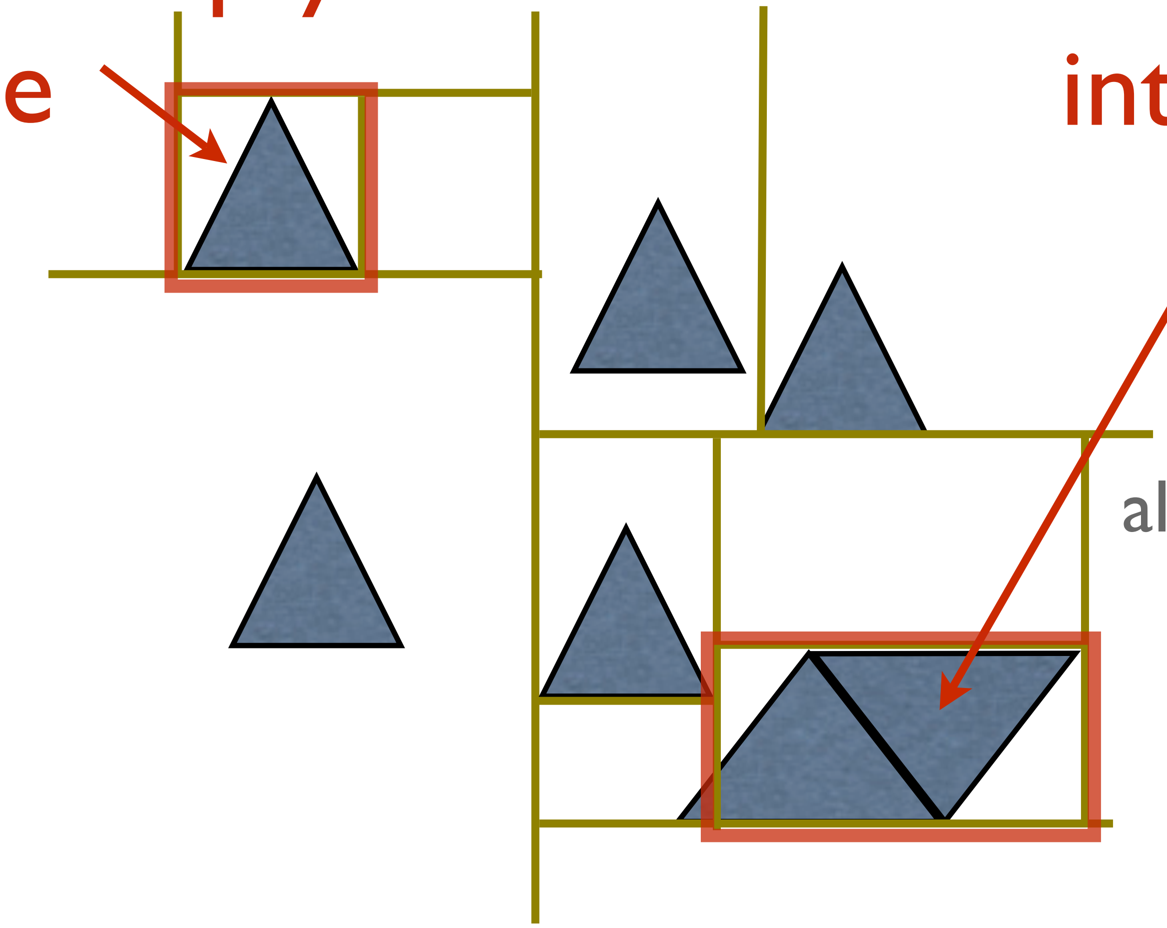
- BSP with axis-aligned splitting planes
- Fast high quality $O(n \log n)$ SAH build
- Often the fastest acceleration structure for static scenes



half the node still has empty space

kd-trees

- BSP with axis-aligned splitting planes
- Fast high quality $O(n \log n)$ SAH build
- Often the fastest acceleration structure for static scenes



Cutting out empty space

Cannot split triangles into separate nodes

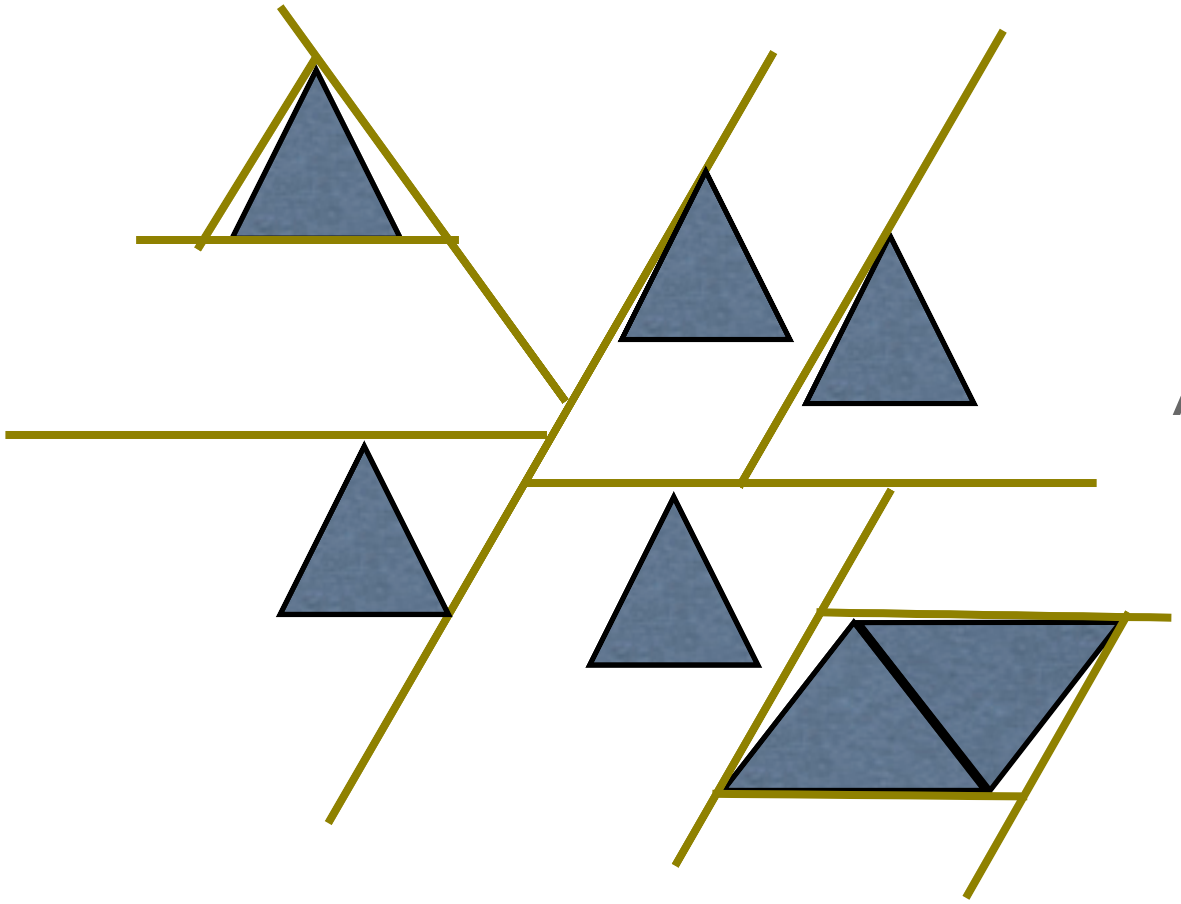
Allow only axis aligned splitting planes

(2 possible splits)

half the node still has empty space

Restricted BSPs

- Kammaje and Mora RT07 (introduction of RBSP)
- Budge et al. RT08 (much faster builds)
- Introduce a fixed number of additional splitting plane normals
- Currently still slower than kd-trees



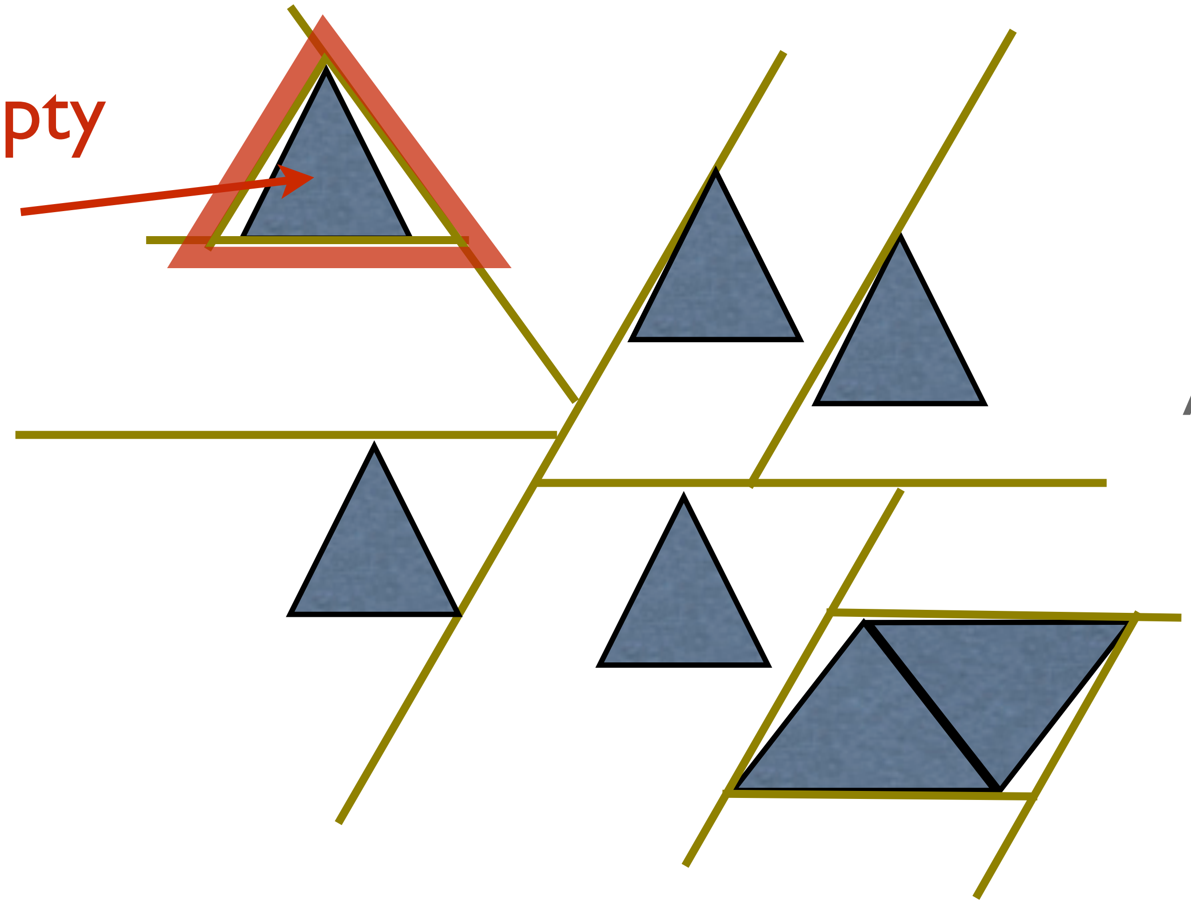
Allow splitting planes at every 30 degrees

(12 possible splits)

Builds are now faster. Might be interesting to use a RBSP build for the top level of the tree and then a real BSP build for the lower levels.

Restricted BSPs

- Kammaje and Mora RT07 (introduction of RBSP)
- Budge et al. RT08 (much faster builds)
- Introduce a fixed number of additional splitting plane normals
- Currently still slower than kd-trees



Cutting out empty space

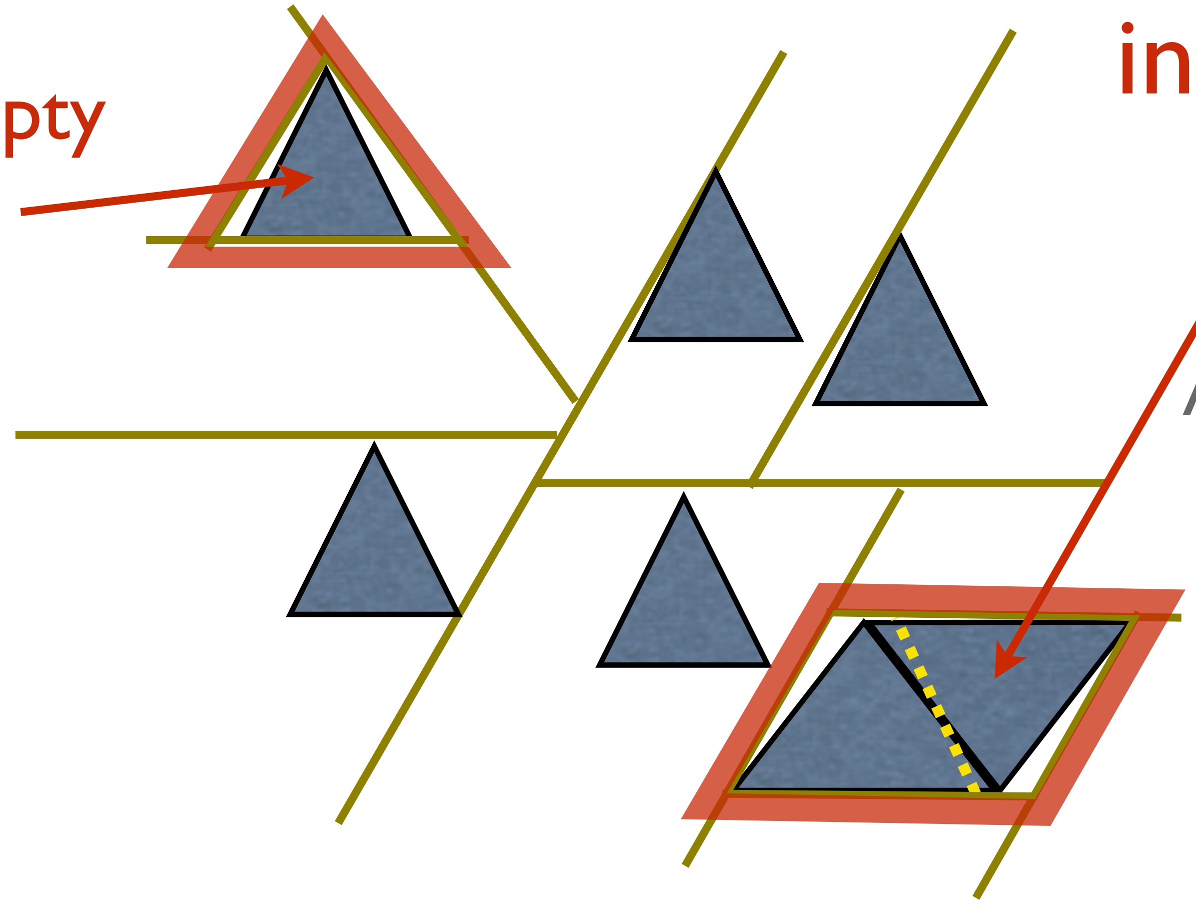
Allow splitting planes at every 30 degrees

(12 possible splits)

Builds are now faster. Might be interesting to use a RBSP build for the top level of the tree and then a real BSP build for the lower levels.

Restricted BSPs

- Kammaje and Mora RT07 (introduction of RBSP)
- Budge et al. RT08 (much faster builds)
- Introduce a fixed number of additional splitting plane normals
- Currently still slower than kd-trees



Cannot split triangles into separate nodes

Cutting out empty space

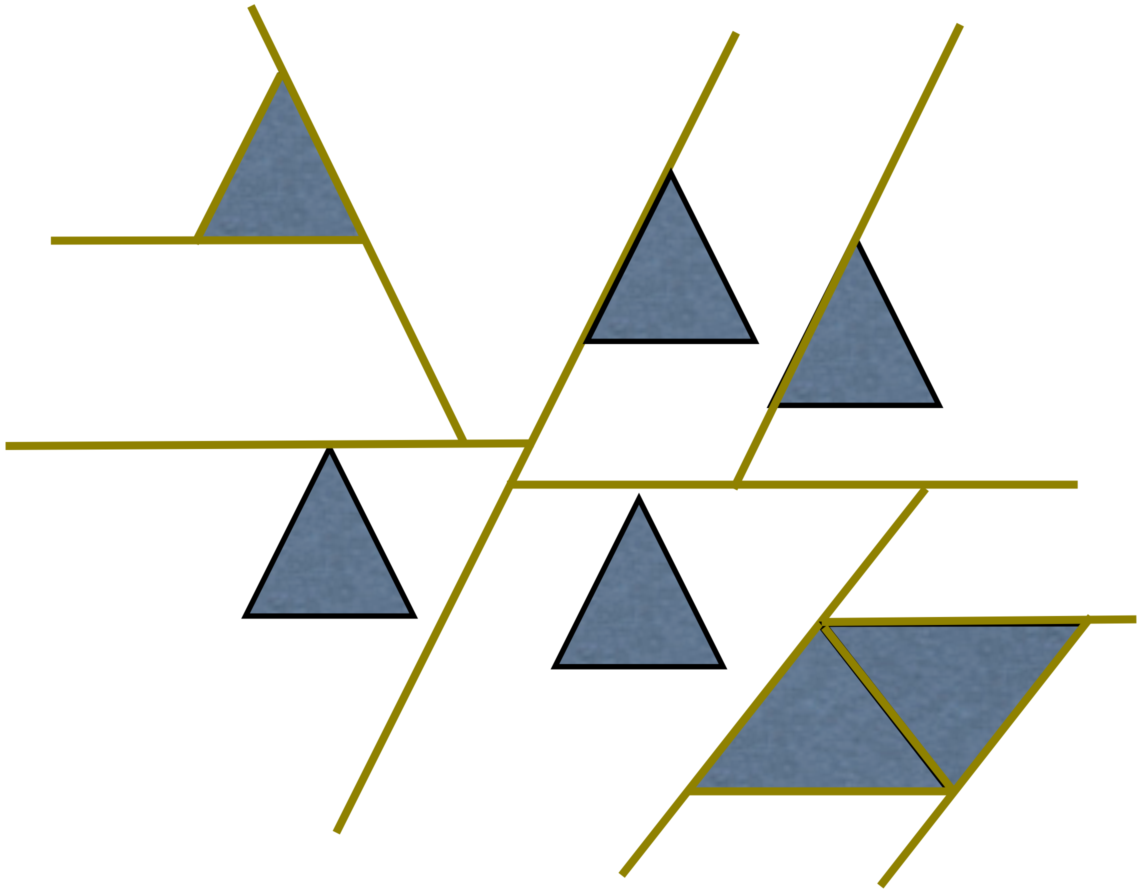
Allow splitting planes at every 30 degrees

(12 possible splits)

Builds are now faster. Might be interesting to use a RBSP build for the top level of the tree and then a real BSP build for the lower levels.

BSPs

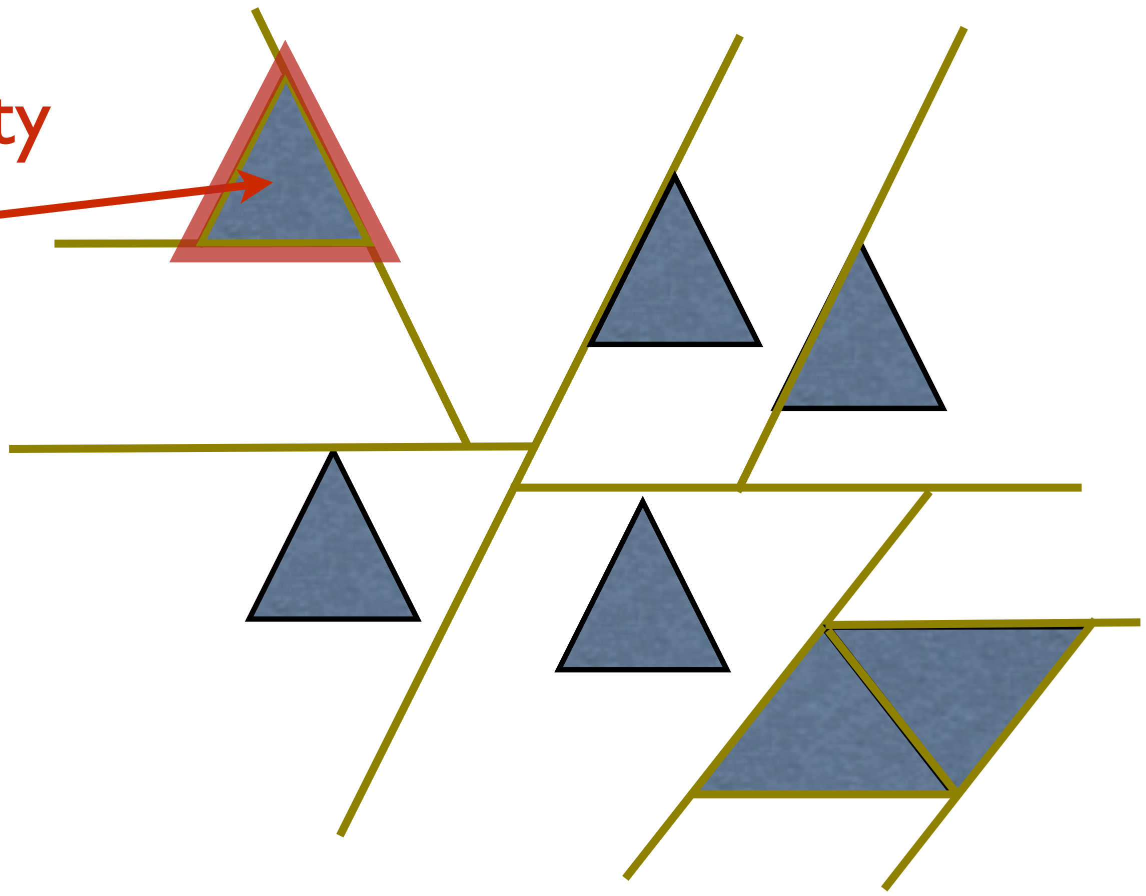
- Allow arbitrary splitting plane normals
- Builds are slow
- Faster than kd-trees for rendering



Any splitting plane is allowed

BSPs

- Allow arbitrary splitting plane normals
- Builds are slow
- Faster than kd-trees for rendering

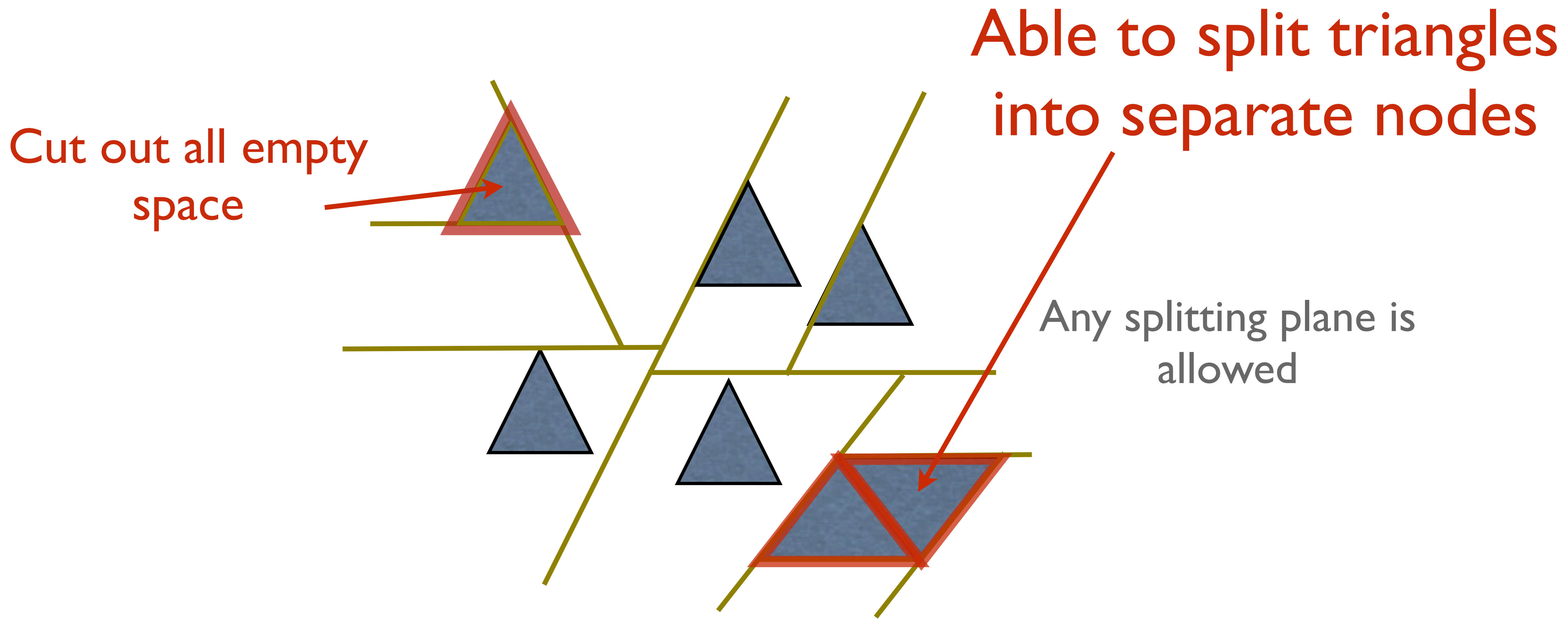


Cut out all empty space

Any splitting plane is allowed

BSPs

- Allow arbitrary splitting plane normals
- Builds are slow
- Faster than kd-trees for rendering

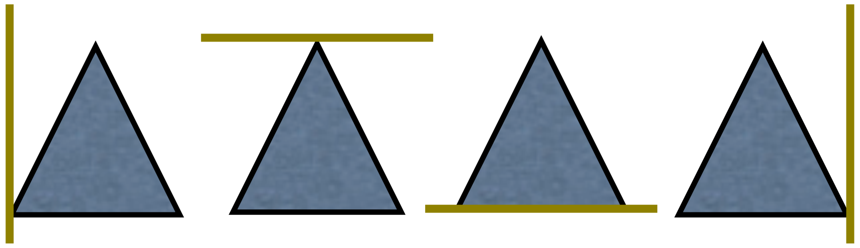

Able to split triangles into separate nodes
Cut out all empty space
Any splitting plane is allowed

Build

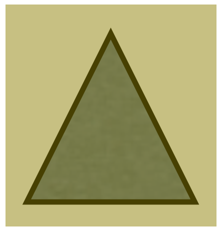
- Root node starts in axis-aligned bounding box (like kd-tree)
- Use the naive $O(n^2)$ SAH kd-tree build (no optimizations)
  - For each triangle, pick candidate splitting planes
  - Evaluate cost of using that splitting plane
    - Find surface areas of child nodes
    - Count number of triangles on each side of splitting plane
  - Split using the candidate splitting plane with lowest cost (SAH)

Build: splitting planes

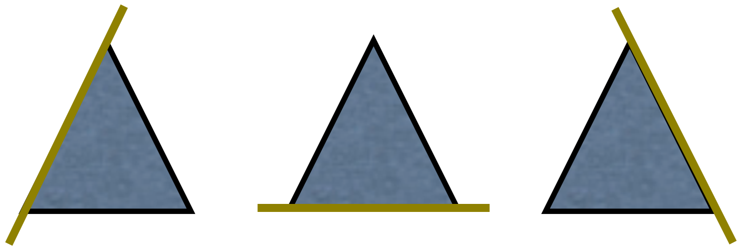
- Which splitting planes to try?
- Axis aligned planes used in a kd-tree



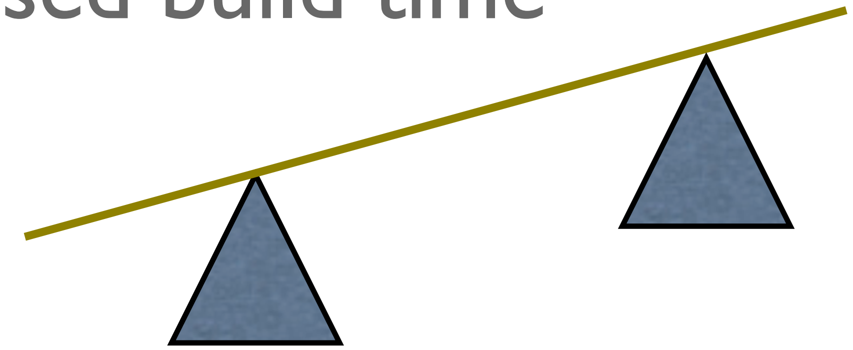
- Use triangle face as a plane



- Use triangle edges as planes



- Could do other planes at cost of increased build time



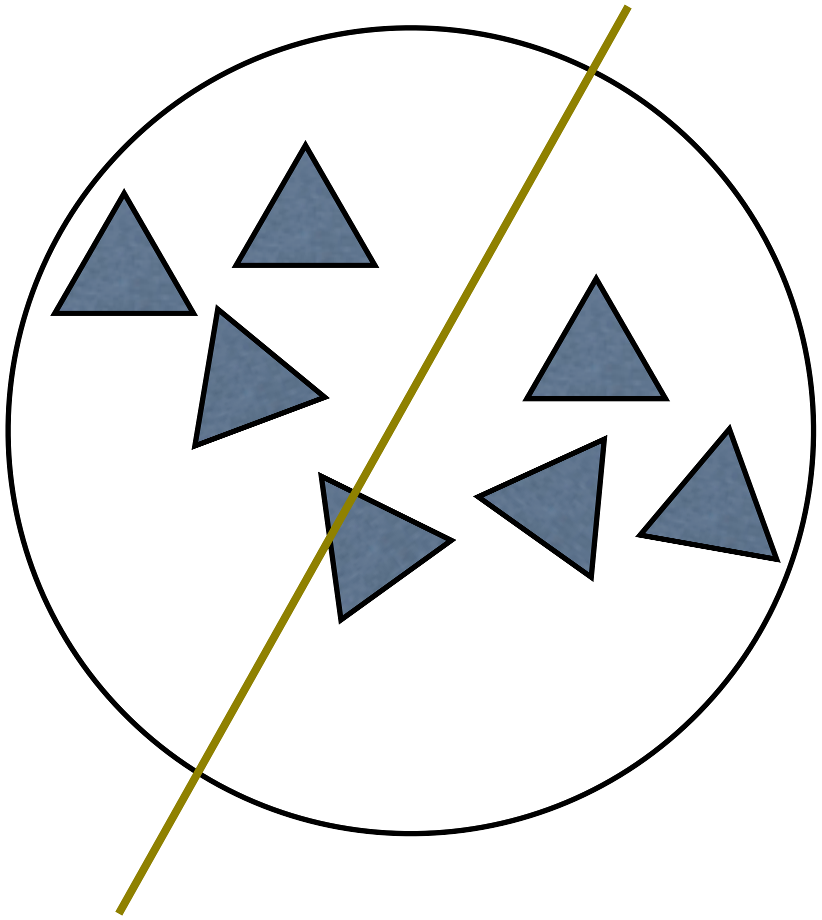
planes on triangle edge are made orthogonal to triangle face

Build: surface area

- Compute actual node geometry after splitting parent node
- Find area of each face of child node to get total area
- This is the expensive step

Build: counting triangles with a BVH

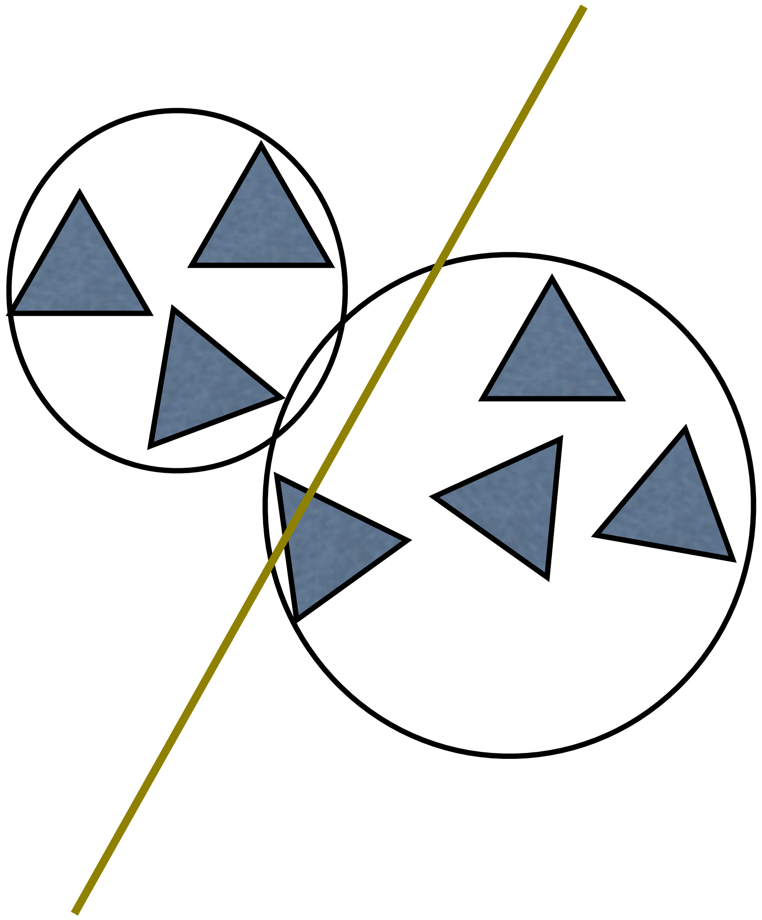
- Use a BVH to quickly count tri on each side of split
- Use spheres for bounding volume (BSH)
- Build BSH exactly like a standard axis aligned BVH
- Lowers complexity from $O(n^2)$ to probably $O(n \log^2 n)$



Counting is required by SAH
we use spheres for efficiency.
Refit after a split is performed (fast to do)

Build: counting triangles with a BVH

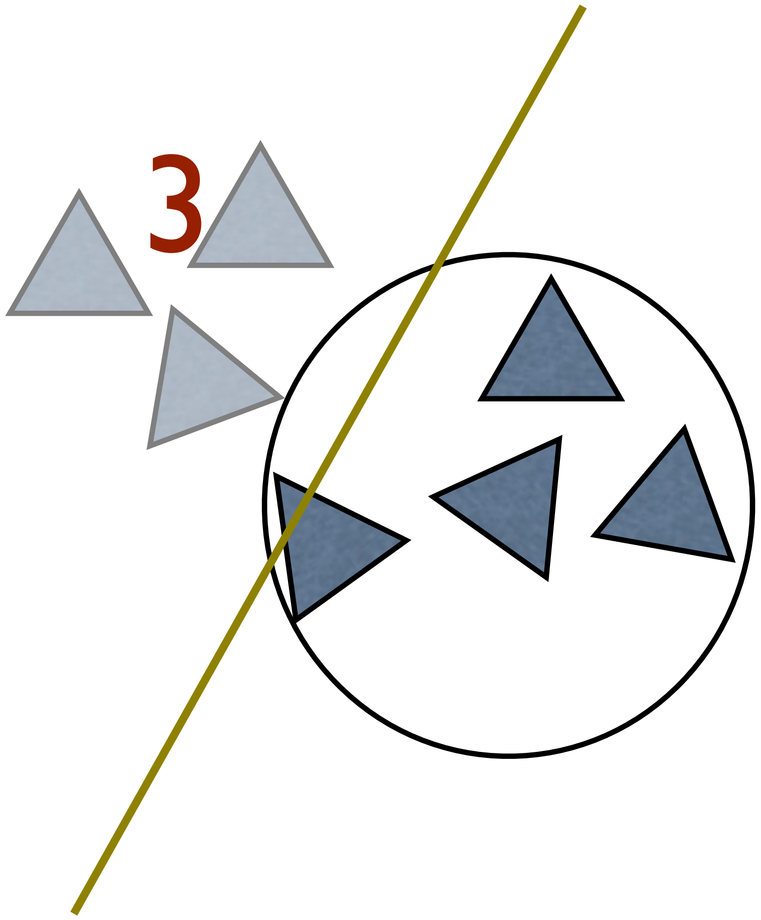
- Use a BVH to quickly count tri on each side of split
- Use spheres for bounding volume (BSH)
- Build BSH exactly like a standard axis aligned BVH
- Lowers complexity from $O(n^2)$ to probably $O(n \log^2 n)$



Counting is required by SAH
we use spheres for efficiency.
Refit after a split is performed (fast to do)

Build: counting triangles with a BVH

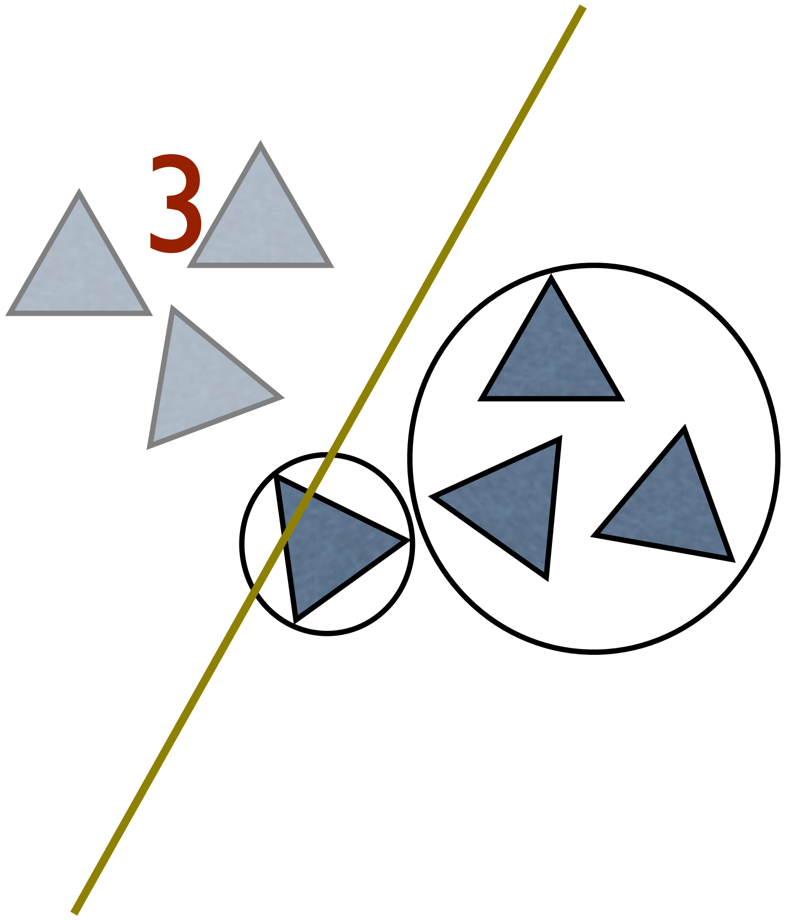
- Use a BVH to quickly count tri on each side of split
- Use spheres for bounding volume (BSH)
- Build BSH exactly like a standard axis aligned BVH
- Lowers complexity from $O(n^2)$ to probably $O(n \log^2 n)$



Counting is required by SAH
we use spheres for efficiency.
Refit after a split is performed (fast to do)

Build: counting triangles with a BVH

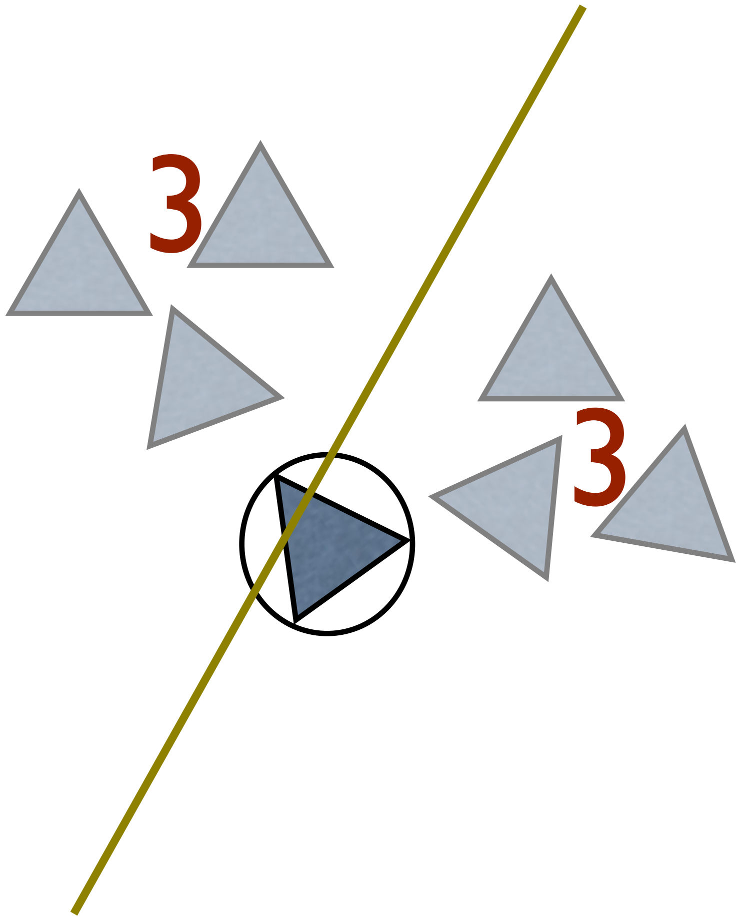
- Use a BVH to quickly count tri on each side of split
- Use spheres for bounding volume (BSH)
- Build BSH exactly like a standard axis aligned BVH
- Lowers complexity from $O(n^2)$ to probably $O(n \log^2 n)$



Counting is required by SAH
we use spheres for efficiency.
Refit after a split is performed (fast to do)

Build: counting triangles with a BVH

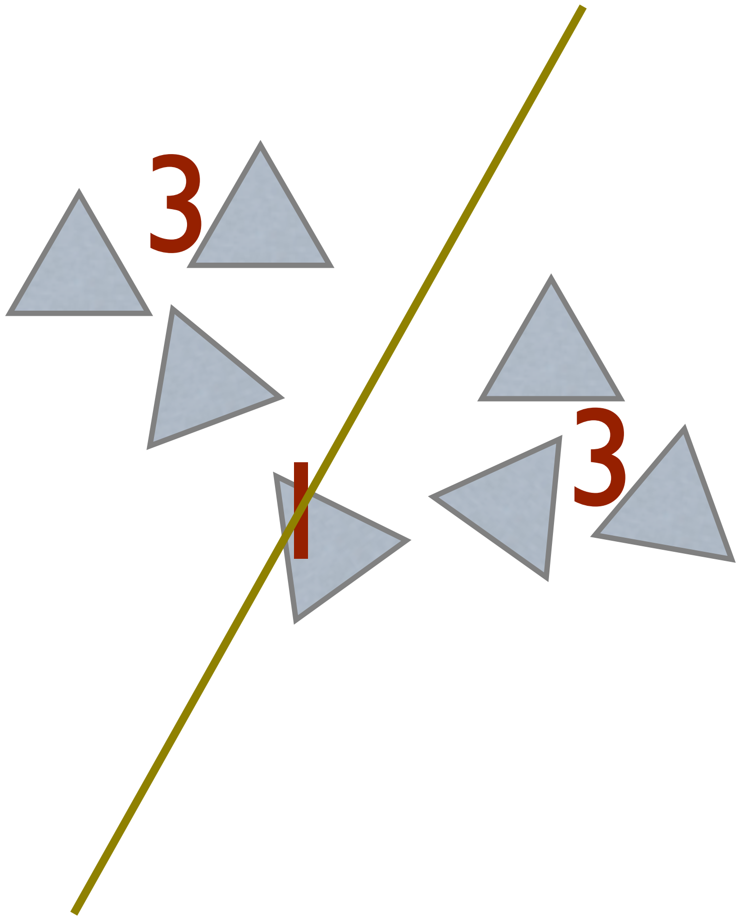
- Use a BVH to quickly count tri on each side of split
- Use spheres for bounding volume (BSH)
- Build BSH exactly like a standard axis aligned BVH
- Lowers complexity from $O(n^2)$ to probably $O(n \log^2 n)$



Counting is required by SAH
we use spheres for efficiency.
Refit after a split is performed (fast to do)

Build: counting triangles with a BVH

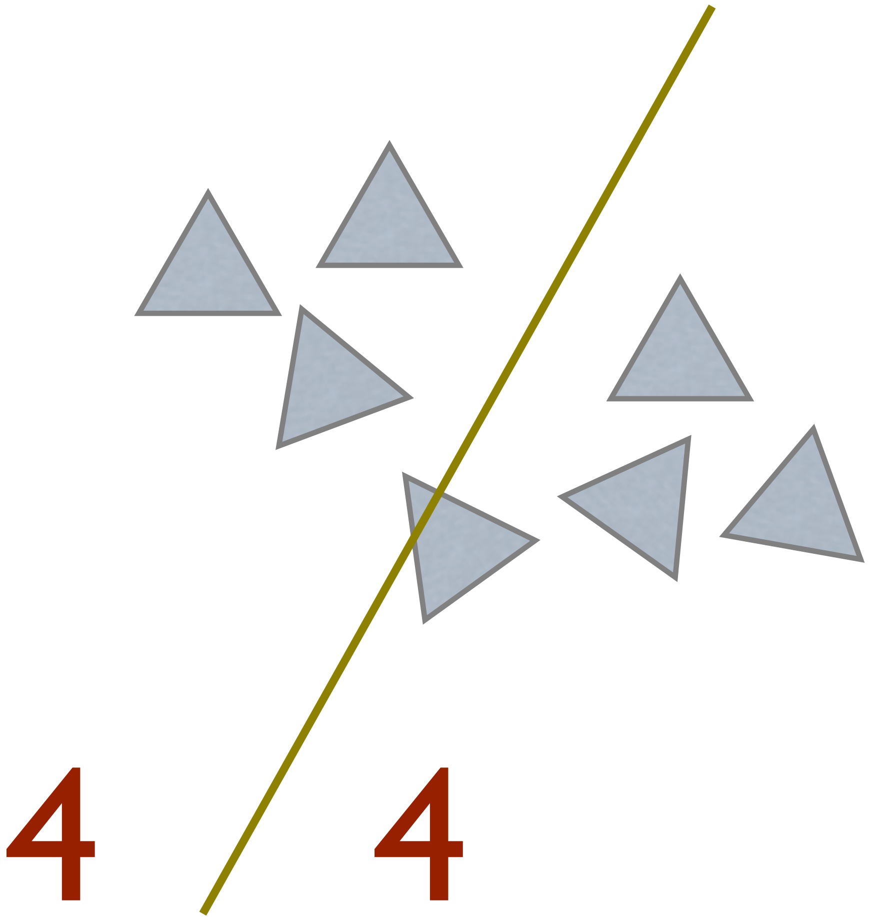
- Use a BVH to quickly count tri on each side of split
- Use spheres for bounding volume (BSH)
- Build BSH exactly like a standard axis aligned BVH
- Lowers complexity from $O(n^2)$ to probably $O(n \log^2 n)$



Counting is required by SAH
we use spheres for efficiency.
Refit after a split is performed (fast to do)

Build: counting triangles with a BVH

- Use a BVH to quickly count tri on each side of split
- Use spheres for bounding volume (BSH)
- Build BSH exactly like a standard axis aligned BVH
- Lowers complexity from $O(n^2)$ to probably $O(n \log^2 n)$



Counting is required by SAH
we use spheres for efficiency.
Refit after a split is performed (fast to do)

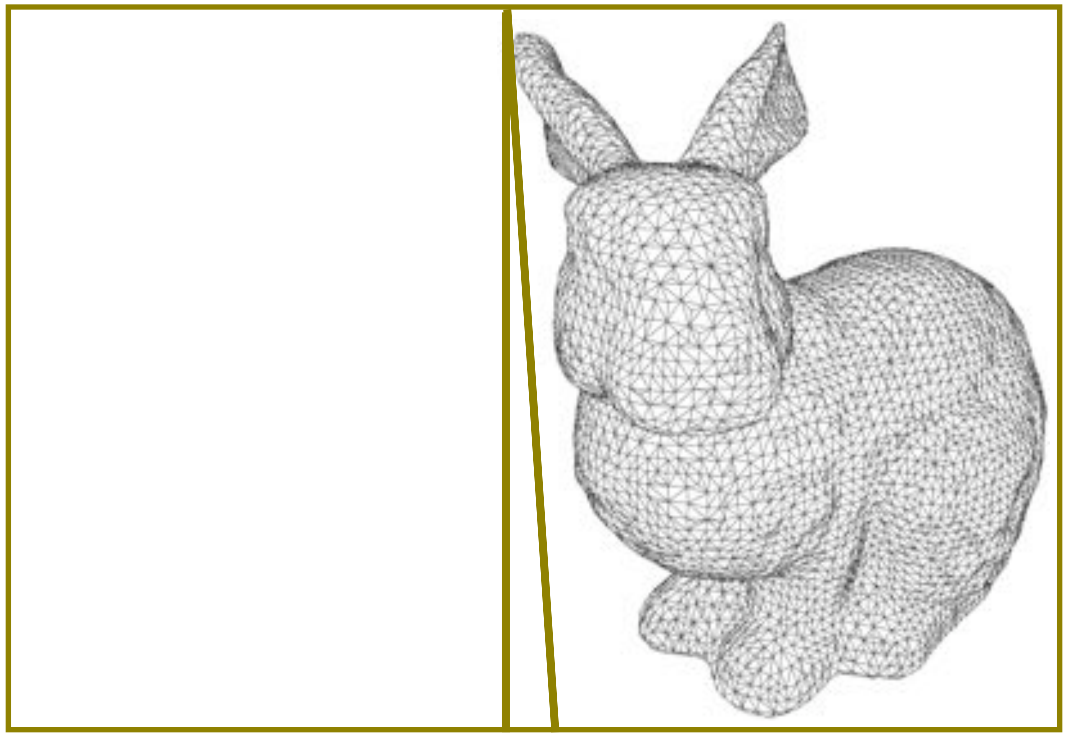
Build: SAH

$$\mathrm{Cost} = P_l n_l C_{\mathrm{tri}} + P_r n_r C_{\mathrm{tri}} + C_{\mathrm{traversal}}$$

probabilities come from ratio of child SA to parent SA

Kd-trees are still good

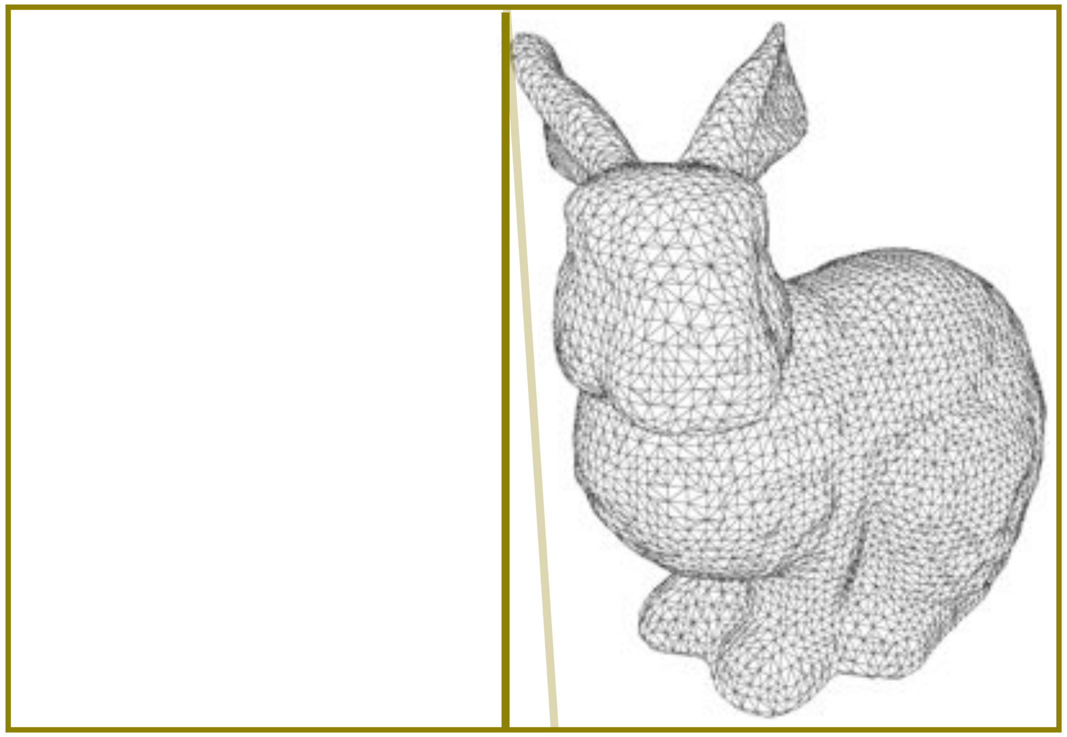
- kd-tree traversals are cheaper than BSP traversals.
- kd-tree splitting planes can sometimes be almost as high quality as a general BSP split
- Cheaper traversal + slightly increased cost of rendering node might make kd-tree split better than the BSP split
- Use SAH to determine which type of split is cheaper
- Could just directly use a C_{BSP} and a $C_{\text{kd-tree}}$ in SAH



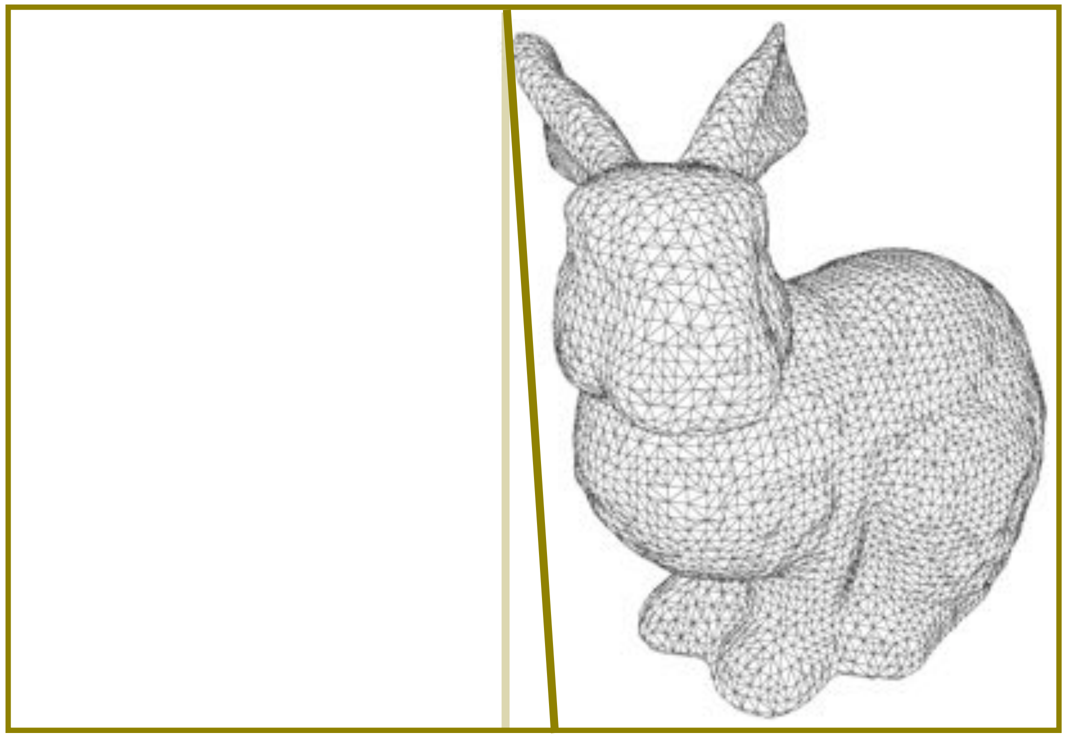
Build: combined SAH

- So what value of C_{BSP} and $C_{\text{kd-tree}}$ should we use?

kd-tree



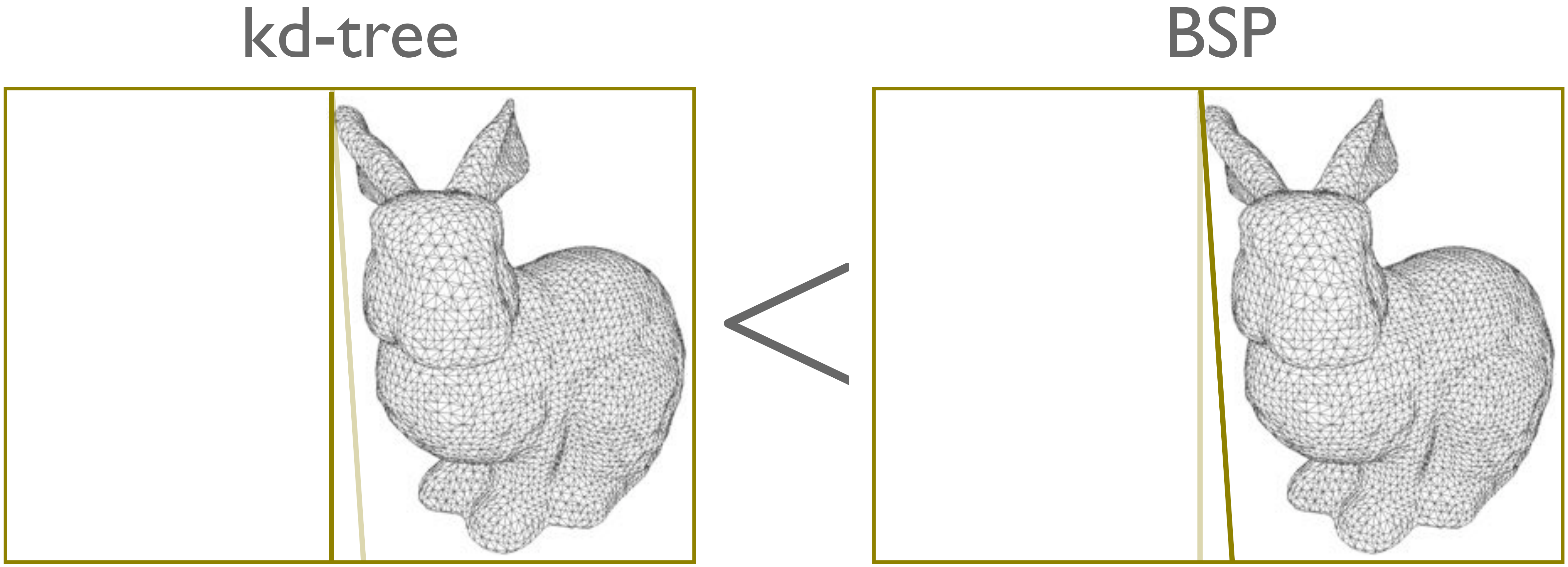
BSP



kd-tree splits are likely to be most useful near root of tree because there are lots of triangles and very accurate splits are not needed.
Near root of trees this SAH will almost never pick the kd-tree split.

Build: combined SAH

- So what value of C_{BSP} and $C_{\text{kd-tree}}$ should we use?



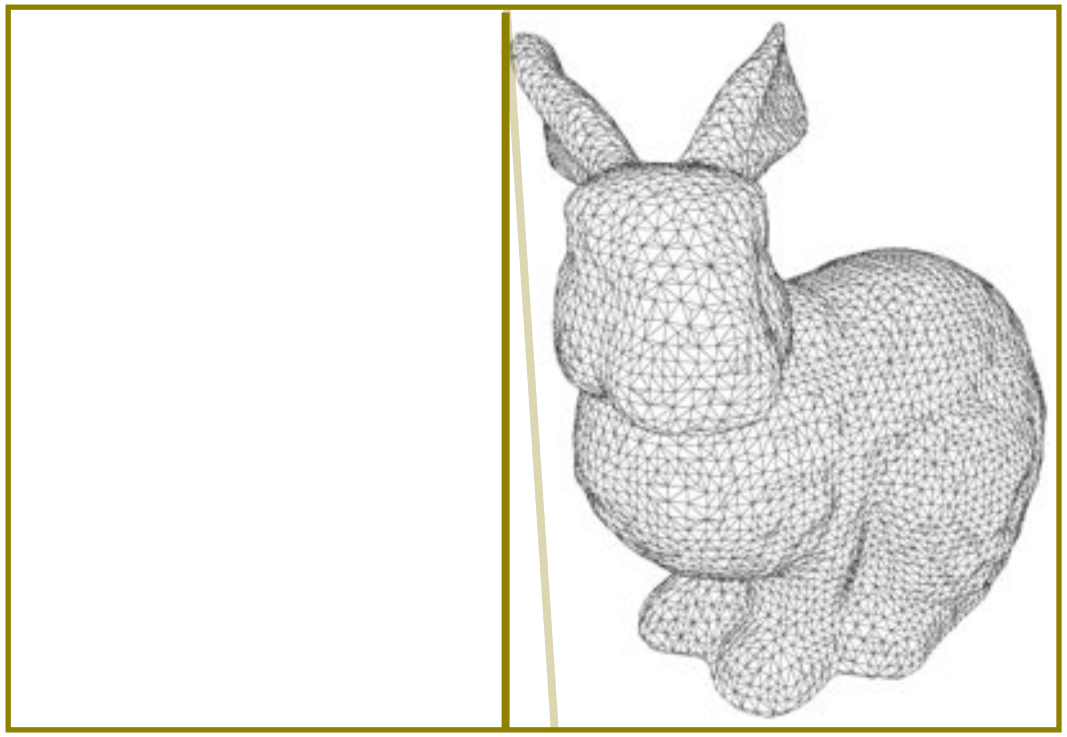
kd-tree $<$ BSP

kd-tree splits are likely to be most useful near root of tree because there are lots of triangles and very accurate splits are not needed.
Near root of trees this SAH will almost never pick the kd-tree split.

Build: combined SAH

- So what value of C_{BSP} and $C_{\text{kd-tree}}$ should we use?

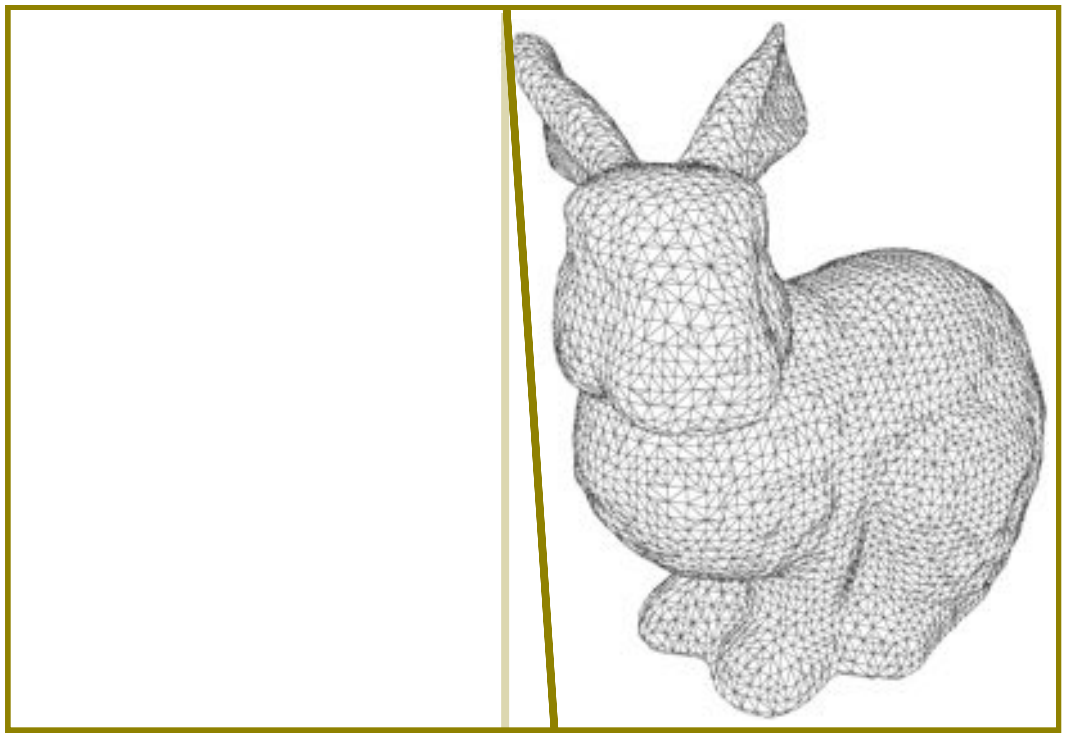
kd-tree



$0.75 \cdot 0C_{\mathrm{tri}} + 0.75 \cdot 10000C_{\mathrm{tri}} + C_{\text{kd-tree}}$

$<$

BSP

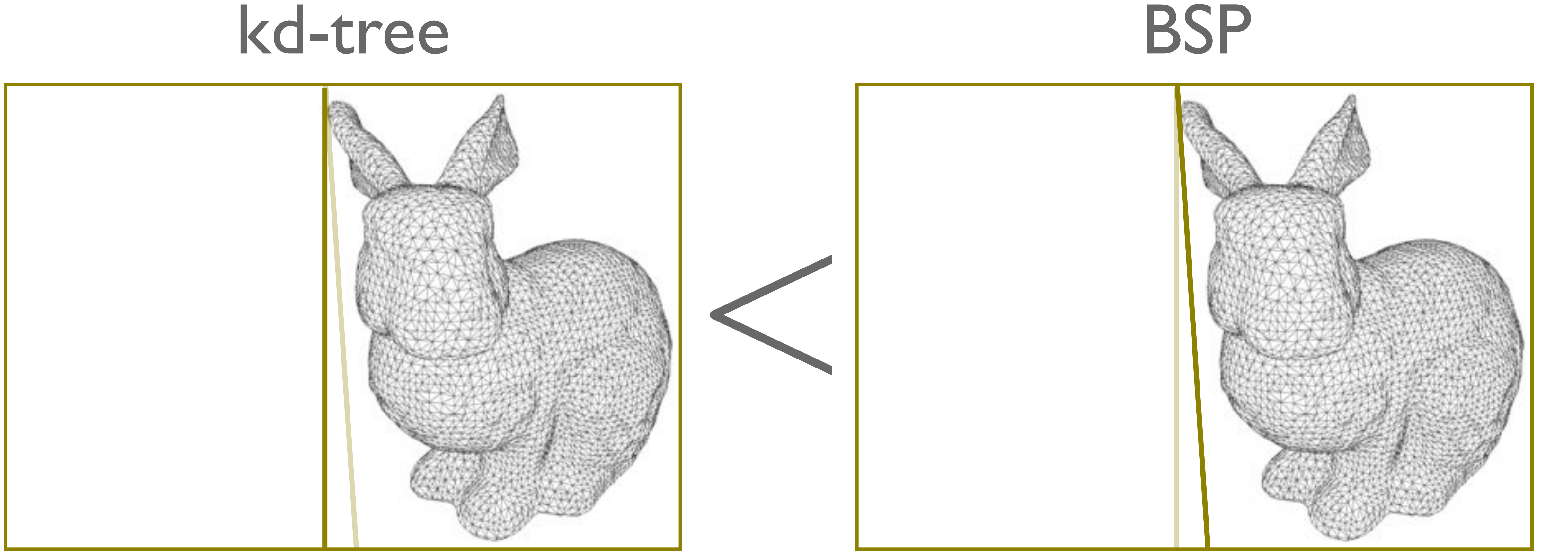


$0.76 \cdot 0C_{\mathrm{tri}} + 0.74 \cdot 10000C_{\mathrm{tri}} + C_{\mathrm{BSP}}$

kd-tree splits are likely to be most useful near root of tree because there are lots of triangles and very accurate splits are not needed.
Near root of trees this SAH will almost never pick the kd-tree split.

Build: combined SAH

- So what value of C_{BSP} and $C_{\text{kd-tree}}$ should we use?



$$0.75 \cdot 0C_{\text{tri}} + 0.75 \cdot 10000C_{\text{tri}} + C_{\text{kd-tree}}$$
$$7500C_{\text{tri}} + C_{\text{kd-tree}}$$

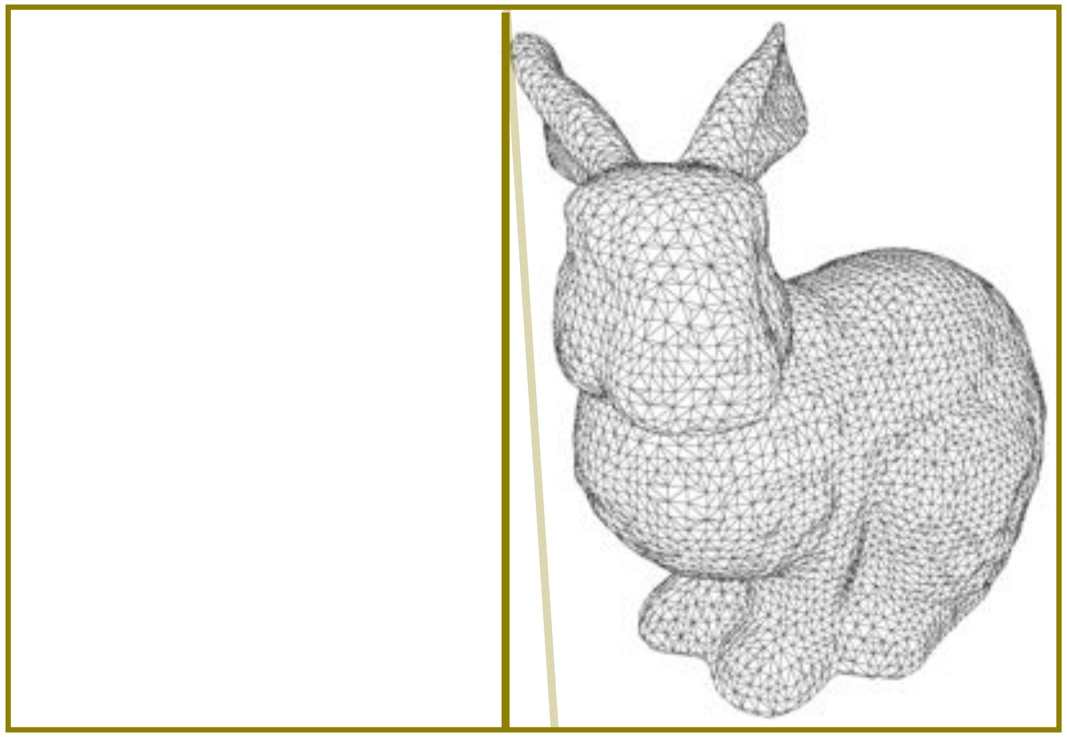
$$0.76 \cdot 0C_{\text{tri}} + 0.74 \cdot 10000C_{\text{tri}} + C_{\text{BSP}}$$
$$7400C_{\text{tri}} + C_{\text{BSP}}$$

kd-tree splits are likely to be most useful near root of tree because there are lots of triangles and very accurate splits are not needed.
Near root of trees this SAH will almost never pick the kd-tree split.

Build: combined SAH

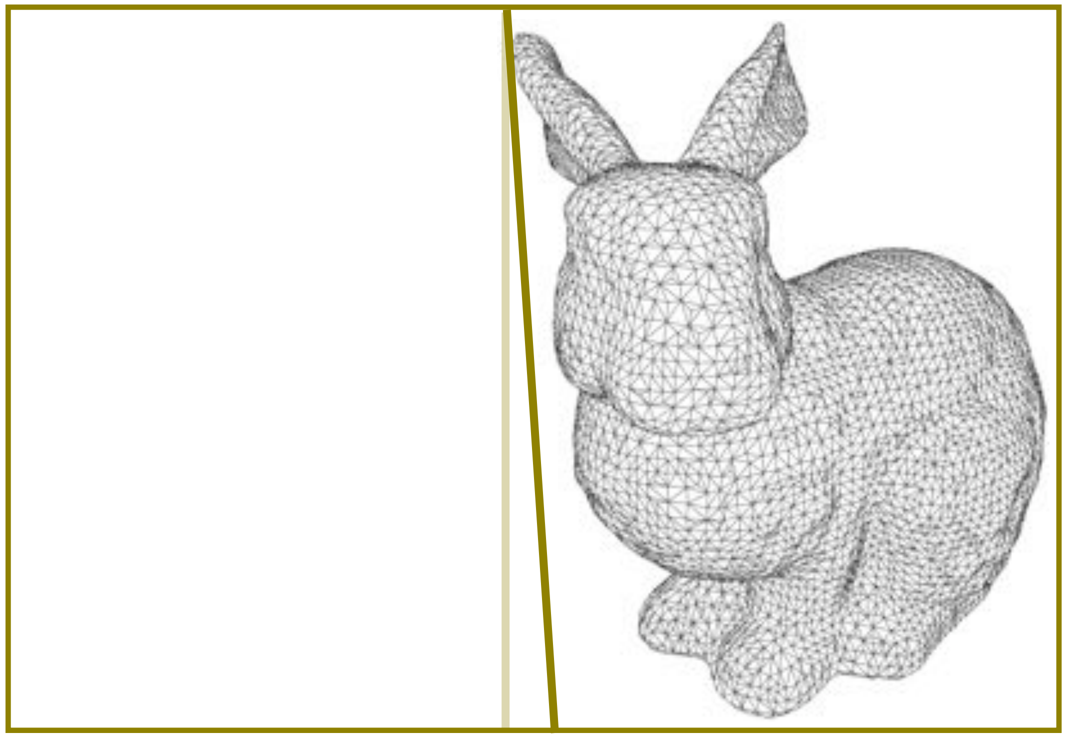
- So what value of C_{BSP} and $C_{\text{kd-tree}}$ should we use?

kd-tree



$<$

BSP



$$\begin{aligned} 0.75 \cdot 0C_{\text{tri}} + 0.75 \cdot 10000C_{\text{tri}} + C_{\text{kd-tree}} \\ 7500C_{\text{tri}} + C_{\text{kd-tree}} \end{aligned}$$

$$\begin{aligned} 0.76 \cdot 0C_{\text{tri}} + 0.74 \cdot 10000C_{\text{tri}} + C_{\text{BSP}} \\ 7400C_{\text{tri}} + C_{\text{BSP}} \end{aligned}$$

$$7500C_{\text{tri}} + C_{\text{kd-tree}} < 7400C_{\text{tri}} + C_{\text{BSP}}$$

$$100C_{\text{tri}} + C_{\text{kd-tree}} < C_{\text{BSP}}$$

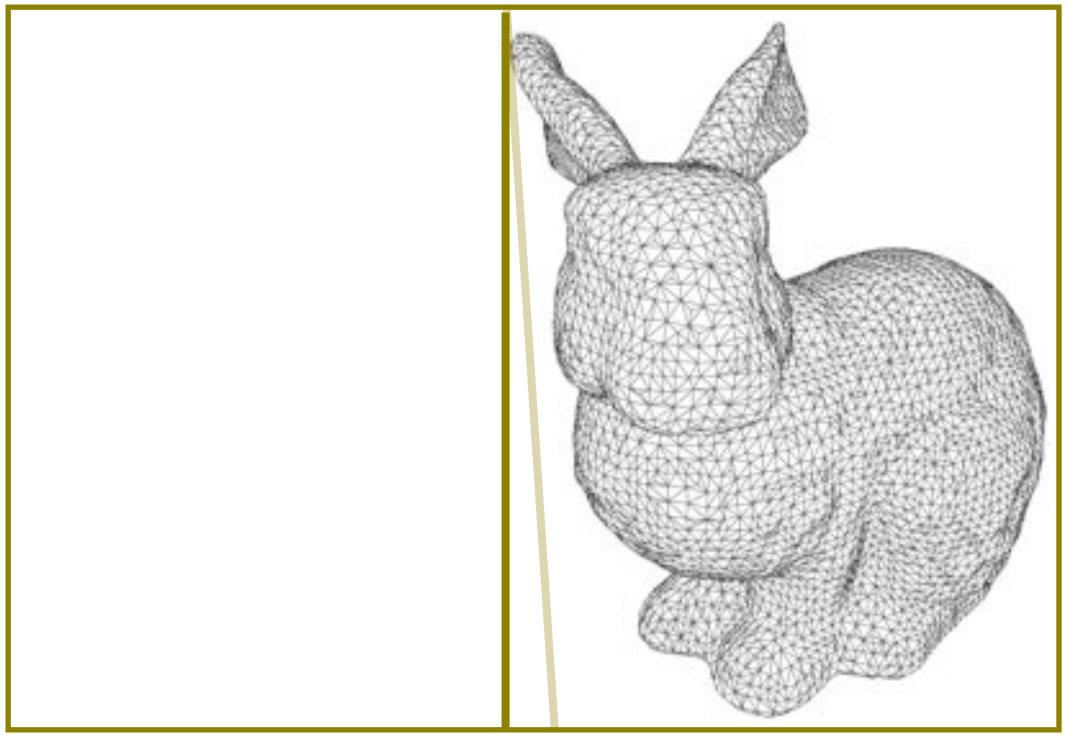
kd-tree splits are likely to be most useful near root of tree because there are lots of triangles and very accurate splits are not needed.

Near root of trees this SAH will almost never pick the kd-tree split.

Build: combined SAH

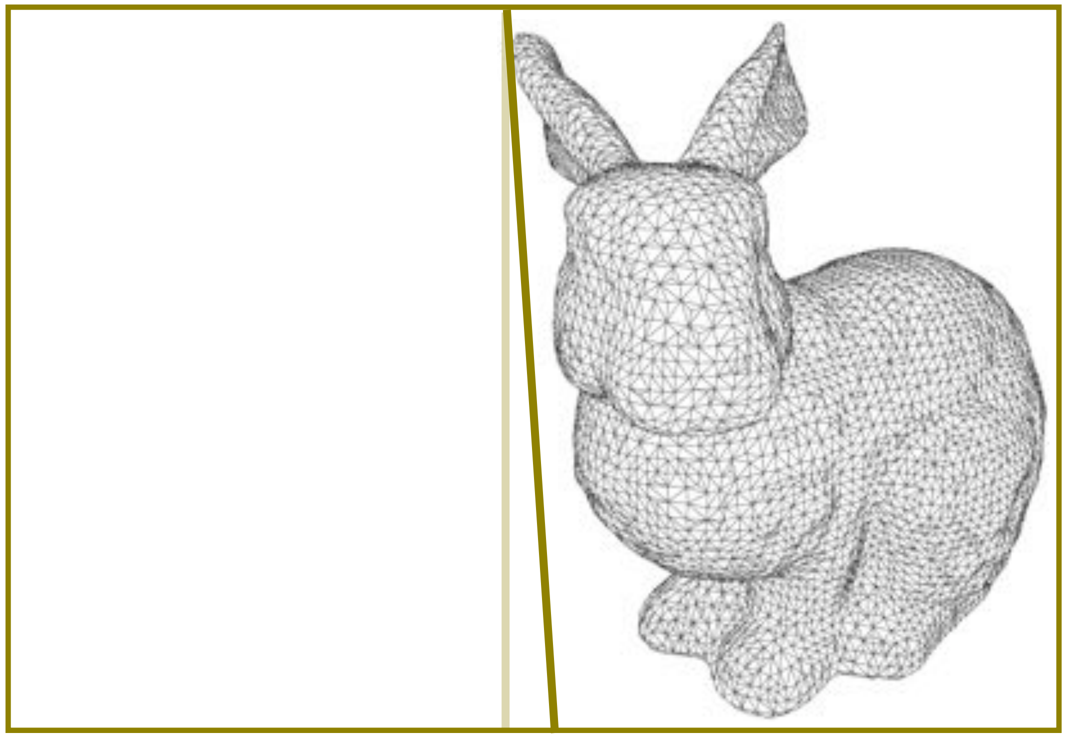
- So what value of C_{BSP} and $C_{\text{kd-tree}}$ should we use?

kd-tree



$<$

BSP



$$0.75 \cdot 0C_{\text{tri}} + 0.75 \cdot 10000C_{\text{tri}} + C_{\text{kd-tree}}$$
$$7500C_{\text{tri}} + C_{\text{kd-tree}}$$

$$0.76 \cdot 0C_{\text{tri}} + 0.74 \cdot 10000C_{\text{tri}} + C_{\text{BSP}}$$
$$7400C_{\text{tri}} + C_{\text{BSP}}$$

$$7500C_{\text{tri}} + C_{\text{kd-tree}} < 7400C_{\text{tri}} + C_{\text{BSP}}$$
$$100C_{\text{tri}} + C_{\text{kd-tree}} < C_{\text{BSP}}$$

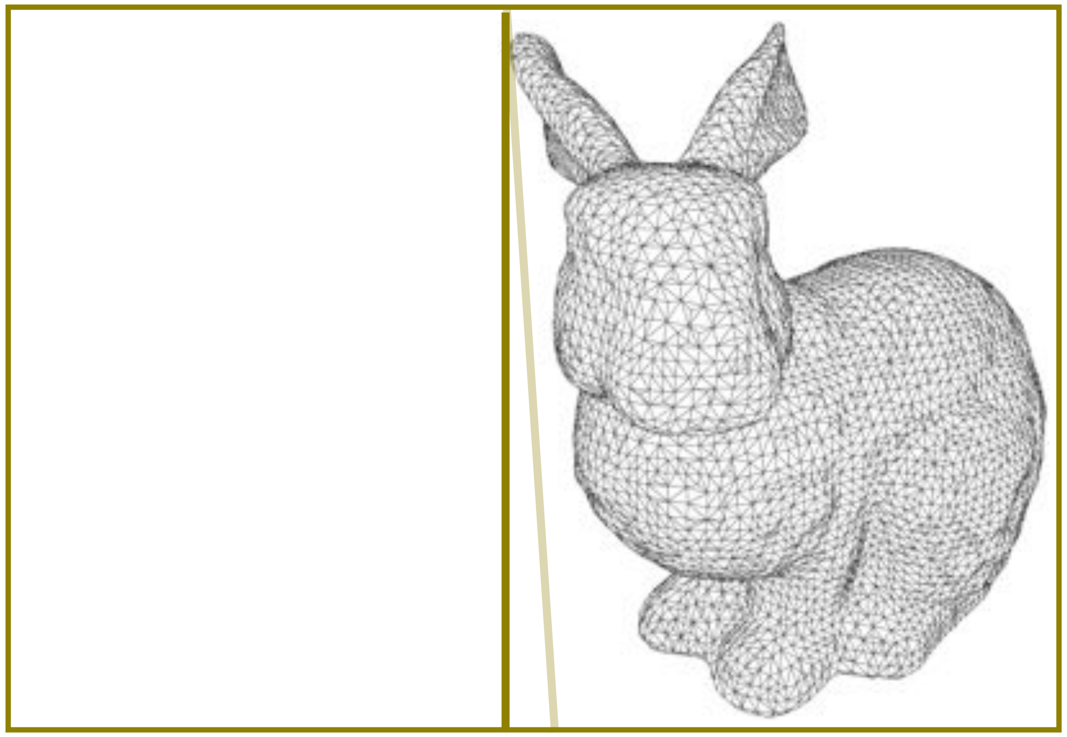
$$.01nC_{\text{tri}} + C_{\text{kd-tree}} < C_{\text{BSP}}$$

kd-tree splits are likely to be most useful near root of tree because there are lots of triangles and very accurate splits are not needed.
Near root of trees this SAH will almost never pick the kd-tree split.

Build: combined SAH

- So what value of C_{BSP} and $C_{\text{kd-tree}}$ should we use?

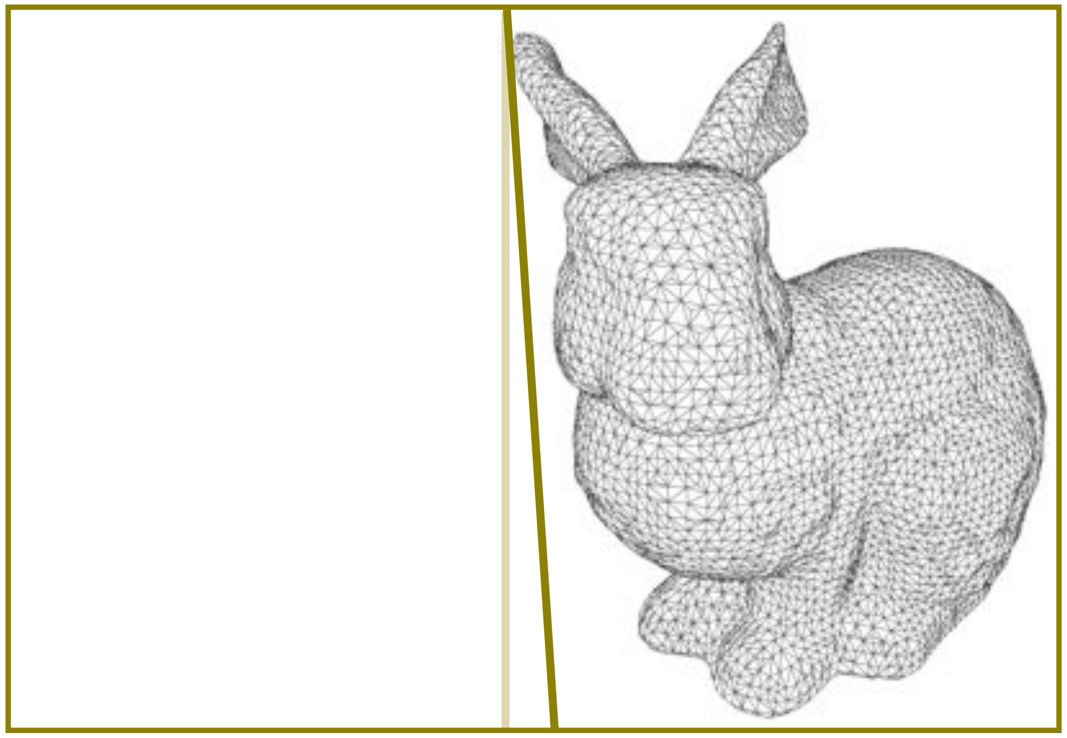
kd-tree



$$0.75 \cdot 0C_{\text{tri}} + 0.75 \cdot 10000C_{\text{tri}} + C_{\text{kd-tree}}$$
$$7500C_{\text{tri}} + C_{\text{kd-tree}}$$

$<$

BSP



$$0.76 \cdot 0C_{\text{tri}} + 0.74 \cdot 10000C_{\text{tri}} + C_{\text{BSP}}$$
$$7400C_{\text{tri}} + C_{\text{BSP}}$$

$$7500C_{\text{tri}} + C_{\text{kd-tree}} < 7400C_{\text{tri}} + C_{\text{BSP}}$$
$$100C_{\text{tri}} + C_{\text{kd-tree}} < C_{\text{BSP}}$$
$$.01nC_{\text{tri}} + C_{\text{kd-tree}} < C_{\text{BSP}}$$

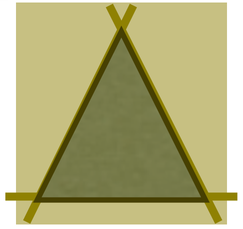
C_{BSP} varies linearly with number of triangles

kd-tree splits are likely to be most useful near root of tree because there are lots of triangles and very accurate splits are not needed.
Near root of trees this SAH will almost never pick the kd-tree split.

Traversal

- Identical to kd-tree except:
- Distance to plane computation more expensive
- Epsilon test when comparing distance
- Optimize by still using kd-tree traversal for axis aligned splits

Traversal based triangle intersection

- BSP can tightly contain a triangle 
- If ray hits a leaf node with a tightly contained triangle, triangle must be hit
- Ray distance already computed during traversal, so we know where triangle is hit
- % time spent in triangle intersection for BSP is already low, so this offers only modest speedups (more interesting from academic perspective)

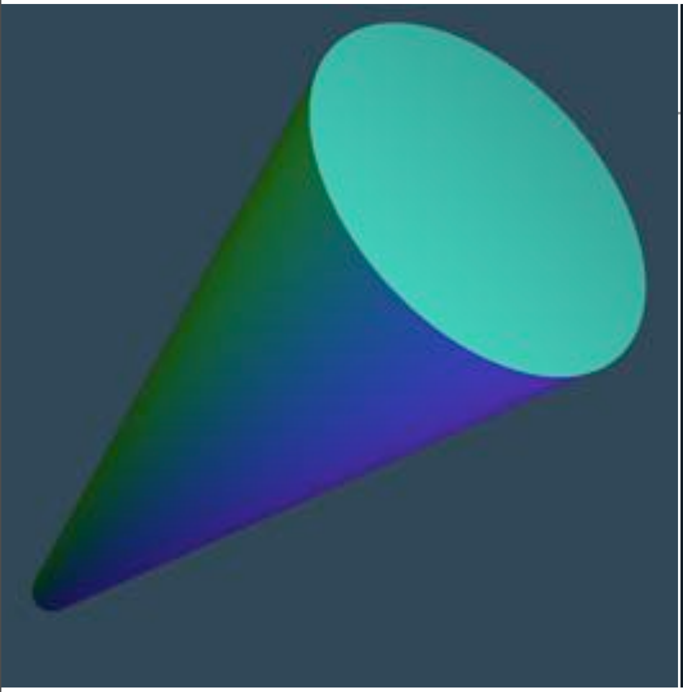
Results

- Ray cast
  - Test against very coherent rays
  - Use ray packets, SSE, etc...
- Path trace
  - Progressively more incoherent with each bounce
  - Single ray traversal

Results

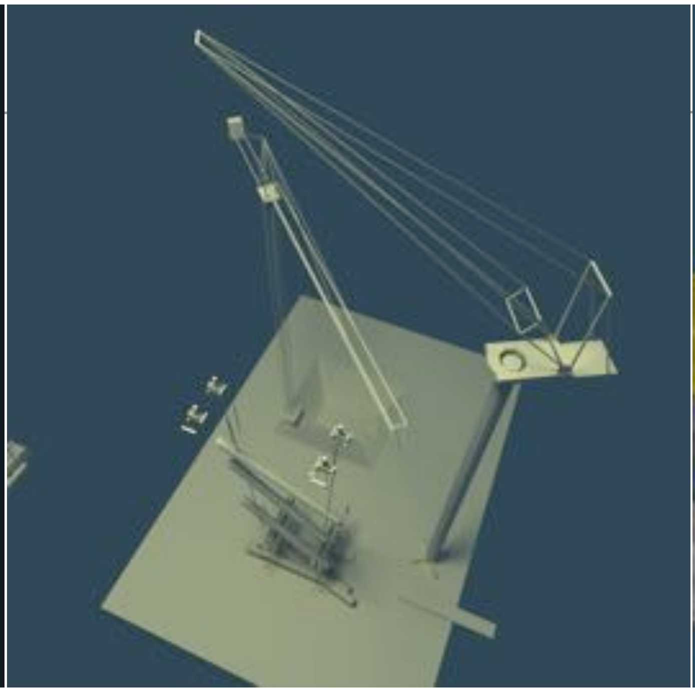
- Compare against a highly optimized kd-tree
  - Packetized SSE for ray casting
  - Single ray for path tracing
  - SAH with perfect splits

Results: Build times

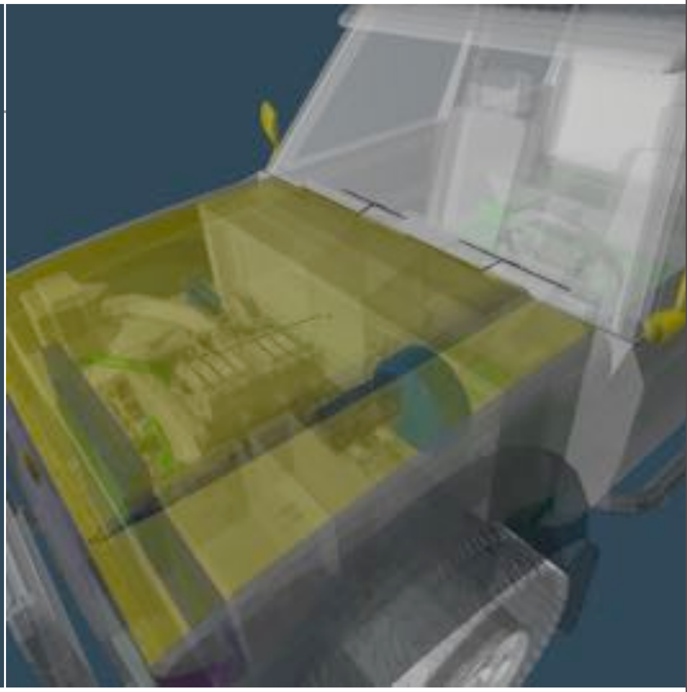


9sec 65tri/s
596 tri

19min 60tri/s
69K tri



36min 56tri/s
122K tri



65min 47tri/s
183K tri

112min 42tri/s
283K tri

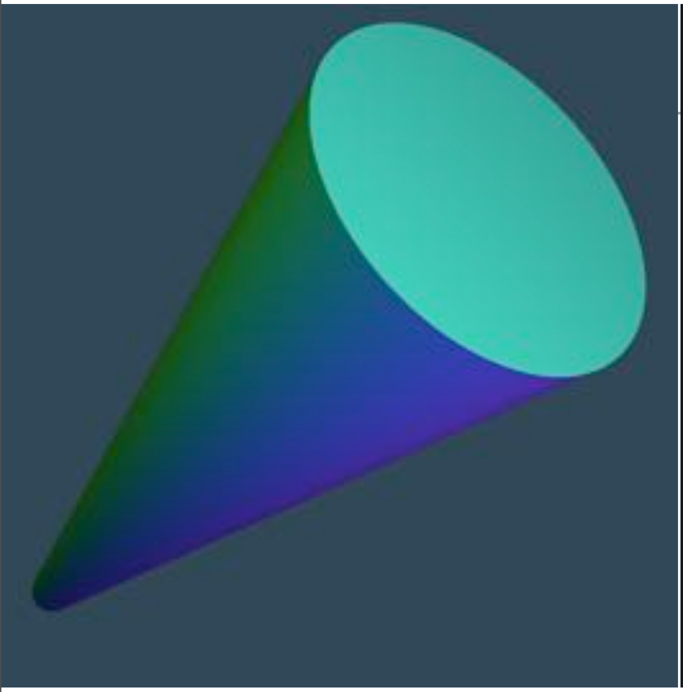
23.6h 25tri/s
2.14M tri

23.6h 25tri/s
2.14M tri

23.6h 25tri/s
2.14M tri

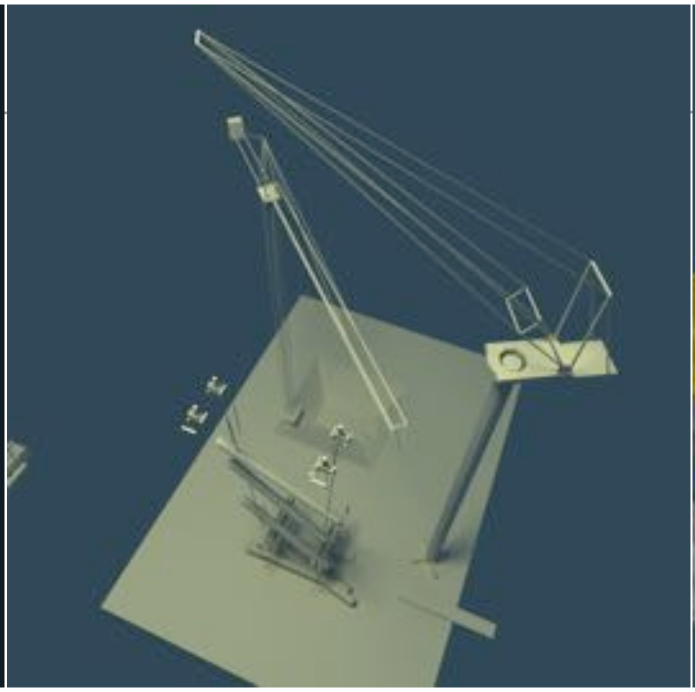
slow build, but ok as a *preprocess*
clearly less than O(n^2)
kd-tree about 100x faster (kd-tree sodahall 6 minutes)

Results: Path traced/raycasted improvements

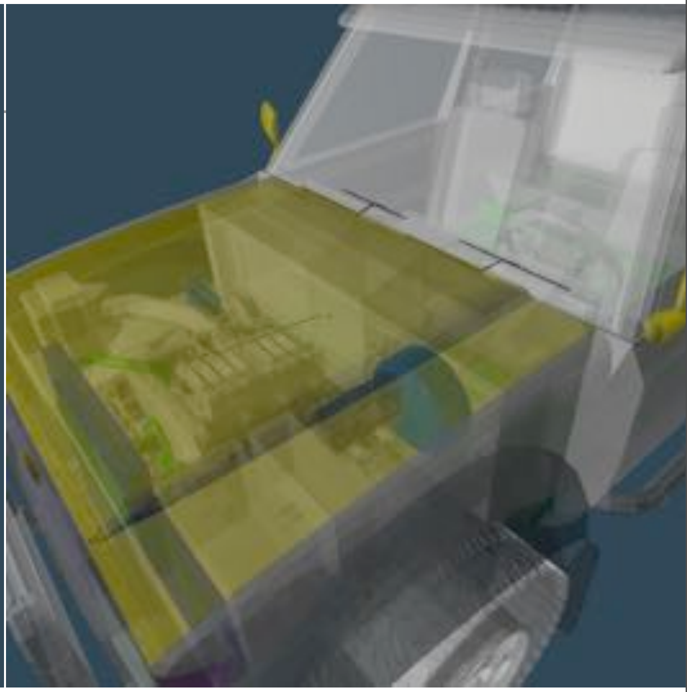


~/26x

1.2x/1.2x



2.4x/3.3x



~/1.3x

1.1x/1.1x

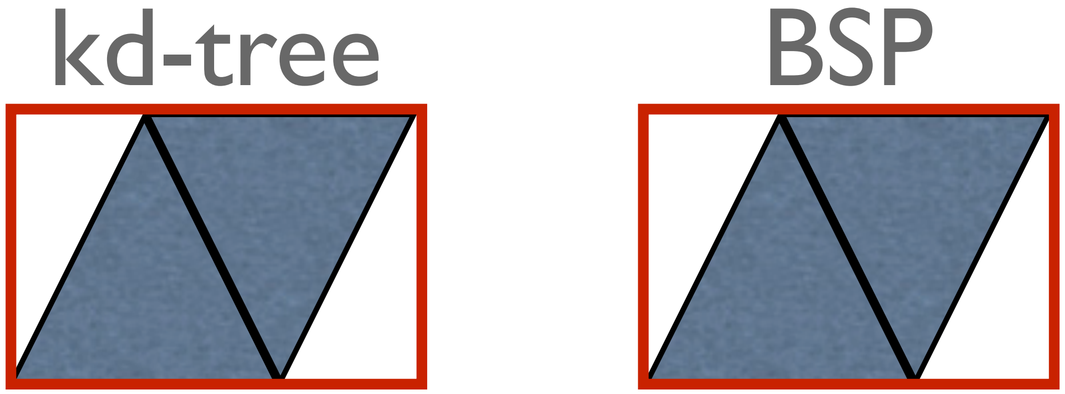
2.5x/2.4x

1.3x/1.8x

1.2x/1.2x

Results: Why is it faster?

- Time spent traversing actually went slightly up
  - A BSP traversal is more expensive than a kd-tree traversal
  - More node traversals performed

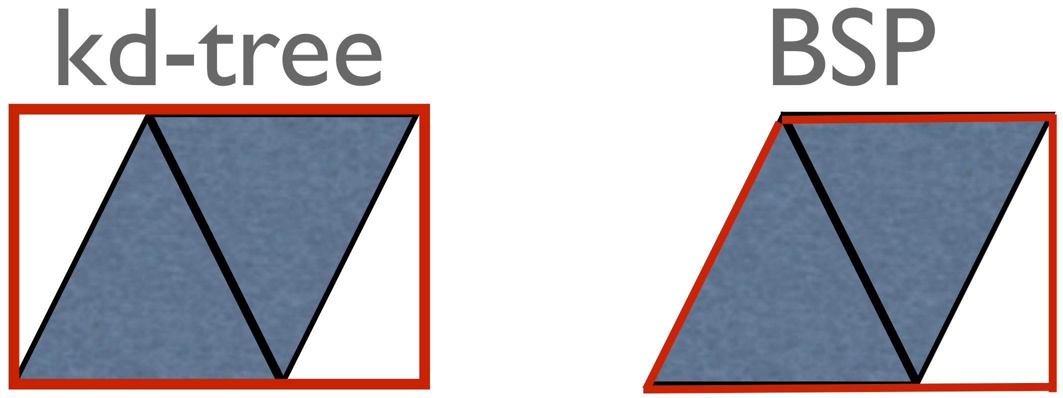


- Time spent intersecting triangles went down (2x-50x)
  - Able to better handle "complicated" geometry
  - Most leaf nodes refined down to only 1 triangle

Example reduces # of tri intersections for that root node by 3x
More node traversals == less triangle intersections.

Results: Why is it faster?

- Time spent traversing actually went slightly up
  - A BSP traversal is more expensive than a kd-tree traversal
  - More node traversals performed

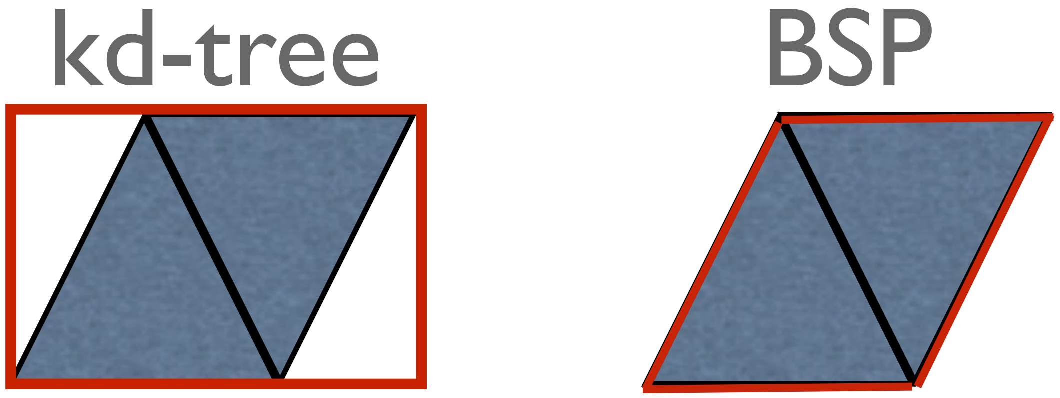


- Time spent intersecting triangles went down (2x-50x)
  - Able to better handle "complicated" geometry
  - Most leaf nodes refined down to only 1 triangle

Example reduces # of tri intersections for that root node by 3x
More node traversals == less triangle intersections.

Results: Why is it faster?

- Time spent traversing actually went slightly up
  - A BSP traversal is more expensive than a kd-tree traversal
  - More node traversals performed

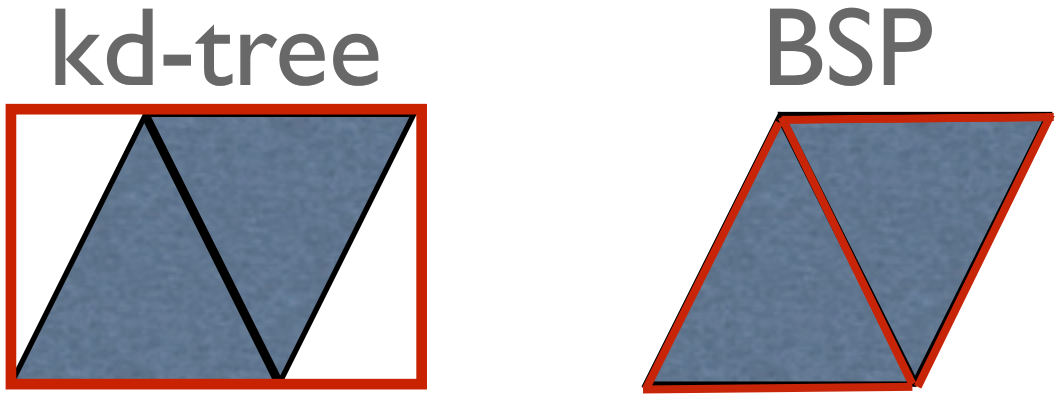


- Time spent intersecting triangles went down (2x-50x)
  - Able to better handle "complicated" geometry
  - Most leaf nodes refined down to only 1 triangle

Example reduces # of tri intersections for that root node by 3x
More node traversals == less triangle intersections.

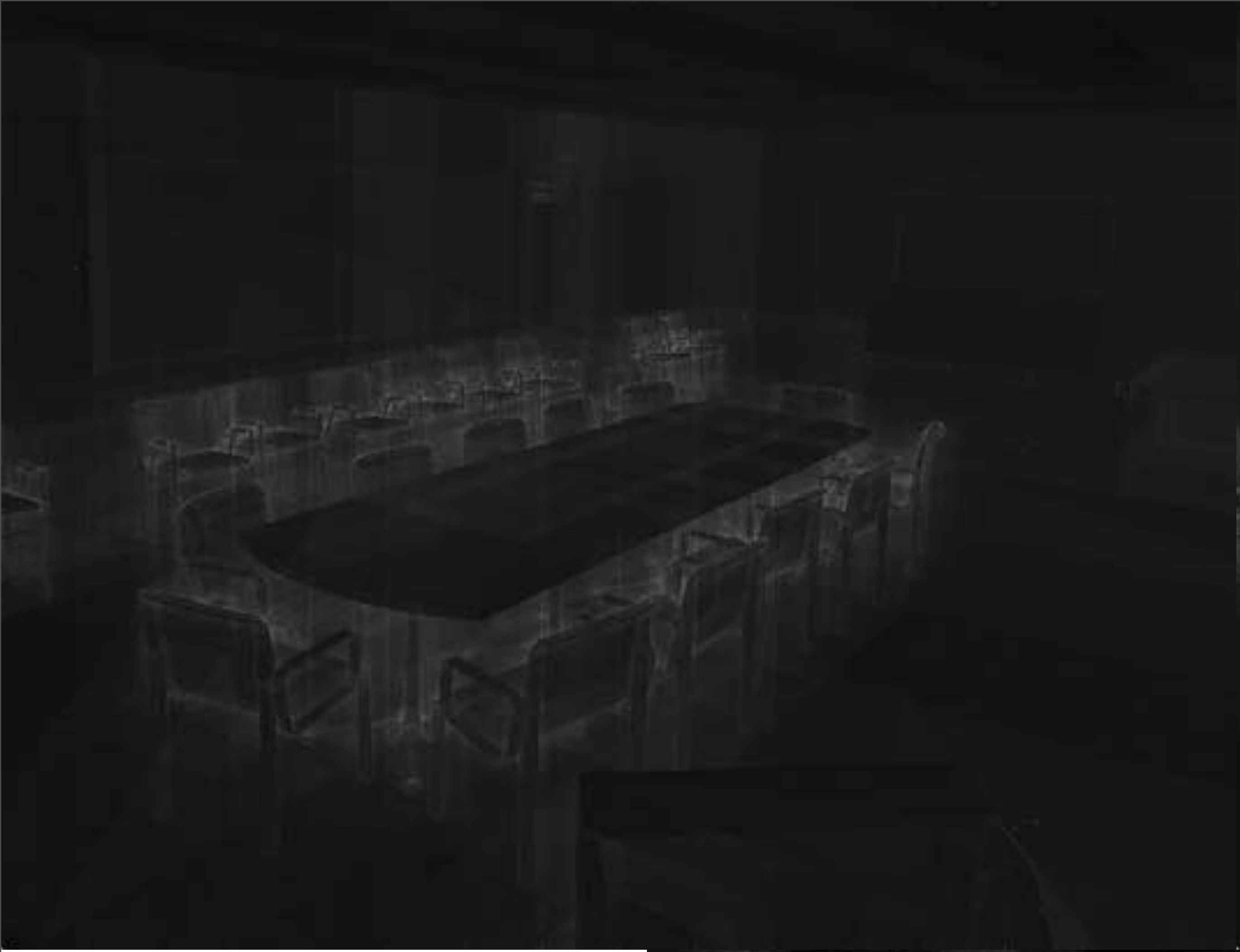
Results: Why is it faster?

- Time spent traversing actually went slightly up
    - A BSP traversal is more expensive than a kd-tree traversal
    - More node traversals performed

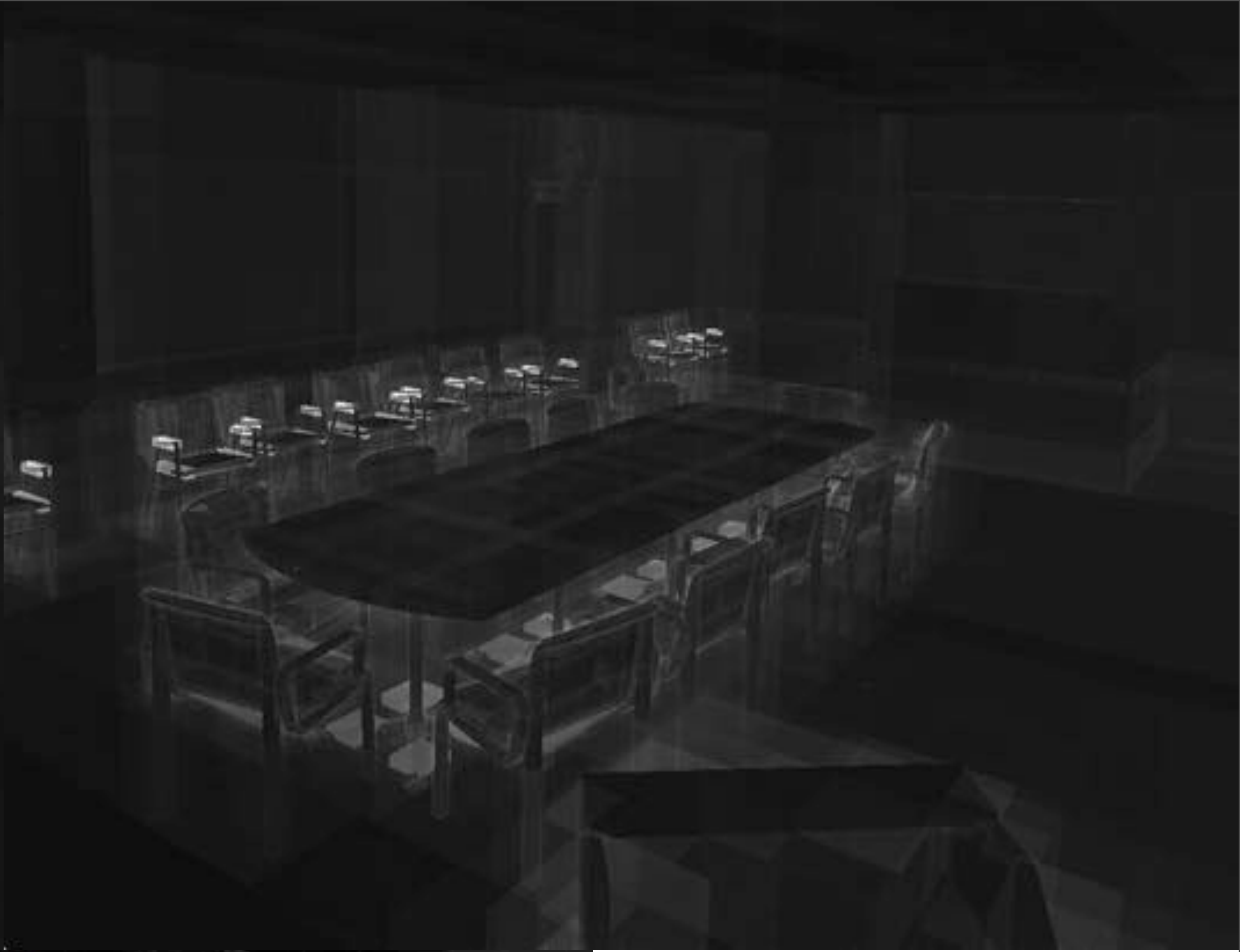


- Time spent intersecting triangles went down (2x-50x)
    - Able to better handle "complicated" geometry
    - Most leaf nodes refined down to only 1 triangle

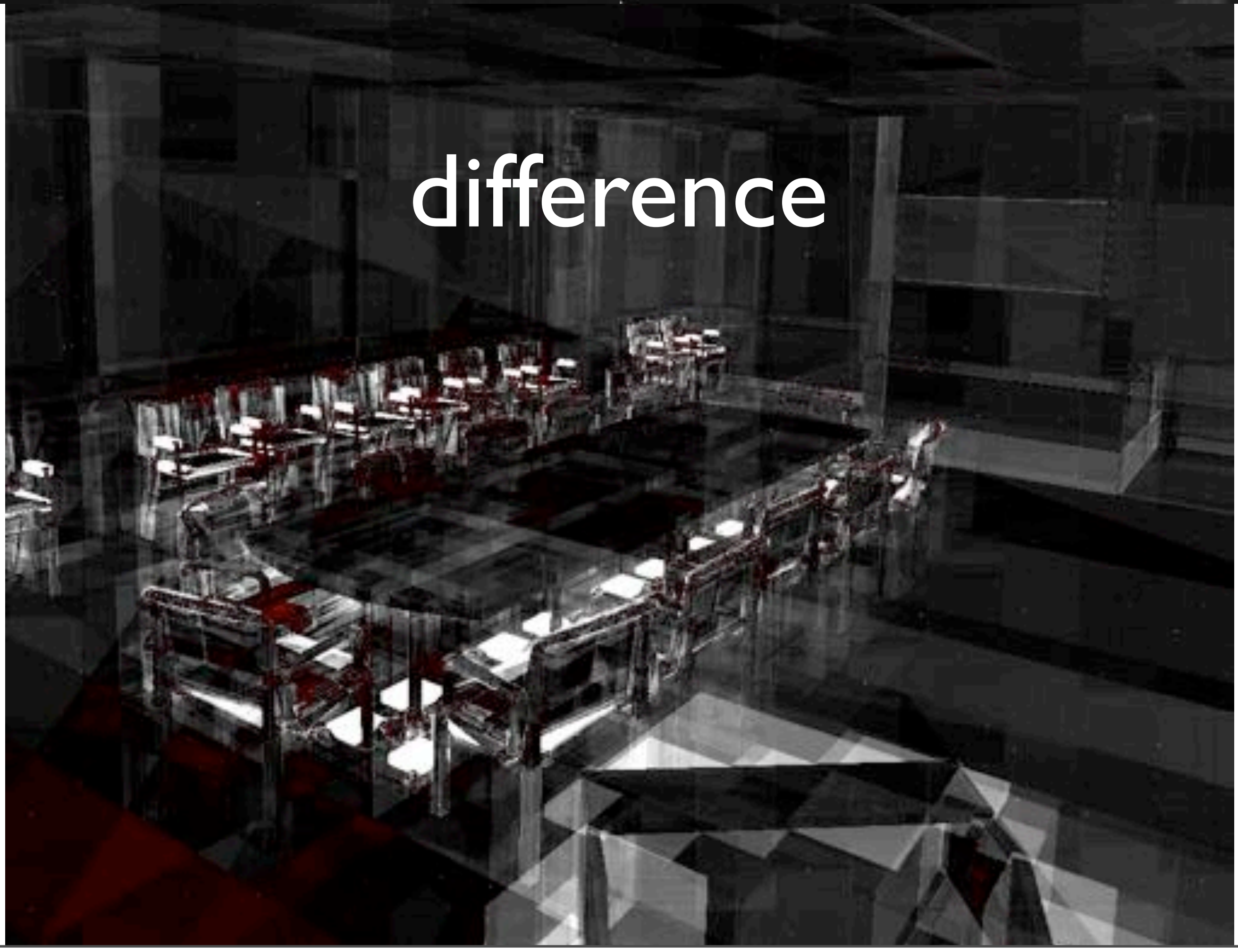
Example reduces # of tri intersections for that root node by 3x
More node traversals == less triangle intersections.



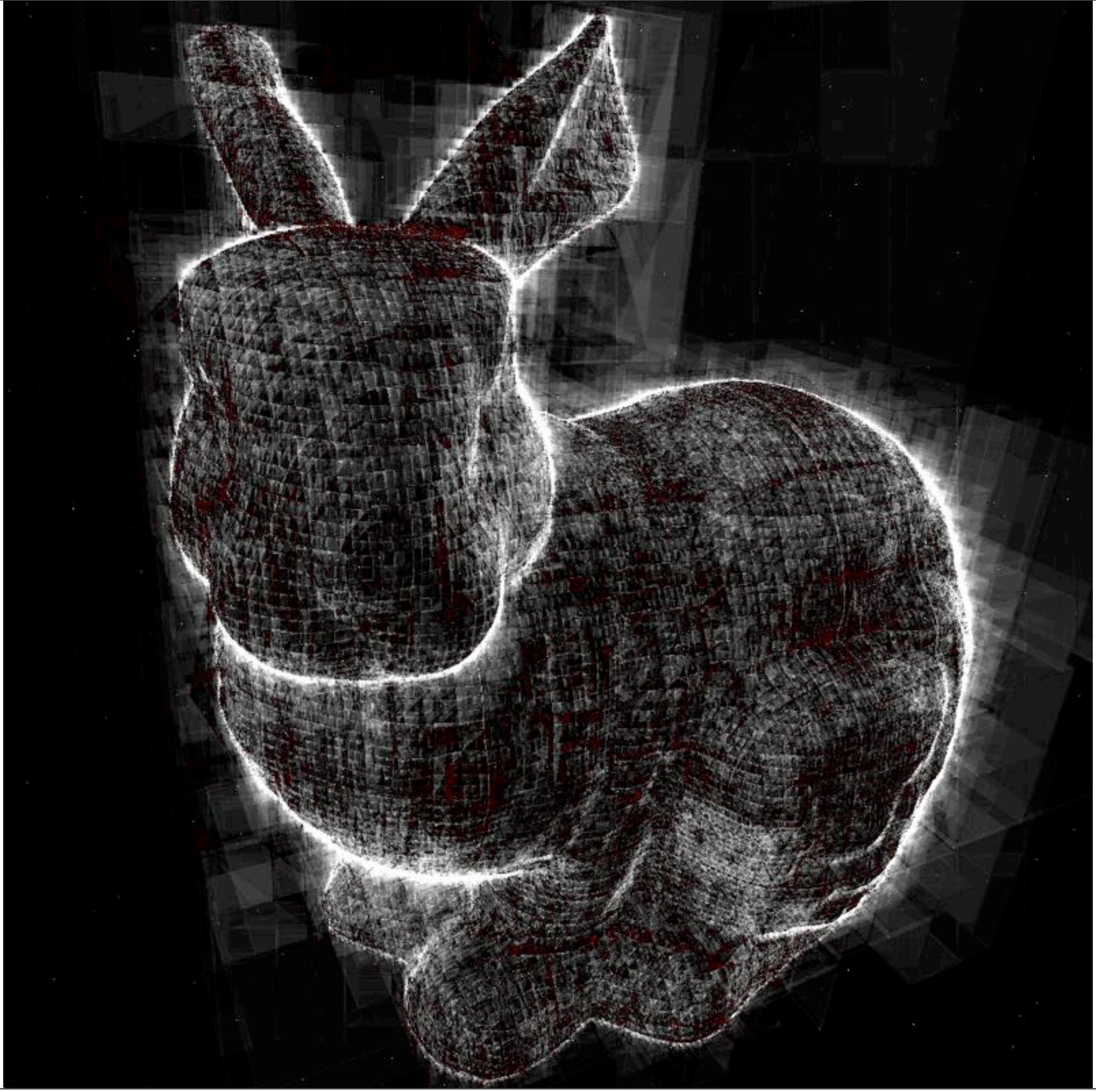
BSP



kd-tree



pixel intensity == time spent rendering pixel
1.1x

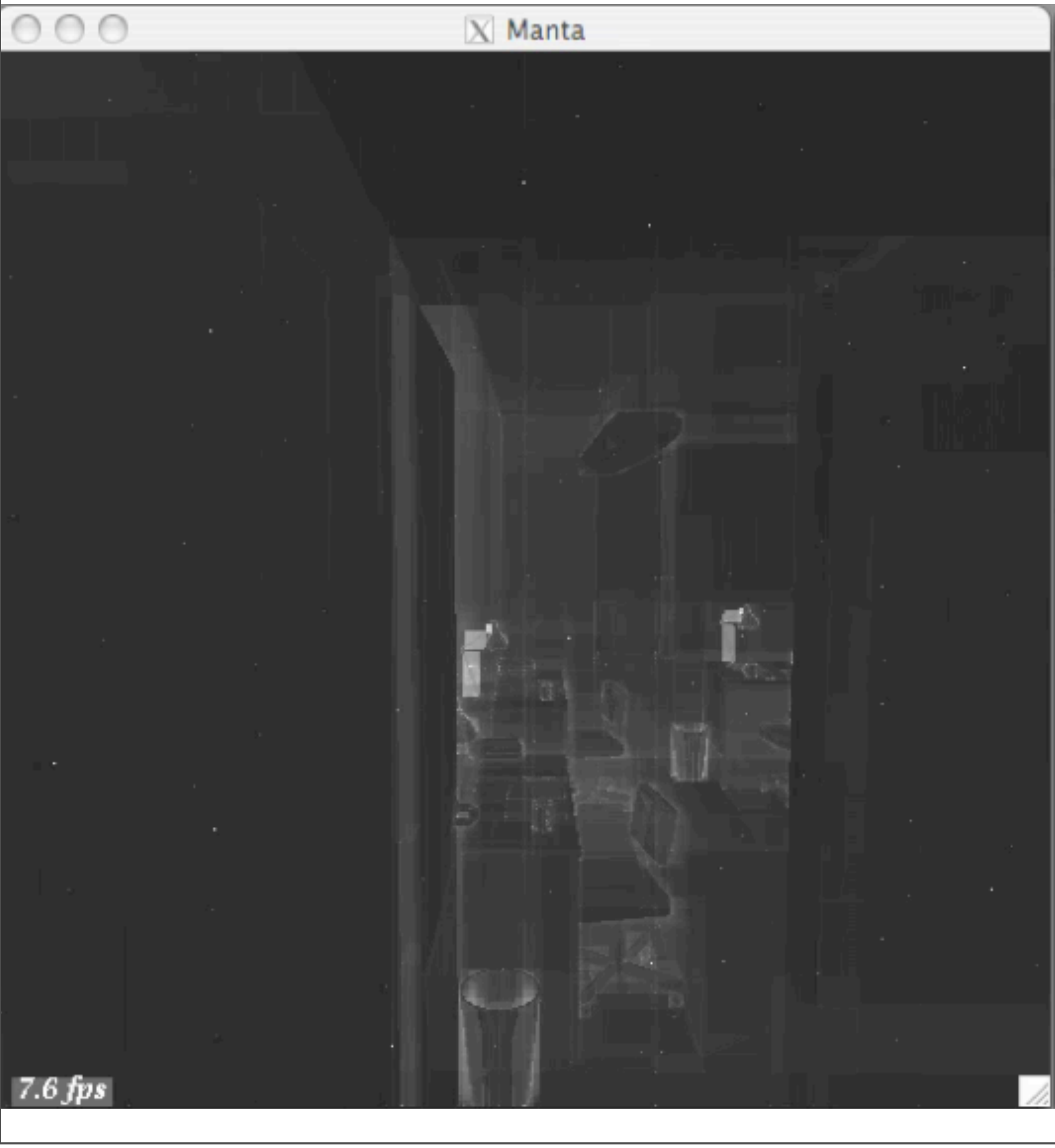


1.2x

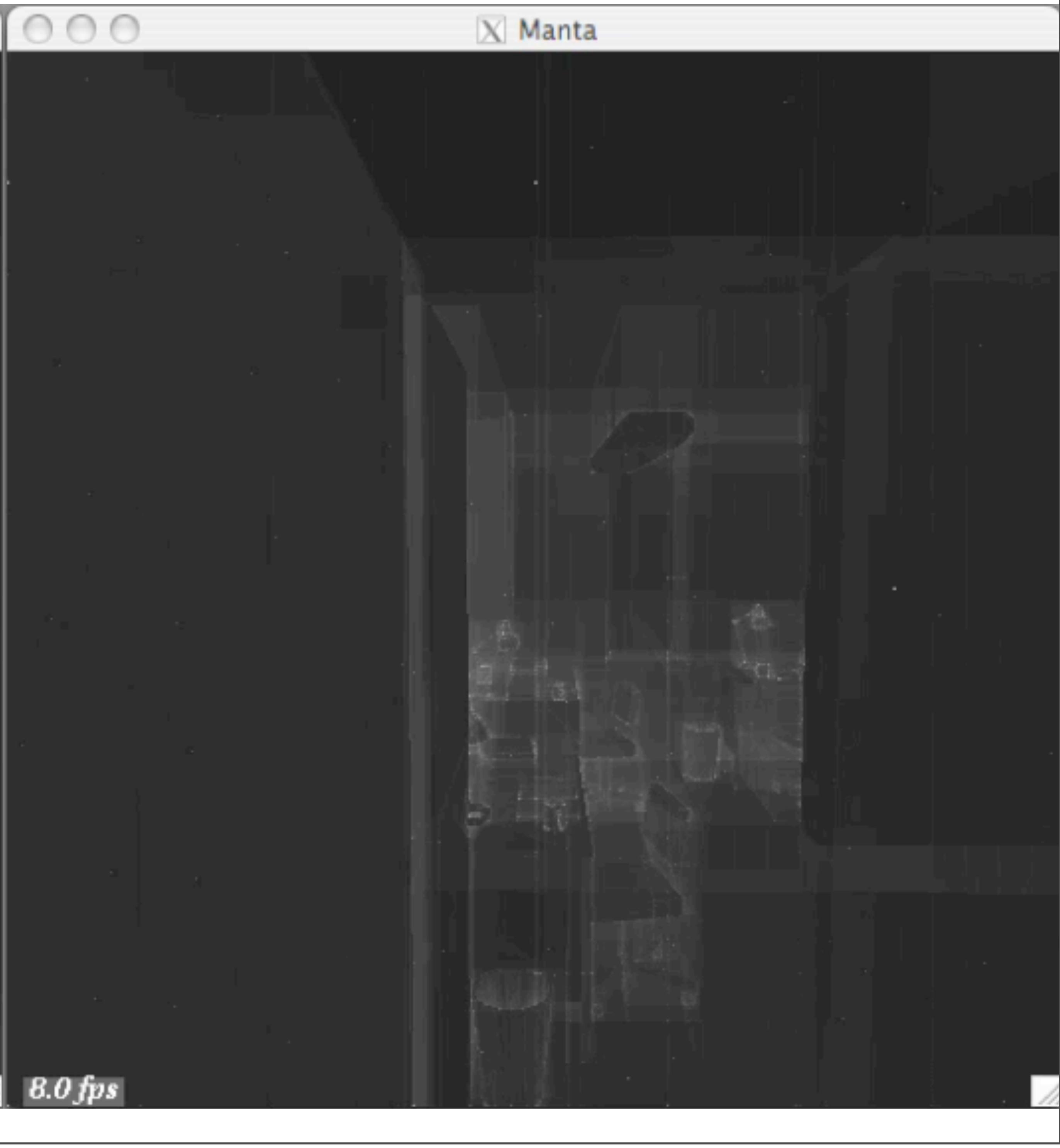
1.2x

Performance robustness

kd-tree



BSP



```
bin/mantar -model ~/work/DynRT-SIGGRAPH/Models/sodahall.iw -as BSP -load ~/data/BSPs/sodahall.bbsp -noload ~/data/KDTree/sodahall.kdtreer  -np 4 -lightOrigin  -0.488 0.4905 0.067566 -ui
"camerapath(-file ../soda-room.path -delta_t 0.01 -delta_time 0.03 -behavior loop)" --timeview
```

Future work and Conclusion

- Faster build!
    - RBSP for top level, full BSP for lower level of tree (possibly huge savings)
    - Parallel build (soda hall in 3 hours on 8 core machine)
    - Optimize the build (2x speedup?)
    - Use BSP *only* for nodes kd-tree cannot further refine
- Adapting to work with other primitives
- BSP is very useful *if* build can be done as an offline preprocess