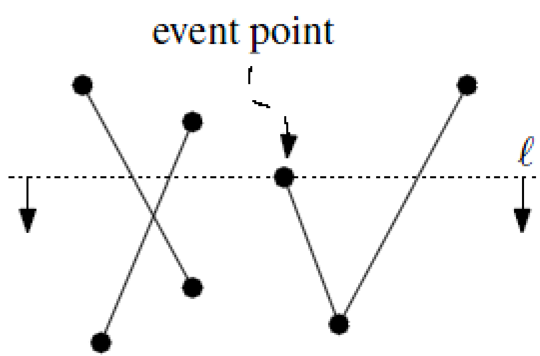
Line Segment Intersection

Gen Nishida

Static Set of Line Segments

Map Overlay

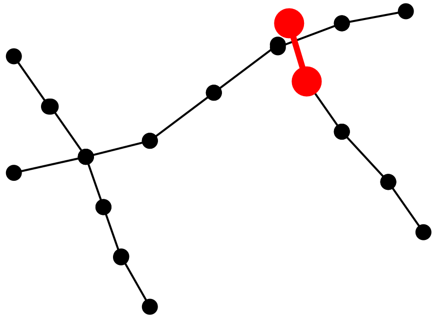
- ❑ Intersections do not change
- ❑ Sweep line algorithm can find all the intersections in $O(n \log n)$ time



[1] Berg, M. D., Cheong, O., Kreveld, M. V., and Overmars, M. Computational Geometry Algorithms and Applications Third Edition. 2008. ISBN 3-54077-973-6

Dynamic Set of Line Segments

- Road generation by procedural modeling
  - Line segments can be removed, added, and moved.
  - Given n line segments, we want to efficiently find whether a newly added line segment intersects the existing segments.



Ray Tracing

- Static scenes
  - Octree, Bounding Volume Hierarchy, Grid
    - $O(\log N)$ query time
    - $O(N\log^2 N)$ or $O(N^2)$ construction time

- Dynamic scenes
  - K-d tree
    - $O(\log N)$ query time
    - $O(N\log^2 N)$ or $O(N\log N)$ construction time

Dynamic Set of Line Segments

- Brute force approach
  - $O(n)$ to find an intersection for a given line segment
- Sweep line algorithm
  - $O(n \log n)$ to find an intersection for a given line segment.

K-d tree[2] for line segments

- How to choose a splitting line?
  - Spatial median splitting
    evenly split the space
  - Randomized algorithm
    randomly choose a line segment as a splitting line

[2] Bentley, J. L. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), 1975.

My Approach

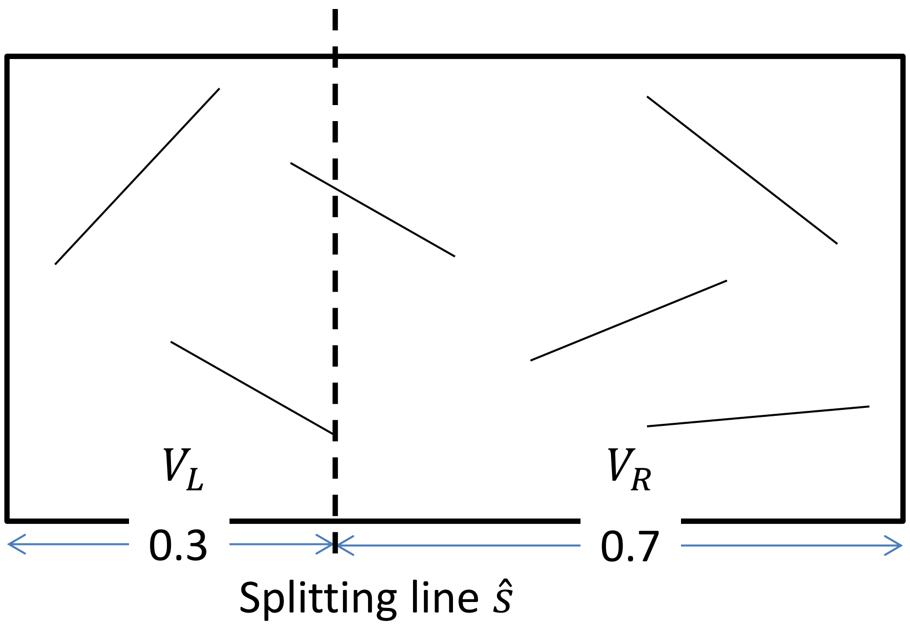
- Cost-based approach
  Choose a splitting line $\hat{s}$ which minimizes the following cost function.

$$C(\hat{s}) = P_L N_L + P_R N_R$$

P_L: probability to traverse the left child node
P_R: probability to traverse the right child node
N_L: size of the left sub tree
N_R: size of the right sub tree

Probability Estimation

- Geometric probability theory[3]
  - $Pr(V_{sub}|V) = \frac{Area(V_{sub})}{Area(V)}$



$$C(\hat{s}) = 0.3 \times 3 + 0.7 \times 4$$
$$= 3.7$$

[3] Glassner, A. 1989. An Introduction to Ray Tracing. *Morgan Kaufmann*, 1989. ISBN 0-12286-160-4.

Splitting Line Candidates

- The splitting lines that pass one of two end points of each line segment are the only candidates that we have to consider.
  - For any pair of splitting lines (s_0, s_1) between which N_L and N_R do not change, $C(s)$ is linear in the position of s.
  - $C(s)$ has its minima only at these candidates.

K-d Tree Construction

- Recursive call of the following function.

BuildKdTree($S = \{s_1, s_2, \cdots, s_n\}$ of segments**)**

1. **if** the number of S is less than a threshold
2. **then** create a leaf node consisting of the set S and return it.
3. **else**
4. Compute the cost for each segment.
5. Find the best segment $\hat{s}$ that minimizes the cost.
6. $S^- \leftarrow \{s \cap l(s_m)^- : s \in S\}$
7. T_L ←BuildKdTree(S^-)
8. $S^+ \leftarrow \{s \cap l(s_m)^+ : s \in S\}$
9. T_R ←BuildKdTree(S^+)
10. Create a tree consisting of a node that contains s_m and two sub trees T_L and T_R and return it.

Cost Computation

- Naïve algorithm
  - For each k-d tree node, $O(N)$ time to compute N_L and N_R of all the candidates.
  - Total computation time is $O(N^2)$
- Sweep line algorithm
  - Consider the end points of line segments as events
  - Computing N_L and N_R incrementally by sweeping achieves $O(N \log N)$ time
  - Total computation time is $O(N \log^2 N)$

Improved Cost Computation

- Sort the events E only one time
- Maintain the events order in $O(N)$ without re-sorting

UpdateEvents ($E, \hat{s}$**)**

1. **for all** $e \in E$
2. **if** the position of e is left of $\hat{s}$ **then**
3. $E_L \leftarrow E_L \cup e$
4. **else**
5. $E_R \leftarrow R_R \cup e$

- Total computation time is $O(N \log N)$

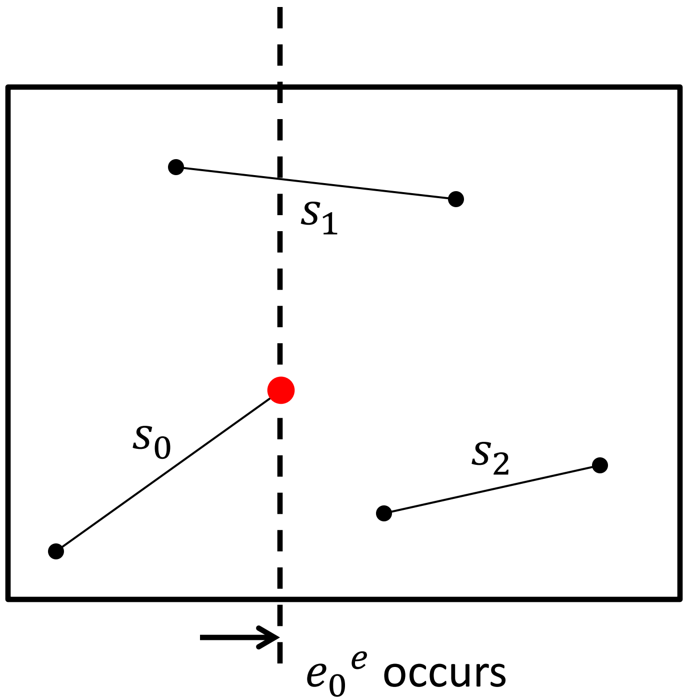
Classification

- For each line segment, determine whether it belongs to S^-, S^+, or both.
  - Start with $S^- = \emptyset$ and $S^+ = S$
  - Incrementally update S^- and S^+

Classify (N, E**)**

```
for all e ∈ E
    if e_type is the start of a line segment then
        S⁻ ← S⁻ ∪ s(e)
    else
        S⁺ ← S⁺ ∪ s(e)
```

Classification



$$E^x = \{e_0{}^s < e_1{}^s < e_0{}^e < e_2{}^s < e_1{}^e < e_2{}^e\}$$
$$E^y = \{e_0{}^s < e_2{}^s < e_2{}^e < e_0{}^e < e_1{}^e < e_1{}^s\}$$

$$S^- = \{s_0, s_1\},\ S^+ = \{s_0, s_1, s_2\}$$

$$S^- = \{s_0, s_1\},\ S^+ = \{s_1, s_2\}$$

K-d Tree Query

- Recursive call of the following function.

Query(T, s)

1. **if** s intersects the segment of the root node $s(T)$
2. **then** return true
3. **else**
4. **if** s is completely on the left half plane of $l\left(s(T)\right)$
5. **then** return query(T_L, s)
6. **if** s is completely on the left half plane of $l\left(s(T)\right)$
7. **then** return query(T_R, s)
8. (s_L, s_R)←split($s, l\left(s(T)\right)$)
9. return query(T_L, s_L) or query(T_R, s_R)

- Computation time is $O(\log N)$

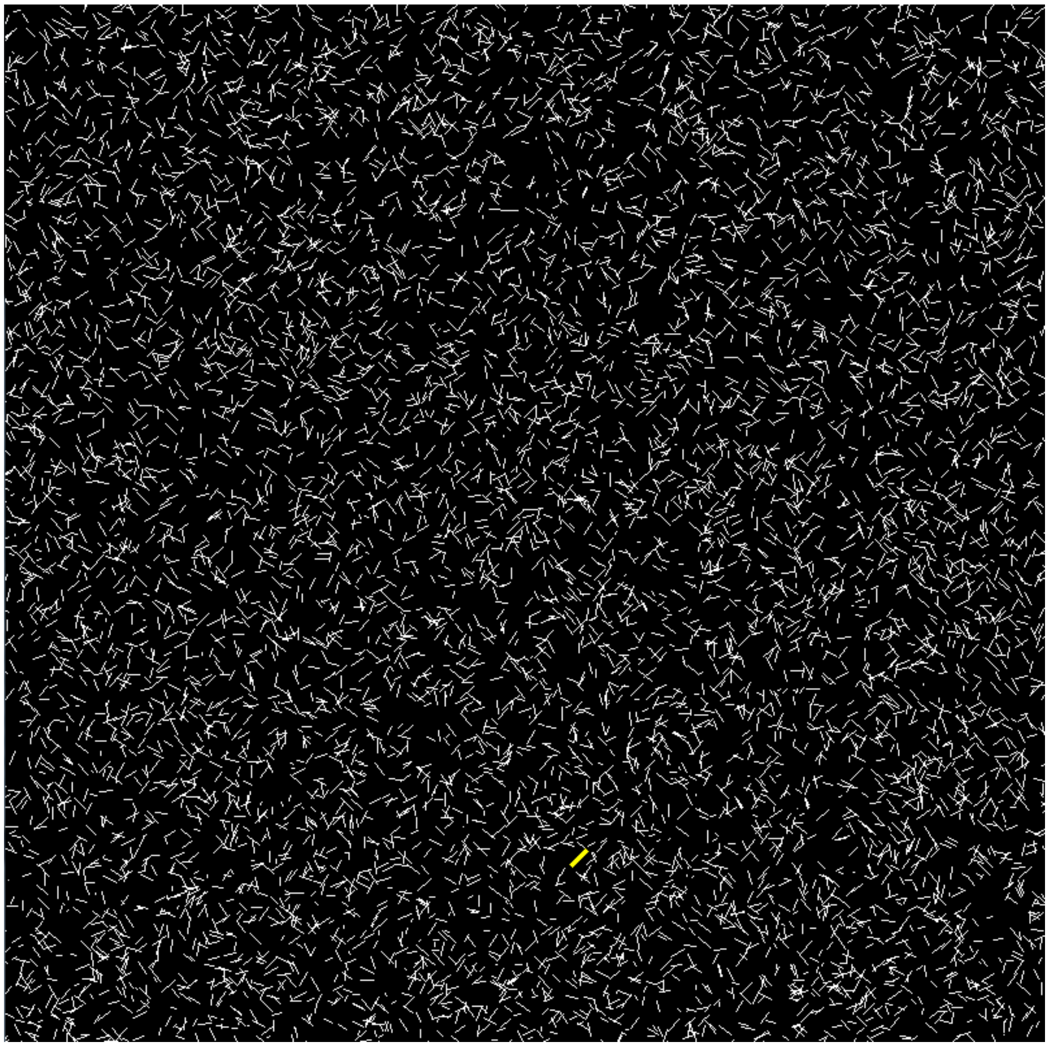
Experiments

- C++ implementation of k-d tree[*] using ACP library.

- 10,000 line segments with average 10 units long randomly spatially distributed over the 1000 units × 1000 units of 2D space.
- Compute the average computation time of 1000 queries.

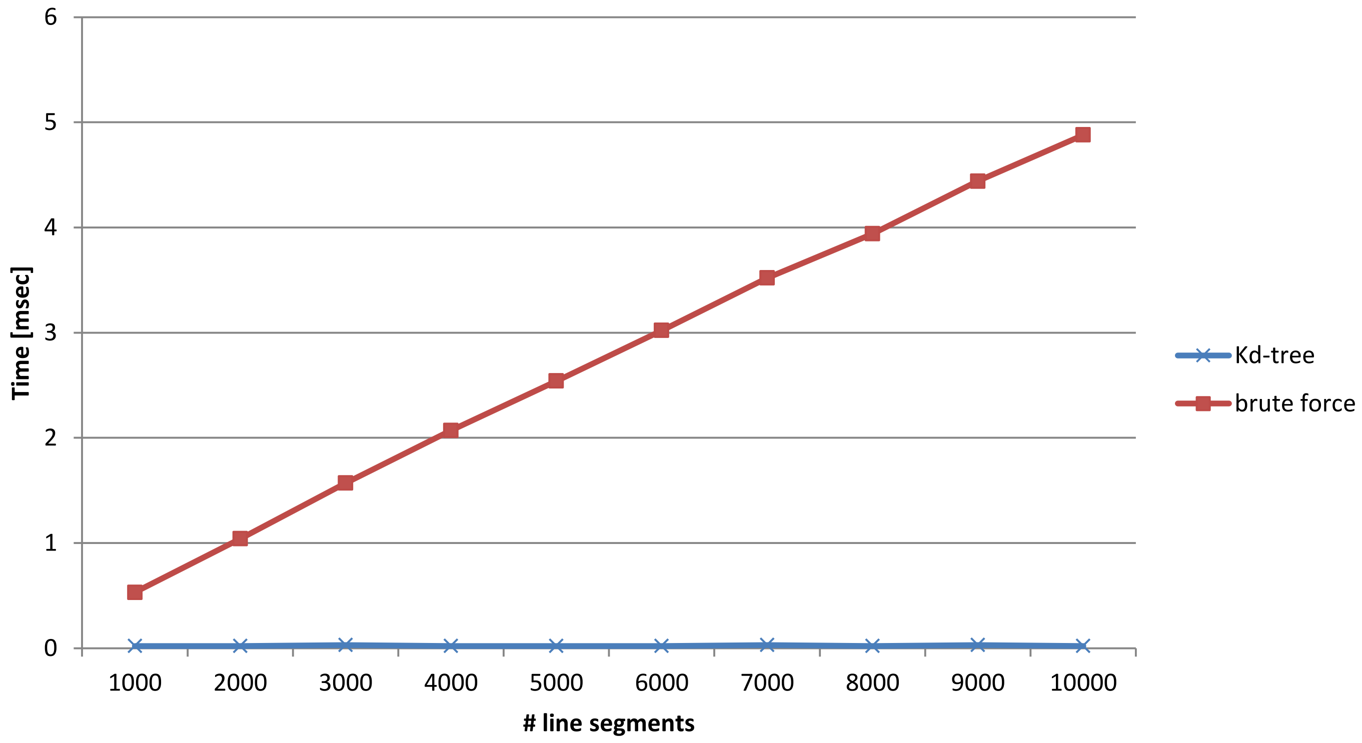
* For the tree construction, only the naïve approach was implemented.

One Example of Test Data



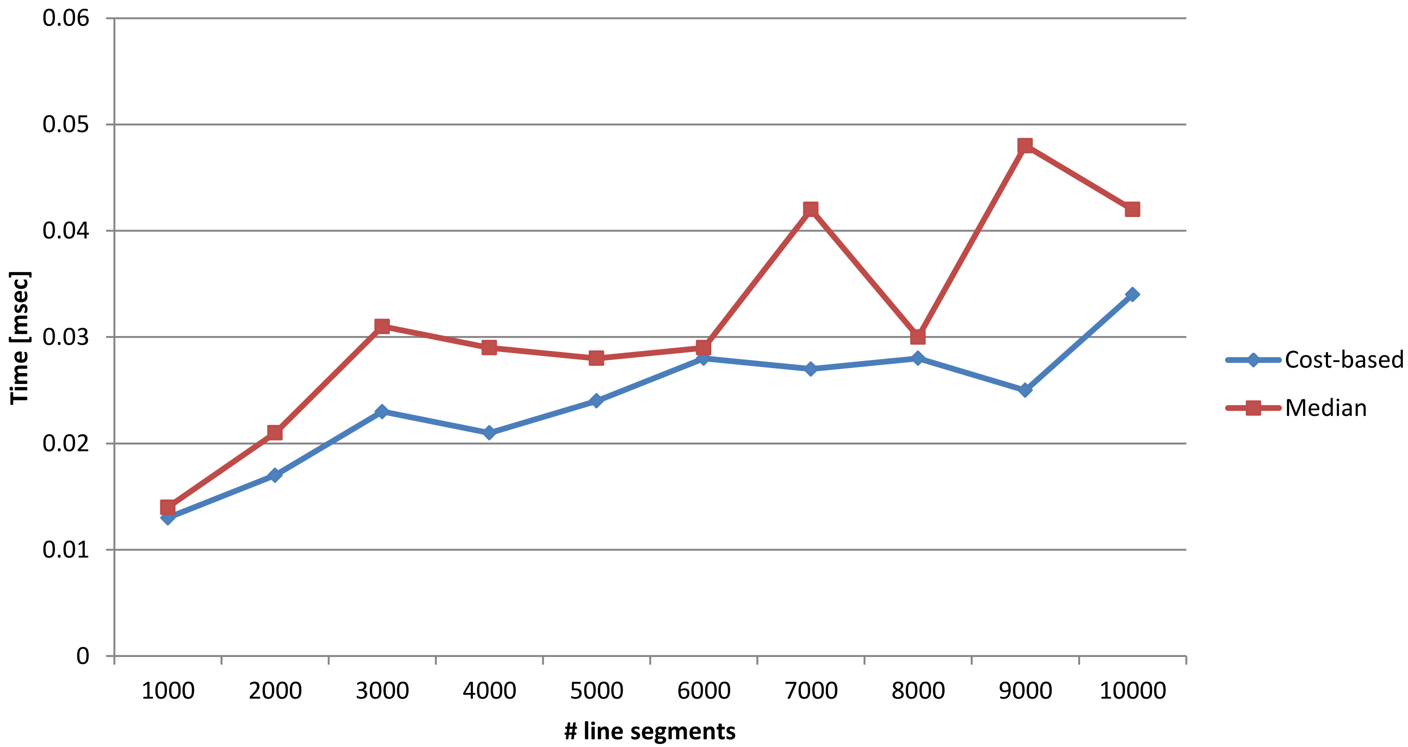
Results

- K-d tree versus brute force approach



Results

- Cost-based splitting versus spatial median splitting



Conclusions

- K-d tree for line segments using cost-based splitting
- Theoretical construction time is $O(N \log N)$
- Better query performance than the spatial median splitting by average 20%.

Thank you

Appendix

- Naïve algorithm for cost computation

$$T(N) = N^2 + 2T\left(\frac{N}{2}\right) = N^2 + \frac{N^2}{2} + \frac{N^2}{4} + \cdots + \frac{N^2}{2^{\log N}} + 2^{\log N+1}$$
$$= N^2\left(2 - \frac{1}{N}\right) + 2N$$
$$= O(N^2)$$

- Sweep line algorithm for cost computation

$$T(N) = N\log N + 2T\left(\frac{N}{2}\right) = N\log N + N(\log N - 1) + \cdots + N(1) + N$$
$$= N\frac{\log N(\log N+1)}{2} + N$$
$$= O(N\log^2 N)$$

- Improved algorithm

$$T(N) = N + 2T\left(\frac{N}{2}\right) = N + N(\log N - 1) + \cdots + N(1) + N$$
$$= \sum_{i=0}^{\log N} N$$
$$= O(N\log N)$$