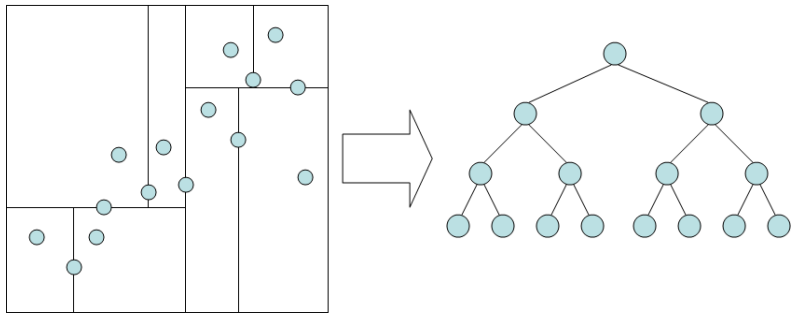
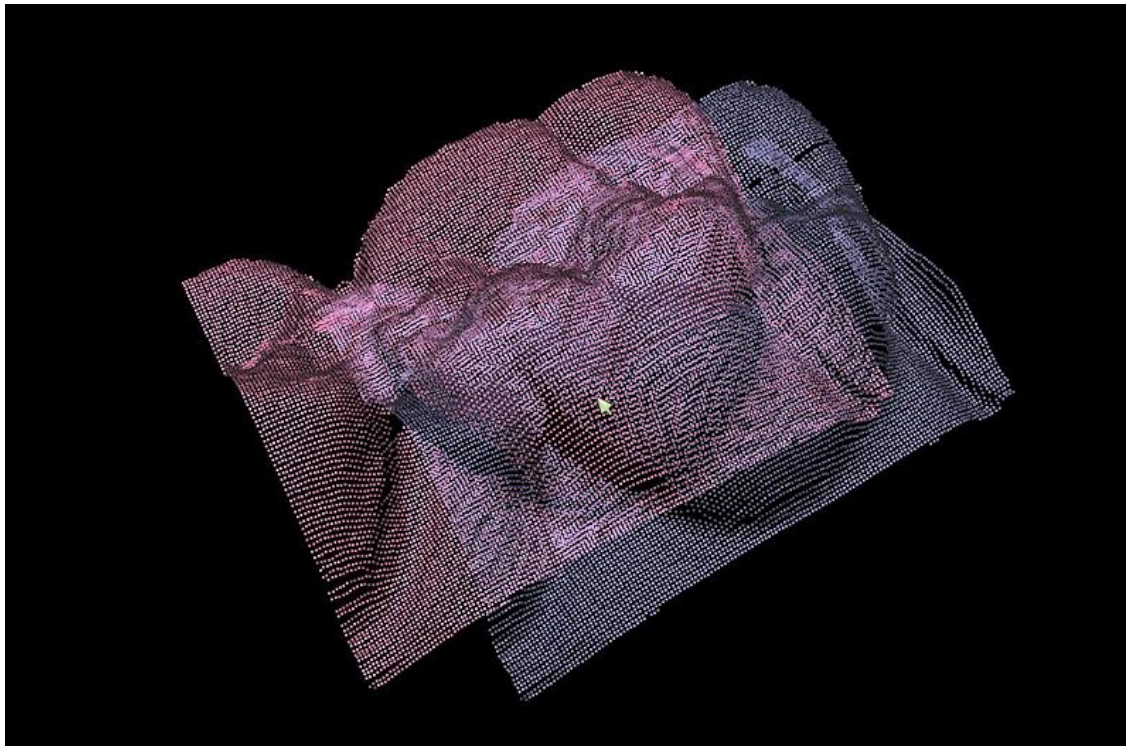
Warped parallel nearest neighbor searches using kd-trees

Roman Sokolov, Andrei Tchouprakov
D4D Technologies

Kd-trees

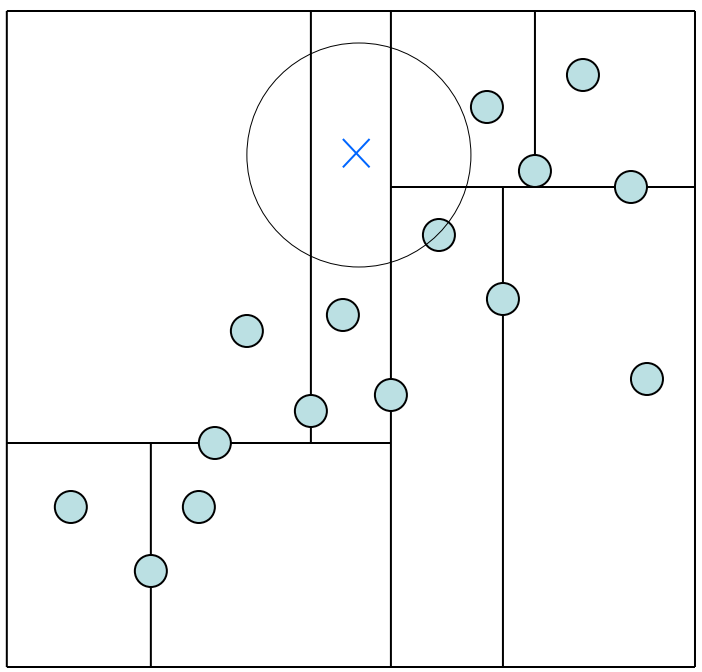
- Binary space partitioning tree
- Used for nearest-neighbor search, range search
- Application: surface alignment, ray-tracing

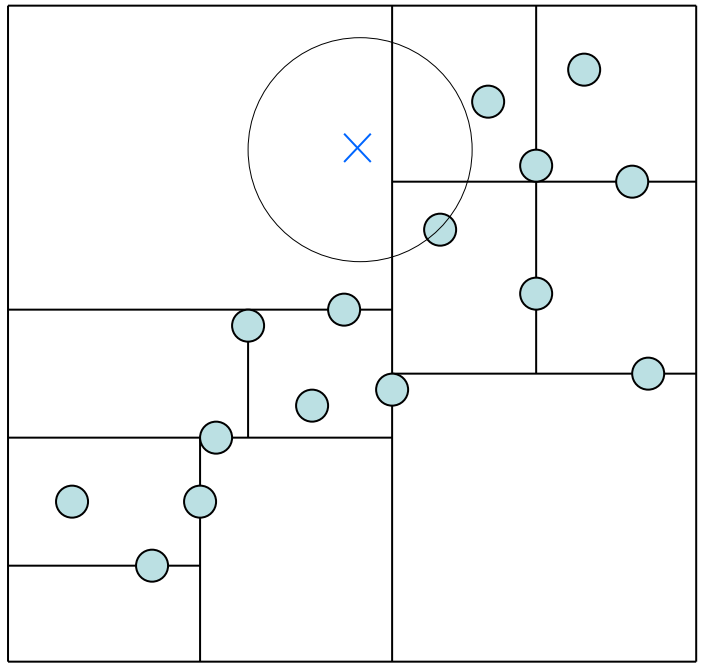




Building a kd-tree: median split and sliding-point rules

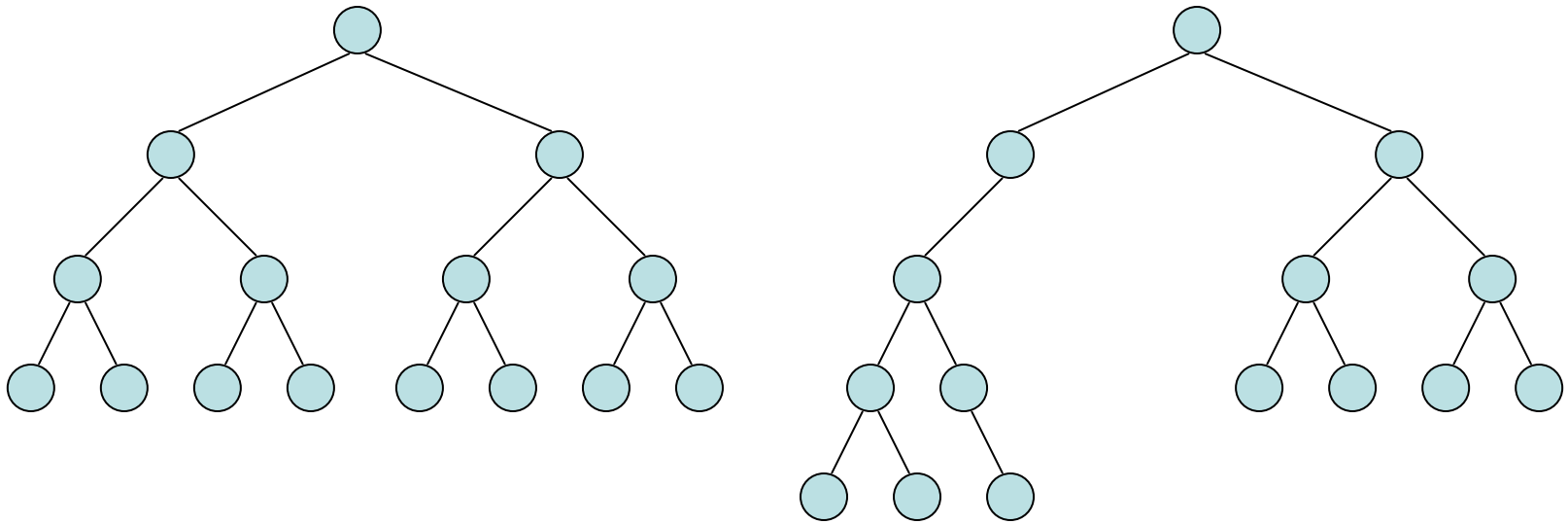
Median split produces high aspect ratio cells – more leaves to inspect





Building a kd-tree: median split and sliding-point rules

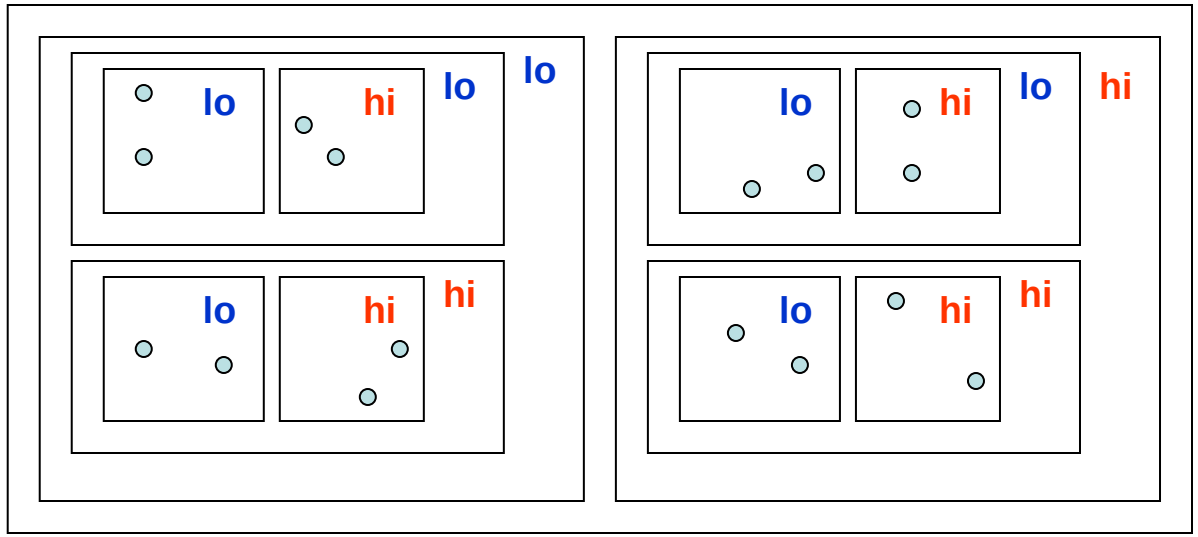
Sliding point rule produces unbalanced trees

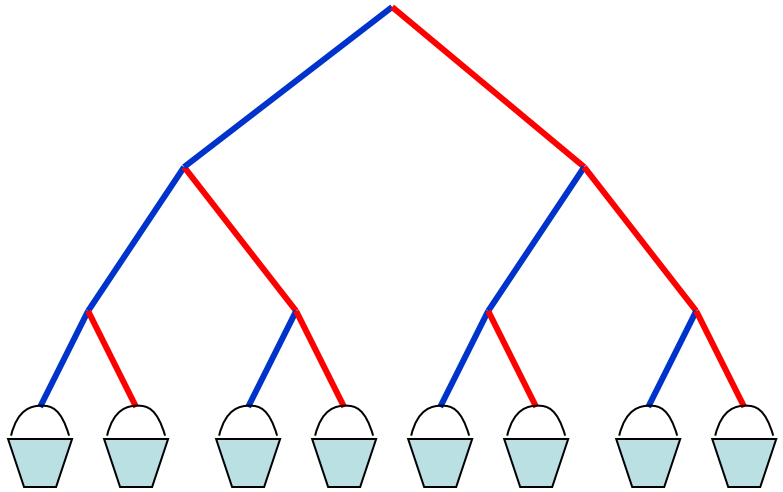


Another way to represent a kd-tree: as an hierarchy of arrays

Leaves are represented as arrays of points (buckets)

Nodes are represented as pointers to arrays



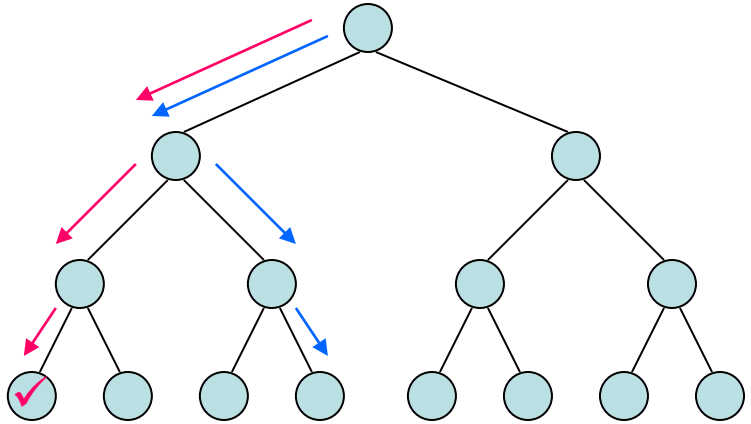


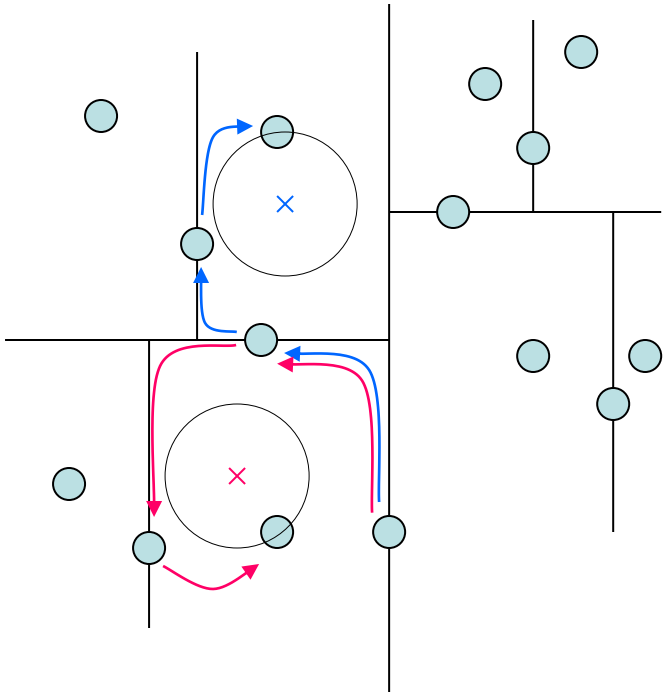
Depth first NN-search on a kd-tree

1. Starting with the root, inspect the child nodes recursively
2. Determine, on which side of the cutting plane lies the query point
3. Inspect the corresponding sub-tree, find minimal distance estimate
4. If the distance to the cutting plane is less than minimal distance, inspect the other sub-tree

NN-search on a kd-tree

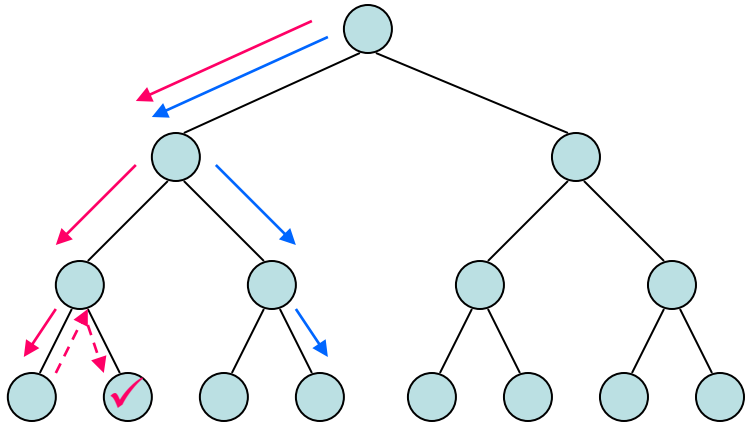
The simplest case: minimal distance found is less than the distance to all nodes

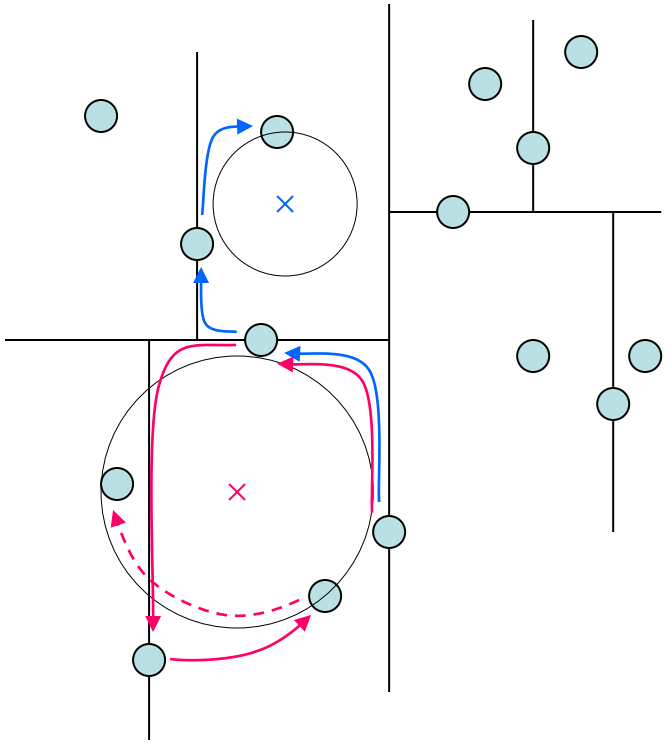




NN-search on a kd-tree

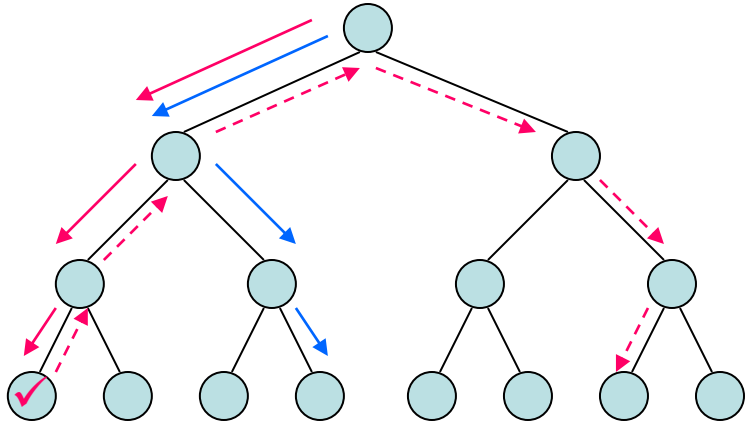
Need to inspect the neighboring leaf

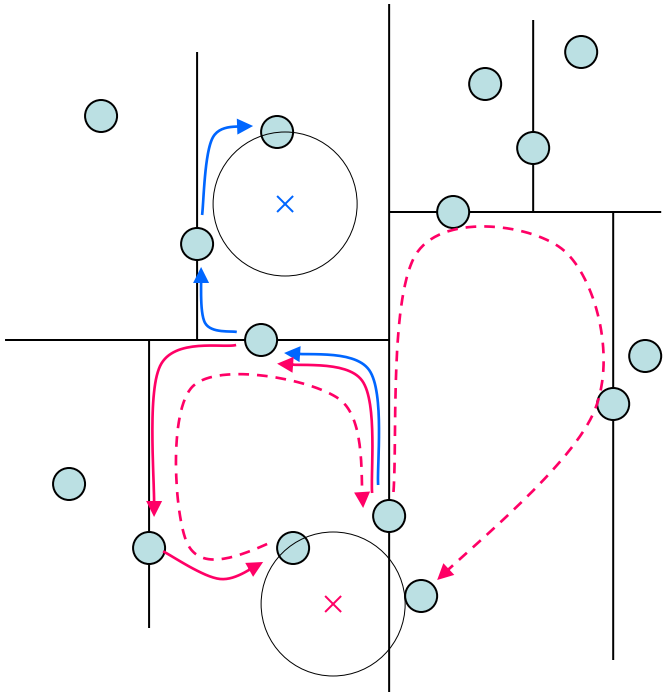




NN-search on a kd-tree

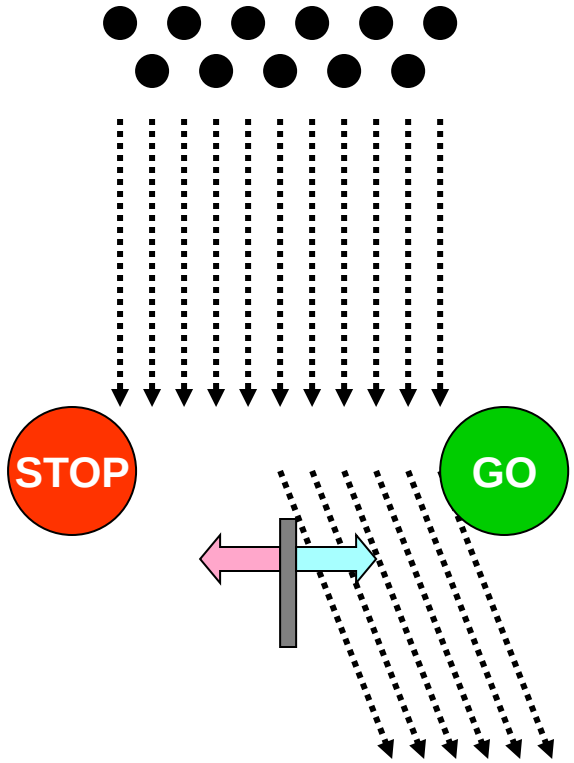
Sometimes we have to inspect far away leaves





NN-search on a kd-tree using GPU

- Set of query points
- One query point per thread
- Problem on GPU: thread divergence
- 32 Threads per warp



Previous work: *GPU Nearest Neighbor Searches using a Minimal kd-tree.* Brown, Shoeyink GTC2010

Thread divergence in a warp

individual threads:

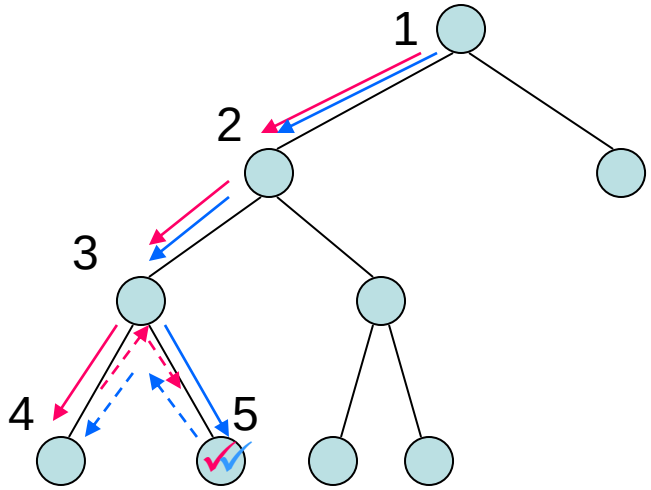
Path1: 1 2 3 **4** **5**
Path2: 1 2 3 **5** **4**

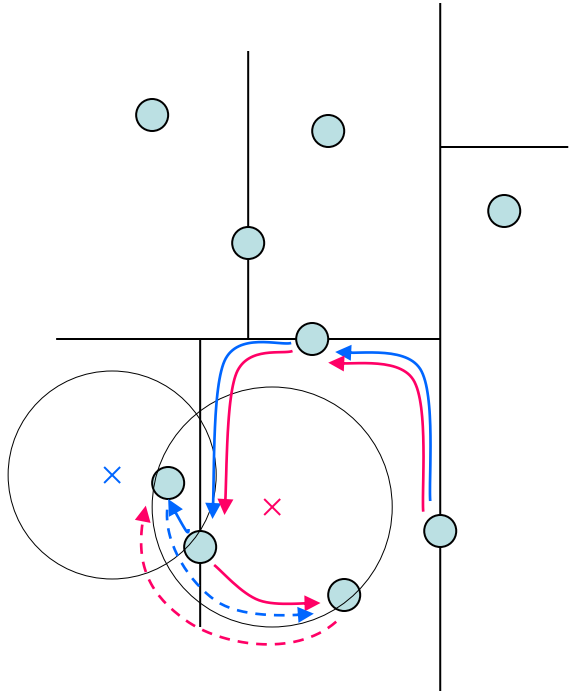
in a warp – divergence:

Path1: 1 2 3 **4** **5**
Path2: 1 2 3 **5** **4**

can save a step:

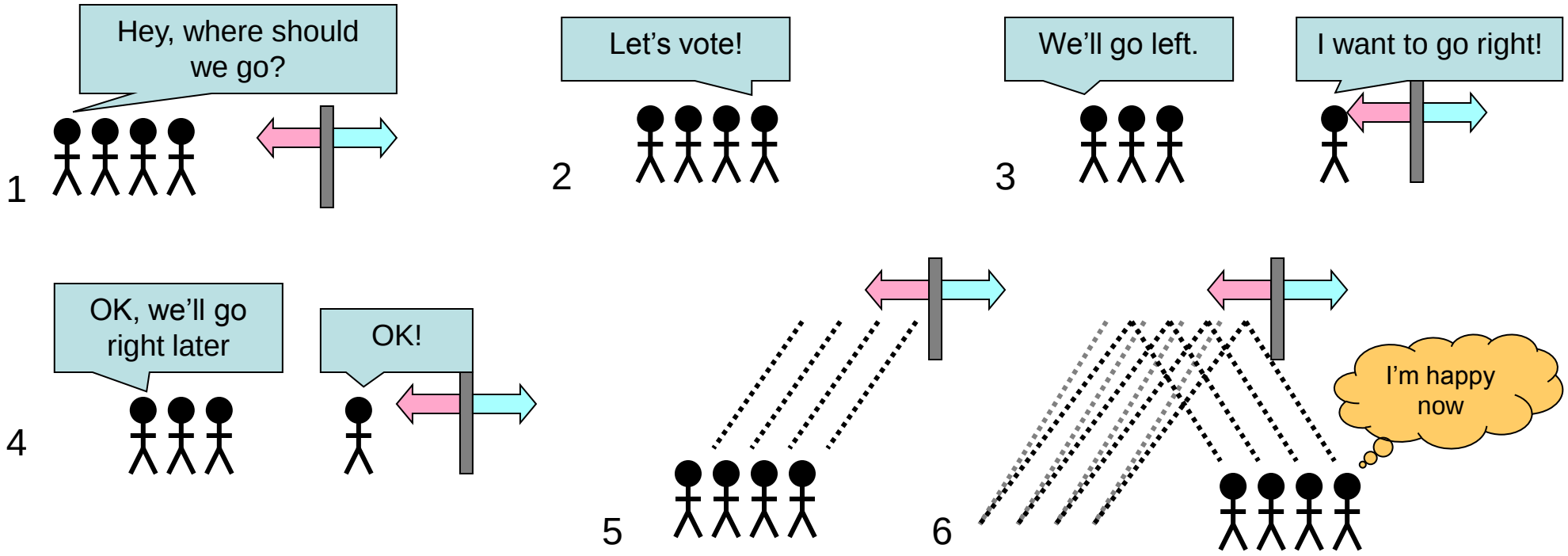
Path1: 1 2 3 **4** **5**
Path2: 1 2 3 **4** **5**





Warped kd-tree search

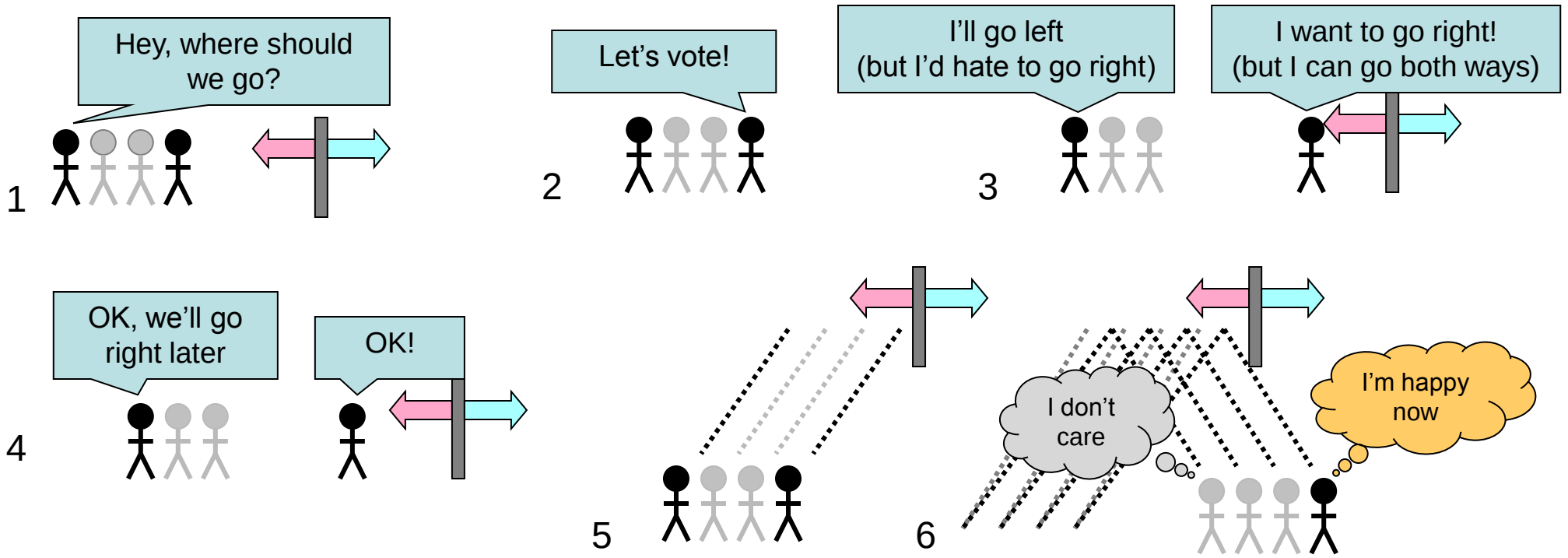
- To prevent divergence, all threads in a warp should perform the same operation
- Use CUDA warp voting functions to decide which sub-tree to inspect first



Warped kd-tree search

- Some threads may be inactive – only active threads vote

- Threads become inactive when they follow a path that does not need to be inspected



- Search is finished when all threads become inactive

NN-search on a kd-tree using recursion

naive:

```
inspect(node)
{
   if (is_leaf(node))
         bruteforce(node),return
   if (inside(node.lo))
   {
      inspect(node.lo)
      if(distance_to(node.hi) < min_dist)
      inspect(node.hi)
   }
   else
   {
   ……
   }
}
```

warped:

```
inspect(node, mask)
{
   if (is_leaf(node))
         bruteforce(node),return
   mask = mask & ballot(inside(node.lo))
   if (mask)
   {
      inspect (node.lo,mask)
      if(any(distance_to(node.hi) < min_dist))
         inspect(node.hi,mask)
   }
   else
   {
   ……
   }
}
```

Warped kd-tree search

- Use stack to implement recursion

- Single stack per wrap, stored in shared memory

Using CUDA `__ballot()` function:

- to decide which path to follow

- to create mask of active threads in a warp

(requires CUDA compute capability 2.0 or greater)

Advantages:

- A leaf on the other side of the splitting plane might have to be inspected anyway later. Might as well do it now and save time.

- Single stack for all threads in a warp.

Drawback:

- More complex logic. Slower performance for highly divergent paths.

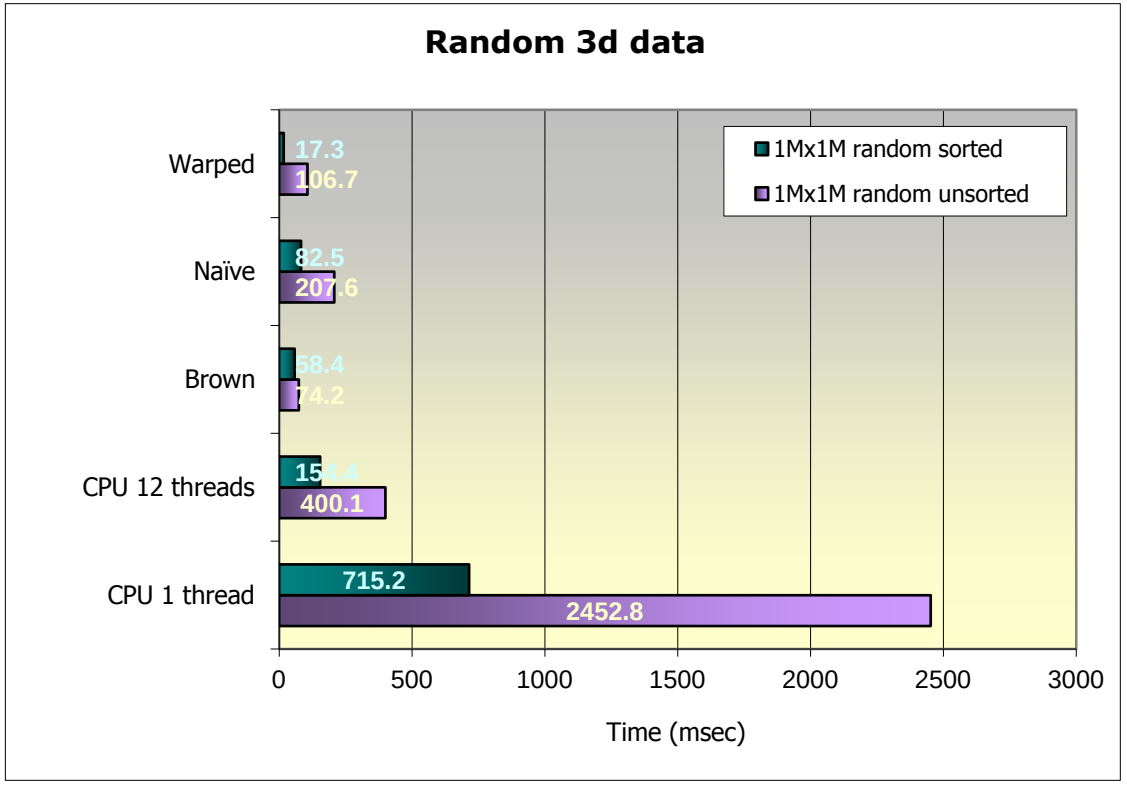
Performance evaluation

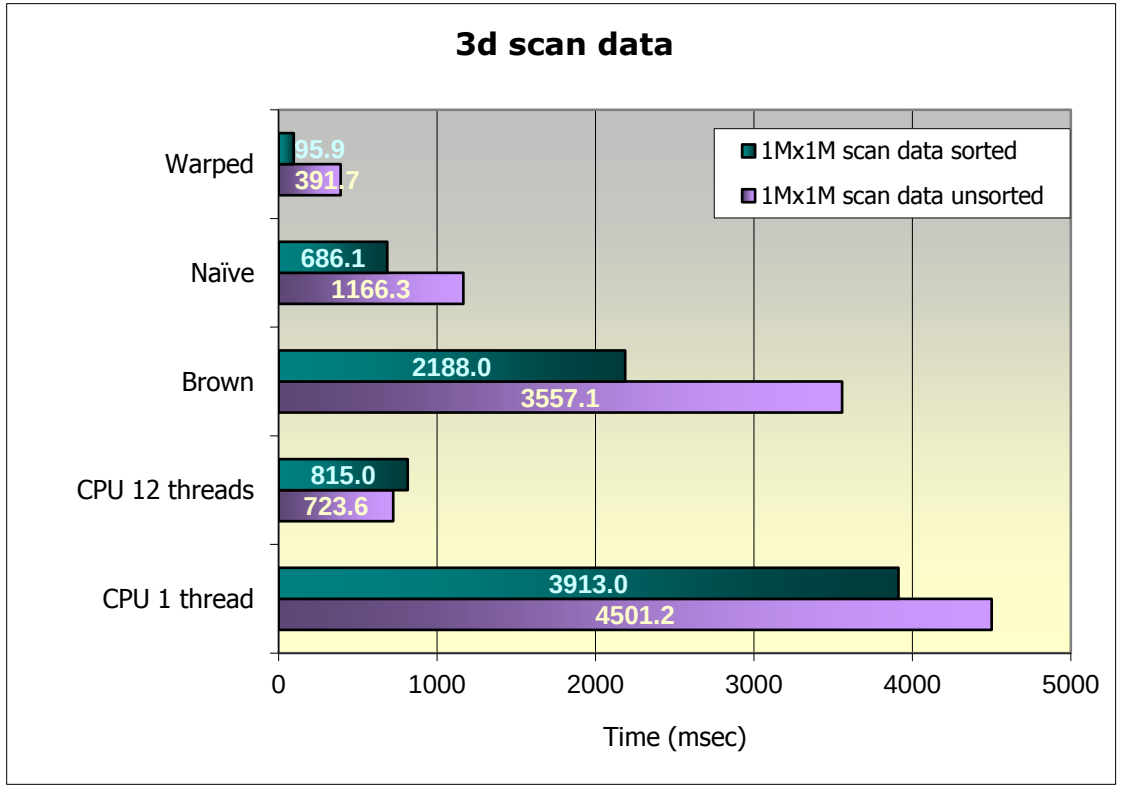
- 1M query and 1M search points
- Random points in a cube
- Data from 3d scans of dental models
- Query points: unsorted and sorted

Results

CPU: Intel Xeon X5650 6 Cores 12 Threads

GPU: GK104 GeForce GTX 680

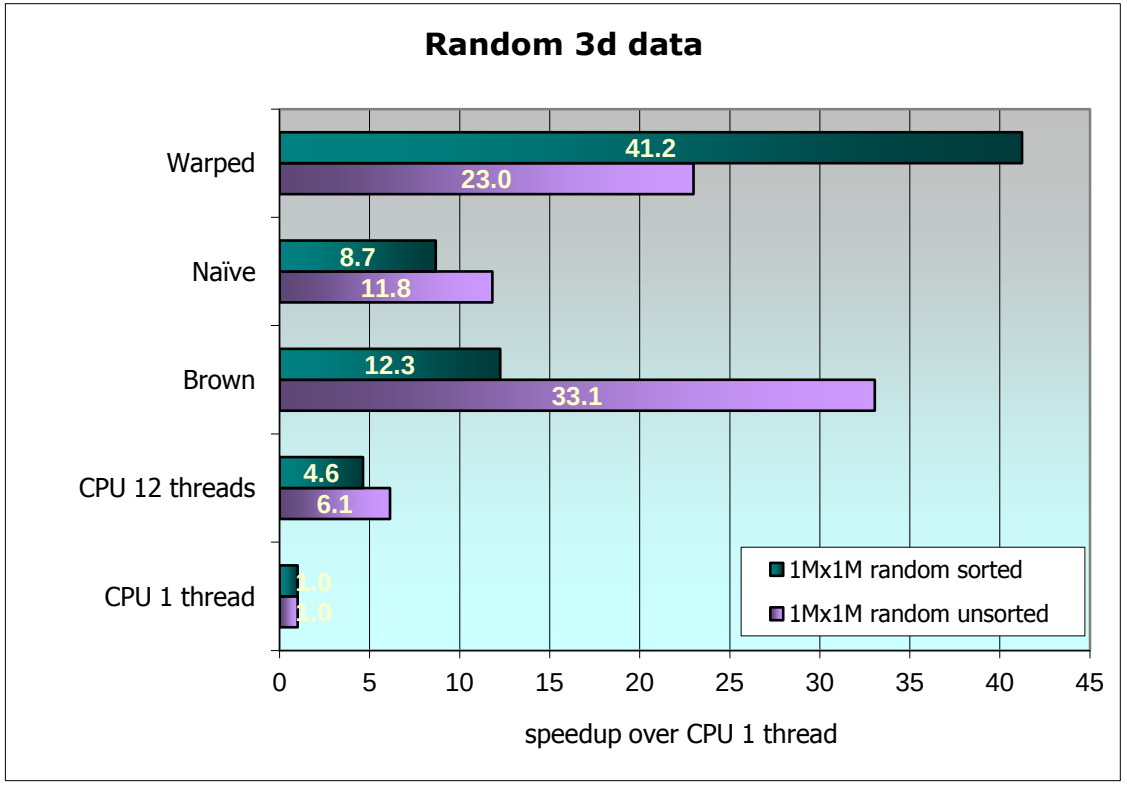


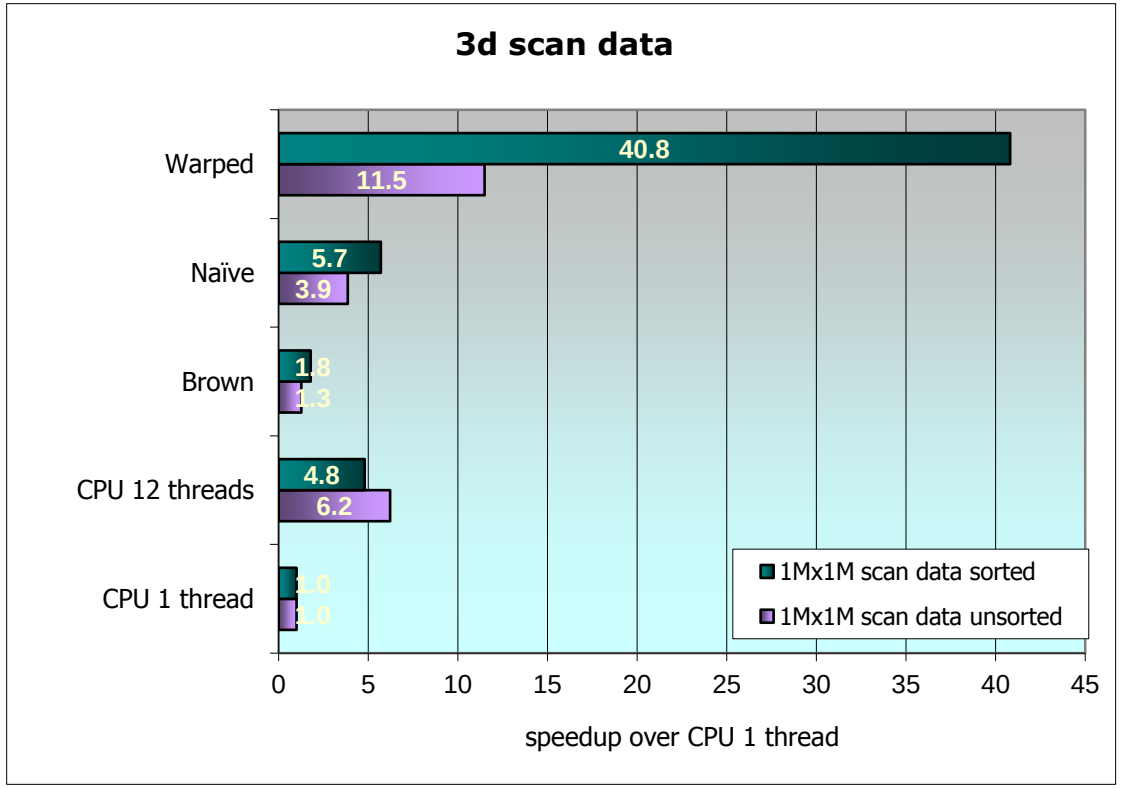


Results

CPU: Intel Xeon X5650 6 Cores 12 Threads

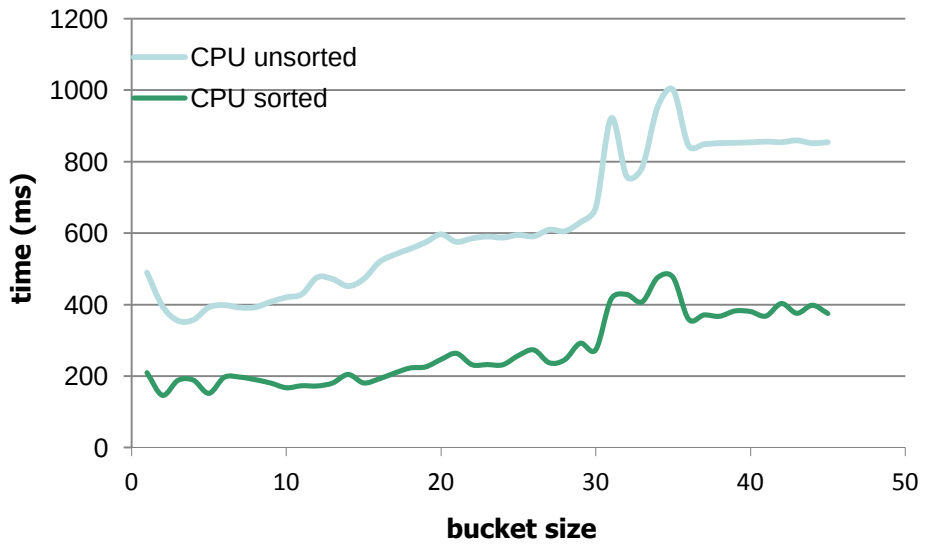
GPU: GK104 GeForce GTX 680

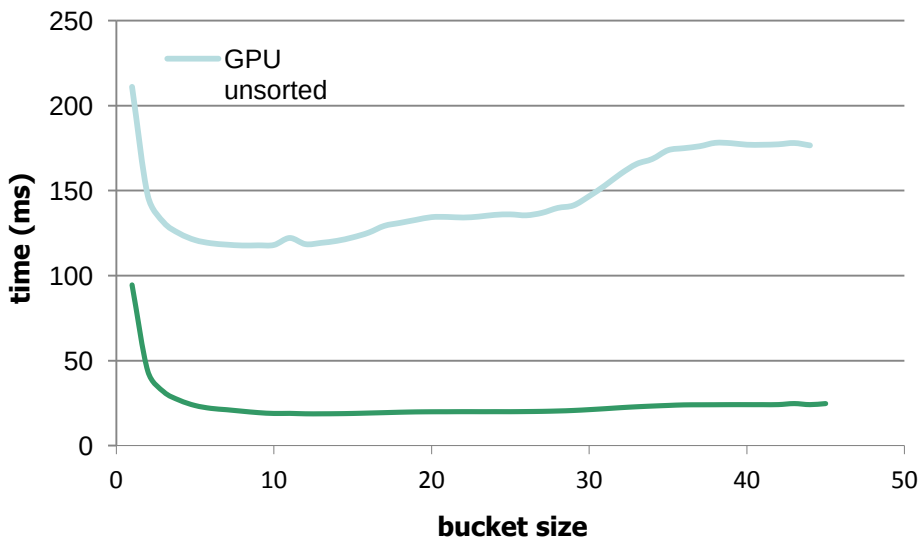




Results

Tree depth vs. bucket size: tradeoff between node traversal and brute force search





Future work

- Improve memory access to kd-tree nodes, utilize shared memory
- Build kd-tree on GPU
- Utilize inactive threads during brute-force search (reduction)
- Try using single stack per block