HIERARCHICAL REPRESENTATIONS OF LINE DATA

Hanan Samet

Computer Science Department and
Center for Automation Research and
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742
e-mail: hjs@umiacs.umd.edu

BACKGROUND

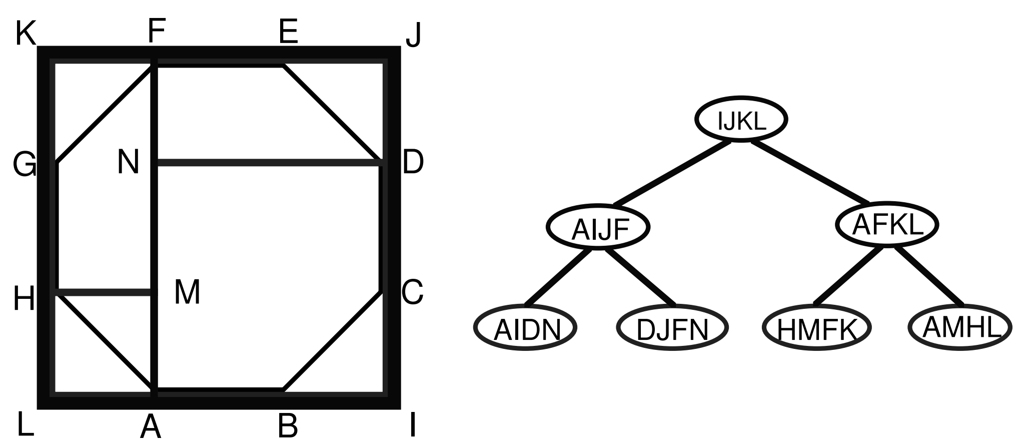
- Goals
  1. hierarchical representation of both a region and its boundary
  2. no thickness associated with a line segment
  3. exact representation
- Vector data vs: raster data
  1. vector keeps track of the boundary
     - polyline: chain of points with implicit lines between them
     - polygon: an area bounded by a closed polyline
     - the distance from a polygon to a point inside it is zero while the distance from the point to the polyline that forms the boundary of the polygon is nonzero
     - chain code: approximation by unit vectors in 4 (or 8) principal directions
     - compact
  2. raster keeps track of the pixels intersected by line segments
     - requires much space as increase the resolution
- Advantages of hierarchical representations:
  1. enable focussing computational resources where they are needed
  2. permit a quick exit (termination) when no more information can be gained

EXAMPLES OF QUERIES ON LINE SEGMENT DATABASES

- Queries about line segments
  1. All segments that intersect a given point or set of points
  2. All segments that have a given set of endpoints
  3. All segments that intersect a given line segment
  4. All segments that are coincident with a given line segment
- Proximity queries
  1. The nearest line segment to a given point
  2. All segments within a given distance from a given point (also known as a range or window query)
- Queries involving attributes of line segments
  1. Given a point, find the closest line segment of a particular type
  2. Given a point, find the minimum enclosing polygon whose constituent line segments are all of a specified type
  3. Given a point, find all the polygons that are incident on it

BSPR (Burton)
Binary Searchable Polygonal Representation

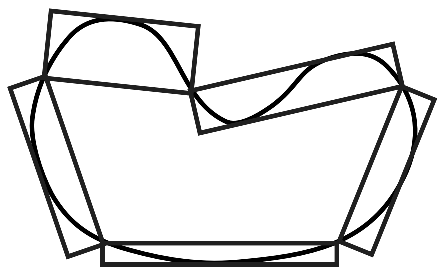
- Bottom-up hierarchical curve approximation using upright rectangles
- Decompose curve into simple sections
- Simple section = segment of a curve whose *x* and *y* coordinate values are monotonic
  1. analogous to a 2-d suite of values (i.e., run) in merge sorting
  2. preferable to joining successive pairs of line segments
- Construct a tree by pairing adjacent simple sections to yield compound sections
- Process terminates when the entire curve can be approximated by one compound section
- Similar to an R-tree where the objects are monotonic curve segments thereby yielding an object hierarchy
- Ex:

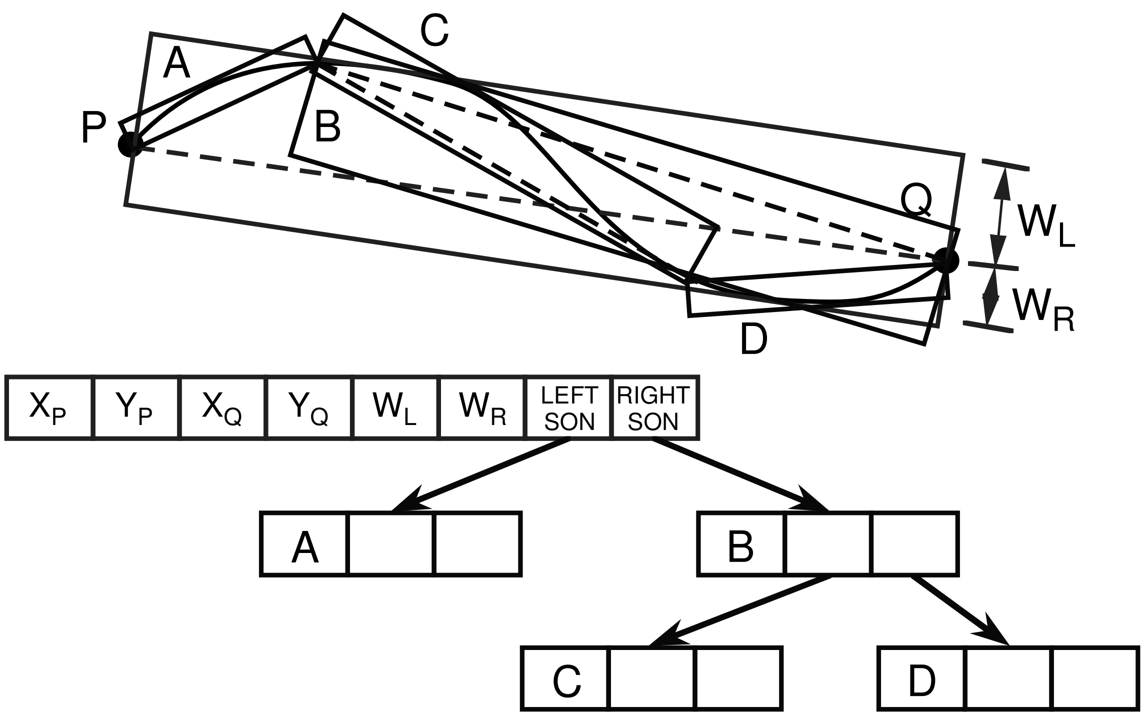


- Operations:
  1. point-in-polygon determination
  2. polygon intersection

STRIP TREE (Ballard, Peucker)

- Top-down hierarchical curve approximation
- Rectangle strips of arbitrary orientation
- Assume curve is continuous
- Ex:

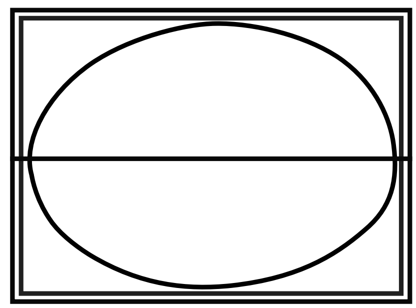




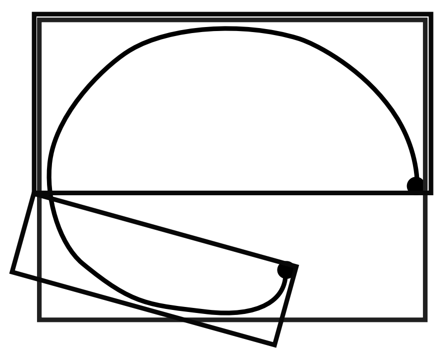
- *Contact points* = where the curve touches the box
  1. not tangent points
  2. curve need not be differentiable - just continuous
- Terminate when all rectangles are of width $\leqslant$ W

SPECIAL CASES

1. Closed curve



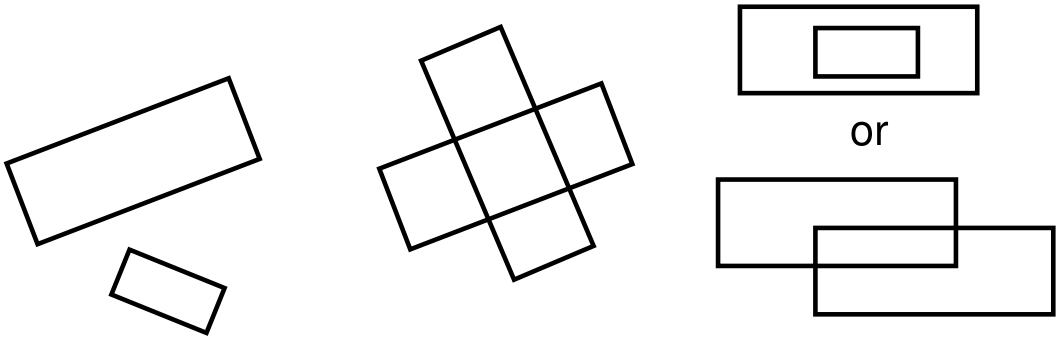
2. Curve extends beyond its endpoints



- enclosed by a rectangle
- split into two rectangular strips

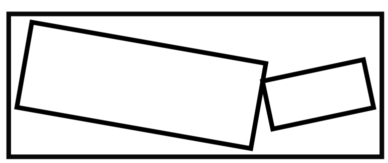
APPLICATIONS

1. Curve intersection



NULL CLEAR POSSIBLE

2. Union of two curves

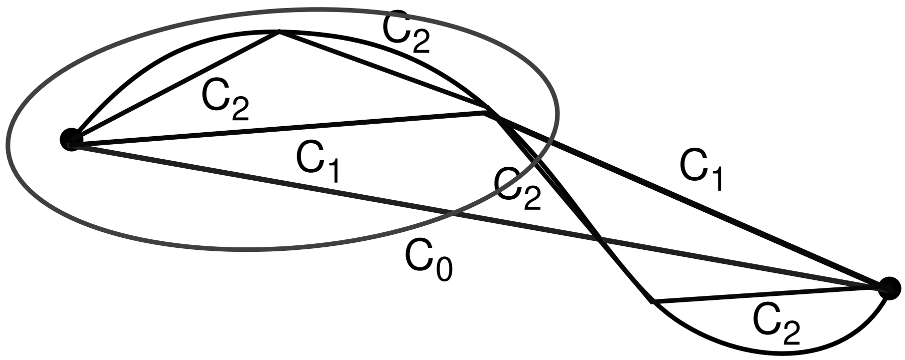


- not possible as the result may fail to be continuous

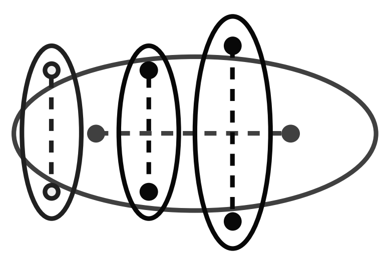
3. Others
   - length
   - area of a closed curve
   - intersection of curves with areas
   - etc.

ARC TREE (Günther)

- Recursive decomposition based on arc length
- Complete binary tree
- n^{th} level approximation has 2^n elements
- Decomposition process stops when straight line approximation of each subarc is within a given tolerance
- Drawback: computing arc length requires evaluating an elliptical integral
- Ex:



- Use ellipses as bounding boxes instead of rectangles
  1. place the foci at the endpoints of each subarc
  2. the principal axis is the length of the subarc
  3. advantage over the strip tree as no need for special provision for closed curves or curves that extend past their endpoints
- Two curves are guaranteed to intersect if their bounding ellipses intersect and the two foci of each ellipse are external to the other ellipse



POSSIBLY YES

- Note that intersection of the principal axes of the bounding ellipses is not enough to guarantee intersection

COMPARISON: BSPR, STRIP TREE, ARC TREE

1. All independent of grid system in which they are embedded

2. BSPR and arc tree do not require special treatment for closed curves

3. Uniqueness:
   - no for BSPR as depends on the side of the start
   - no for all when curve is closed

4. BSPR not as flexible as the strip tree or arc tree
   - resolution of approximation is fixed since width of the approximating rectangles cannot be varied
   - resolution drawback is same as that associated with hexagonal-based tiling methods

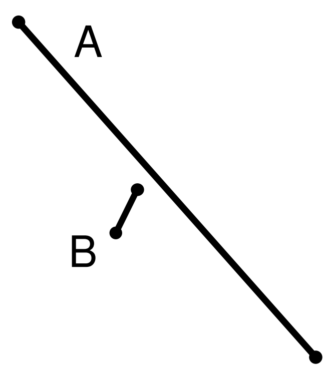
APPLICATION OF THE MX QUADTREE (Hunter)

- A record in a DBMS may be considered a point in multi-dimensional space

  Ex: a line can be a point in 4-d with (x1,y1,x2,y2)

- Good for queries about line segments
- Not good for proximity queries since points not on the line (i.e., the query point) are not mapped into the 4-d space
- The representative points of two lines that are physically close to each other in 2-d space may be very far from each other in 4-d space

example:



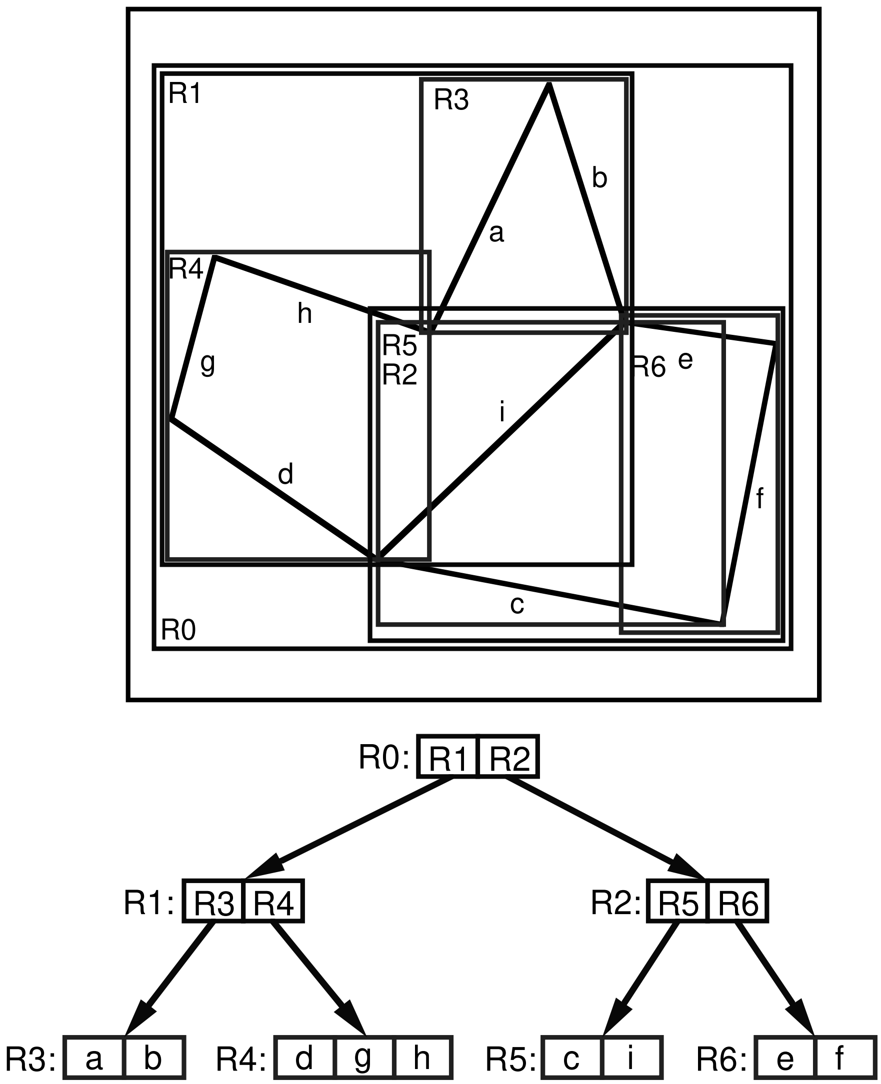
- Problem is that the transformation only transforms the space occupied by the lines of the 2-d space and not the rest of the space (e.g., the query point)

ONE APPROACH

- A data structure based upon spatial occupancy is the best solution
  1. decompose the space from which the data is drawn into regions called buckets
  2. with each block only store whether or not it is occupied by the object or a part of it
     - i.e., a pointer to a descriptor of the object
  3. interest is in methods that are designed specifically for the spatial data type being stored
- Four basic approaches to decomposing space
  1. minimum bounding rectangles
     - non-disjointness forces longer search since several bounding rectangles may cover an object, yet the object will only be associated with one box
  2. disjoint cells
     - object associated with possibly many cells thereby causing it to be represented more than once!
     - Q: how to report each object only once!
  3. uniform grid
  4. quadtrees

MINIMUM BOUNDING RECTANGLES

- Objects grouped into hierarchies, stored in another structure such as a B-tree
- Does not result in disjoint decomposition of space
- Object has single bounding rectangle, yet area that it spans may be included in several bounding rectangles
- Examples include the R-tree and the R*-tree
- Order (m,M) R-tree
  1. between $m \leqslant \lceil M/2 \rceil$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node

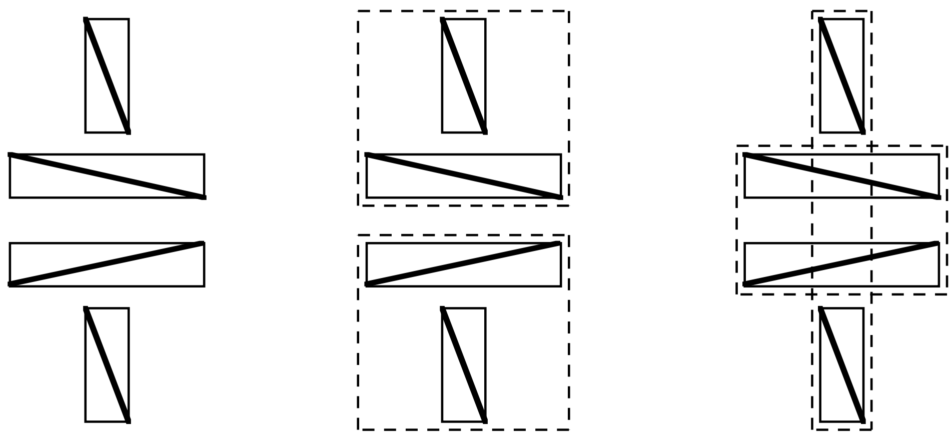


INSERTING A LINE SEGMENT IN AN R-TREE

1. Determine the appropriate node
   - traverse the tree starting at the root
   - choose the subtree whose minimum bounding rectangle needs to be enlarged the least (areawise)
2. Once leaf node has been determined
   - add the line segment to the node
   - check if insertion causes overflow
   - split node if necessary and redistribute records
   - propagate split up the tree

SPLITTING AN R-TREE NODE ON OVERFLOW

1. Reduce likelihood that both nodes are examined in subsequent searches
   - minimize area common to both nodes (overlap)
2. Reduce likelihood the node will be visited in subsequent searches
   - minimize total area of covering rectangles (coverage)



Line Segments   Goal 1   Goal 2

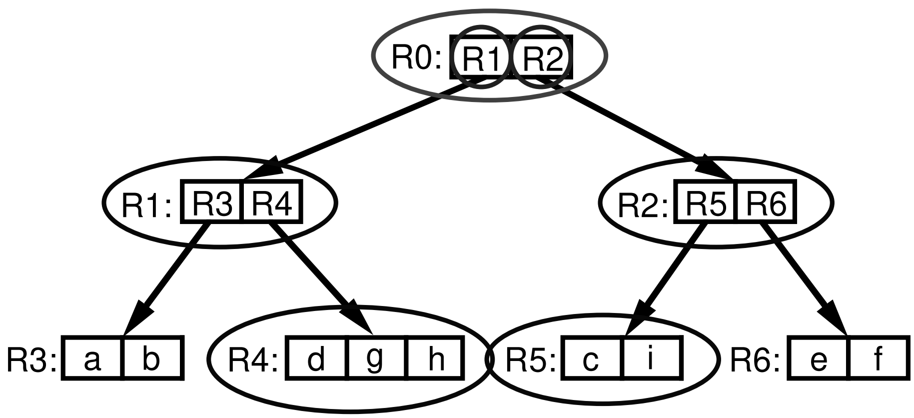
R*-TREE

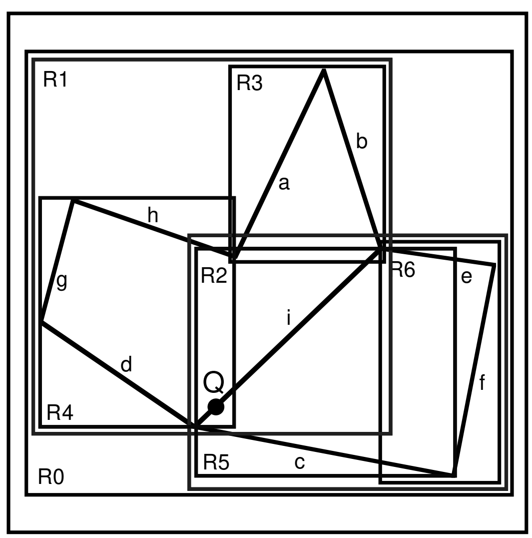
- R-tree variant with more sophisticated node insertion and splitting algorithms
- Node insertion
  1. choose the subtree whose minimum bounding rectangle has the minimum increase of amount of overlap with its brothers (i.e., the other nodes pointed at by its father) : satisfies goal 1
  2. better than choosing the node whose minimum bounding rectangle would have to be enlarged the least : satisfies goal 2
  3. rationale: reduce likelihood that remaining nodes are examined in subsequent searches
- Node splitting - a node has $M+1$ objects
  1. determine axis (x or y)
     - examine all possible vertical and horizontal splits (so each resulting node has at least m and at most $M+1-m$ bounding rectangles)
     - choose the split axis for which the sum of the perimeters of the two resulting nodes from all the possible splits is minimized
  2. determine the position of the split
     - $M-2m+2$ possibilities
     - choose the split that minimizes the overlap between the two new nodes

SEARCHING FOR A POINT OR LINE SEGMENT IN AN R-TREE

- May have to examine many nodes since a line segment can be contained in the covering rectangles of many nodes yet its record is contained in only one leaf node (e.g., i in R2, R3, R4, and R5)

Ex: Search for a line segment containing point Q





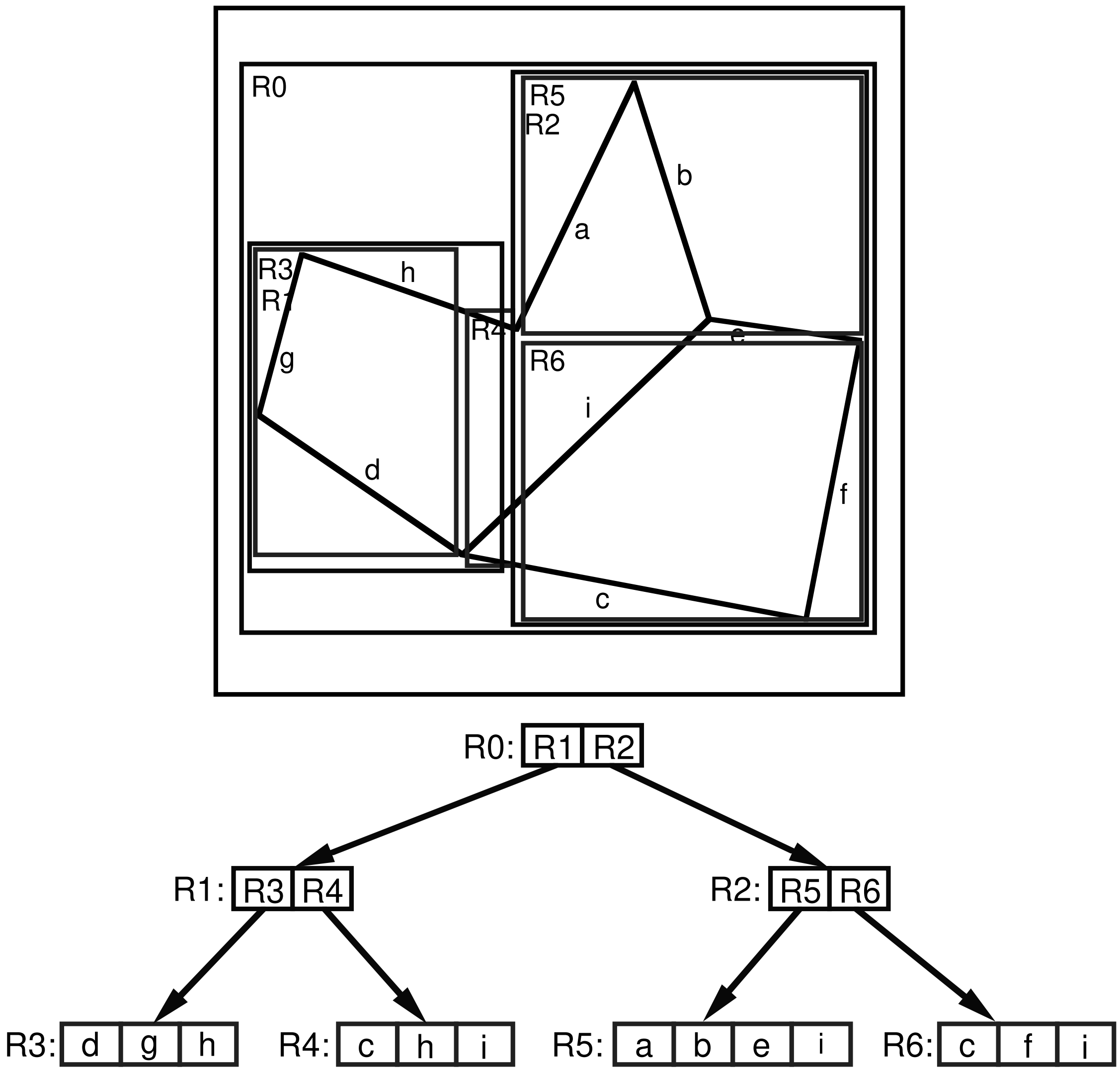
- Q is in R0
- Q can be in both R1 and R2
- Searching R1 first means that R4 is searched but this leads to failure even though Q is part of i which is in R4
- Searching R2 finds that Q can only be in R5

DISJOINT CELLS

- Objects decomposed into disjoint subobjects; each subobject in different cell
- Techniques differ in degree of regularity
- In order to determine area covered by object, must retrieve all cells that it occupies
- Deletion is achieved in a similar manner

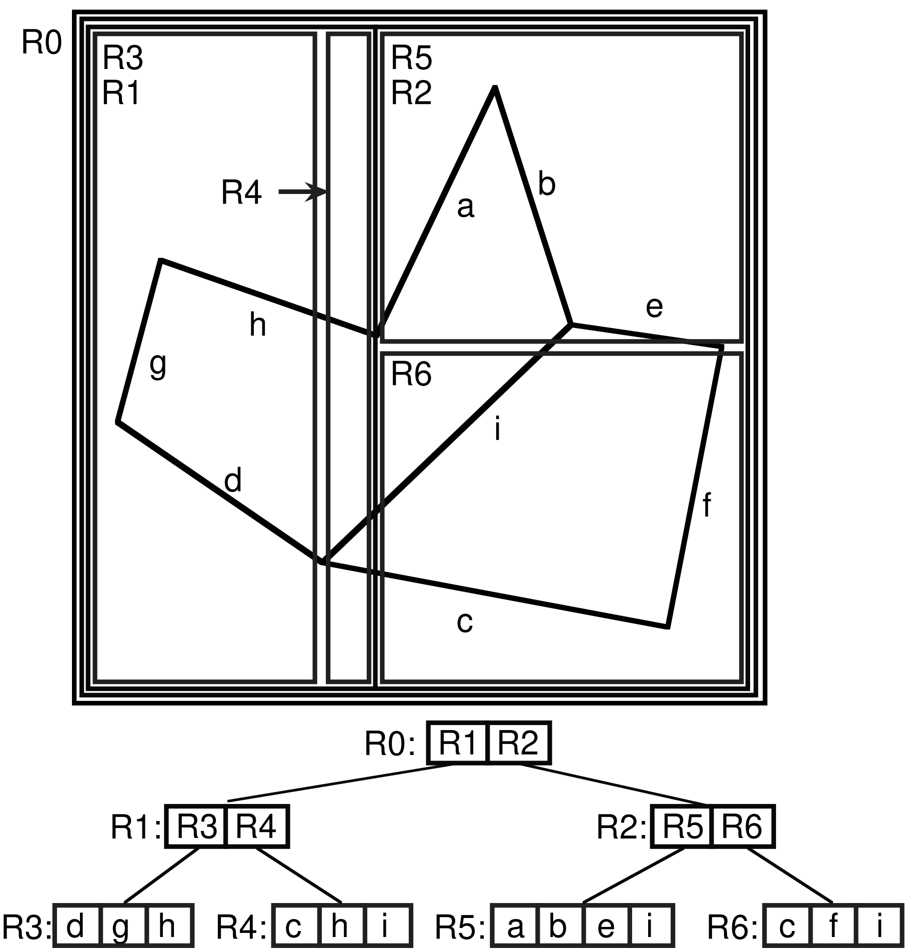
R+-TREES

- Extension of the k-d-B-tree
- Motivated by a desire to avoid overlap among the minimum bounding rectangles
- Pages are not guaranteed to be *m/M* full without very complicated node insertion and deletion procedures



DISJOINT CELLS: K-D-B-TREE

1. Same principle as R$^+$-tree but developed much earlier
   - in R$^+$-tree, rectangle at depth *i* is a minimum bounding rectangle of contained rectangles at depth *i*+1
   - aggregates blocks of k-d tree partition of space rather than minimum bounding rectangles into nodes of finite capacity
   - both use minimum bounding rectangles for objects at deepest level
2. R$^+$-tree reduces number of false hits compared to k-d-B-tree
3. Same drawback of duplicate reporting as in R$^+$-tree

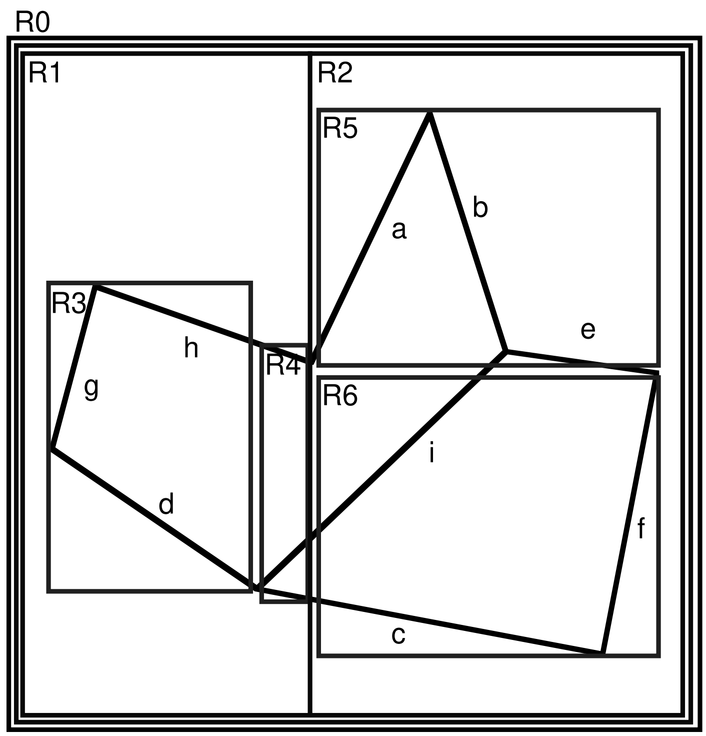


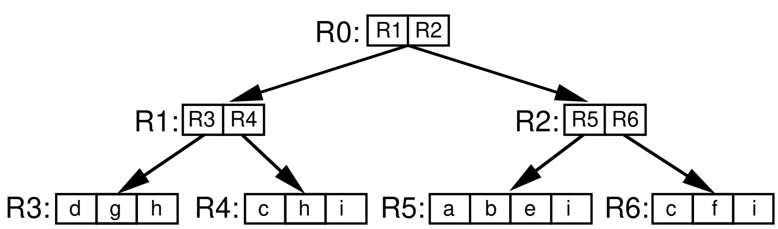
COMPARISON OF R+-TREES AND K-D-B-TREES

1. k-d-B-tree has faster building times than R+-tree
2. Storage costs are the same
3. R+-tree minimizes dead space
   - point search queries are thus faster as failure is detected earlier
   - range and nearest line segment queries are faster as the minimum bounding rectangles lead to more pruning

HYBRID R+-TREES

- Use minimum bounding rectangle for the line segments in the leaf nodes while not in the nonleaf nodes
- Simplified construction algorithm
- Absence of minimum bounding rectangles in the nonleaf nodes is not so costly given that the number of leaf nodes is much greater than the number of nonleaf nodes
- Use this variant in all tests



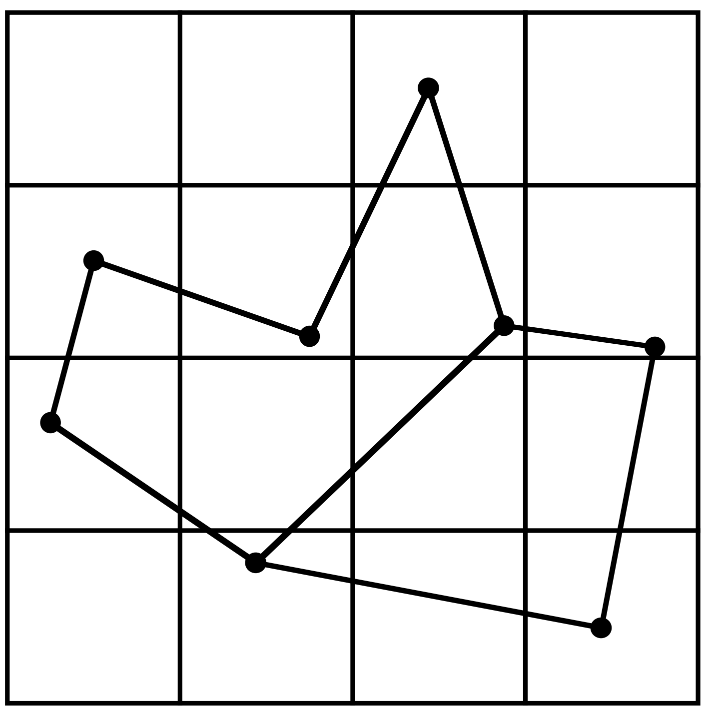


INSERTING A LINE SEGMENT IN A HYBRID R+-TREE

- Place line segment in every leaf node that it intersects
- Check if nodes should be split if overflow takes place
  1. split so that the resulting total number of portions of line segments (bounding rectangles if nonleaf node) is minimized
  2. must try all possible horizontal and vertical split lines
     - for each split line calculate number of line segments (or bounding rectangles) intersected by the line
     - select the line with the minimum number of intersections
     - in case of a tie, select the one that yields the most even distribution of line segments (or bounding rectangles) among the two constituent nodes

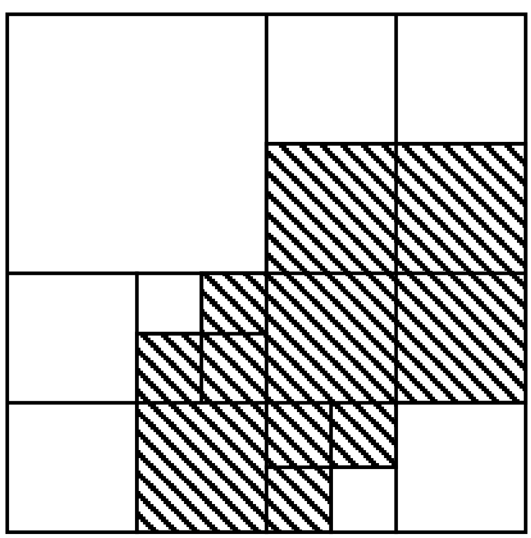
UNIFORM GRID

- Ideal for uniformly distributed data
- Supports set-theoretic operations
- Spatial data (e.g., line segment data) is rarely uniformly distributed



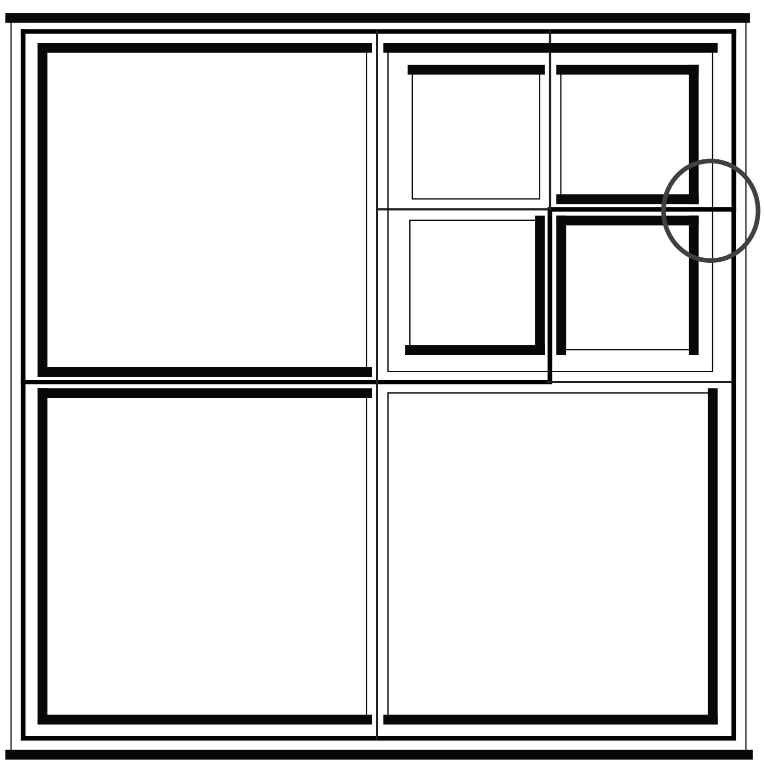
QUADTREES

- Hierarchical variable resolution data structure based on regular decomposition
- Many different decomposition schemes and applicable to different data types:
  1. points
  2. lines
  3. regions
  4. rectangles
  5. surfaces
  6. volumes
  7. higher dimensions including time
- Supports set-theoretic operations among different maps and different data types
- Shape is usually independent of order of inserting the data
- Ex: region quadtree



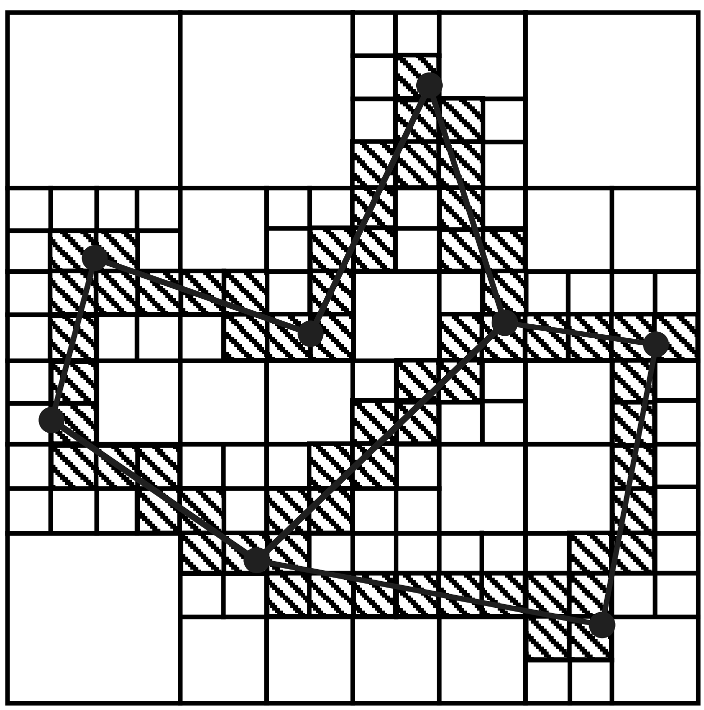
LINE QUADTREE

- Hierarchical encoding of both the interior and the boundary of a region
- Only works for polygonal data with orthogonal edges
- Repeatedly subdivide until the leaf nodes represent regions with no line segments passing through their interiors
- At each leaf node a code indicates which of its four sides forms a boundary (not a partial border of a region)
- Hierarchical because the information is also recorded for nonleaf nodes as long as there is no T-junction at the interior side of the boundary of the block corresponding to the node



MX QUADTREE

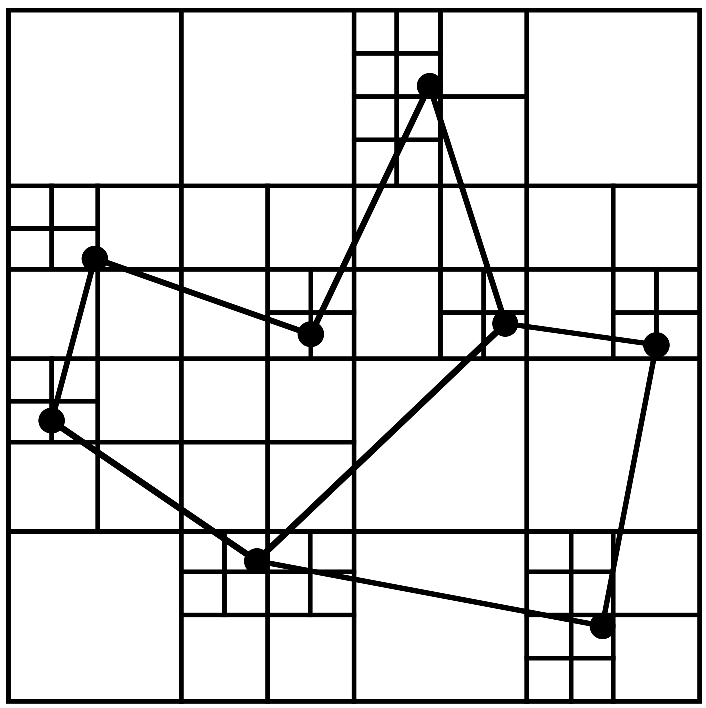
- Represent the boundary as a sequence of BLACK pixels in a region quadtree
- Useful for a simple digitized polygon (i.e., non-intersecting edges)
- Three types of nodes
  1. interior - treat like WHITE nodes
  2. exterior - treat like WHITE nodes
  3. boundary - the edge of the polygon passes through them and treated like BLACK nodes
- Disadvantages
  1. a thickness is associated with the line segments
  2. cannot have more than 4 lines meet at a point



- Raster-to-vector conversion is very difficult

EDGE QUADTREES (Shneier, Warnock)

- Repeatedly subdivide until block contains a single curve that can be approximated by a single straight line
- Special nodes at vertices which are at the maximum level of resolution
- Ex:



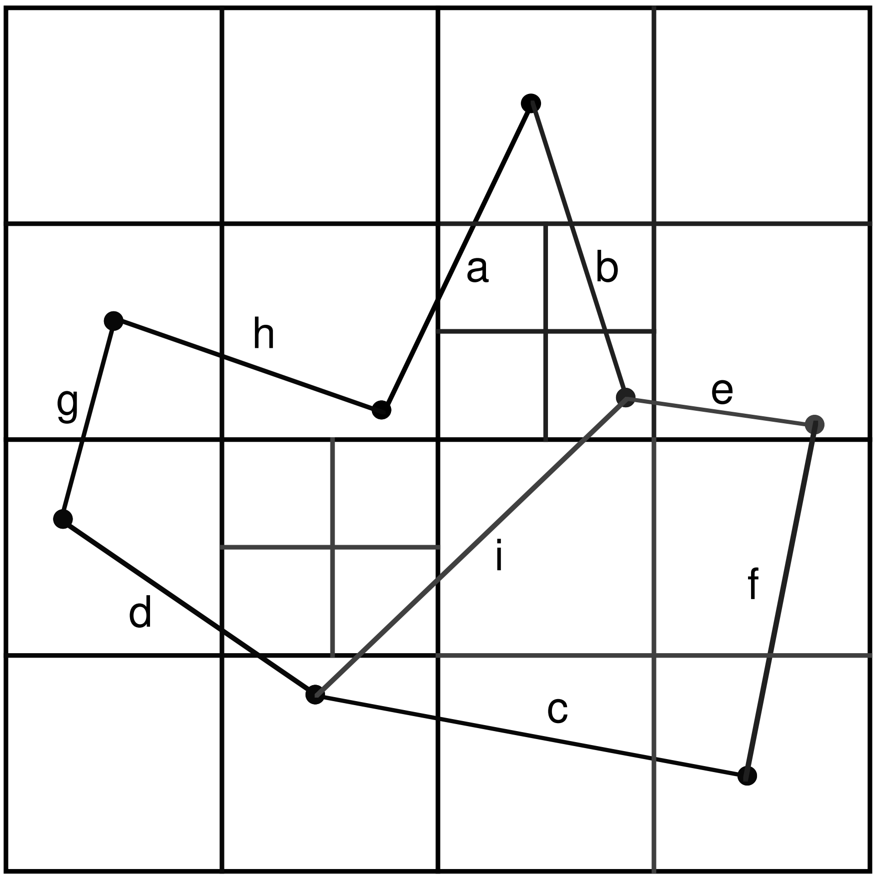
- Martin's least square quadtree: curve in block can be approximated by *k* straight lines within a least square tolerance

PM QUADTREES

- Store a collection of line segments
- Divide space into blocks using some rule
  1. vertex-based
     - limit number of vertices per block (usually one)
  2. edge-based
     - involves number of edges per block
- Store line segments intersecting each block
- Generally require variable size nodes
- Enables storing vector data (i.e., quadtree is not limited to raster data)

PM1 QUADTREE

- Vertex-based (one vertex per block)



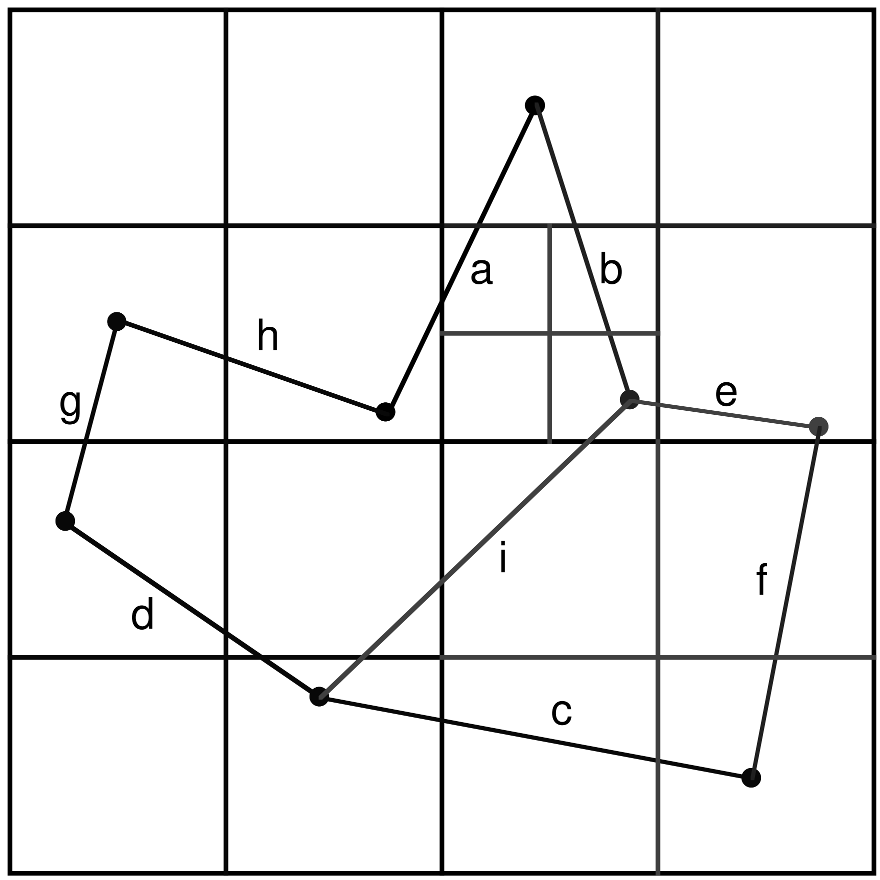
DECOMPOSITION RULE:

Partitioning occurs when a block contains more than one segment unless all the segments are incident at the same vertex which is also in the same block

- Shape independent of order of insertion

PM2 QUADTREE

- Vertex-based (one vertex per block)



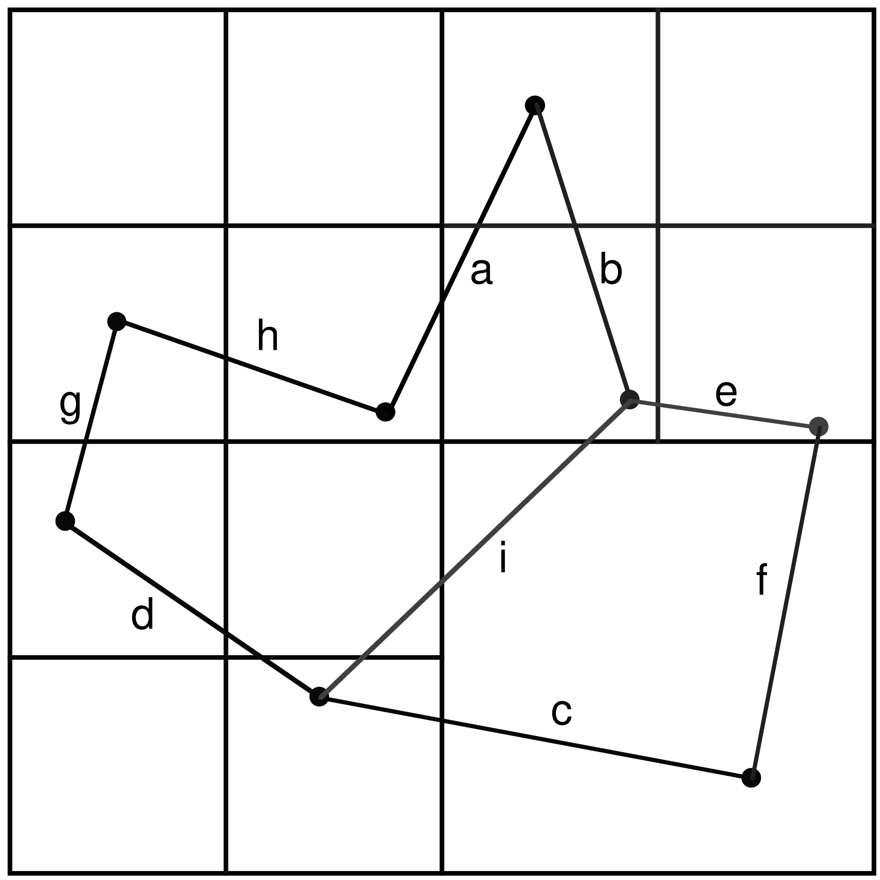
DECOMPOSITION RULE:

Partitioning occurs when a block contains more than one line segment unless <u>all</u> the segments are incident at the <u>same</u> vertex (the vertex can be in another block!)

- Shape independent of order of insertion

PM3 QUADTREE

- Vertex-based (one vertex per block)



DECOMPOSITION RULE:

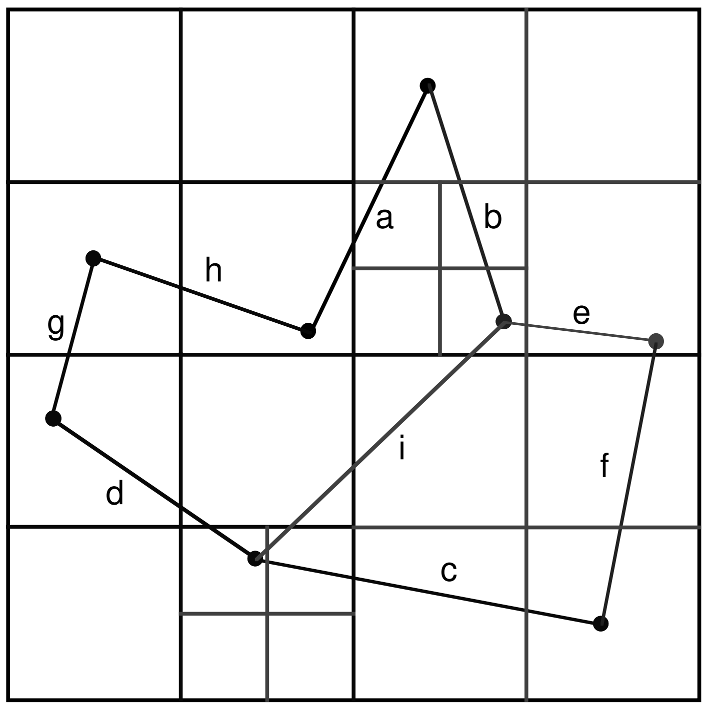
Partitioning occurs when a block contains more than one vertex (i.e., a PR quadtree with edges)

- Shape independent of order of insertion

PMR QUADTREE

- Edge-based
- Avoids having to split many times when two vertices or lines are very close as in PM1 quadtree
- Probabilistic splitting and merging rules
- Uses a splitting threshold value — say N

Ex: $N = 2$



DECOMPOSITION RULE:

Split a block *once* if upon insertion the number of segments intersecting a block exceeds N

Merge a block with its siblings if the total number of line segments intersecting them is less than N

- Merges can be performed more than once
- Does not guarantee that each block will contain at most N line segments
- Splitting threshold is not the same as bucket capacity
- Shape depends on order of insertion

ORGANIZATION OF LEAF NODES IN PM QUADTREES

1. PM1 quadtree
   - one segment
   - set of segments incident at a vertex
2. PM2 quadtree
   - one segment
   - set of segments incident at a vertex
3. PM3 quadtree
   - up to 7 classes of information describing the edges passing through it
     a. incident at the vertex
     b. crossing the NW, NE, SE, and SW corners of the node's block
     c. crossing the EW and NS sides of the node's block
   - each class can be stored in a balanced binary tree but a sequential list is usually sufficient unless the number is unusually high
4. PMR quadtree
   - set of segments passing through the block
   - store in a sequential list but a splitting threshold increases (i.e., > 8), performance suffers and probably preferable to sort
     a. slope
     b. *x* and/or *y* intercept values
     c. orientation around a common point
     d. other?

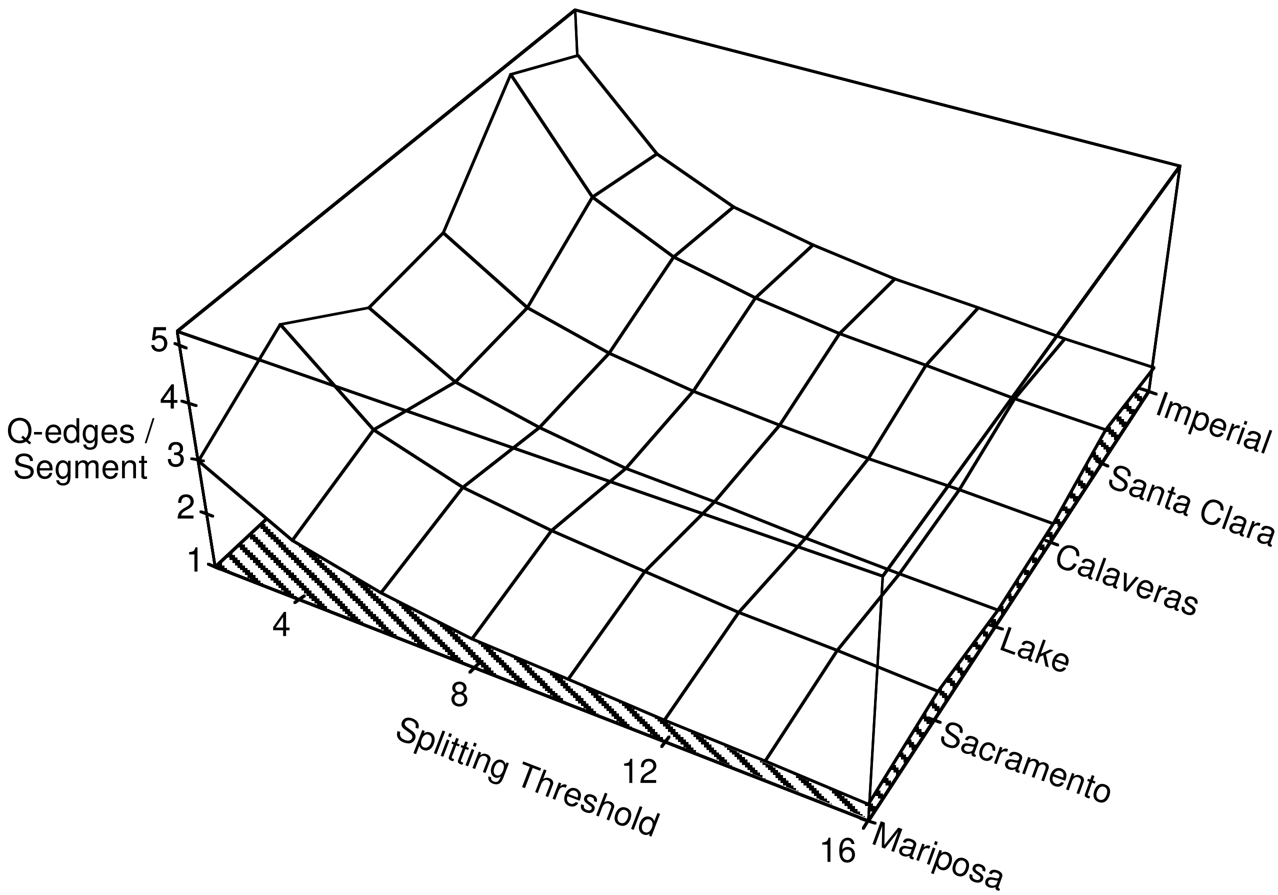
SAMPLE CALIFORNIA TIGER DATASETS

County	Segments	Q-edges / Segment	Blocks	Q-edges / Block
Mariposa	92843	1.58	42295	4.43
Sacramento	104502	1.92	52918	4.31
Lake	107708	1.61	49462	4.41
Calaveras	112529	1.65	51781	4.36
Santa Clara	113564	2.14	62227	4.26
Imperial	133049	1.93	74188	4.27

- Splitting threshold value of 8
- Image resolution (i.e., map size) is 16K x 16K

Q-EDGES PER LINE SEGMENT

- The average number of q-edges per line segment in the PMR quadtree as a function of the splitting threshold
- Image resolution size (i.e., map size) is 16K x 16K

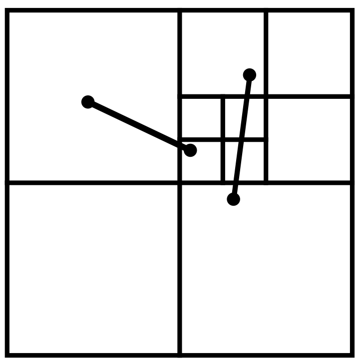


- Observe that as the splitting threshold increases, most of the line segments are contained in one or two buckets thereby explaining the asymptotic value of 1.4

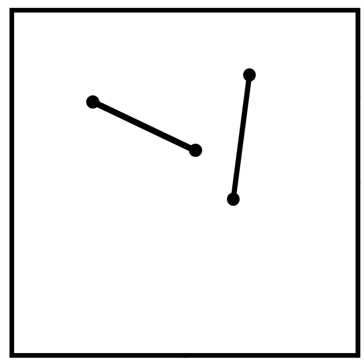
ADVANTAGES OF EDGE-BASED METHODS

- The decomposition is focussed where the line segments are the densest
- Handles the situation that several non-intersecting lines are very close to each other

  Example:



PM1

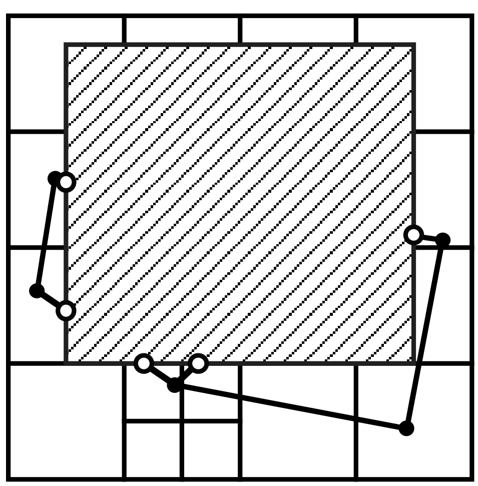


PMR (N=2)

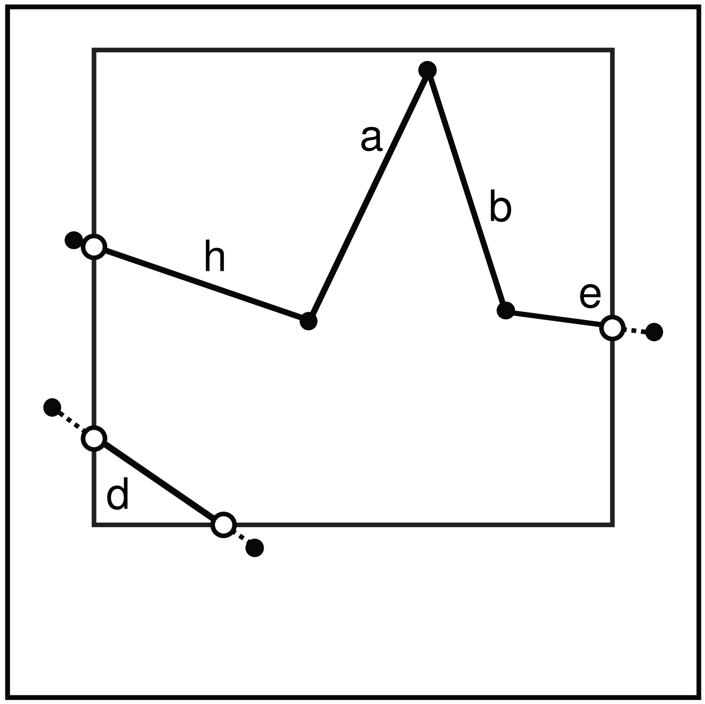
- Good average behavior in terms of node occupancy
- Able to represent nonplanar line segment data

CONSISTENCY OF PM APPROACH

- Stores lines exactly
  1. not a digitized representation
  2. no thickness associated with line segments
- Updates can be made in a consistent manner - i.e., when a vector feature is deleted, the database can be restored to the state it would have been in had the deleted feature never been inserted
- Each line segment is represented by a pointer to a record containing its endpoints
- Uses the decomposition induced by the quadtree to specify what parts of the line segments (i.e., q-edges) are actually present
- The line segment descriptor stored in a block only implies the presence of the corresponding q-edge - it does not mean that the entire line segment is present as a lineal feature
- Useful for representing fragments of lines such as those that result from the intersection of a line map with an area map
- Ex:



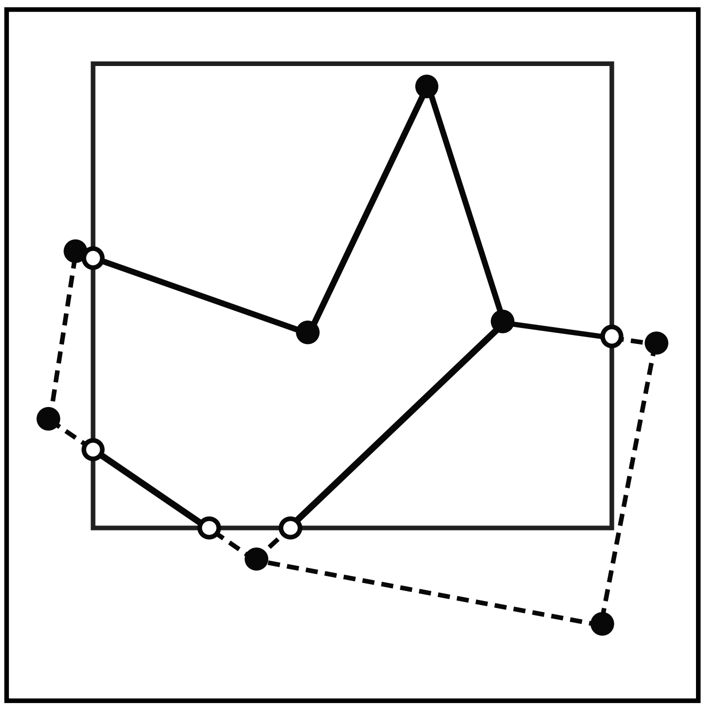
FRAGMENTS



- When a line segment is split or clipped, a line segment *fragment* is produced
- Fragment ≡ any connected piece of a line segment (including the whole)
- Fragments are represented by a collection of q-edges

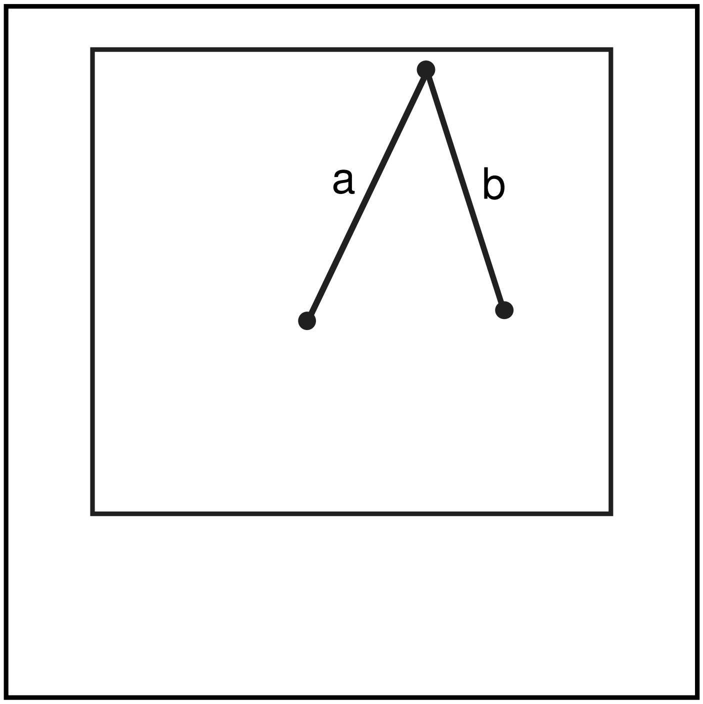
STORING FRAGMENTS USING PMR QUADTREES

- Split to localize cutpoints
- Use PMR splitting rule to prevent excessive node occupancy
- Ex:



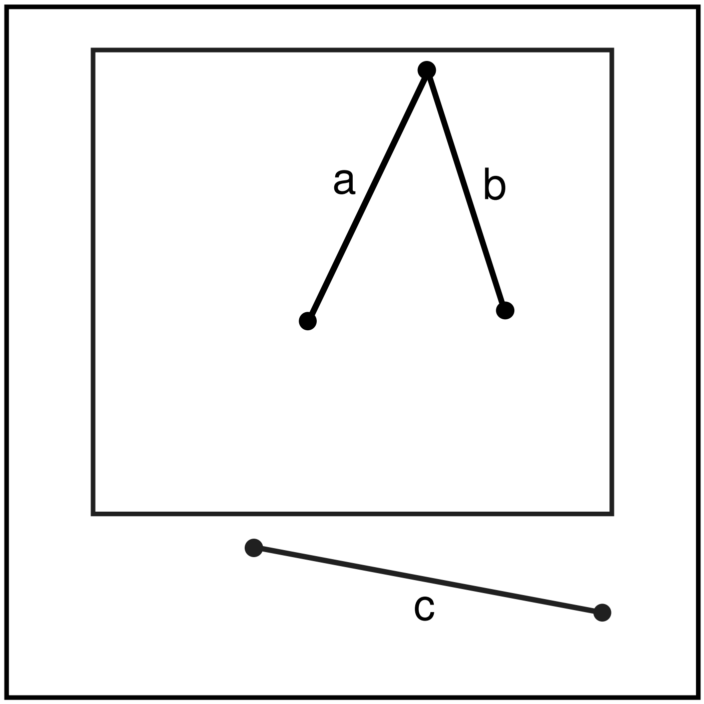
- Assume a splitting threshold value of 2

INSERTION OF LINE SEGMENTS a AND b



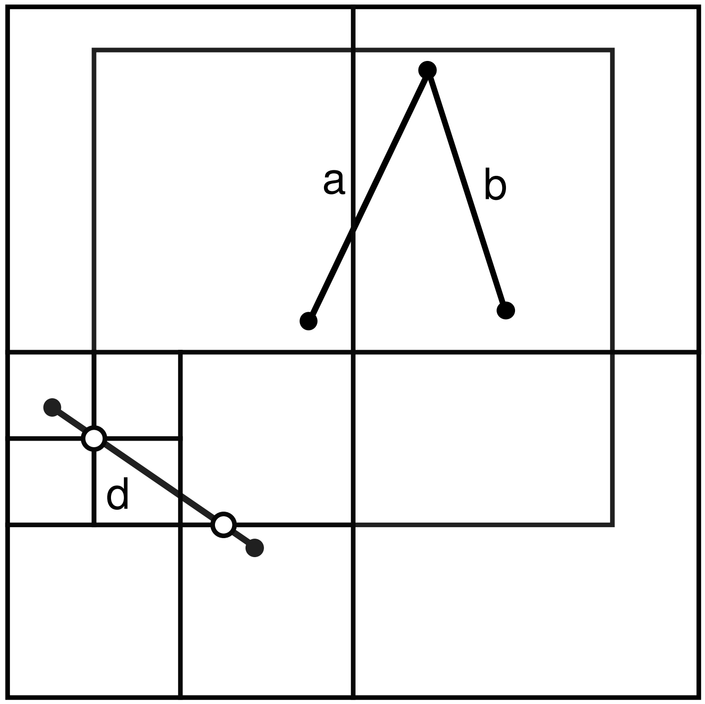
- No splitting as a and b are entirely in the window and the splitting threshold has not been exceeded

INSERTION OF LINE SEGMENT c



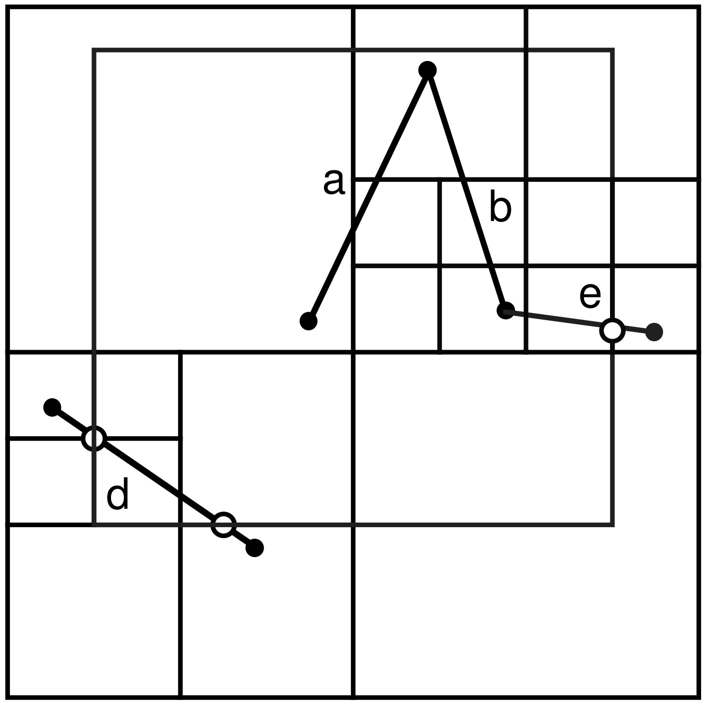
- No splitting as c falls outside the window

INSERTION OF LINE SEGMENT d



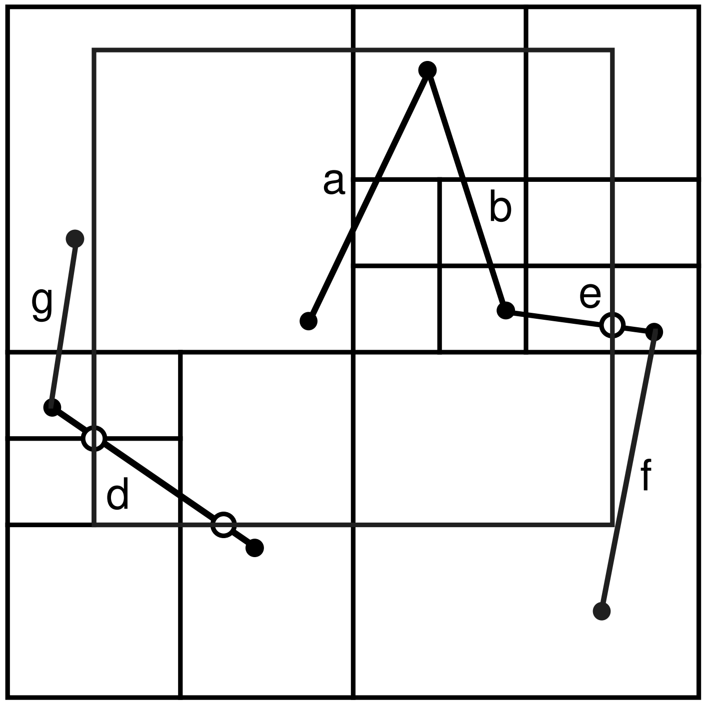
- Three splits to localize the two cutpoints of d as the endpoints of d fall outside the window

INSERTION OF LINE SEGMENT e



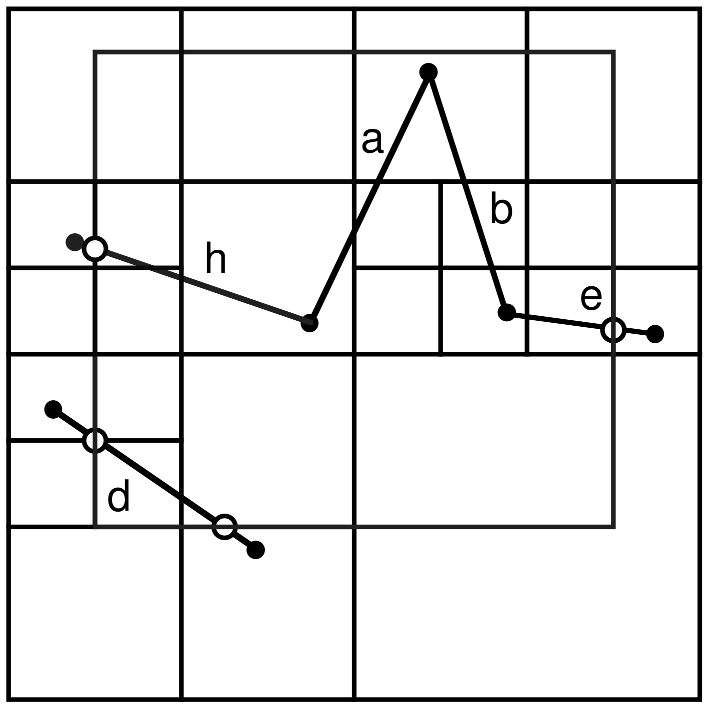
- Two splits to localize the cutpoint of e as the endpoint falls outside the window
- One PMR split as 3 q-edges (from a, b, and e) are in the same block

INSERTION OF LINE SEGMENTS f AND g



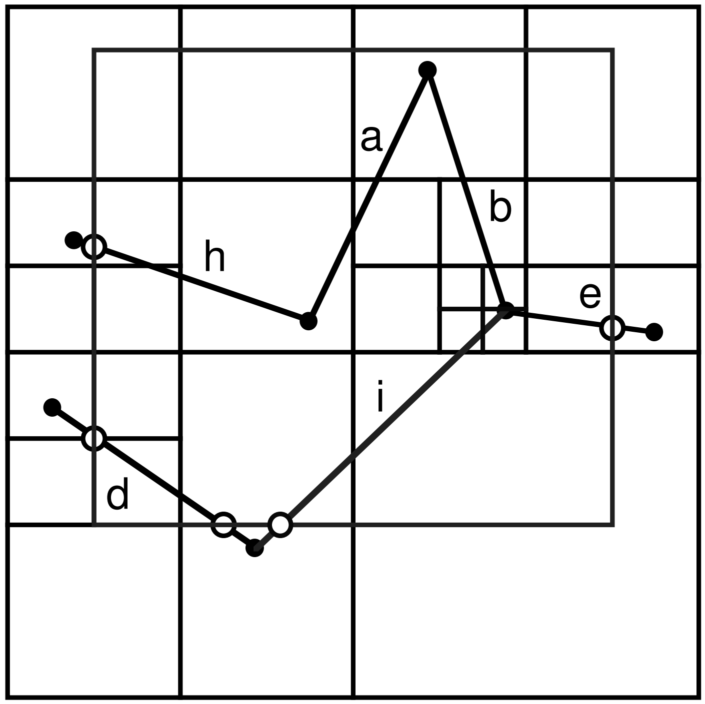
- No splitting as f and g fall outside the window

INSERTION OF LINE SEGMENT h



- Two splits to localize the cutpoint of h as the endpoint falls outside the window
- No PMR split necessary as the splitting thresholds in the NW quadrant are not exceeded

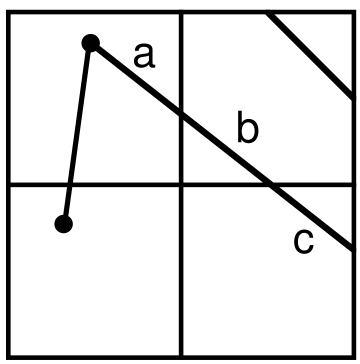
INSERTION OF LINE SEGMENT i

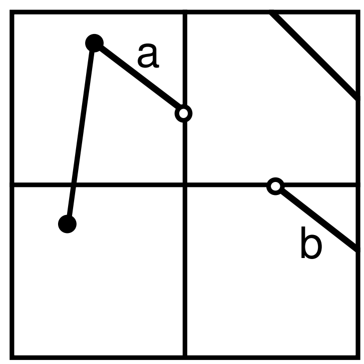


- No cutpoint localization split is necessary as the cutpoint falls on the boundary of an existing block
- One PMR split as 3 q-edges (from b, e, and i) meet at a vertex and thus are in the same block

MERGING BLOCKS CONTAINING FRAGMENTS

- Must be careful not to destroy decomposition that localizes cutpoints
- Only merge if siblings contain fewer than *N* (splitting threshold value) line segment references and are *compatible*

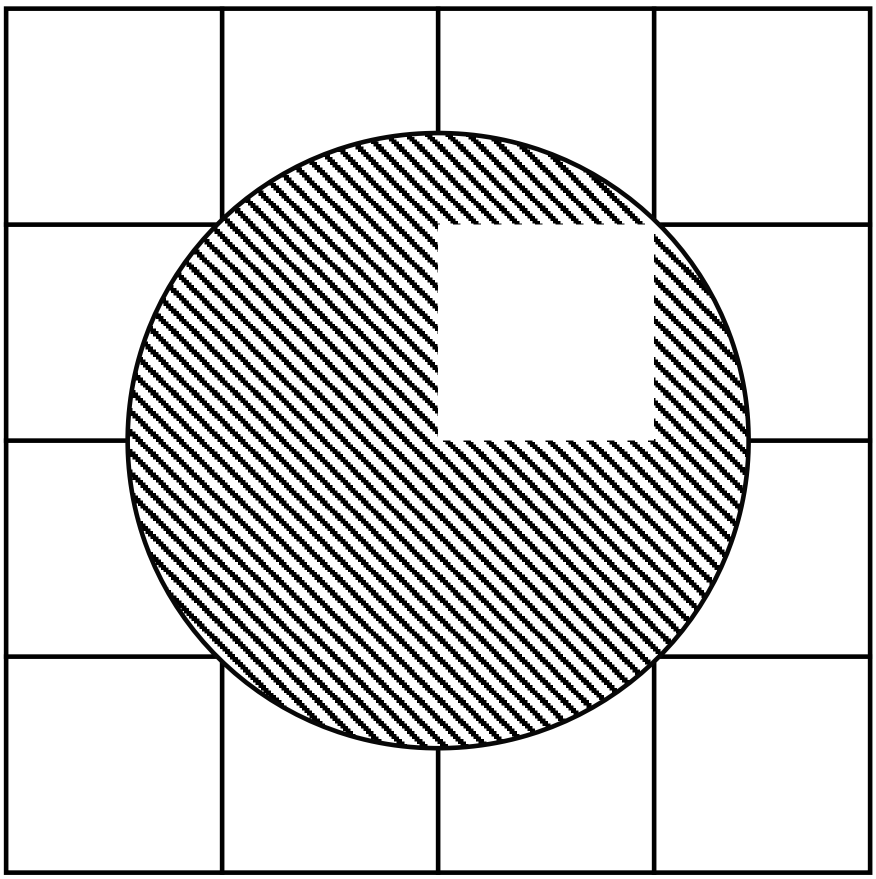




- Siblings are *incompatible* if the inner boundaries contain cutpoints

MAXIMAL SEARCH RADIUS

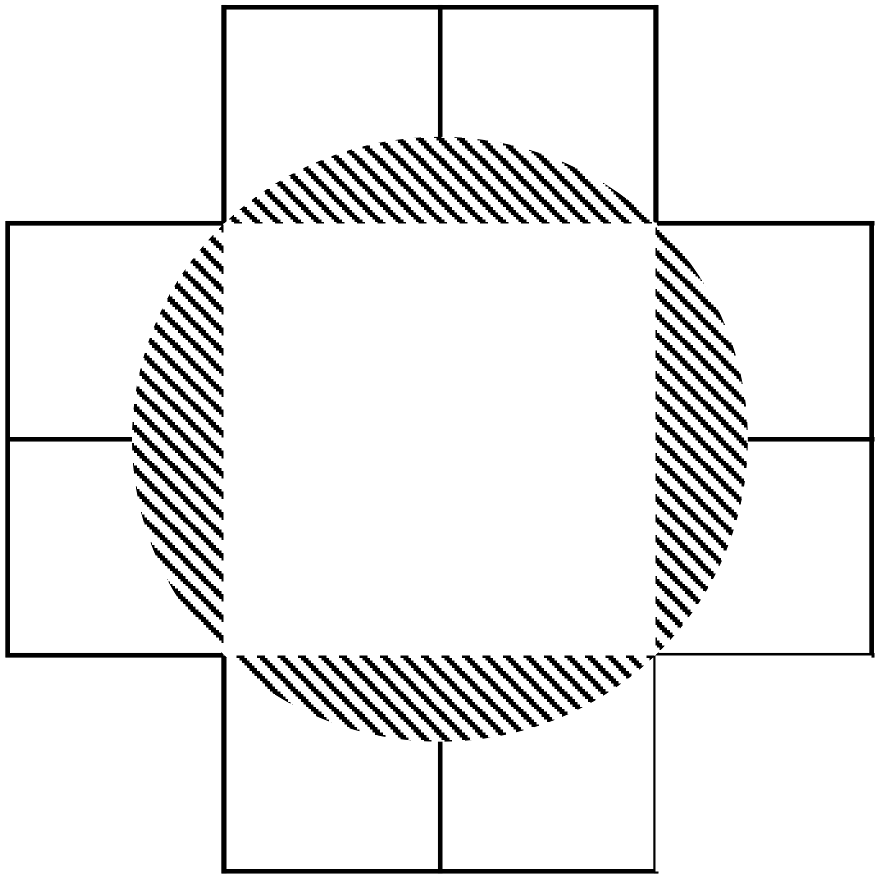
- Properties of the PM quadtree family (PM1, PMR, etc.) greatly localize the search area for nearest line segment
- Assume that the query point P falls in the SW corner of the small highlighted block



- By virtue of the existence of a block of this size, we are guaranteed that at least one of the remaining three siblings contains a line segment

NEAREST LINE SEGMENT ALGORITHM

- A four stage intelligent search process
- Maximal search radius equal to length of parent node's diagonal



- Basic algorithm:
  1. Search the block containing the query point
  2. Search the three siblings
  3. Search the three regions of size equal to that of the parent that are incident to the block containing the query point
  4. Search the final four groups of two adjacent blocks to the previous step

EMPIRICAL TESTS OF THE ALGORITHM

- What is the relationship between map segment density and the search times for finding the nearest line segment?
- What is the average execution time for our implementation of the algorithm?
- What is the effect of changing the value of the splitting threshold on the execution time?
- What is the effect of changing the value of the splitting threshold on the storage requirements of the PMR quadtree?

TESTING ENVIRONMENT

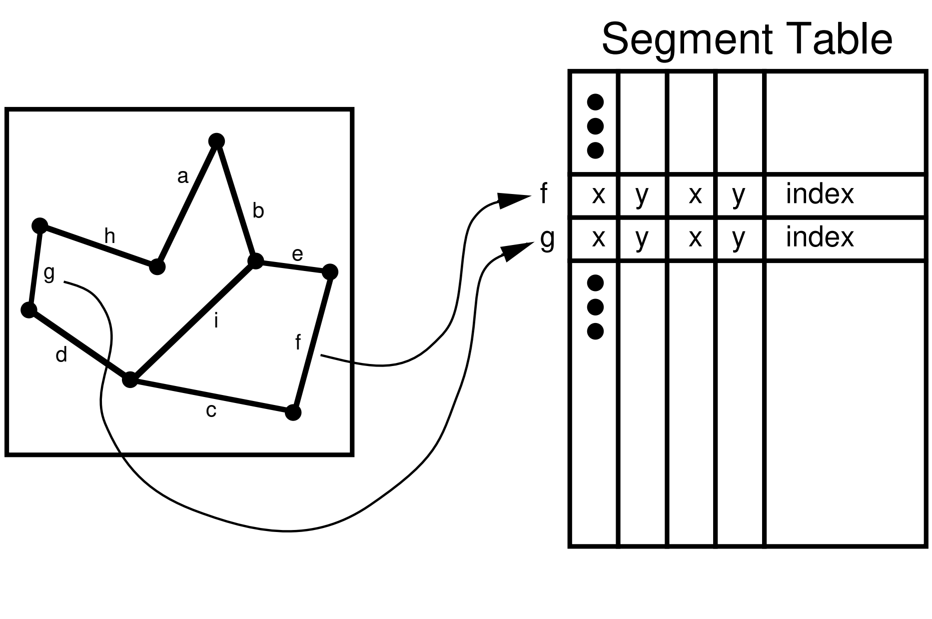
- Choice of query points
  1. Not a uniform distribution of points
  2. Use a two stage process
     a. uniform distribution of blocks in a particular map to yield a block containing the query point
     b. uniform distribution within a block to yield the query point
- PMR quadtree with a varying splitting threshold (i.e., N = 2,4,...,16)
- PMR quadtree is implemented as a linear quadtree and stored in a B+-tree
- Each B+-tree node is stored in a page of size 1K bytes
- Buffer size of 16 pages
- Data sets are Census Bureau TIGER / Line files
- Image resolution (i.e., map size) is 16K x 16K
- Sun SPARCstation 1+ (13.8 SPECint92, 11.1 SPECfp92)

NEAREST LINE SEGMENT PERFORMANCE

County	Segments	Pages Accessed	Line Seg Comps	CPU Seconds
Mariposa	92843	5.71	19.42	0.0154
Sacramento	104502	5.93	17.85	0.0156
Lake	107708	5.81	19.29	0.0155
Calaveras	112529	5.83	18.91	0.0163
Santa Clara	113564	5.90	17.06	0.0148
Imperial	133049	6.13	17.86	0.0158

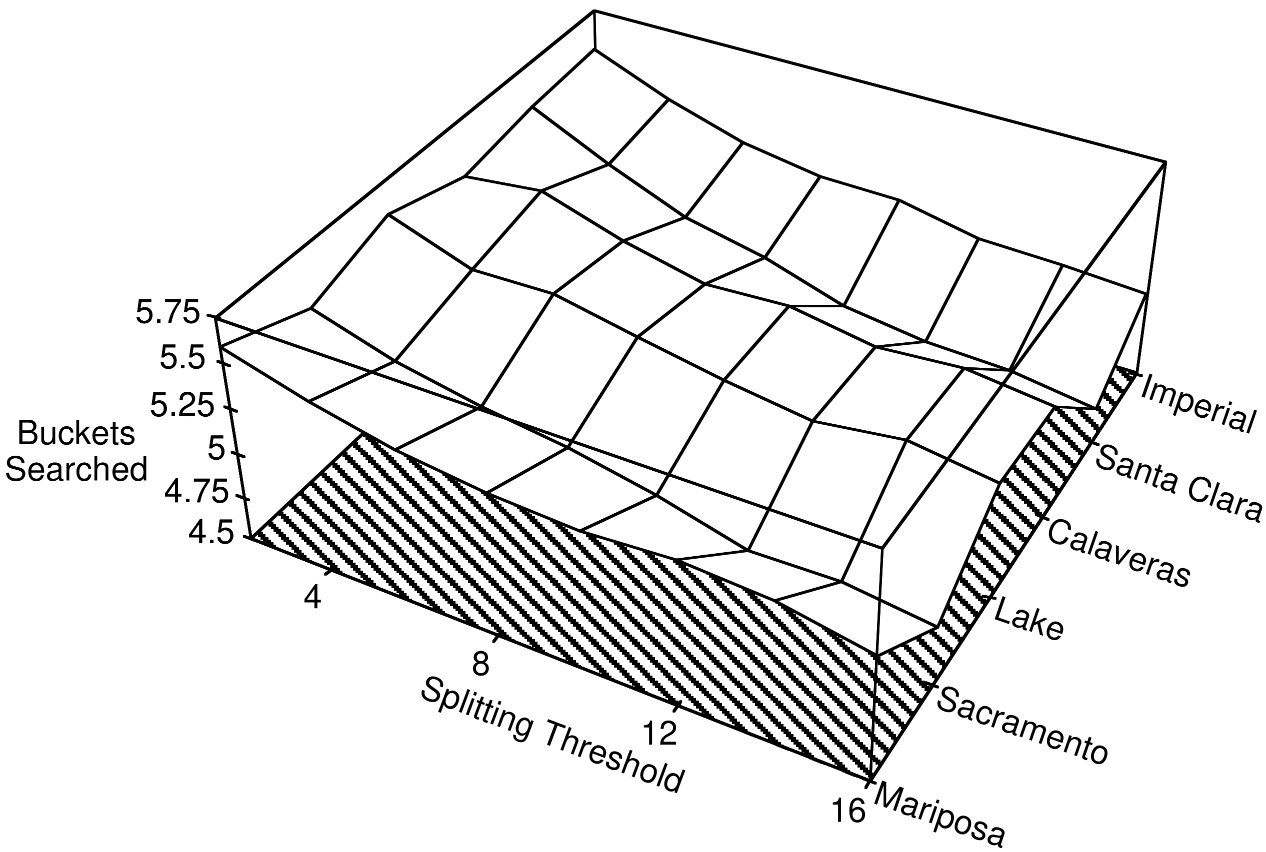
(N = 8)

- Observe that the number of comparisons for Imperial County appears to be ~$\log_2$(number of line segments)
- Sun SPARCstation 1+
- Pages accessed includes the B-tree structure and an auxiliary segment table.



BUCKETS SEARCHED

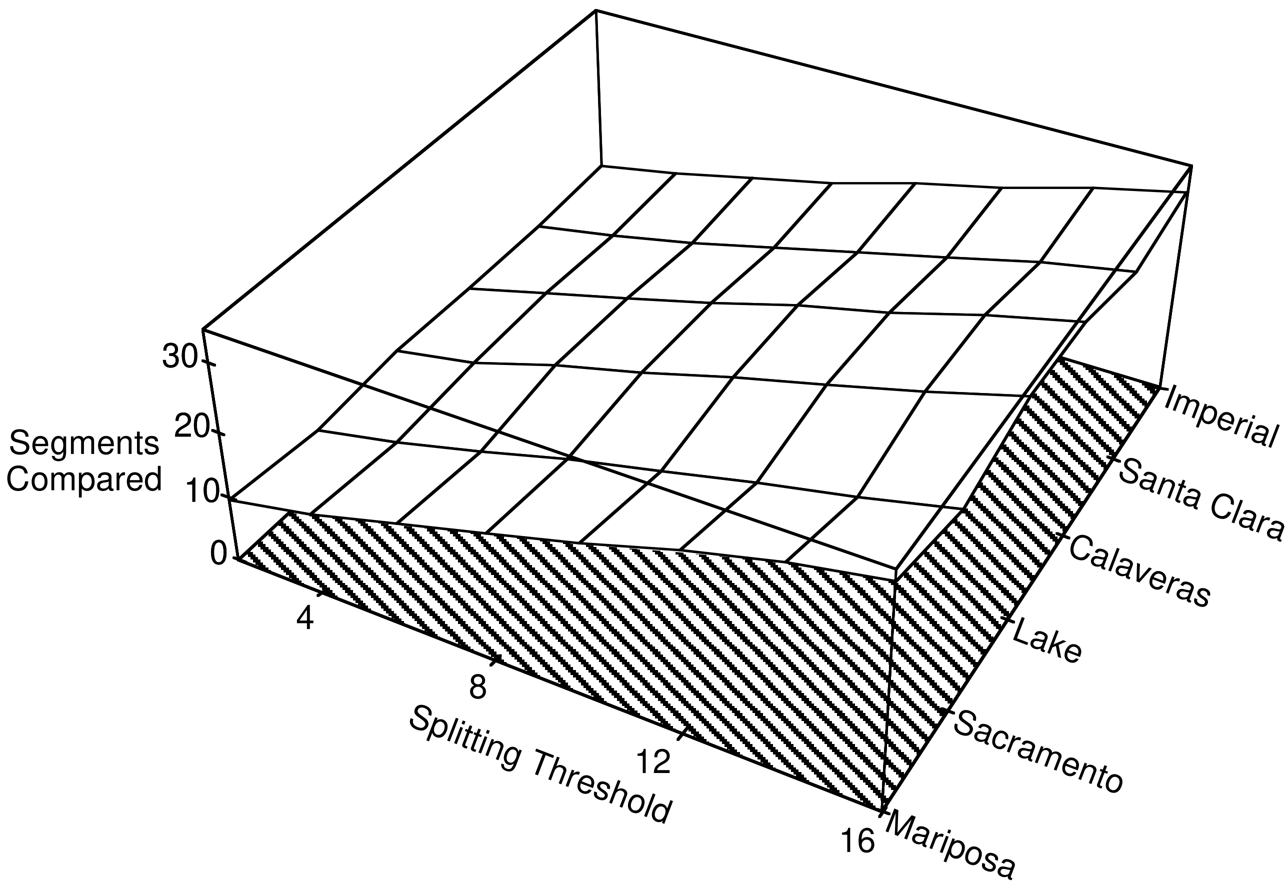
- The average number of buckets (blocks) searched for each random nearest line segment query in the PMR quadtree as a function of the splitting threshold



- The values are stable since the closest line segment is usually in the query block or its siblings

LINE SEGMENTS COMPARED

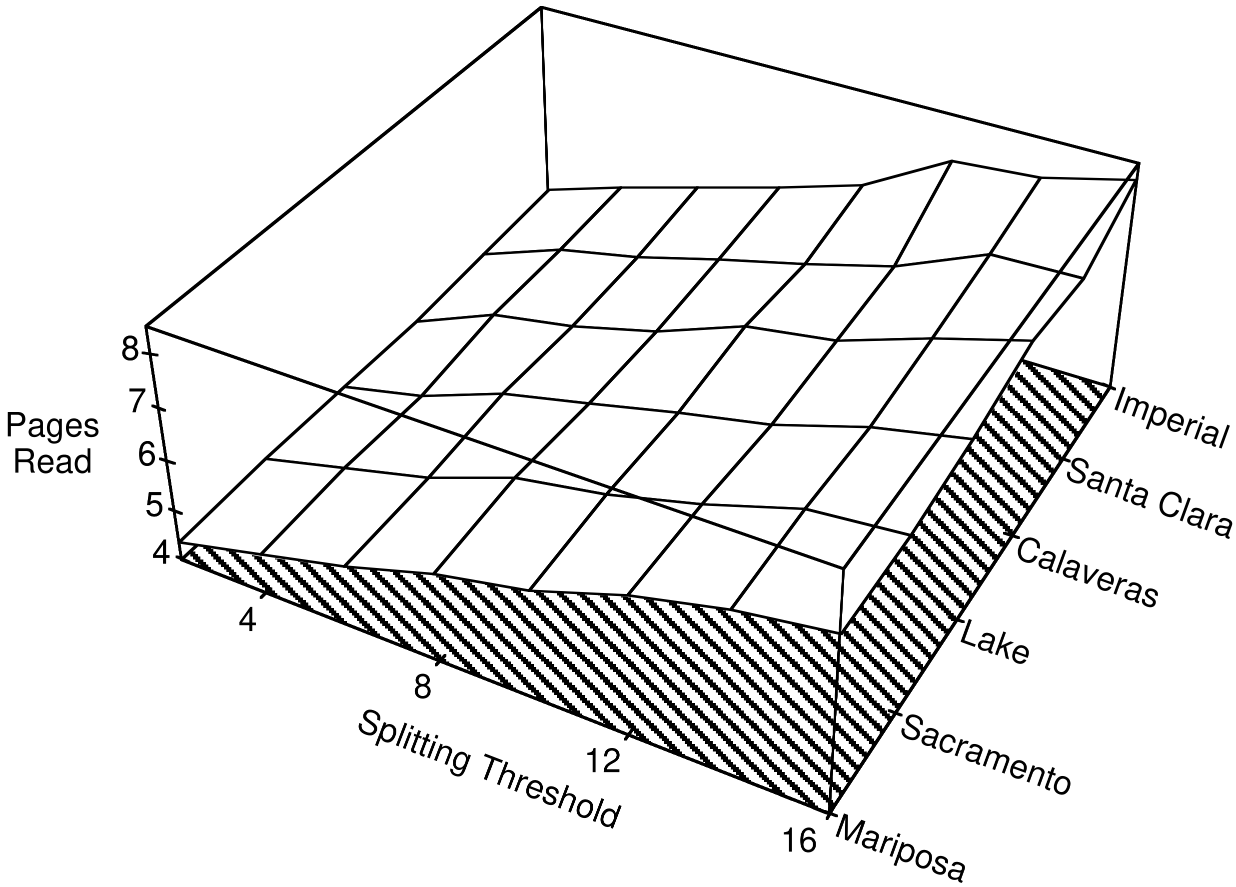
- The average number of line segments compared for each random nearest line segment query in the PMR quadtree as a function of the splitting threshold



- Number of comparisons increases with the splitting threshold since the average bucket size increases
- Uses sequential search in each bucket which causes many line segments to be needlessly accessed and can be reduced by sorting the line segments in each bucket

DISK I/O

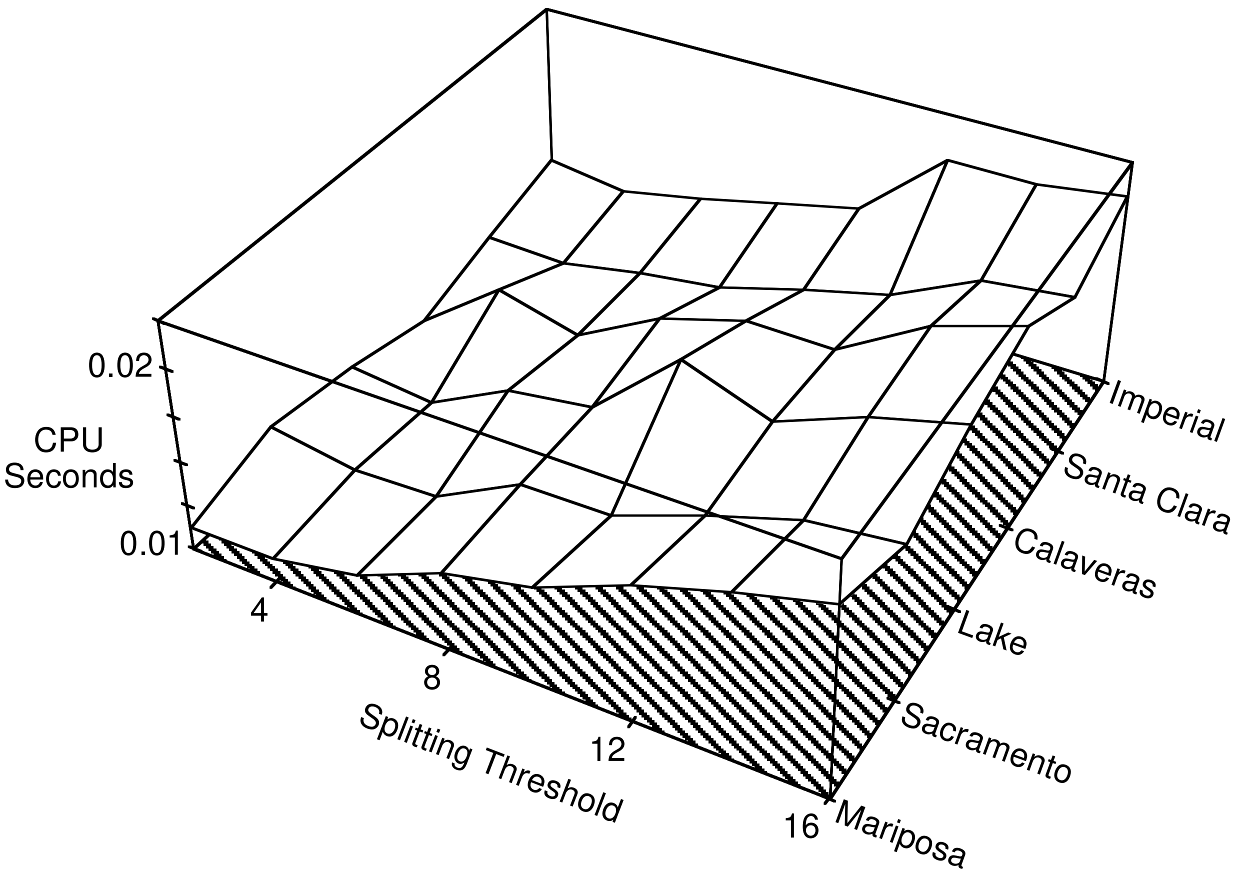
- The average number of pages (1K page size) read from disk for each random nearest line segment query in the PMR quadtree as a function of the splitting threshold.



- Increases with splitting threshold as buckets contain more line segments
- Also includes the disk pages that must be read to fetch the line segment data from the segment table which is disk-resident
- Uses sequential search in each bucket which causes many line segments to be needlessly accessed and can be reduced by sorting the line segments in each bucket

CPU TIME

- The average number of cpu seconds (on a Sun SPARCstation 1+) required for each random nearest line segment query in the PMR quadtree as a function of splitting threshold



- Increases with splitting threshold as buckets contain more line segments
- Uses sequential search in each bucket which causes many line segments to be needlessly accessed and can be reduced by sorting the line segments in each bucket

COMPARISON OF BUILDING TIMES AND SPACE REQUIREMENTS

Structure	Build Time (secs.)	Leaf Nodes
MX quadtree	22.55	19699
edge quadtree	20.48	7723
PM3 quadtree	29.38	3939
PMR quadtree	19.08	2078

- VAX 11/785
- Splitting threshold value of 4
- No method overwhelmingly superior with repect to build time
- PMR quadtree as good or better than other methods

COMPARATIVE INTERSECTION TIMES

- Ex: Intersect road map with floodplain

Structure	Time (secs.)
MX quadtree	4.10
edge quadtree	5.60
PM3 quadtree	6.83
PMR quadtree	8.02

- VAX 11/785
- Execution times are inversely proportional to the storage requirements
- As node complexity increases, so does the execution time
- PM quadtree execution times can be improved slightly by sorting the line segments in each block instead of performing sequential search in each block (result is a two-level storage hierarchy)

IMPLEMENTATION ISSUES IN MAKING COMPARISONS

- Often details are left out
- Ex. R+-tree description is silent on building algorithm and ideal parameters
- Danger that compare implementations rather than data structures
- Goal is to implement data structures in way that make them look best
- Use implementations described in the literature unless can find something better

MAKING THE TESTING ENVIRONMENTS SIMILAR

- Try to ensure that the data structures all use the same amount of storage
- Not always possible
  1. R*-tree always uses less than R+-tree (and PMR quadtree) since each line segment is stored in only one block
  2. can make node (i.e., page) sizes the same
     - PMR quadtree page can store more line segments than R-tree variants since no need for bounding box information
- If a time vs. space issue, then opt for an implementation that is more time-efficient at the cost of increasing space
  1. e.g., node format for R-tree variants
     - one bounding box for each node, OR
     - one bounding box for each line segment
  2. prefer second choice as otherwise an extra disk access is needed for each access to a line segment
  3. PMR quadtree:
     - each block is a bounding box but is not a minimum bounding box as for the R-tree variants
     - bounding box can be encoded by its locational code (i.e., *x* and *y* coordinate values of a corner and its size)

ACTUAL IMPLEMENTATION ENVIRONMENT

- Assume database is disk resident and 1K byte nodes
- PMR quadtree
  1. linear quadtree (only leaf nodes retained)
  2. each line segment passing through a block is a 2-tuple (*L,O*)
     - *L* is the locational code of the block (4 bytes)
     - *O* is a pointer to a segment table entry (on disk) for the line segment
  3. can store 120 line segments in each page
  4. embedded in the QUILT spatial database system
  5. splitting threshold is 4 since it is rare for more than 4 roads to intersect
- R-tree variants
  1. each line segment and bounding box is a 2-tuple (*R,O*)
     - *R* is minimum bounding box (4 values of 4 bytes)
     - *O* points to segment table entry for a leaf node or a son for a nonleaf node (4 bytes)
  2. can store 50 line segments in each page
  3. m is 40% of M

QUERIES

1. Given an endpoint of a line segment, find all the line segments that are incident at it (*point query*)
   - simple search query that does not require that the space occupied by the line segments be sorted
2. Given an endpoint of a line segment, find all the line segments that are incident at the other endpoint of the line segment
3. Given a point in the two-dimensional space containing the line segments, find the nearest line segment using a Euclidean distance metric
   - ex: find nearest subway line to a given house
   - more efficient when the line segments are sorted
4. Given a point in the two-dimensional space containing the line segments, find the minimal enclosing polygon by outputting its constituent line segments
5. Given a rectangular window, find all the line segments in the window (*range query* or *window query*)
   - ex: find all roads passing through a given region

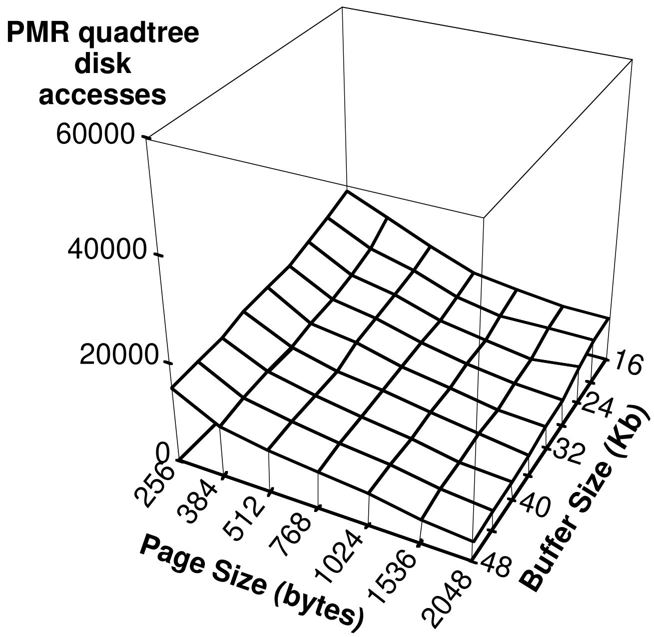
ACTUAL TESTING ENVIRONMENT

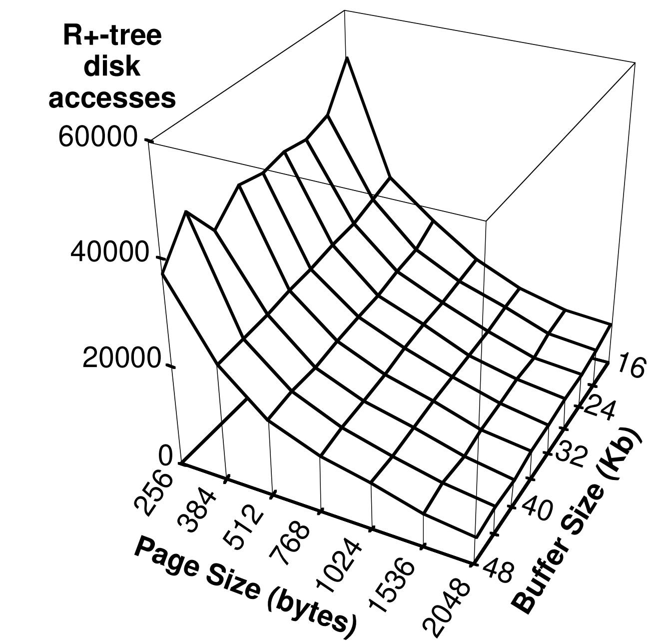
- Main statistic is number of disk accesses (actually where a potential for them exists)
  1. can distinguish between operations that access the same page and those that do not
  2. meaningful execution times hard to obtain
- Choice of query points
  1. uniform distribution
     - drawback is that many of the query points lie outside the boundary of the map or in large empty areas
  2. two-stage process
     - uniform distribution of blocks in a particular map to yield the block containing the query point
     - uniform distribution within the block to yield the query point
- Window queries used .01% of total area
- Data sets are Census Bureau TIGER/Line files
- HP 720 (58.2 SPECfp92, 36.4 SPECint92)
- 16K x 16K image
- Buffer pool of 16 1K pages using an LRU page replacement policy

DATA STRUCTURE BUILDING STATISTICS

- Observations:
  1. R+-tree is the fastest
  2. PMR quadtree requires sorting contents of B-tree nodes and hence must move data around
     - ~50% slower for PMR than R+
  3. R*-tree slow because of expensive node overflow handler
     - must reinsert about 30% of bounding boxes
     - slower by about a factor of 8!
  4. number of disk accesses are comparable with PMR quadtree usually smaller
  5. R*-tree uses the least space but not by much
     - 10 - 30% less space
  6. space requirements of PMR quadtree and R+-tree are comparable

EFFECT OF BUFFER AND PAGE SIZES





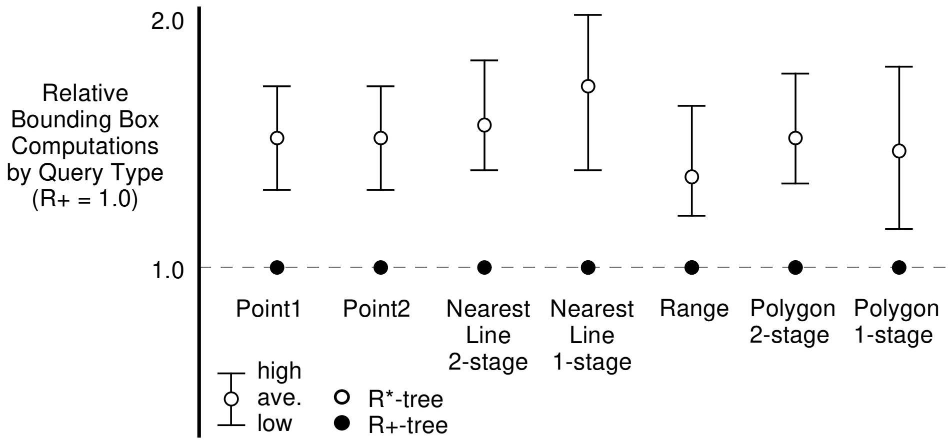
- Disk accesses decrease as the page size and the size of the buffer pool increase
- Lower values for the PMR quadtree since R+-tree pages contain fewer line segment tuples

MAP DATA

- Line segment distributions for maps were quite different
  1. urban: 5-6 line segments per polygon (Baltimore)
  2. suburban: 19 line segments per polygon (Anne Arundel)
  3. rural: 132 line segments per polygon (Charles)
- Normalize R-tree variants with respect to PMR quadtree
- Sample un-normalized data (Charles)

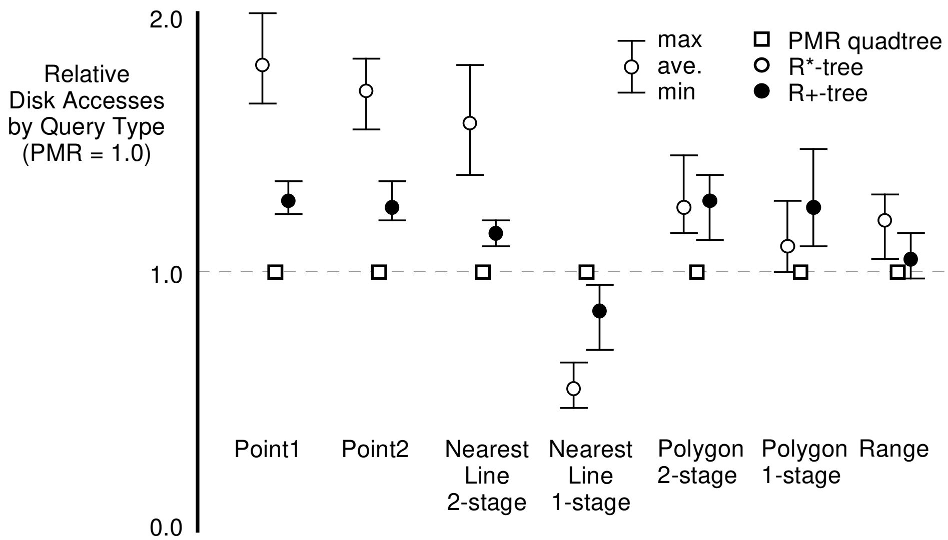
Query	Metric	PMR	R+	R*
Point1	disk I/O	1.55	2.07	2.74
	seg comps	3.48	2.43	2.39
	bbox/bucket comps	1.00	105.02	149.89
Point2	disk I/O	1.72	2.29	2.90
	seg comps	4.43	3.38	3.35
	bbox/bucket comps	2.00	209.75	299.10
Nearest Line (2-stage)	disk I/O	2.21	2.52	3.35
	seg comps	11.23	27.02	36.16
	bbox/bucket comps	5.33	248.01	389.05
Nearest Line (1-stage)	disk I/O	7.18	6.75	3.38
	seg comps	22.32	75.08	40.35
	bbox/bucket comps	8.77	387.86	765.98
Polygon (2-stage)	disk I/O	13.19	18.46	14.07
	seg comps	451.43	388.23	389.85
	bbox/bucket comps	185.98	16996.69	23730.10
Polygon (1-stage)	disk I/O	12.62	18.67	13.43
	seg comps	368.10	347.95	333.55
	bbox/bucket comps	152.35	14101.58	20387.28
Range	disk I/O	2.93	3.24	3.50
	seg comps	14.70	8.17	6.88
	bbox/bucket comps	16.57	149.24	179.76

BOUNDING BOX COMPUTATIONS



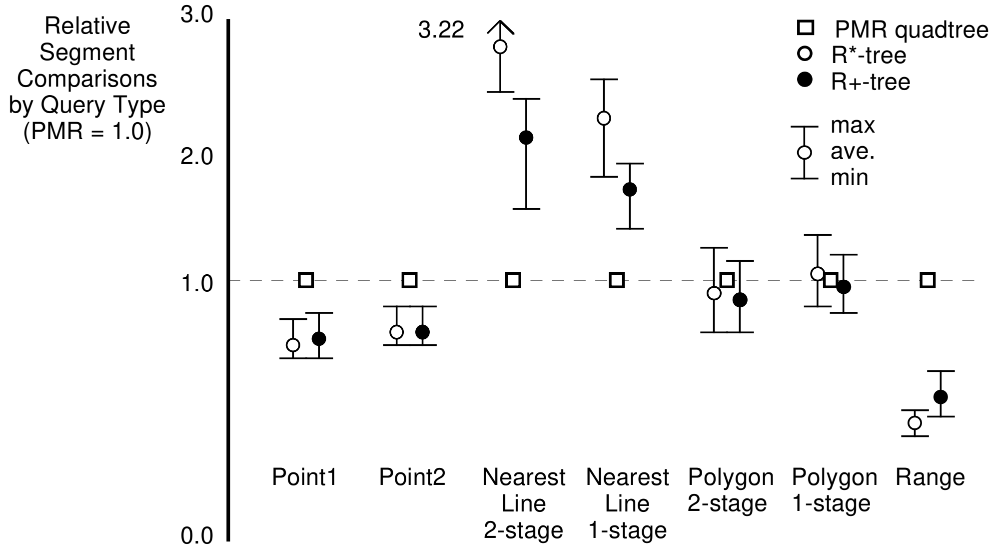
- Bounding bucket comparisons for PMR quadtree is analogous to bounding box for R-trees but omitted since 2 orders of magnitude difference in favor of the PMR quadtree
- Contents of a PMR quadtree B-tree node (i.e., page) are sorted by locational code and thus no need to do sequential search within the B-tree node as in the R-variants
- We did not count the logarithmic search within each B-tree node for the right PMR quadtree node
- Could define R-tree variants to also sort the bounding boxes but build times will now be slower
- R+-tree better than R*-tree since a disjoint decomposition of space

DISK ACCESSES



- Exclude accesses to segment table
- PMR quadtree usually had a slight edge over R-tree variants
- When the R+-tree is better than the R*-tree, the reason is the disjoint decomposition of space
- Exceptions:
  1. R*-tree is better for the polygon query due to the effect of locality on the repeated application of the point query
  2. R*-tree is better for 1-stage nearest line query due to the emptyness of the regions queried thereby forcing larger initial search radii
     - large initial search radii meant more disk accesses for PMR and R+ as pages are not organized by locality to the same extent as the R*

SEGMENT COMPARISONS



- Implies access to segment table which is disk-resident
- Little actual difference in disk activity as segments are usually in close proximity
- Insignificant differences for point and polygon queries
- PMR quadtree is significantly better than R-trees for the nearest line segment query since it sorts the line segments and hence can prune the search space
- Look at sum of segment comparisons and bounding box and bucket comparisons as no bounding boxes stored in PMR quadtree
  1. bounding bucket comparisons are very small for the PMR quadtree
  2. PMR quadtree is superior to R-tree variants by several orders of magnitude
  3. poor range query performance of PMR quadtree is due to absence of a good bounding box mechanism

CONCLUSIONS

1. No overwhelming superiority for any particular data structure
2. Choice of data structure depends on repertoire of operations
3. If operations involve search, then the R+-tree and PMR quadtree are best as they yield a disjoint decomposition of space
4. If results are to be composed with the results of other operations, then the PMR quadtree is best as it uses a regular decomposition
5. R*-tree is most compact space-wise but performance is not as good as the R+-tree due to non-disjointness of the decomposition induced by it
6. Performance could be improved by addressing the issue of how to organize line segments in each bucket or node (e.g., sort them)
7. Splitting threshold plays similar role to bucket capacity
   - choose a splitting threshold value that yields an average bucket (node) occupancy similar to the average page occupancy in an R-tree