Build and Travel KD-Tree with CUDA

Meng Zhou

MengZhou@cmail.carleton.ca

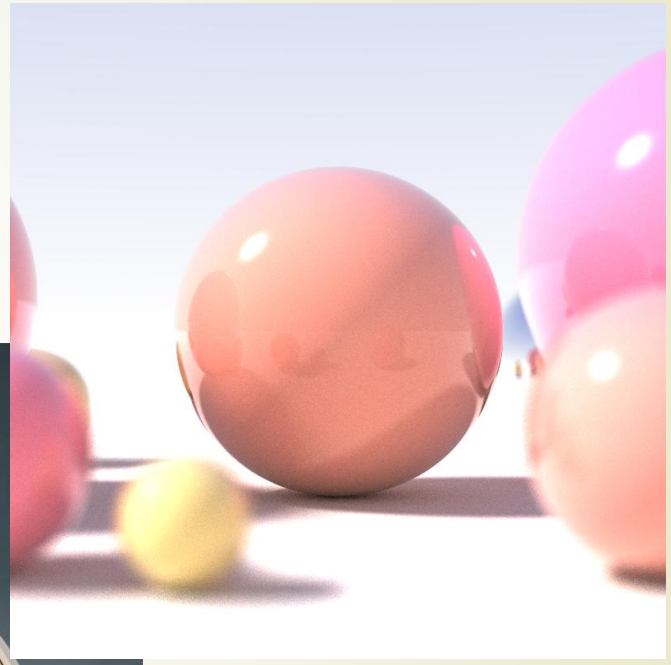
COMP 5704

School of Computer Science Carleton University

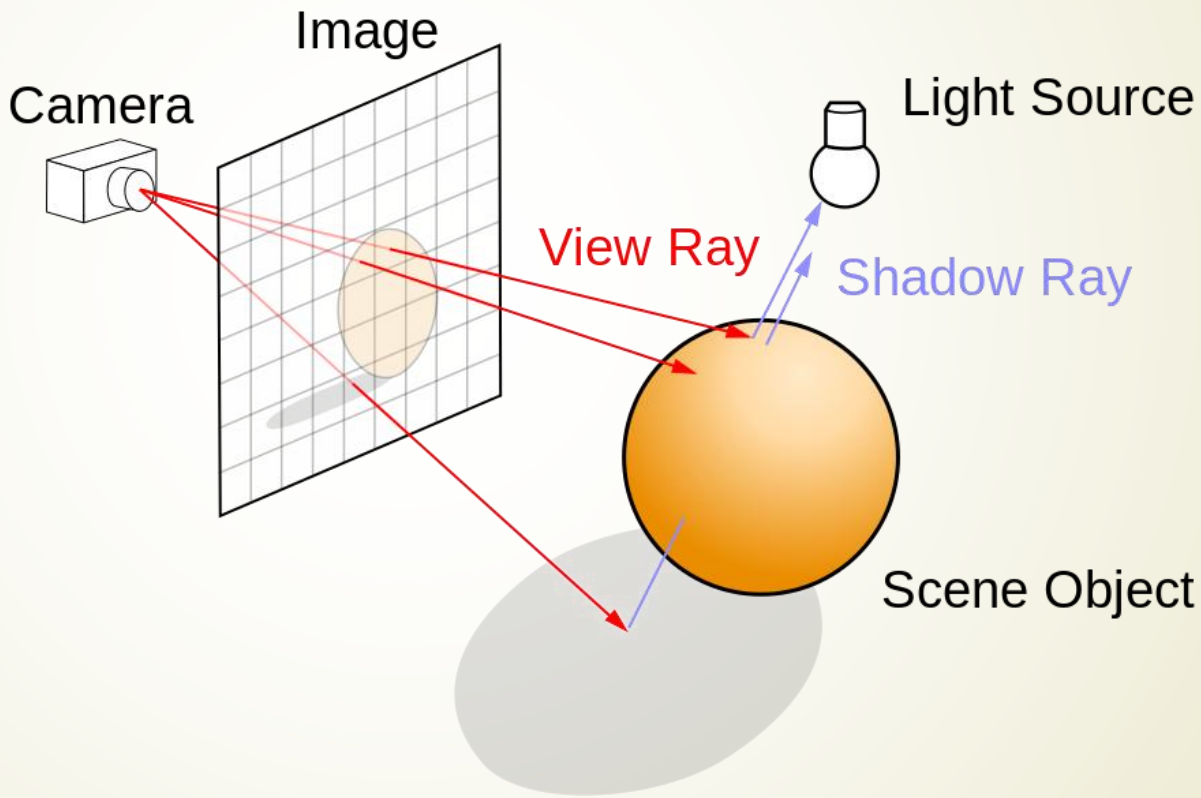
Contents

- Introduction to ray tracing and KD-Tree
- Sequential building and traveling algorithm
- Parallel building KD-Tree on GPU
- Parallel traveling KD-Tree on GPU
- Algorithm Analysis
- Implementation using CUDA
- Performance

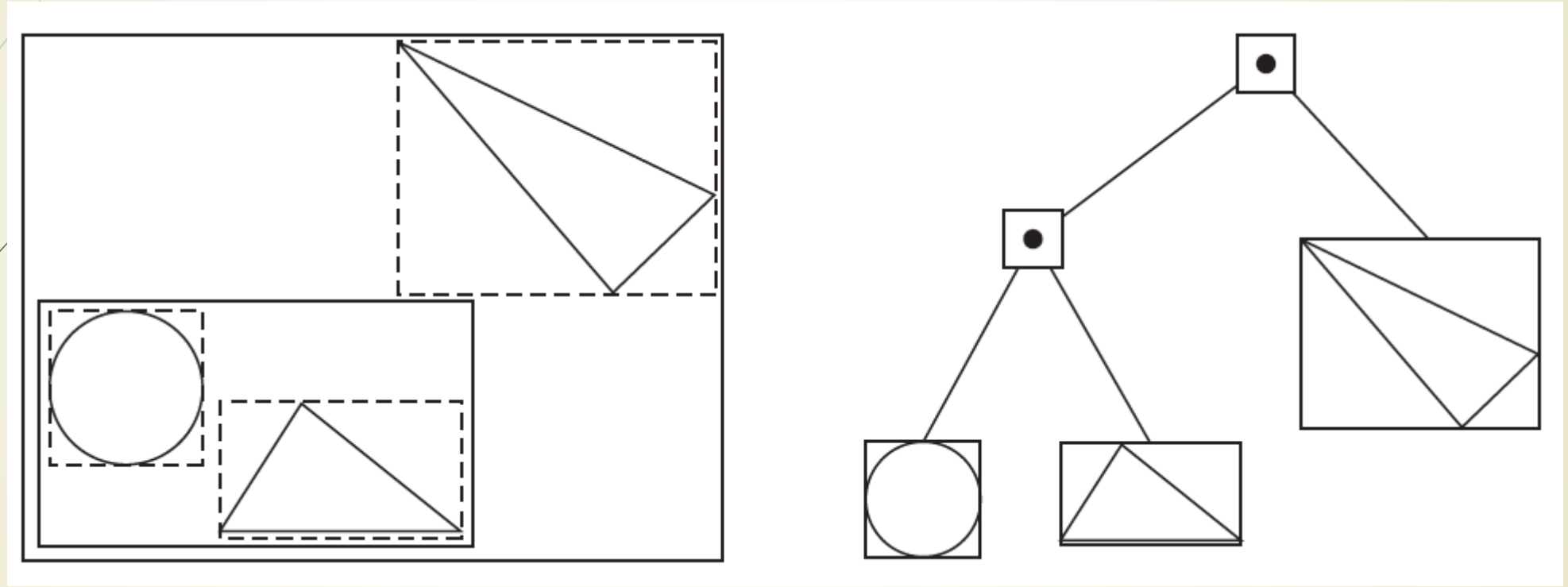
Introduction to ray tracing



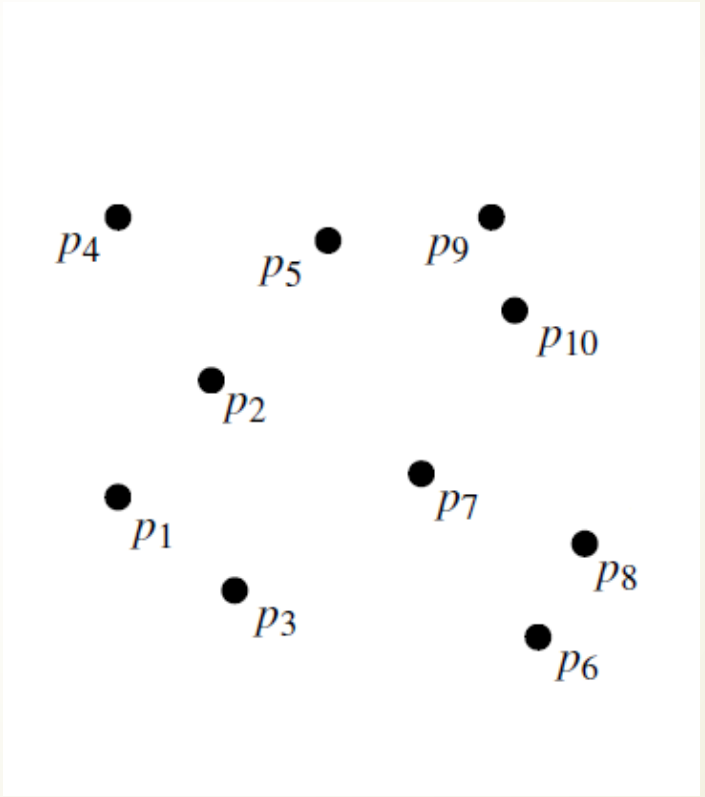
Introduction to ray tracing



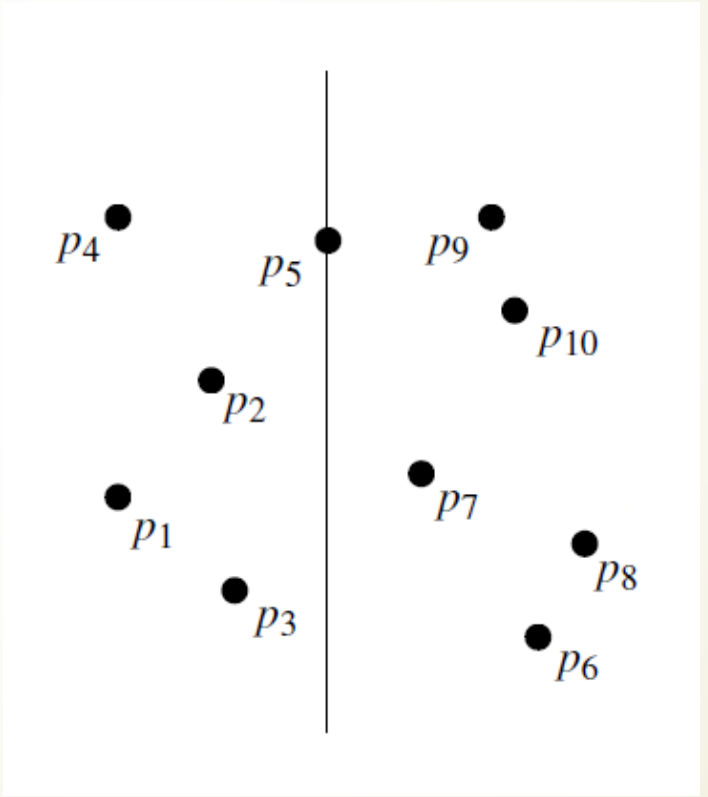
Space partition



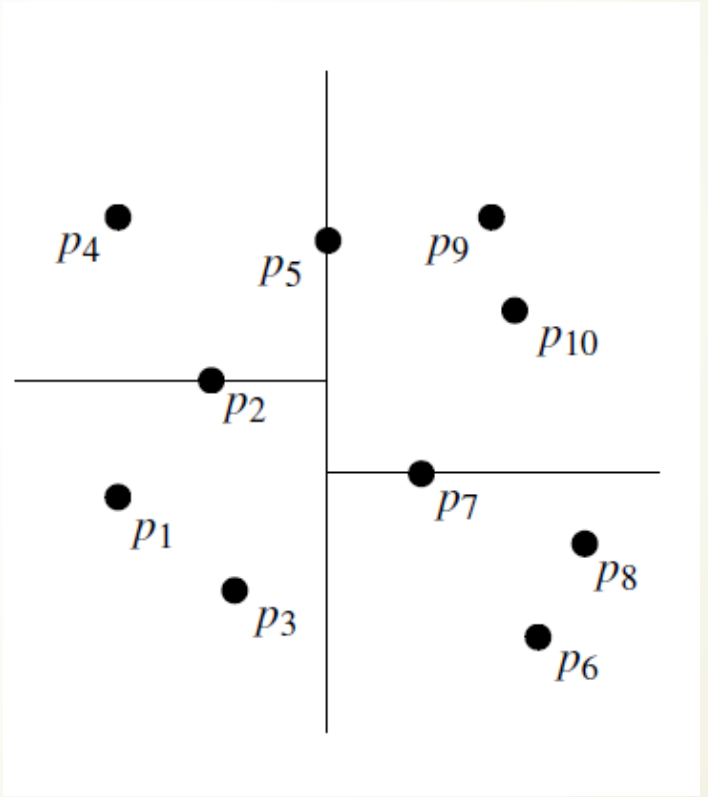
Building KD-Tree



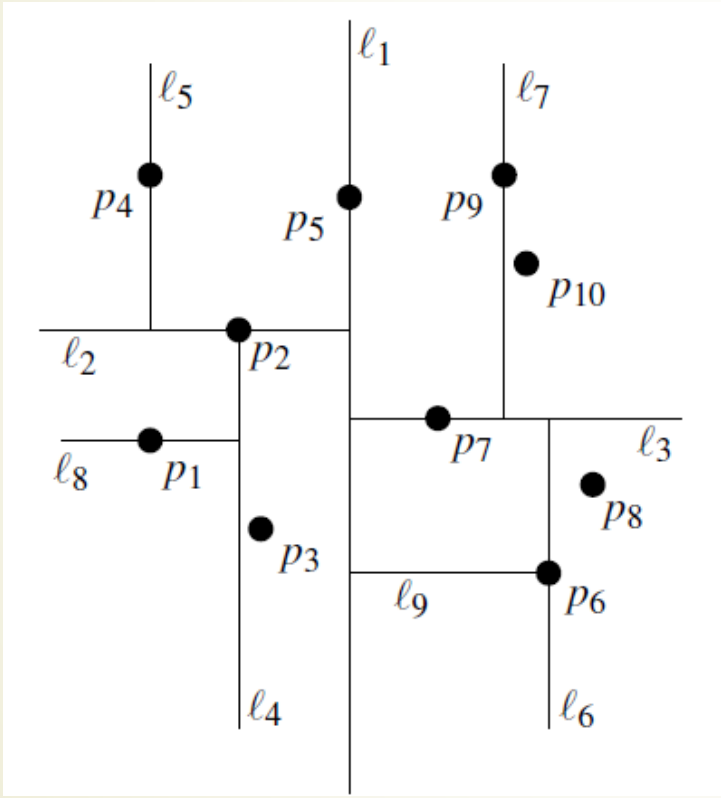
Building KD-Tree

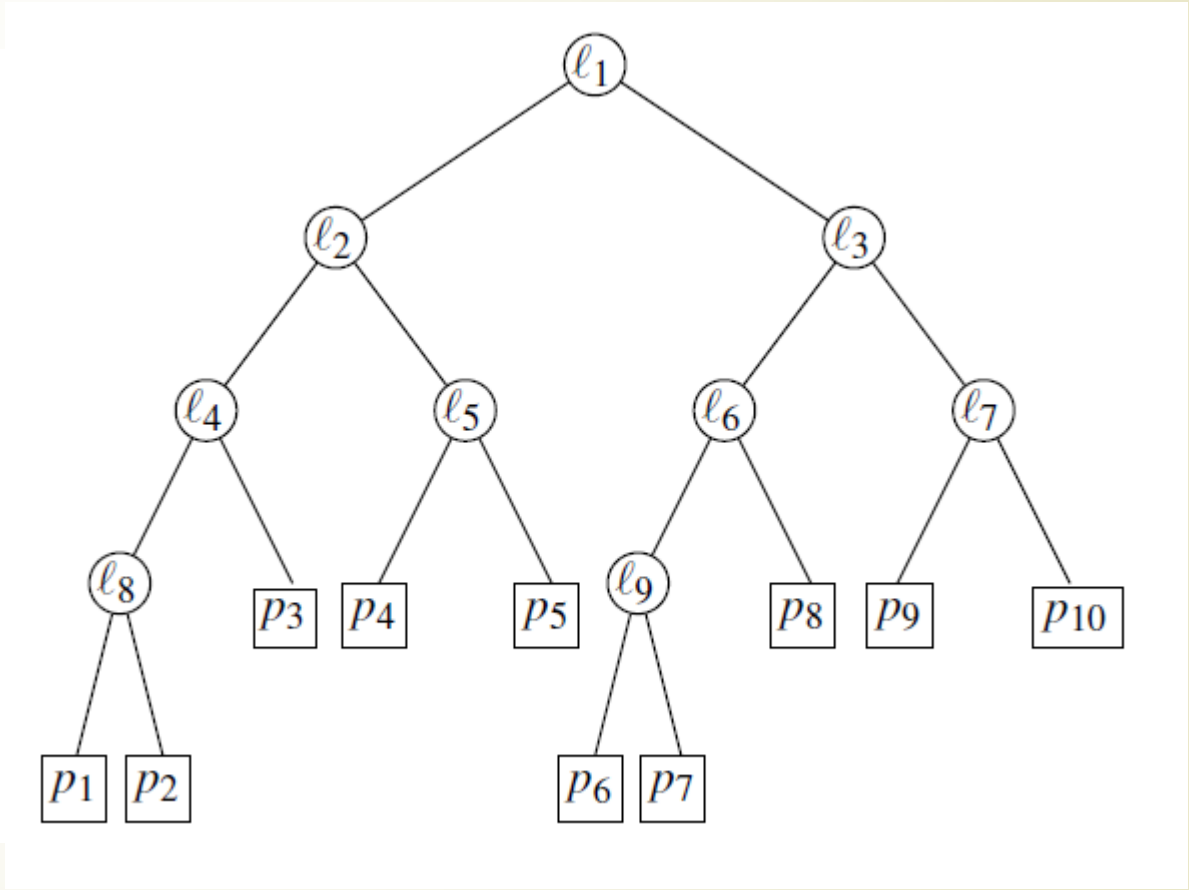


Building KD-Tree

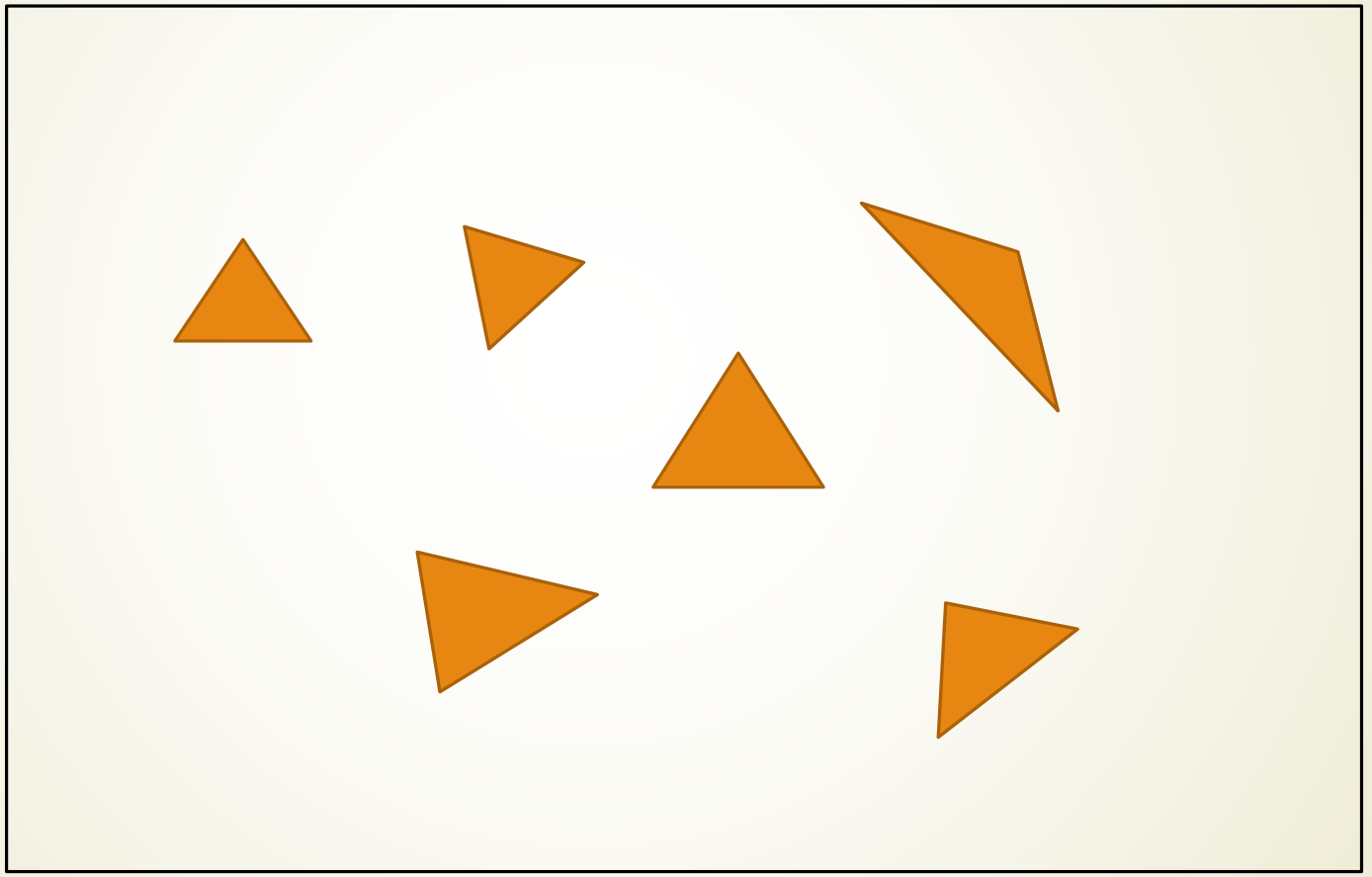


Building KD-Tree

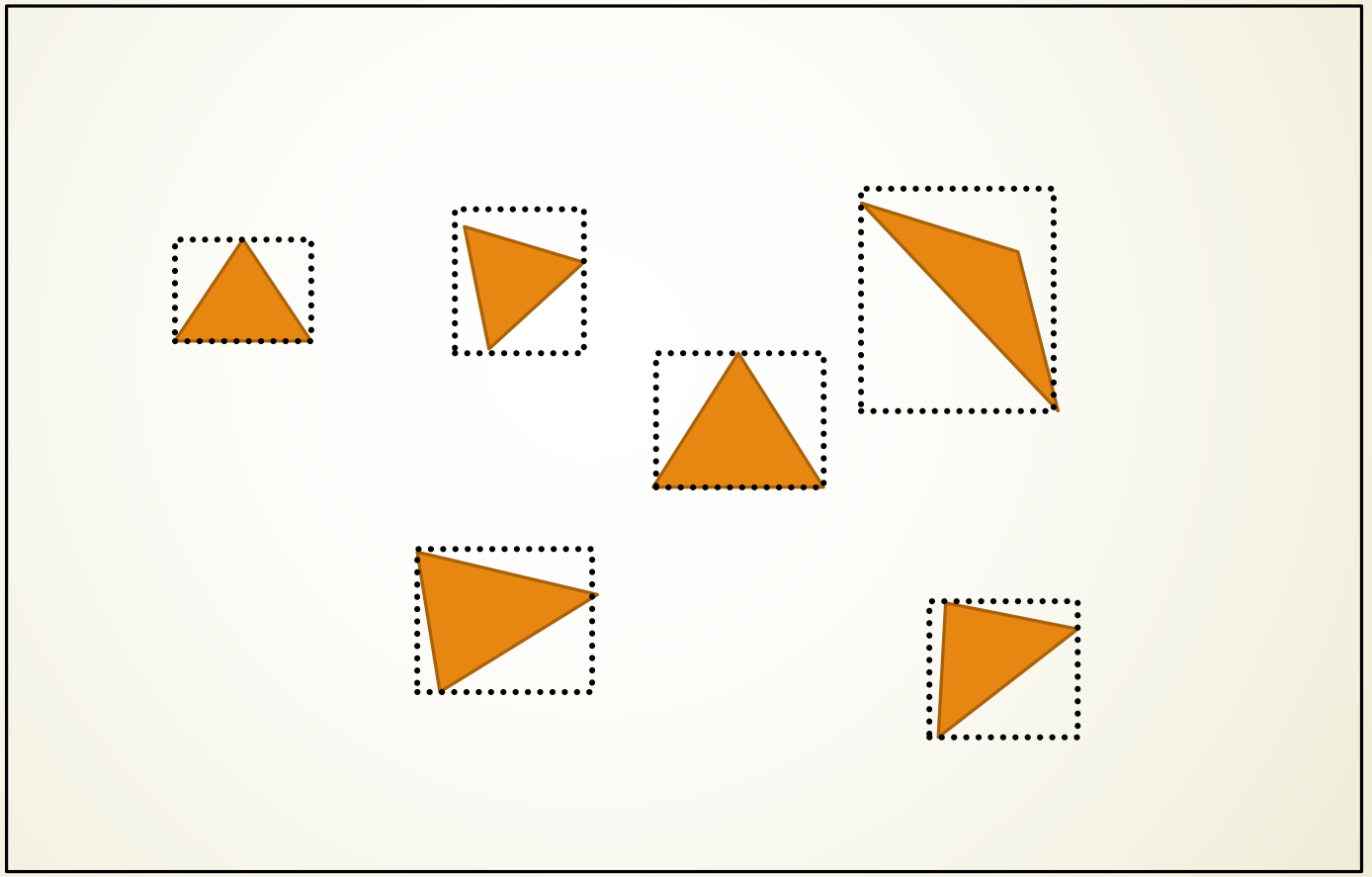




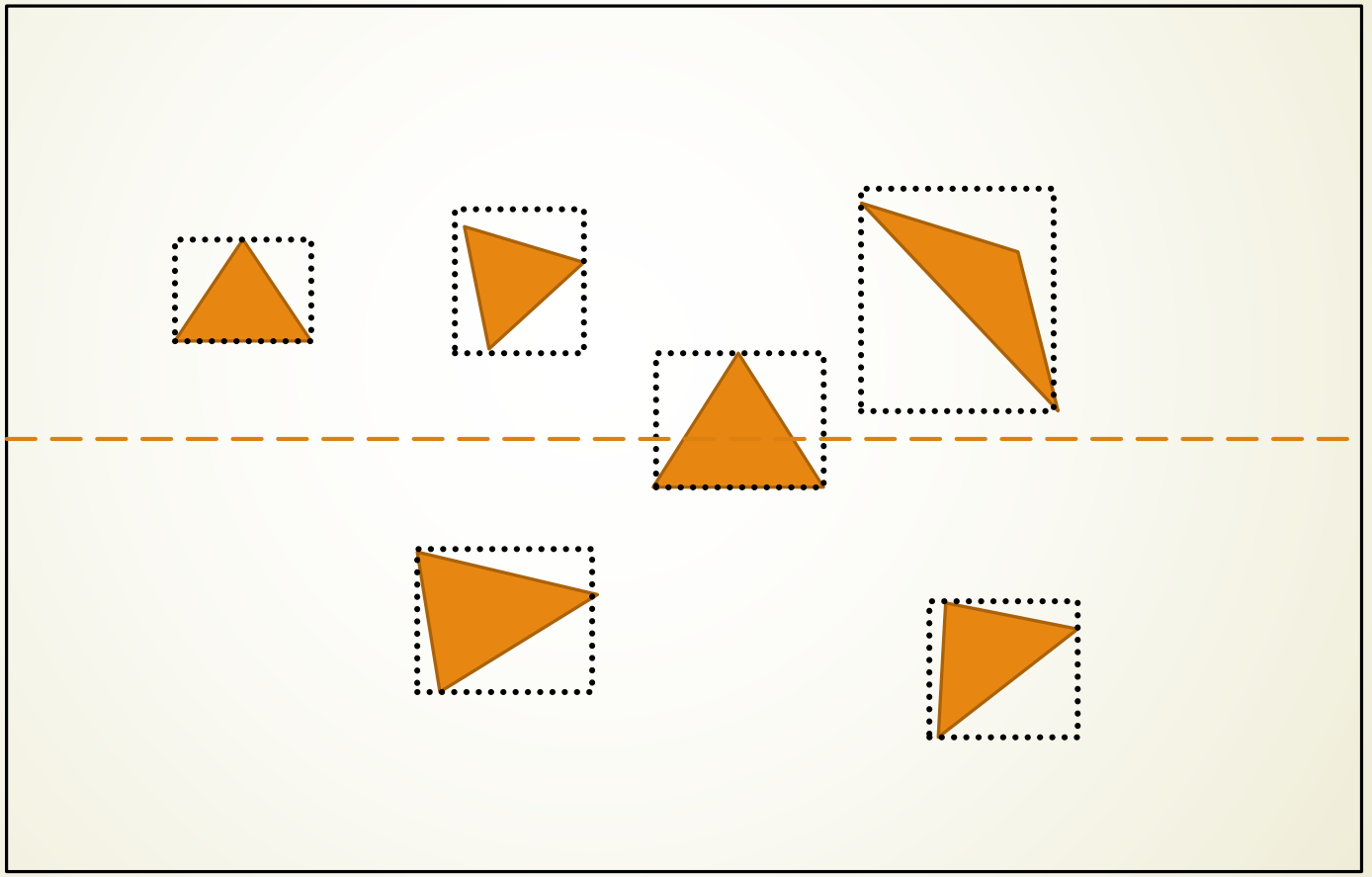
Building KD-Tree



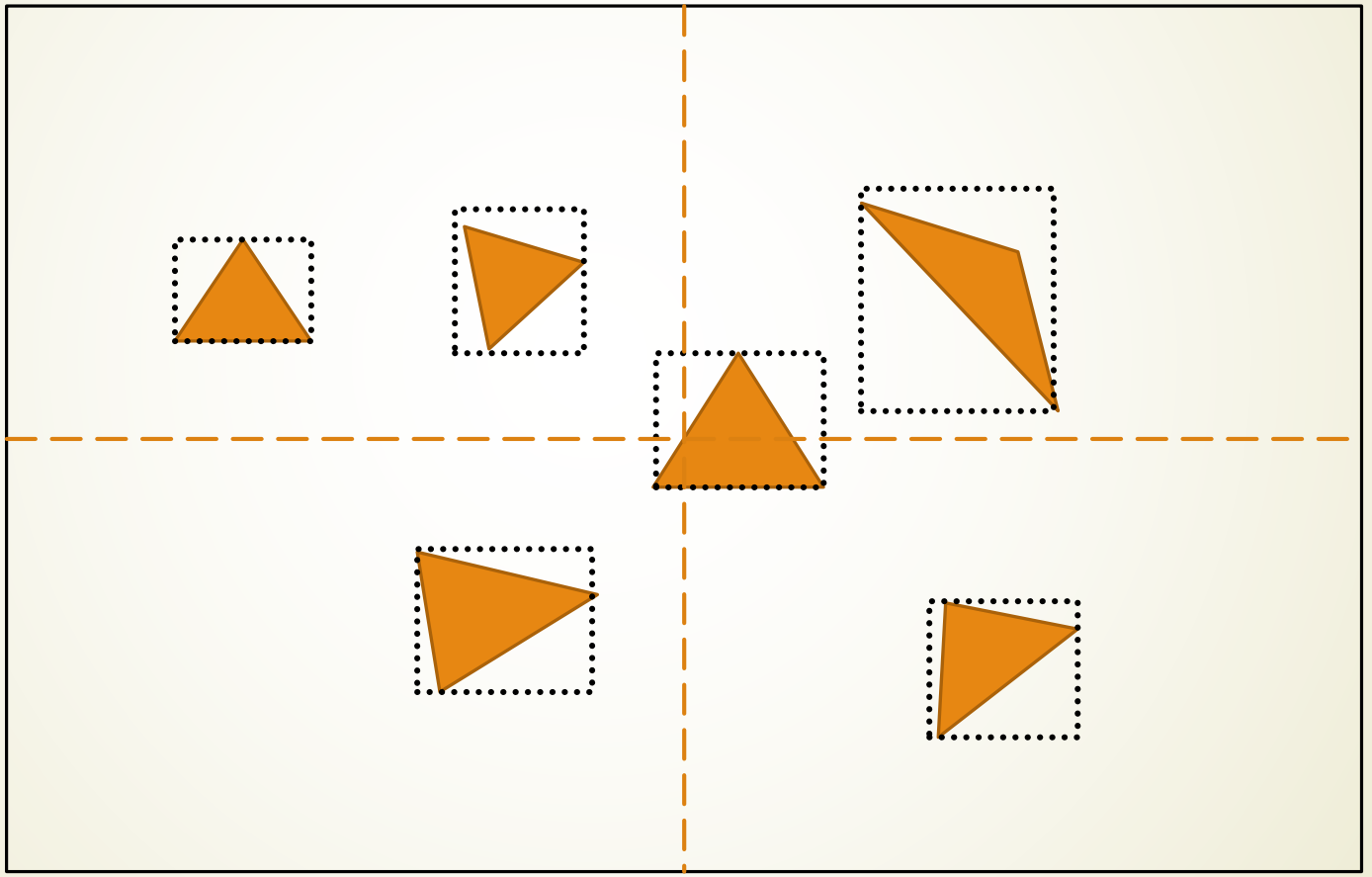
Building KD-Tree



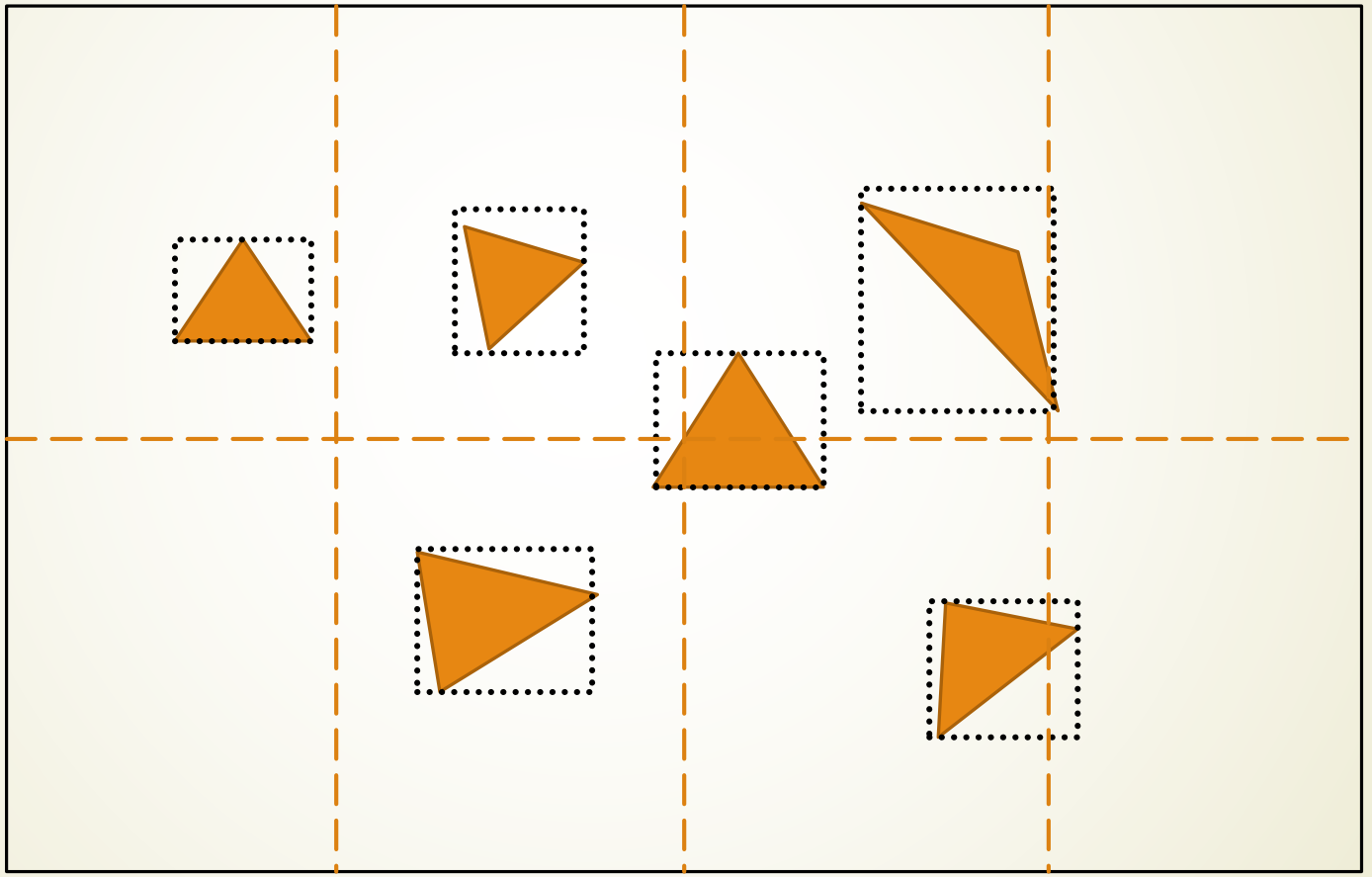
Building KD-Tree



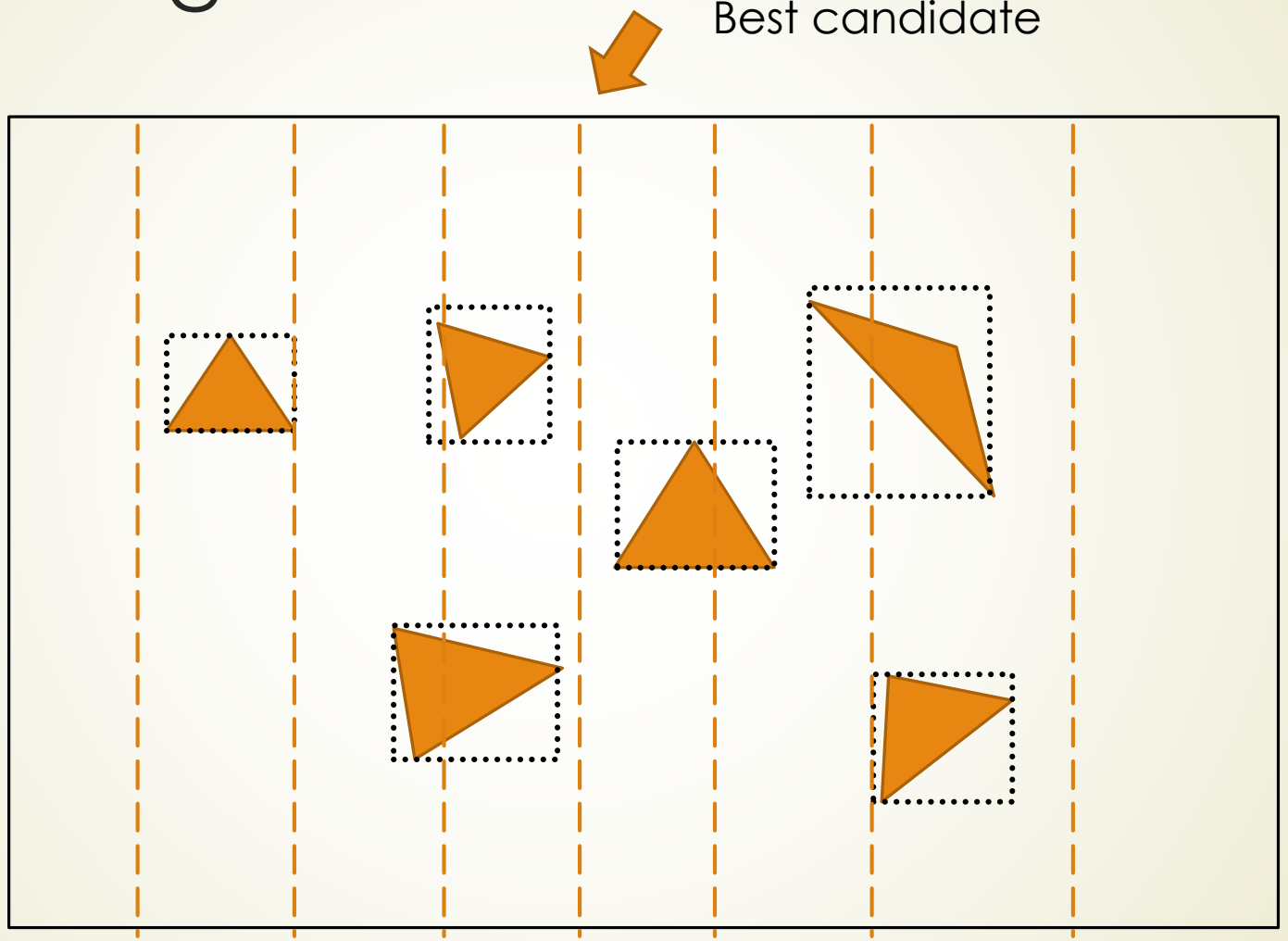
Building KD-Tree



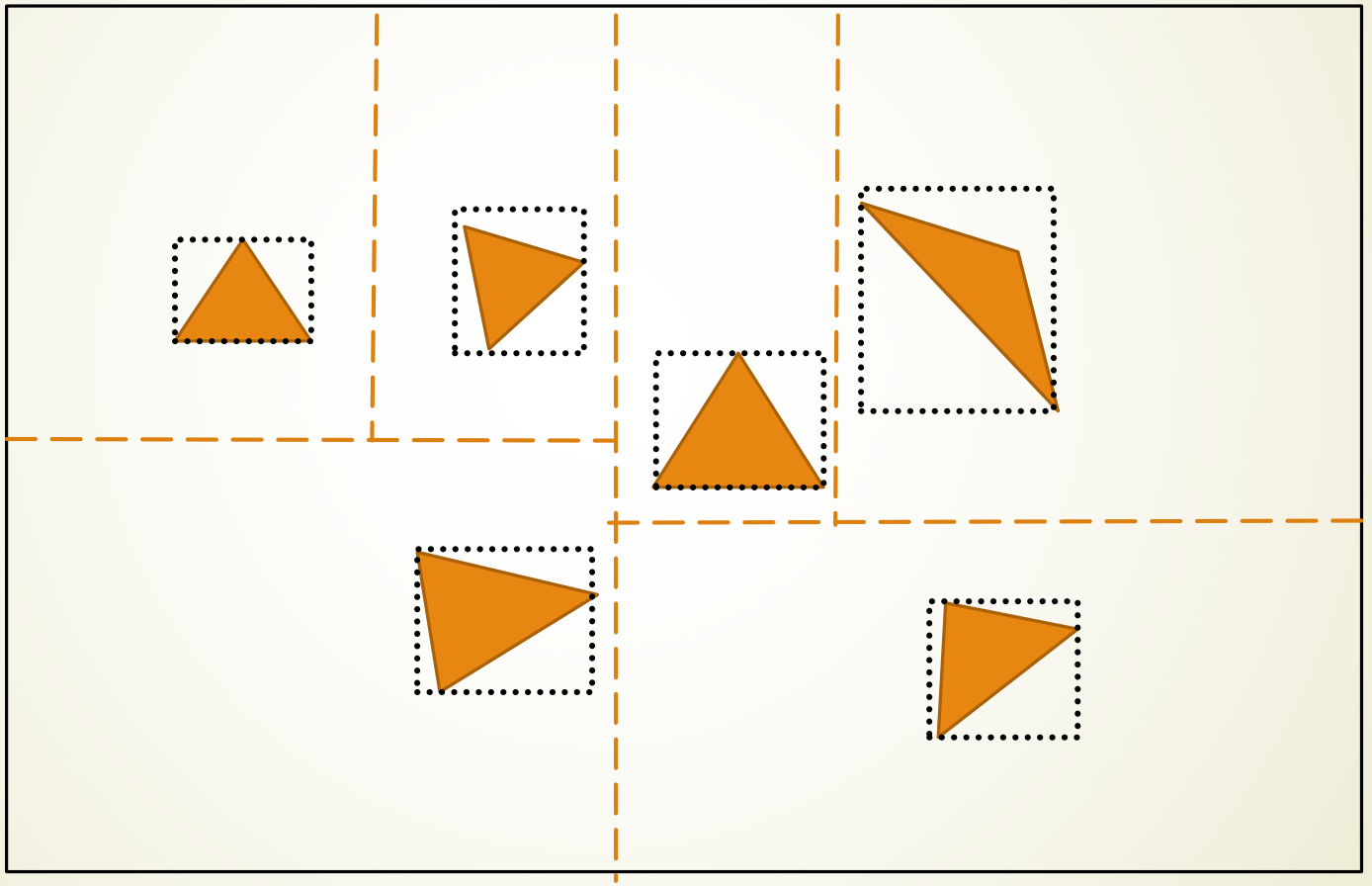
Building KD-Tree



Building KD-Tree



Building KD-Tree



Sequential algorithm overview

- Input: n triangles
- Output: KD-Tree
- Algorithm:

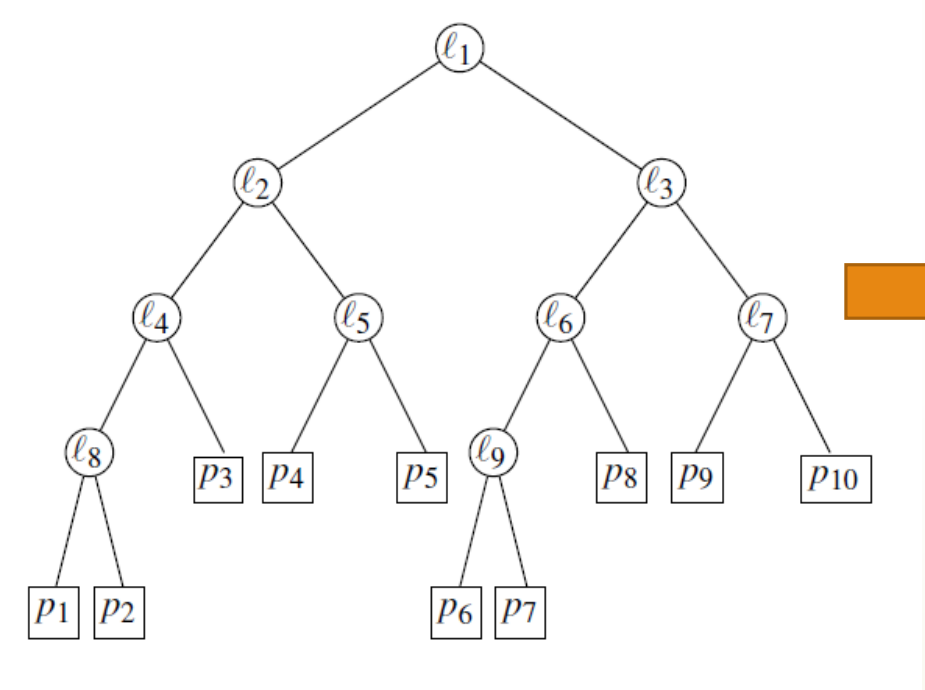
Initialize nodelist, activelist, nextlist;
For each input triangle t
    Compute AABB for t
activelist←root
While activelist not empty
    For each currentnode in activelist
        If #of triangle in currentnode > threshold
            split currentnode;
            Add children to nextlist and nodelist;
        End if
    switch activelist and nextlist;
End while

Parallelizing building algorithm

- Input: n triangles
- Output: KD-Tree
- Algorithm:

Initialize nodelist, activelist, nextlist;
For each input triangle t **in parallel**
    Compute AABB for t
activelist←root
While activelist not empty
    For each currentnode in activelist **in parallel**
        If #of triangle in currentnode > threshold
            split currentnode;
            Add children to nextlist and nodelist;
        End if
    switch activelist and nextlist;
End while

Implementation on CUDA



Node list

l_1	l_2	l_3					.....

Left child

l_2	l_4	l_6					.....

Right child

l_3	l_5	l_7					.....

Node AABB

							.....

Split axis
Split position
Triangle list

Implementation on CUDA

```
template<class T>
class DeviceVector {
__inline__ __device__ static unsigned int push_back(T* d, unsigned int* ptr,T& t){
    unsigned int i = atomicAdd(ptr, 1);
    d[i]=t;
    return i;
}
__inline__ __device__ static void pop(T* d, unsigned int* ptr, T& t){
    unsigned int i = atomicAdd(ptr, -1);
    t=d[i-1];
}
unsigned int* d_ptr; // data size
T* data;
thrust::device_ptr<T> thrustPtr;
}
```

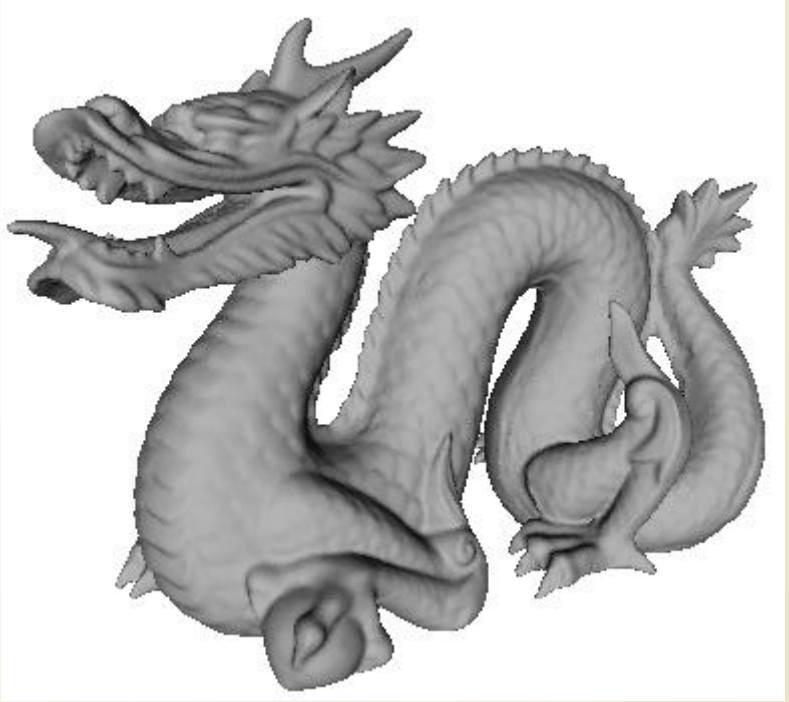
Performance

- Experiment on Intel i5-4670k@3.4Ghz, 16GB DDR3@1800Mhz

Nvidia GeForce GTX 780 Ti 3GB with 2880 cuda cores

- Test scene:

Name	Triangles
Dragon	871 k
Dragon_Res2	202 k
Dragon_Res3	47 k
Dragon_Res4	11 k
elephav	1 k



Performance(build)

Name	CPU	GPU	CUDAMemcpy	Ratio
Dragon	38315	8063	250	4.75
Dragon_Res2	7625	1909	268	3.99
Dragon_Res3	1547	672	258	2.30
Dragon_Res4	313	359	234	0.87

Performance(travel)

Intersecting 50000 rays

Name	CPU	GPU	Ratio
Dragon	3297	344	9.58
Dragon_Res2	2641	265	9.96
Dragon_Res3	2016	219	9.21
Dragon_Res4	1469	172	8.54

References

- Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time kd-tree construction on graphics hardware. In ACM SIGGRAPH Asia 2008 Pa-pers, SIGGRAPH Asia '08, pages 126:1-126:11, New York, NY, USA, 2008. ACM.
- Daniel Reiter Horn, Jeremy Sugerman, Mike Houston, and Pat Hanrahan. Interactive k-d tree gpu raytracing. In Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games, I3D '07, pages 167-174, New York, NY, USA, 2007. ACM.
- Tim Foley and Jeremy Sugerman. Kd-tree acceleration structures for a gpu raytracer. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, HWWS '05, pages 15-22, New York, NY, USA, 2005. ACM.
- Stefan Popov, Johannes G□unther, Hans-Peter Seidel, and Philipp Slusallek. Stackless kd-tree traversal for high performance GPU ray tracing. Com- puter Graphics Forum, 26(3):415-424, September 2007. (Proceedings of Eurographics).