

Sorting in Space: Multidimensional Data Structures for Computer Graphics and Vision Applications

Hanan Samet

`hjs@cs.umd.edu`

`http://www.cs.umd.edu/~hjs`

Department of Computer Science

University of Maryland

College Park, MD 20742, USA

Unless explicitly stated otherwise, the upper-left corner of each slide indicates the page numbers in Foundations of Multidimensional and Metric Data Structures by H. Samet, Morgan-Kaufmann, San Francisco, 2006, where more details on the topic can be found

TITLE:

Sorting in Space: Multidimensional Data Structures for Computer Graphics and Vision Applications

Hanan Samet
Computer Science Department
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742
e-mail: hjs@cs.umd.edu
url: <http://www.cs.umd.edu/~hjs>

SUMMARY STATEMENT:

We show how to represent spatial data using techniques that sort it. They include quadtrees, octrees, and bounding volume hierarchies and are rooted in the intersection between computer vision and graphics. We focus on building these representations and on finding nearest neighbors which is critical when using machine learning methods.

COURSE ABSTRACT:

The representation of spatial data is important in game programming, computer graphics, visualization, solid modeling, computer vision and geographic information systems (GIS). They are rooted in the intersection of computer vision and graphics. Recently, there has been much interest in hierarchical representations such as quadtrees, octrees, and pyramids which are based on image hierarchies, as well methods that use bounding boxes which are based on object hierarchies. Their advantage is that they provide a way to index into space. In fact, they are little more than multidimensional sorts. They save space as well as time and also facilitate operations such as search. In addition, we introduce methods for dealing with recognizing textual specifications of spatial data such as locations in news articles.

This course provides a brief overview of hierarchical spatial data structures and related algorithms that make use of them. We describe hierarchical representations of points, lines, collections of small rectangles, regions, surfaces, and volumes. For region data, we point out the dimension-reduction property of the region quadtree and octree, as how to navigate between nodes in the same tree, thereby leading to the popularity of these representations in ray tracing applications. We also demonstrate how to use these representations for both raster and vector data. We also In the case of nonregion data, we show how these data structures can be used to find nearest neighbors which is critical when using machine learning techniques. We also show how to do it in an incremental fashion so that the number of objects need not be known in advance. In addition, the SAND spatial browser based on the SAND spatial database system, the VASCO JAVA applet illustrating these methods (www.cs.umd.edu/hjs/quadtree/index.html), and the NewsStand system (newsstand.umiacs.umd.edu) will be demonstrated.

PREREQUISITE:

A familiarity with computer terminology and some programming experience.

COURSE LEVEL:

Beginner

INTENDED AUDIENCE:

Practitioners working in computer graphics and computer vision will be given a different perspective on data structures found to be useful in most applications. Game developers and technical managers will appreciate the presentation and methods described herein.

COURSE SYLLABUS:

The representation of spatial data is an important issue in game programming, computer graphics, visualization, solid modeling, and related areas including computer vision and geographic information systems (GIS). It has also taken on an increasing level of importance as a result of the popularity of web-based mapping services such as Bing Maps, Google Maps, Google Earth, and Yahoo Maps, as well as the increasing importance of location-based services. Operations on spatial data are facilitated by building an index on it. The traditional role of the index is to sort the data, which means that it orders the data. However, since no ordering exists in dimensions greater than 1 without a transformation of the data to one dimension, the role of the sort process is one of differentiating between the data, and what is usually done is to sort (i.e., order) the spatial objects with respect to the space that they occupy (e.g., Warnock's algorithm, back-to-front and front-to-back display algorithms, BSP trees for visibility determination, acceleration of ray tracing, bounding box/volume hierarchies that sort the space on the basis of whether it is occupied). The resulting ordering is usually implicit rather than explicit so that the data need not be resorted (i.e., the index need not be rebuilt) when the queries change (e.g., the query reference objects).

There are many representations (i.e., indexes) currently in use. Recently, there has been much interest in hierarchical data structures such as quadrees, octrees, and pyramids, which are based on image hierarchies, as well methods that make use of bounding boxes or volumes, which are based on object hierarchies. They are rooted in the intersection of computer vision and graphics. The key advantage of these representations is that all of them provide a way to index into space. They are compact and depending on the nature of the spatial data, they save space as well as time and also facilitate operations such as search.

In this course we provide a brief overview of hierarchical spatial data structures and related algorithms that make use of them. We describe hierarchical representations of points, lines, collections of small rectangles, regions, surfaces, and volumes. For region data, we point out the dimension-reduction property of the region quadtree and octree, as how to navigate between nodes in the same tree, thereby leading to the popularity of these representations in ray tracing applications. We also demonstrate how to use these representations for both raster and vector data. In the case of nonregion data, we show how these data structures can be used to find nearest neigh-

bors which is critical when using machine learning techniques. We also show how to do it in an incremental fashion so that the number of objects need not be known in advance. We point out that these algorithms can also be used in an environment where the distance is measured along a spatial network rather than being constrained to “as the crow flies” (i.e., the Euclidean distance). We also review a number of different tessellations and show why hierarchical decomposition into squares instead of triangles or hexagons is preferred. In addition, we briefly show how to represent data that lies only in a metric space rather than a vector space and point out the relationship of these representations to those that assume that the data lies in a vector space. In particular, we show that the difference is that the partitions are implicit in the metric space in contrast to being explicit in the vector space. We conclude with a demonstration of the SAND spatial browser based on the SAND spatial database system and of the VASCO JAVA applet illustrating these methods (found at <http://www.cs.umd.edu/hjs/quadtrees/index.html>). We conclude with a demonstration of the SAND spatial browser based on the SAND spatial database system, the VASCO JAVA applet illustrating these methods (found at <http://www.cs.umd.edu/~hjs/quadtrees/index.html>), and the NewsStand system (<http://newsstand.umiacs.umd.edu>) for recognizing textual specifications of spatial data such as locations in news articles.

COURSE SCHEDULE:

0:00-0:15	Introduction
0:15-0:25	Points
0:25-0:30	Lines
0:30-0:35	Regions
0:35-0:45	Bounding Box Hierarchies
0:45-0:55	Rectangles and Moving Object Representations
0:55-1:05	Metric Space Data Structures
1:05-1:15	Incremental Nearest Neighbor Finding
1:15-1:25	Nearest Neighbor Finding in Spatial Networks
1:25-1:40	Demos
1:40-1:45	Questions

COURSE TOPICS

1. Introduction
 - (a) Sorting definition
 - (b) Sample queries
 - (c) Spatial Indexing
 - (d) Sorting approach
 - (e) Minimum bounding rectangles (e.g., R-tree)
 - (f) Disjoint cells (e.g., R+-tree, k-d-B-tree)
 - (g) Uniform grid
 - (h) Region quadtree
 - (i) Space ordering methods

- (j) Pyramid
- (k) Region quadtree vs: pyramids
- 2. Points
 - (a) Point quadtree
 - (b) PR quadtree
 - (c) Sorting points
 - (d) K-d tree
 - (e) PR k-d tree
- 3. Lines
 - (a) Strip tree
 - (b) MX quadtree for regions
 - (c) PM1 quadtree
 - (d) PM2 quadtree
 - (e) PM3 quadtree
 - (f) PMR quadtree
 - (g) Triangulations
- 4. Regions
 - (a) Region quadtree
 - (b) Dimension reduction
 - (c) Tessellations
 - (d) BSP tree
- 5. Bounding Box Hierarchies
 - (a) Overview
 - (b) Minimum bounding rectangles (e.g., R-trees)
 - (c) Searching in an R-tree
 - (d) Node overflows
- 6. Rectangles
 - (a) MX-CIF quadtree
 - (b) Loose quadtree or coverage fieldtree
 - (c) Partition fieldtree
- 7. Surfaces and Volumes
 - (a) Region octree
 - (b) PM octree
- 8. Operations

- (a) Incremental nearest object location
- (b) Incremental nearest object location in spatial networks

9. Demos

- (a) SAND Internet browser
- (b) JAVA spatial data applets
- (c) NewsStand spatiotextual news retrieval

COURSE MATERIALS:

Participants receive a copy of the slides. In addition, there is a web site at <http://www.cs.umd.edu/~hjs/quadtrees/index.html> where applets demonstrating much of the material in the course are available. In order to run these applets you must deal with Java Security. For Windows: 1)Open FireFox 2)Update Java 3)Control panel-> Java->Security-> Add “<http://donar.umiacs.umd.edu/>” to the exception site list. For MAC: 1)Open Firefox 2)Update Java 3)System Preferences -> Java Control Panel -> Security -> Add “<http://donar.umiacs.umd.edu/>” to exception site list. Participants are referred to the text: H.Samet, Foundations of Multidimensional and Metric Data Structures, Morgan-Kaufmann, San Francisco, 2006. Participants also have the opportunity to obtain the book at a discount of 30% as reflected at <http://www.cs.umd.edu/~hjs/multidimensional-book-flyer.pdf> or simply go to the publisher’s website and use code COMP316

SPEAKER BIOGRAPHY:

Hanan Samet (<http://www.cs.umd.edu/~hjs/>) received the BS. degree in engineering from the University of California, Los Angeles, and the M.S. Degree in operations research and the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford, CA. His Ph.D. dissertation founded the field of compiler translation validation and the related concept of proof-carrying code. He is a Fellow of the IEEE, ACM, IAPR (International Association for Pattern Recognition), AAAS, and UCGIS (University Consortium for Geographic Science), an ACM Distinguished Speaker, and was also elected to the ACM Council in 1989-1991 where he served as the Capital Region Representative. He is the recipient of the 2009 UCGIS Research Award, the 2010 University of Maryland College of Computer, Mathematical and Physical Sciences Board of Visitors Distinguished Faculty Award, 2011 ACM Paris Kanellakis Theory and Practice Award, 2014 IEEE Computer Society Wallace McDowell Award, and a Science Foundation of Ireland (SFI) Walton Visitor Award at the Centre for Geocomputation at the National University of Ireland at Maynooth (NUIM).

In 1975 he joined the Computer Science Department at the University of Maryland, College Park, where he is now a Distinguished Professor.. He has held visiting positions at the National University of Singapore, University of Genoa, Genoa (Italy), University of Pavia (Italy), University of Paris (France), Hebrew University of Jerusalem (Israel), and at NTT Basic Research Lab (Japan).

At the University of Maryland he is a member of the Computer Vision Laboratory of the Center for Automation Research and also has an appointment in the University of Maryland Institute for Advanced Computer Studies. At the Computer Vision Laboratory he leads a number of research projects on the use of hierarchical data structures for geographic information systems. His

research group has developed the QUILT system which is a GIS based on hierarchical spatial data structures such as quadrees and octrees, the SAND system which integrates spatial and non-spatial data, the SAND Browser (<http://www.cs.umd.edu/~brabec/sandjava>) which enables browsing through a spatial database using a graphical user interface, the VASCO spatial indexing applet (found at <http://www.cs.umd.edu/~hjs/quadtree/index.html>), a symbolic image database system, and the STEWARD system for spatio-textual retrieval of documents on the web as well as the NewsStand and TwitterStand systems for news and Twitter tweets, respectively. He is the founding chair of the ACM Special Interest Group on Spatial Information (SIGSPATIAL). He has served as the co-general chair of the 15th ACM International Conference on Advances in Geographic Information Systems (ACMGIS'07) and the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACMGIS'08).

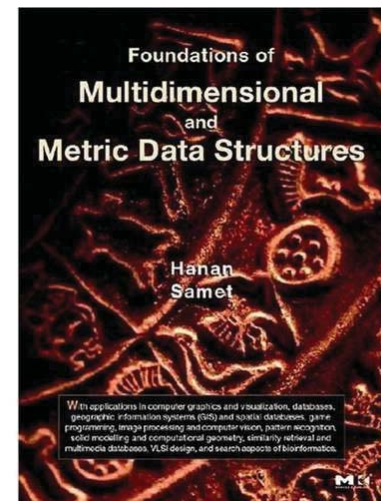
His research interests include data structures, computer graphics, geographic information systems, computer vision, robotics, and database management systems, and is the author of over 300 publications on these topics. He is the author of the recent book titled "Foundations of Multidimensional and Metric Data Structures" (<http://www.cs.umd.edu/~hjs/multidimensional-book-flyer.pdf>) published by Morgan-Kaufmann, an imprint of Elsevier, in 2006, an award winner in the 2006 best book in Computer and Information Science competition of the Professional and Scholarly Publishers (PSP) Group of the American Publishers Association (AAP), and of the first two books on spatial data structures titled "Design and Analysis of Spatial Data Structures", and "Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS", both published by Addison-Wesley in 1990. He is the Founding Editor-In-Chief of the ACM Transactions on Spatial Algorithms and Systems (TSAS) and the founding chair of ACM SIGSPATIAL. He is an Area Editor of "Graphical Models", and on the Editorial Board of "Image Understanding", "Journal of Visual Languages", and "GeoInformatica". He received best paper awards in the 2007 Computers & Graphics Journal, the 2008 ACM SIGMOD and SIGSPATIAL ACMGIS Conferences, the 2012 SIGSPATIAL MobiGIS Workshop, and the 2013 SIGSPATIAL GIR Workshop, as well as a best demo award at the 2011 SIGSPATIAL ACMGIS'11 Conference. His paper at the 2009 IEEE International Conference on Data Engineering (ICDE) was selected as one of the best papers for publication in the IEEE Transactions on Knowledge and Data Engineering. He was elected to the ACM Council as the Capitol Region Representative for the term 1989-1991, and is an ACM Distinguished Speaker. He holds several patents in the spatial and location-based services space.

Sorting in Space

Hanan Samet

`hjs@cs.umd.edu`

Department of Computer Science
Institute for Advanced Computer Studies
Center for Automation Research
University of Maryland
College Park, MD 20742, USA



Flyer

Outline

1. Introduction
2. Points
3. Lines
4. Regions
5. Bounding Box Hierarchies
6. Rectangles and Moving Object Representations
7. Volumes and Surfaces
8. Metric Data Structures
9. Operations
10. Demos

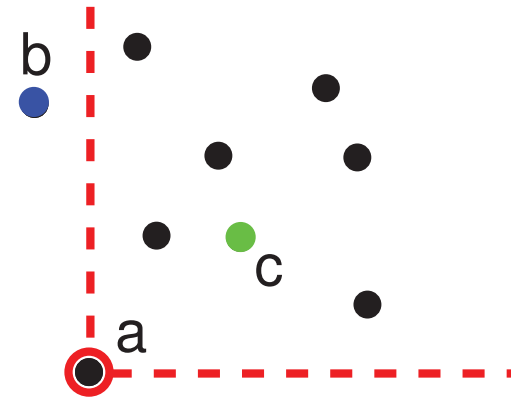
Why Sorting of Spatial Data is Important

- Most operations invariably involve search
- Search is sped up by sorting the data
- *sort* - Definition: verb
 1. to put in a certain place or rank according to kind, class, or nature
 2. to arrange according to characteristics
- Examples
 1. Warnock algorithm: sorting objects for display
 - vector: hidden-line elimination
 - raster: hidden-surface elimination
 2. Back-to-front and front-to-back algorithms
 3. BSP trees for visibility determination
 4. Accelerating ray tracing and ray casting by finding ray-object intersections
 5. Bounding box hierarchies arrange space according to whether occupied or unoccupied

Sorting Implies the Existence of an Ordering

1. Fine for one-dimensional data

- sort people by weight and find closest in weight to Bill and can also find closest in weight to Larry
- sort cities by distance from Chicago and find closest to Chicago but cannot find closest to New York unless resort



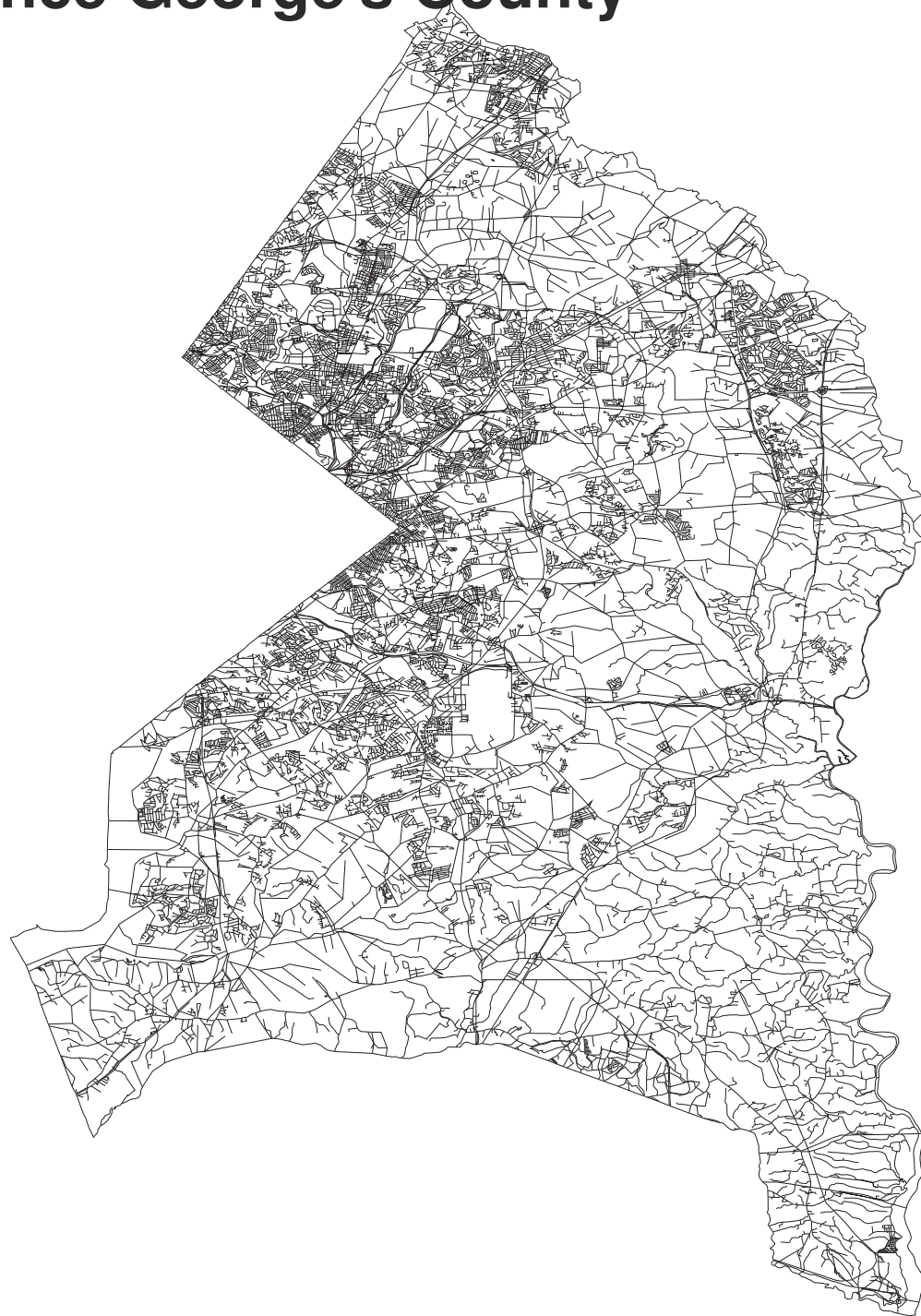
2. Hard for two-dimensions as high as notion of ordering does not exist unless a dominance relation holds

- point $a = \{a_i | 1 \leq i \leq d\}$ dominates point $b = \{b_i | 1 \leq i \leq d\}$ if $a_i \leq b_i, 1 \leq i \leq d$.
- **a** does not dominate **b** but dominates **c**

3. Only solution is to linearize data as in a space-filling curve

- sort is explicit
- need implicit sort so no need to resort if reference point changes

Map of Prince George's County



Example Queries in Line Segment Databases

1. Queries about line segments

- All segments that intersect a given point or set of points
- All segments that have a given set of endpoints
- All segments that intersect a given line segment
- All segments that are coincident with a given line segment

2. Proximity queries

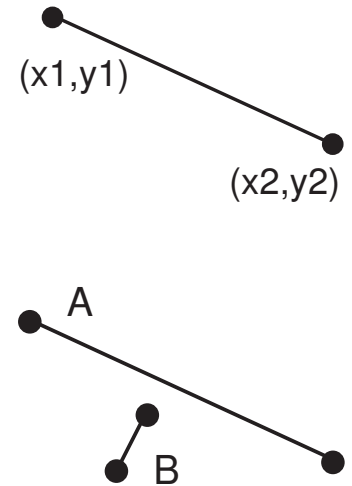
- The nearest line segment to a given point
- All segments within a given distance from a given point (also known as a range or window query)

3. Queries involving attributes of line segments

- Given a point, find the closest line segment of a particular type
- Given a point, find the minimum enclosing polygon whose constituent line segments are all of a given type
- Given a point, find all the polygons that are incident on it

What Makes Continuous Spatial Data Different?

1. Spatial extent of the objects is the key to the difference
2. A record in a DBMS may be considered as a point in a multidimensional space
 - A line can be transformed (i.e., represented) as a point in 4-d space with $(x1, y1, x2, y2)$
 - Good for queries about the line segments
 - Not good for proximity queries since points outside the object are not mapped into the higher dimensional space
 - Representative points of two objects that are physically close to each other in the original space (e.g., 2-d for lines) may be very far from each other in the higher dimensional space (e.g., 4-d)
 - Problem is that the transformation only transforms the space occupied by the objects and not the rest of the space (e.g., the query point)
 - Can overcome by projecting back to original space
3. Use an index that sorts based upon spatial occupancy (i.e., extent of the objects)



Spatial Indexing Requirements

1. Compatibility with the data being stored
2. Choose an appropriate zero or reference point
3. Need an implicit rather than an explicit index
 - a. impossible to foresee all possible queries in advance
 - b. cannot have an attribute for every possible spatial relationship
 - i. derive adjacency relations
 - ii. 2-d strings capture a subset of adjacencies
 - A. all rows
 - B. all columns
 - c. implicit index is better as an explicit index which, for example, sorts two-dimensional data on the basis of distance from a given point is impractical as it is inapplicable to other points
 - d. implicit means that don't have to resort the data for queries other than updates

SORTING ON THE BASIS OF SPATIAL OCCUPANCY

- Decompose the space from which the data is drawn into regions called *buckets* (like hashing but preserves order)
- Interested in methods that are designed specifically for the spatial data type being stored
- Basic approaches to decomposing space
 1. minimum bounding rectangles
 - e.g., R-tree or AABB (axis-aligned) and OBB (arbitrary orientation)
 - good at distinguishing empty and non-empty space
 - drawbacks:
 - a. non-disjoint decomposition of space
 - may need to search entire space
 - b. inability to correlate occupied and unoccupied space in two maps
 2. disjoint cells
 - drawback: objects may be reported more than once
 - uniform grid
 - a. all cells the same size
 - b. drawback: possibility of many sparse cells
 - adaptive grid — quadtree variants
 - a. regular decomposition
 - b. all cells of width power of 2
 - partitions at arbitrary positions
 - a. drawback: not a regular decomposition
 - b. e.g., R⁺-tree
- Can use as approximations in filter/refine query processing strategy



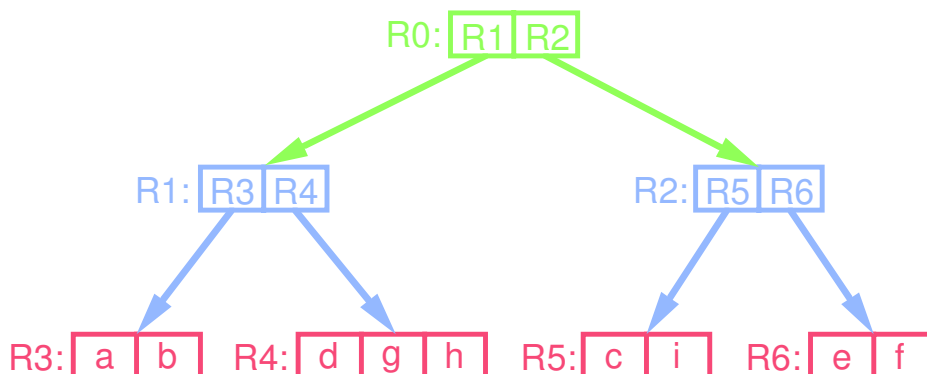
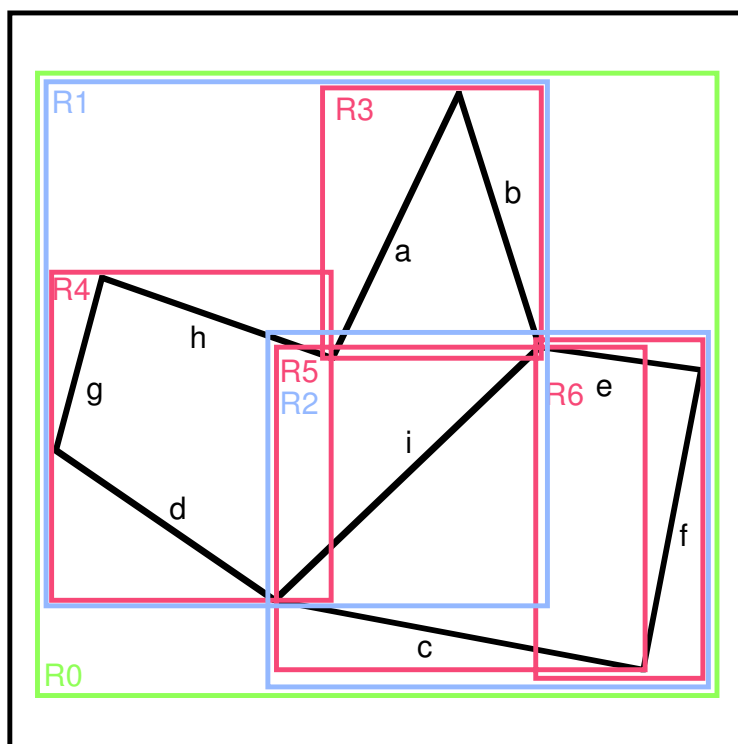
4	3	2	1
g	z	r	b

hi31



MINIMUM BOUNDING RECTANGLES

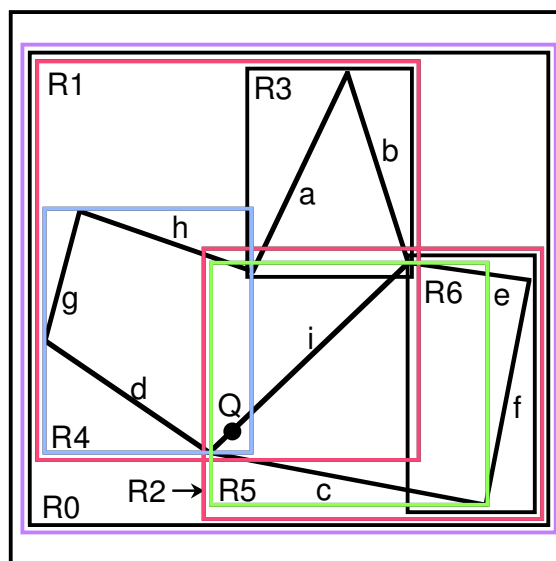
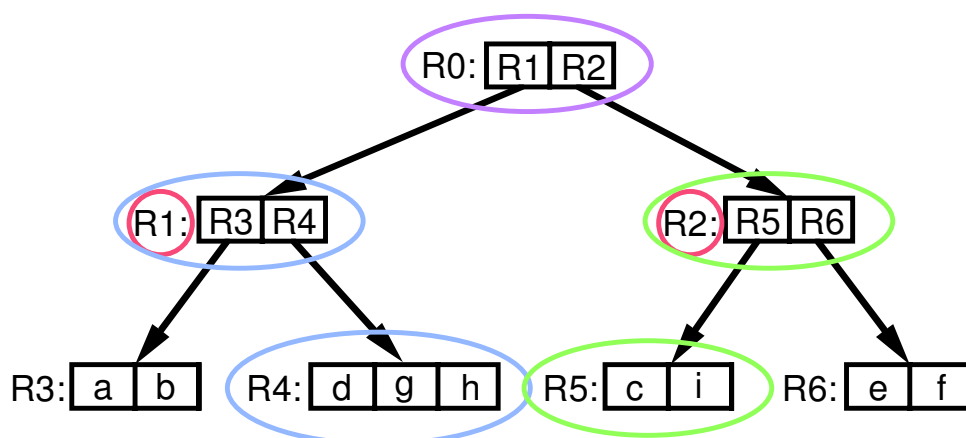
- Objects grouped into hierarchies, stored in a structure similar to a B-tree
- Drawback: not a disjoint decomposition of space
- Object has single bounding rectangle, yet area that it spans may be included in several bounding rectangles
- Examples include the R-tree and the R*-tree
- Order (m, M) R-tree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



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

- Drawback is that 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)

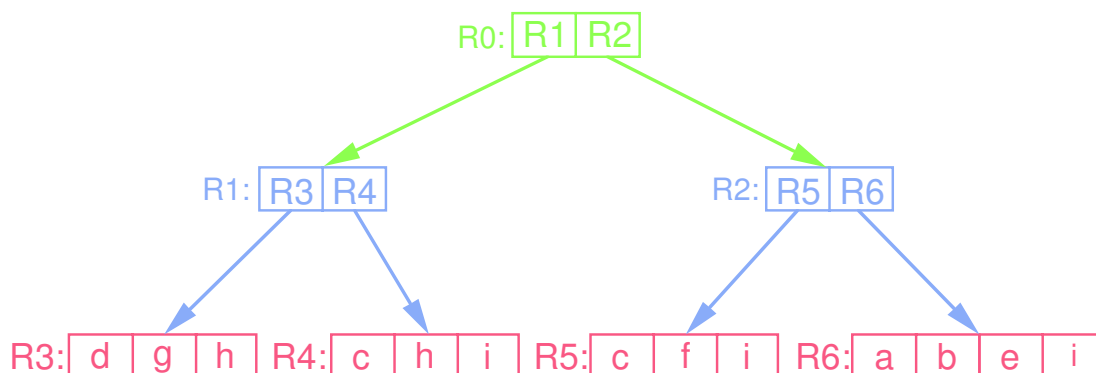
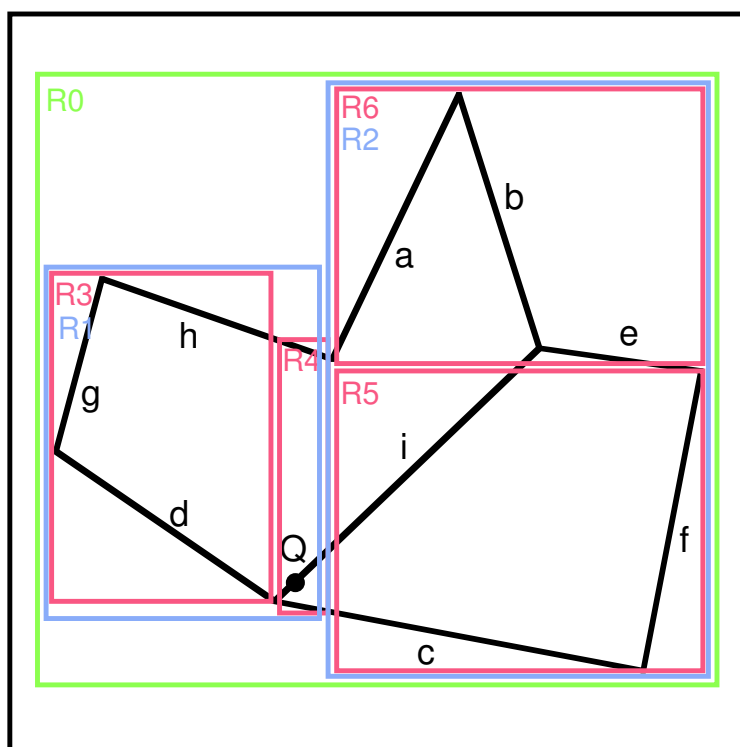
Ex: Search for a line segment containing point Q



- 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

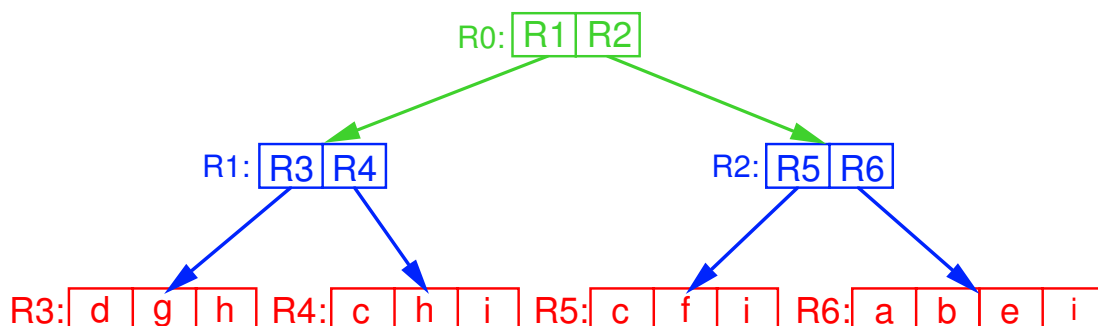
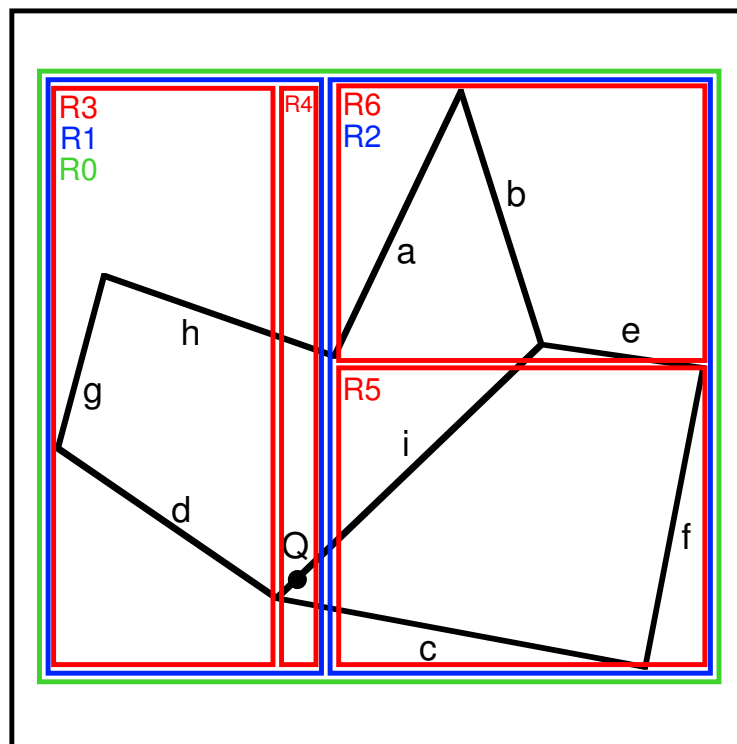
DISJOINT CELLS

- Objects decomposed into disjoint subobjects; each subobject in different cell
- Techniques differ in degree of regularity
- Drawback: in order to determine area covered by object, must retrieve all cells that it occupies
- R+-tree (also k-d-B-tree) and cell tree are examples of this technique



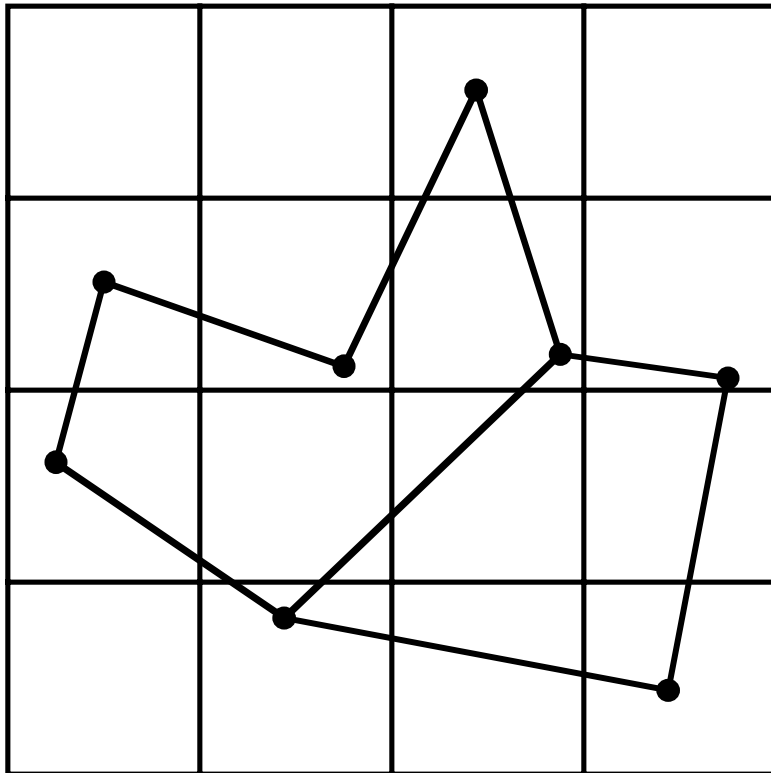
K-D-B-TREES

- Rectangular embedding space is hierarchically decomposed into disjoint rectangular regions
- No dead space in the sense that at any level of the tree, entire embedding space is covered by one of the nodes
- Blocks of k-d tree partition of space are aggregated into nodes of a finite capacity
- When a node overflows, it is split along one of the axes
- Originally developed to store points but may be extended to non-point objects represented by their minimum bounding boxes
- Drawback: in order to determine area covered by object, must retrieve all cells that it occupies



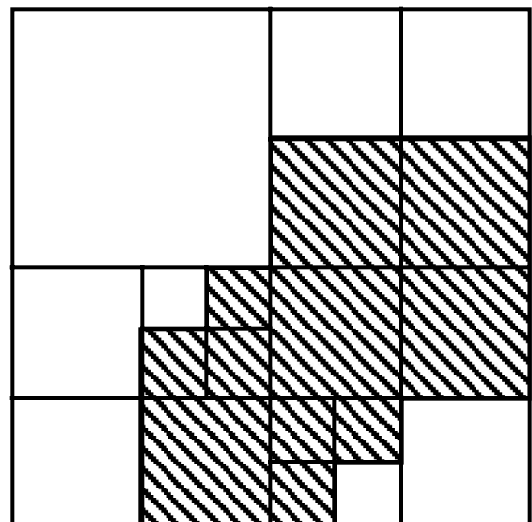
UNIFORM GRID

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



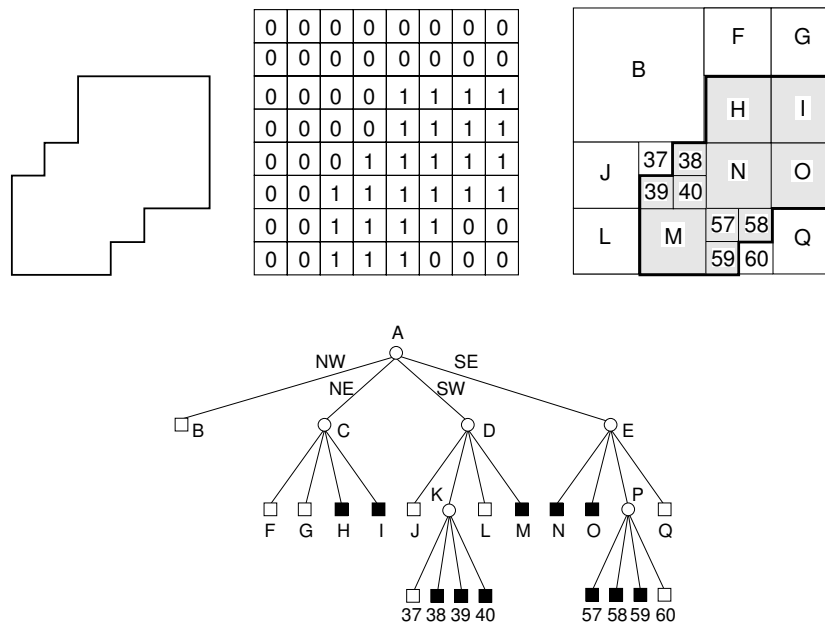
QUADTREES

- 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
 - changes meaning of nearest
 - a. nearest in time, OR
 - b. nearest in distance
- Can handle both raster and vector data as just a spatial index
- Shape is usually independent of order of inserting data
- Ex: region quadtree
- A decomposition into blocks — not necessarily a tree!



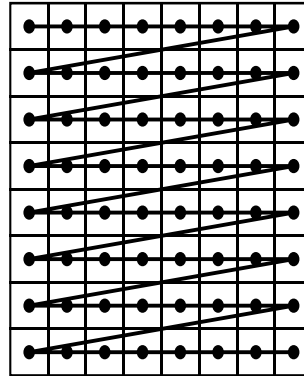
REGION QUADTREE

- Repeatedly subdivide until obtain homogeneous region
- For a binary image (BLACK \equiv 1 and WHITE \equiv 0)
- Can also use for multicolored data (e.g., a landuse class map associating colors with crops)
- Can also define data structure for grayscale images
- A collection of maximal blocks of size power of two and placed at predetermined positions
 1. could implement as a list of blocks each of which has a unique pair of numbers:
 - concatenate sequence of 2 bit codes corresponding to the path from the root to the block's node
 - the level of the block's node
 2. does not have to be implemented as a tree
 - tree good for logarithmic access
- A variable resolution data structure in contrast to a pyramid (i.e., a complete quadtree) which is a multiresolution data structure

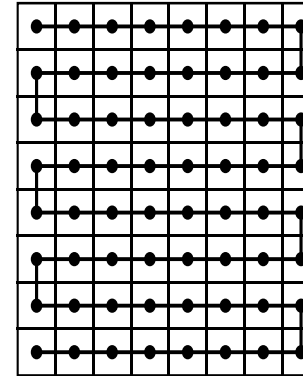


Ordering Space

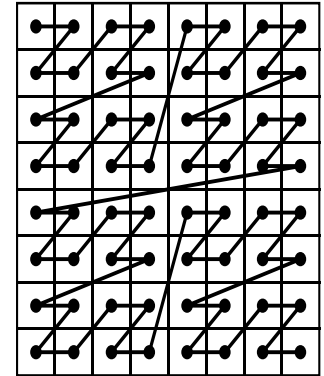
- Many ways of laying out the addresses corresponding to the locations in space of the cells each having its own mapping function
- Can use one of many possible space-filling curves
- Important to distinguish between *address* and *location* or *cell*
- *Address of a location or cell* \equiv physical location (e.g., in memory, on disk, etc.), if any, where some of the information associated with the *location* or *cell* is stored



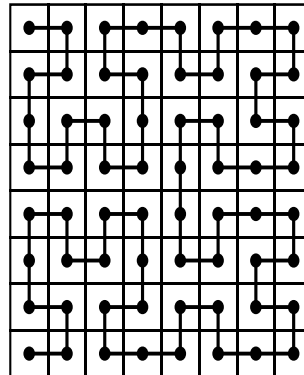
row order



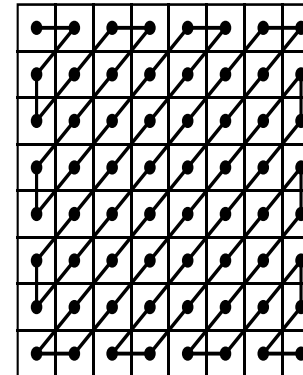
row-prime order



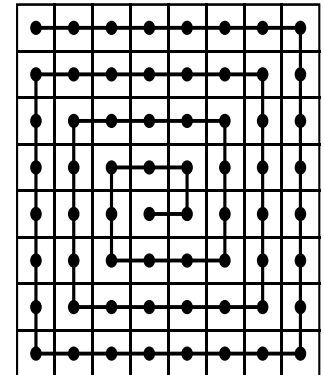
morton order



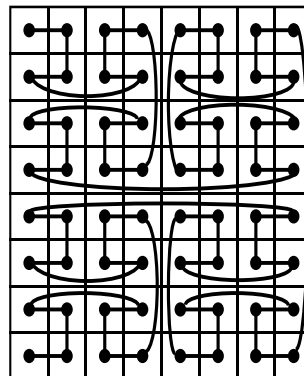
peano-hilbert order



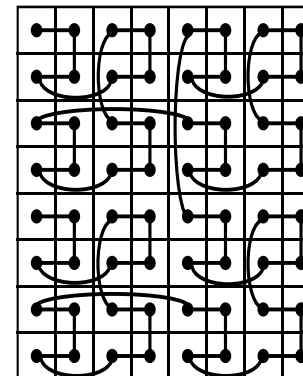
cantor-diagonal order



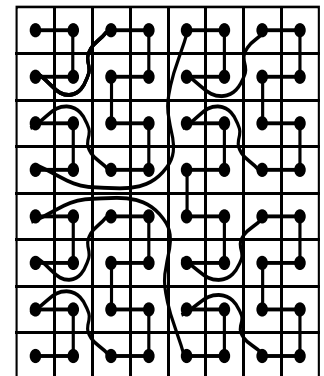
spiral order



gray code



double gray order



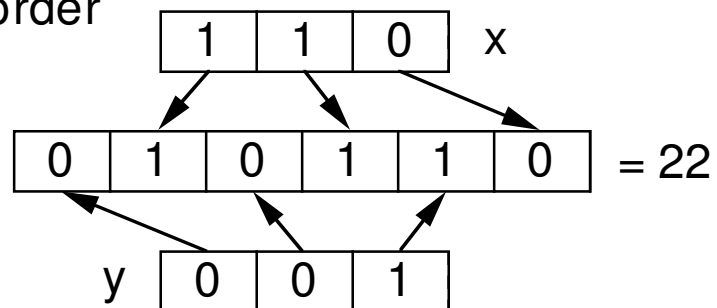
u order

CONVERTING BETWEEN POINTS AND CURVES

- Need to know size of image for all but the Morton order
- Relatively easy for all but the Peano-Hilbert order which is difficult (although possible) to decode and encode to obtain the corresponding x and y coordinate values
- Morton order
 1. use bit interleaving of binary representation of the x and y coordinates of the point

2. also known as Z-order

3. Ex: Atlanta (6,1)

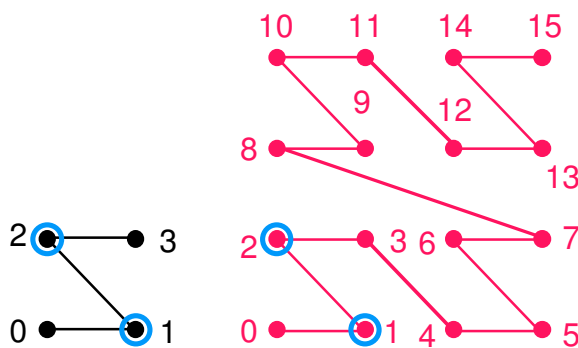




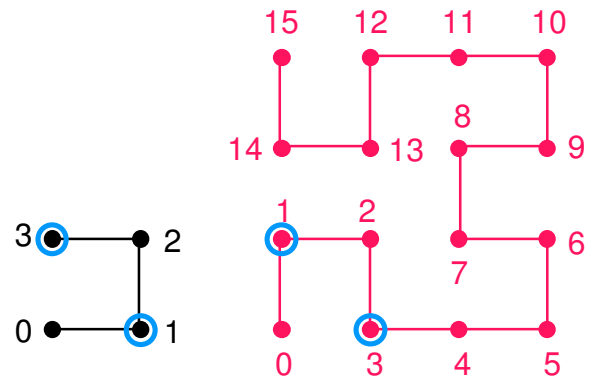
STABILITY OF SPACE ORDERING METHODS

- An order is *stable* if the relative order of the individual pixels is maintained when the resolution (i.e., the size of the space in which the cells are embedded) is doubled or halved
- Morton order is stable while the Peano-Hilbert order is not
- Ex:

Morton:



Peano-Hilbert:



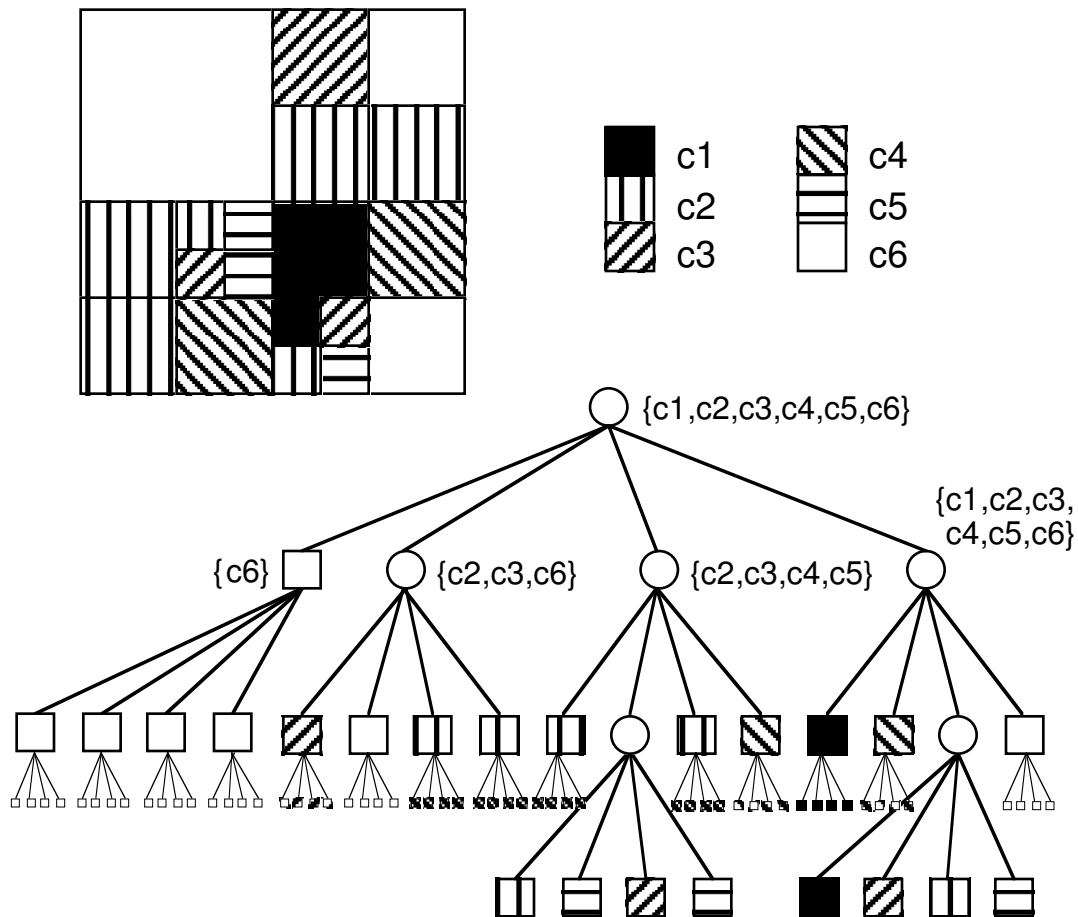
- Result of doubling the resolution (i.e., the coverage) in which case the circled points do not maintain the same relative order in the Peano-Hilbert order while they do in the Morton order

DESIRABLE PROPERTIES OF SPACE FILLING CURVES

1. Pass through each point in the space once and only once
2. Two points that are neighbors in space are neighbors along the curve and vice versa
 - impossible to satisfy for all points at all resolutions
3. Easy to retrieve neighbors of a point
4. Curve should be stable as the space grows and contracts by powers of two with the same origin
 - yes for Morton and Cantor orders
 - no for row, row-prime, Peano-Hilbert, and spiral orders
5. Curve should be admissible
 - at each step at least one horizontal and one vertical neighbor must have already been encountered
 - used by active border algorithms - e.g., connected component labeling algorithm
 - row, Morton, and Cantor orders are admissible
 - Peano-Hilbert order is not admissible
 - row-prime and spiral orders are admissible if permit the direction of the horizontal and vertical neighbors to vary from point to point
6. Easy to convert between two-dimensional data and the curve and vice-versa
 - easy for Morton order
 - difficult for Peano-Hilbert order
 - relatively easy for row, row-prime, Cantor, and spiral orders

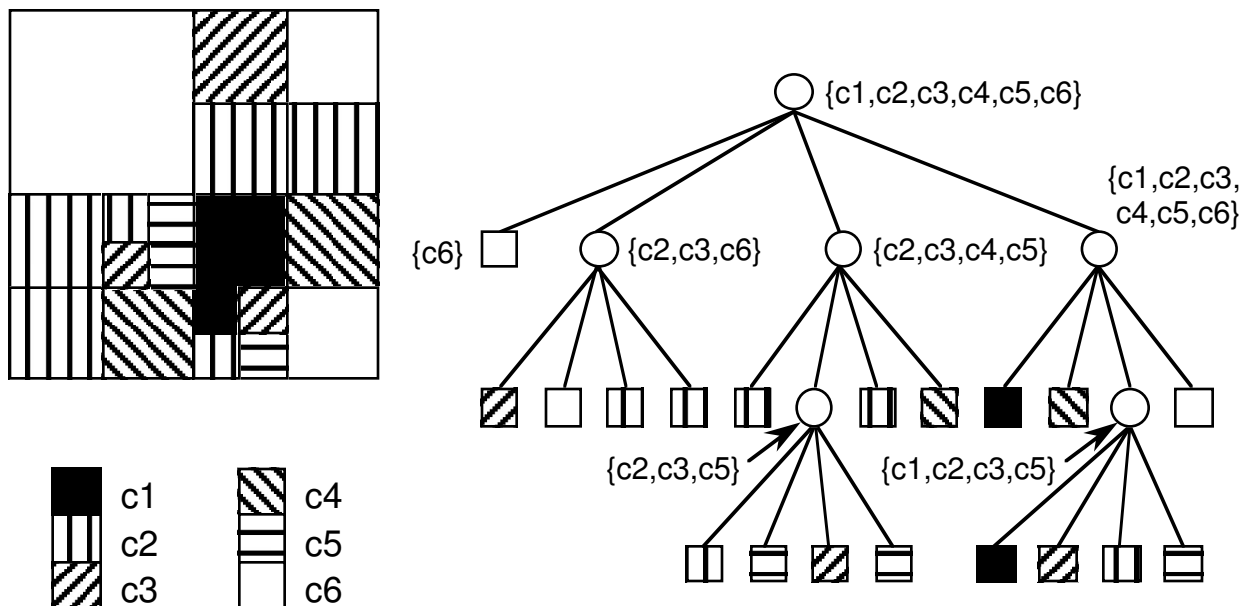
PYRAMID

- Internal nodes contain summary of information in nodes below them
- Useful for avoiding inspecting nodes where there could be no relevant information



QUADTREES VS. PYRAMIDS

- Quadtrees are good for location-based queries
 1. e.g., what is at location x ?
 2. not good if looking for a particular feature as have to examine every block or location asking “are you the one I am looking for?”
- Pyramid is good for feature-based queries — e.g.,
 1. does wheat exist in region x ?
 - if wheat does not appear at the root node, then impossible to find it in the rest of the structure and the search can cease
 2. report all crops in region x — just look at the root
 3. select all locations where wheat is grown
 - only descend node if there is possibility that wheat is in one of its four sons — implies little wasted work
- Ex: truncated pyramid where 4 identically-colored sons are merged



- Can represent as a list of leaf and nonleaf blocks (e.g., as a linear quadtree)

Outline

1. Introduction
2. Points
3. Lines
4. Regions, Volumes, and Surfaces
5. Bounding Box Hierarchies
6. Rectangles
7. Surfaces and Volumes
8. Metric Data
9. Operations
10. Example system



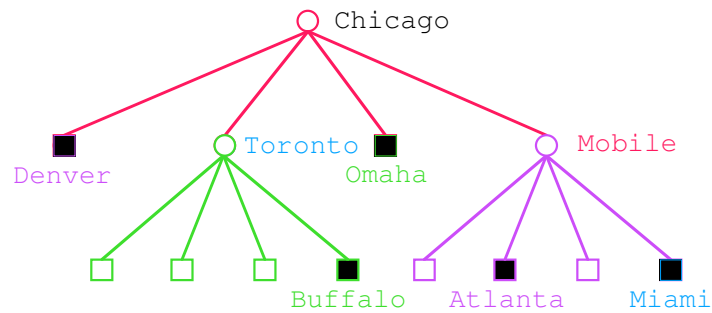
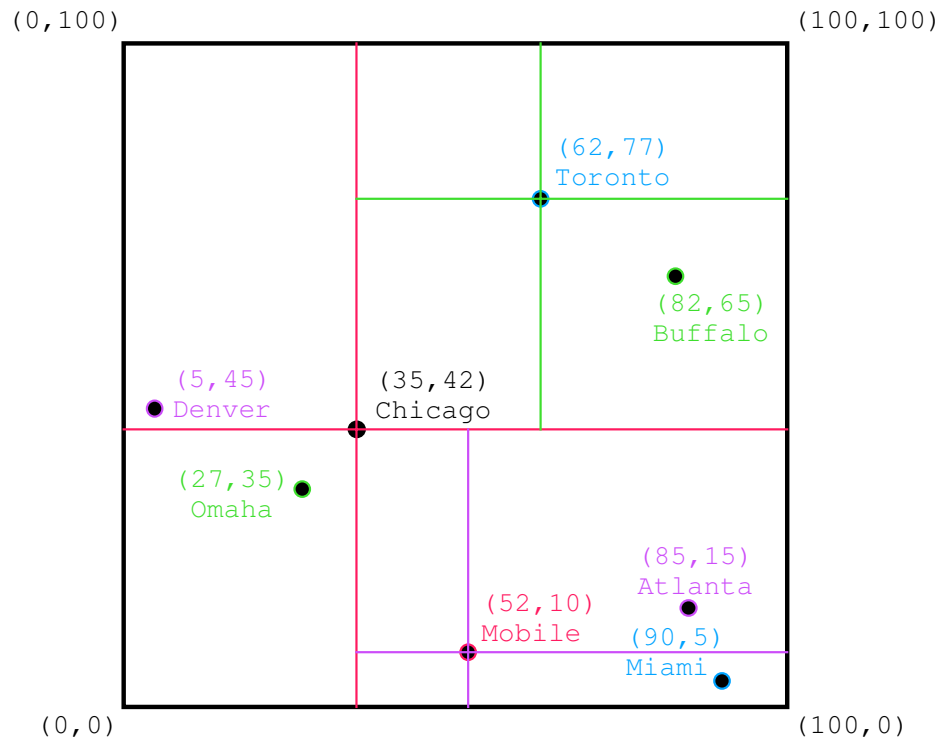
POINT QUADTREE (Finkel/Bentley)

8	7	6	5	4	3	2	1
z	v	g	v	g	z	r	b

hp4



- Marriage between a uniform grid and a binary search tree

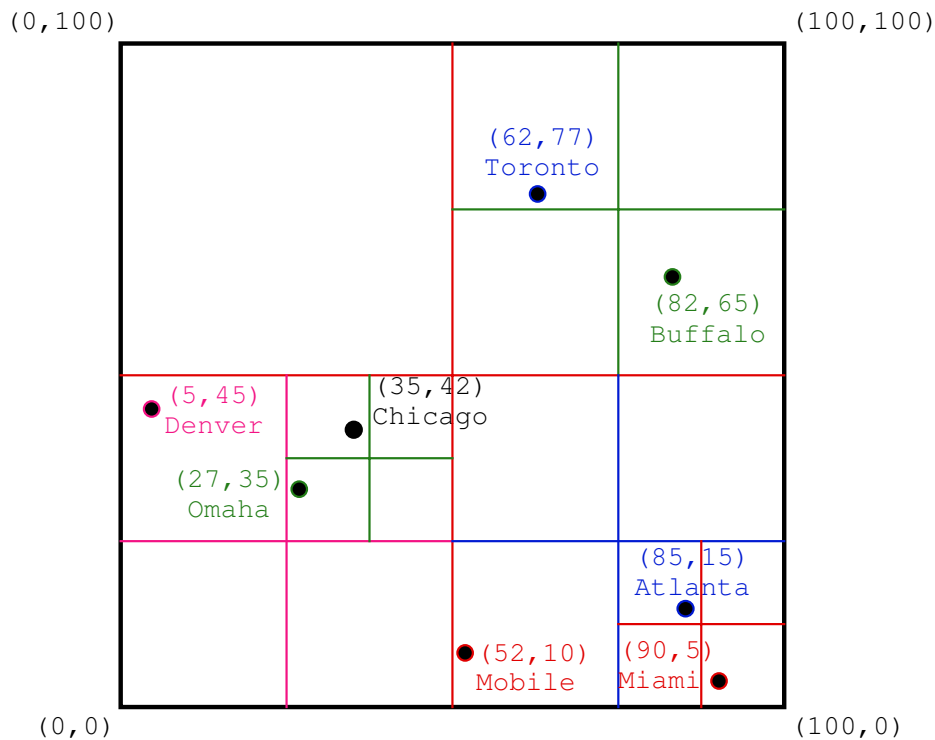


PR QUADTREE (Orenstein)

8	7	6	5	4	3	2	1
r	z	g	v	g	z	r	b

hp9

1. Regular decomposition point representation
2. Decomposition occurs whenever a block contains more than one point
3. Useful when the domain of data points is not discrete but finite
4. Maximum level of decomposition depends on the minimum separation between two points
 - if two points are very close, then decomposition can be very deep
 - can be overcome by viewing blocks as buckets with capacity c and only decomposing the block when it contains more than c points

Ex: $c = 1$ 

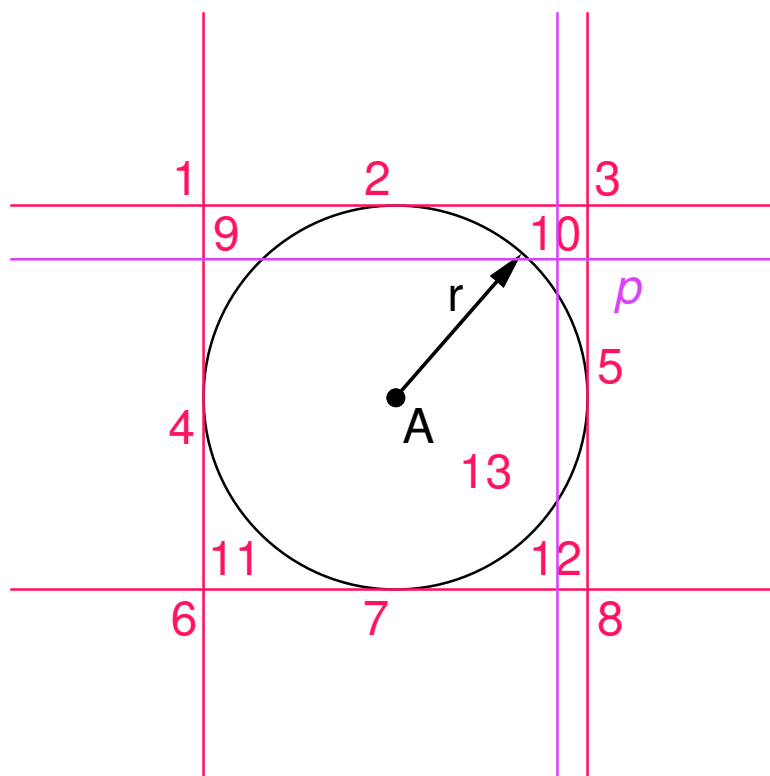


REGION SEARCH

5	4	3	2	1
v	g	z	r	b

hp10

- Ex: Find all points within radius r of point A



- Use of quadtree results in pruning the search space
- If a quadrant subdivision point p lies in a region l , then search the quadrants of p specified by l

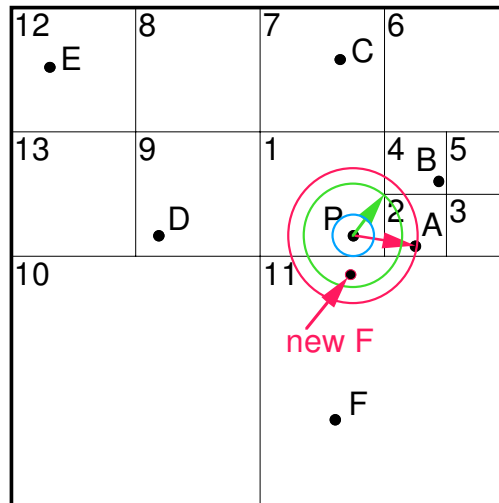
1. SE	6. NE	11. All but SW
2. SE, SW	7. NE, NW	12. All but SE
3. SW	8. NW	13. All
4. SE, NE	9. All but NW	
5. SW, NW	10. All but NE	

FINDING THE NEAREST OBJECT

7	6	5	4	3	2	1
r	z	v	g	z	r	b

zk24

- Ex: find the nearest object to P



- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:
 - start at block 2 and compute distance to P from A
 - ignore block 3 whether or not it is empty as A is closer to P than any point in 3
 - examine block 4 as distance to sw corner is shorter than the distance from P to A; however, reject B as it is further from P than A
 - ignore blocks 6, 7, 8, 9, and 10 as the minimum distance to them from P is greater than the distance from P to A
 - examine block 11 as the distance from P to the southern border of 1 is shorter than the distance from P to A; however, reject F as it is further from P than A
- If F was moved, a better order would have started with block 11, the southern neighbor of 1, as it is closest



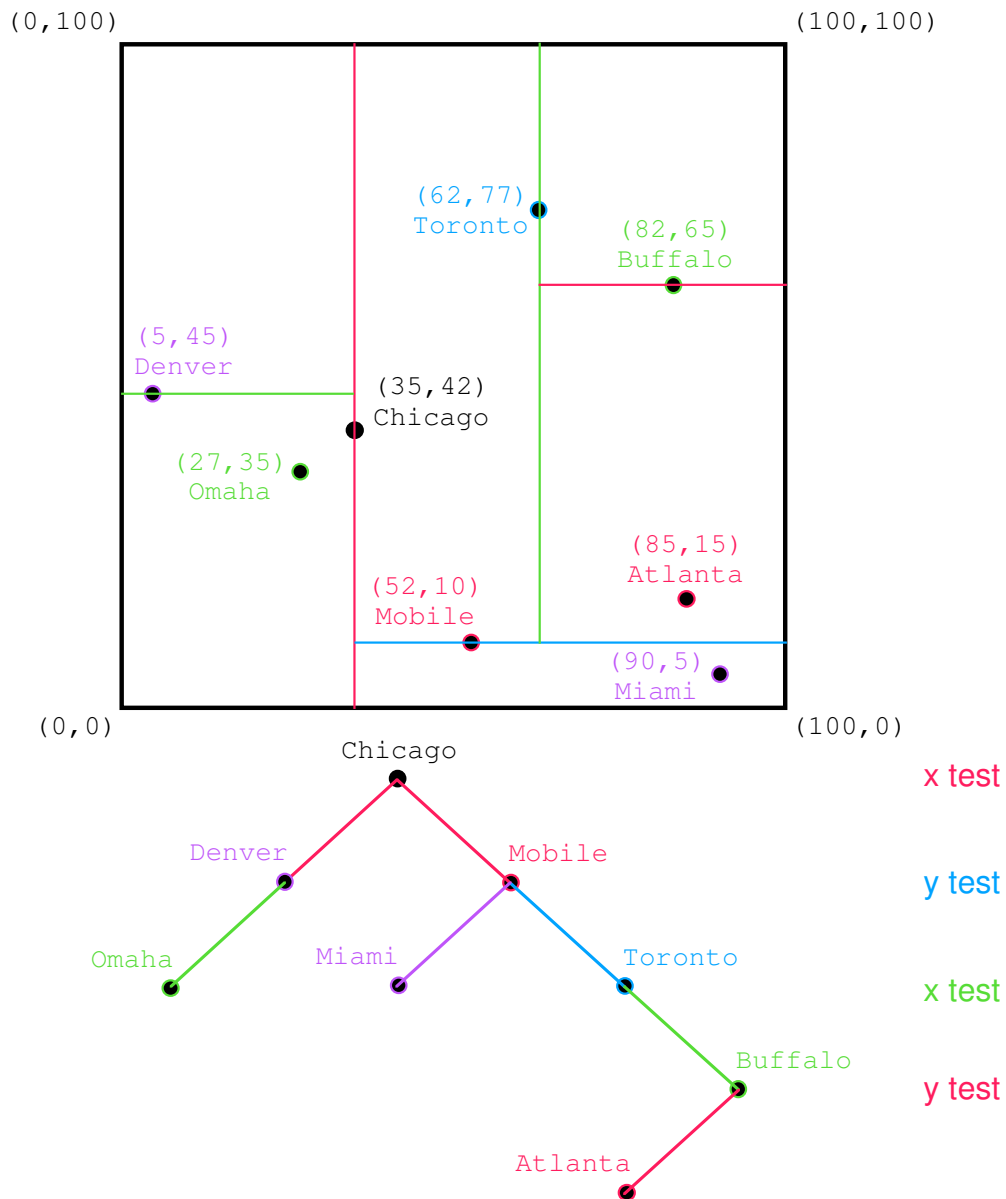
K-D TREE (Bentley)

8	7	6	5	4	3	2	1
v	r	g	v	g	z	r	b

hp15



- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered





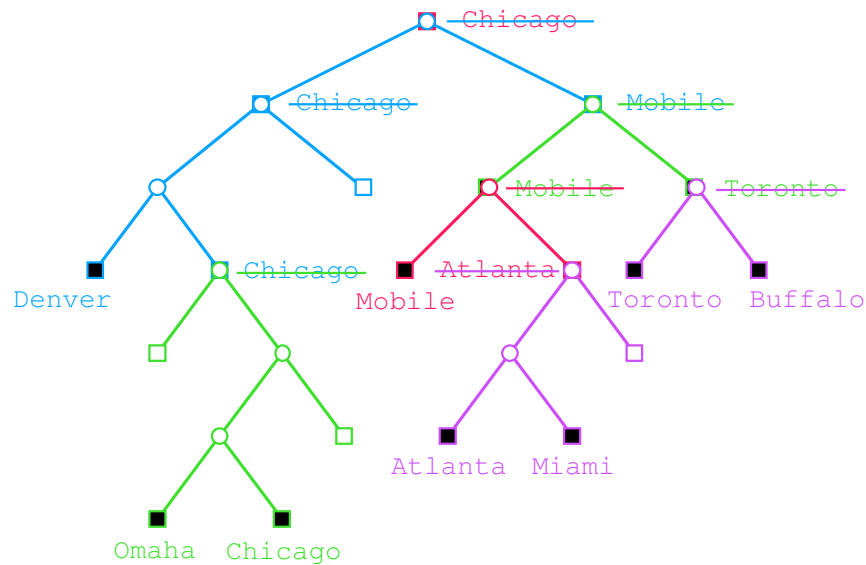
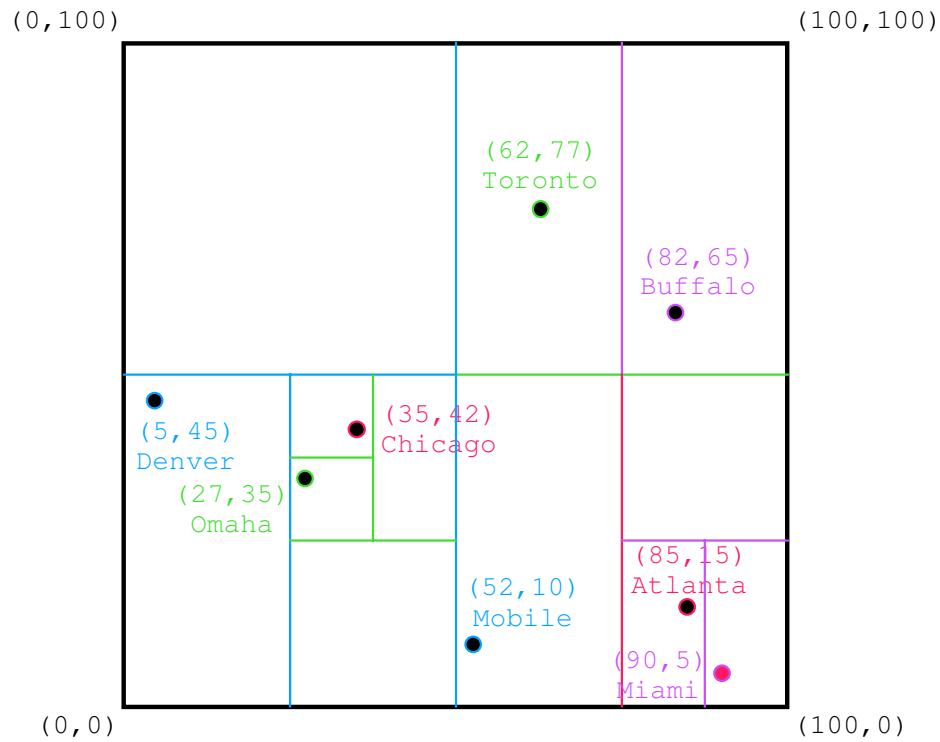
PR K-D TREE (Knowlton)

9	8	7	6	5	4	3	2	1
v	r	g	z	v	g	z	r	b

hp19



- A region contains at most one data point
- Analogous to EXCELL with bucket size of 1



Outline

1. Introduction
2. Points
3. Lines
4. Regions, Volumes, and Surfaces
5. Bounding Box Hierarchies
6. Rectangles
7. Surfaces and Volumes
8. Metric Data
9. Operations
10. Example system

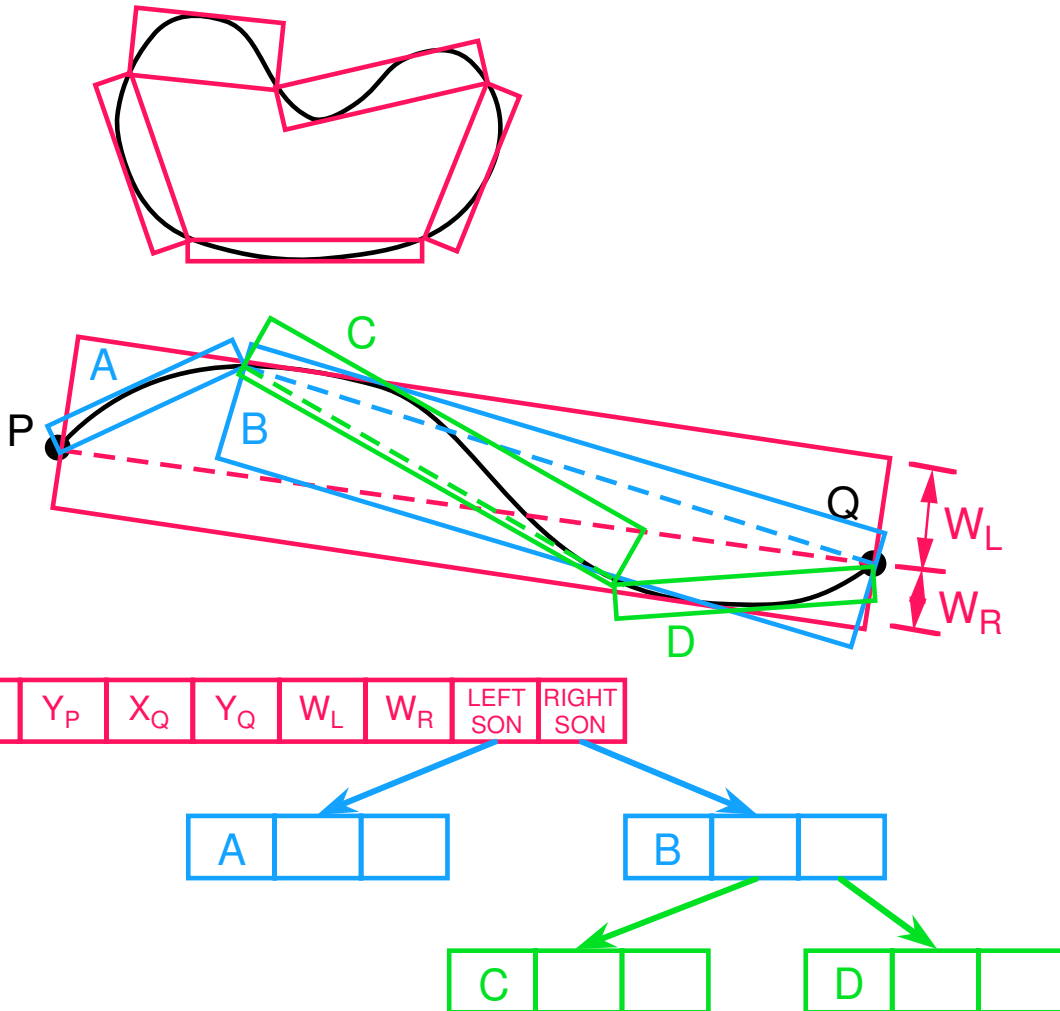


STRIP TREE (Ballard, Peucker)

5	4	3	2	1
v	g	z	r	b

cd4

- 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 $\leq W$

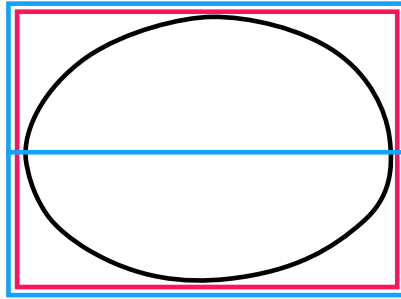


SPECIAL CASES

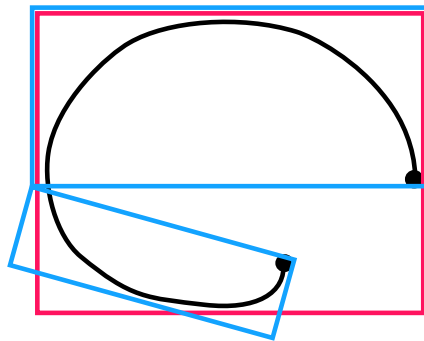
3	2	1
z	r	b

cd5 

1. Closed curve



2. Curve extends beyond its endpoints



- enclosed by a rectangle
- split into two rectangular strips

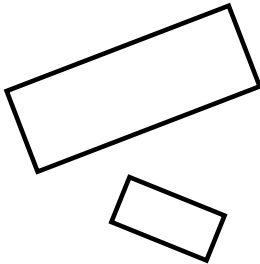


APPLICATIONS

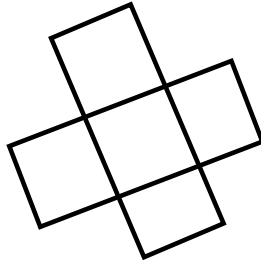
4	3	2	1
g	z	r	b

cd6

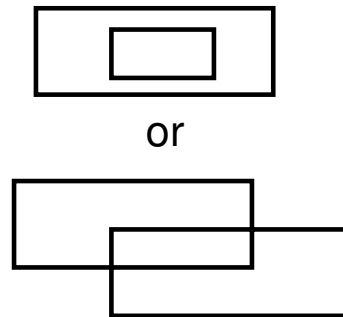
1. Curve intersection



NULL

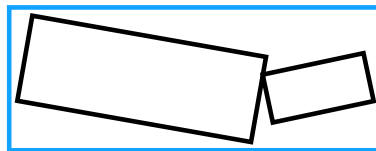


CLEAR



POSSIBLE

2. Union of two curves



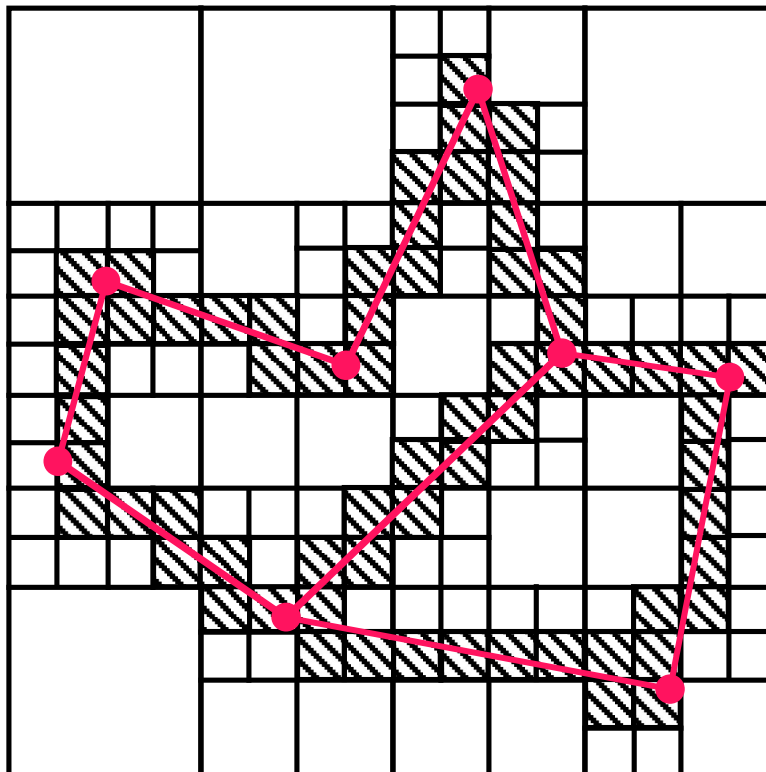
- not possible as the result may fail to be continuous

3. Others

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

MX QUADTREE FOR REGIONS (Hunter)

- 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. no more than 4 lines can meet at a point





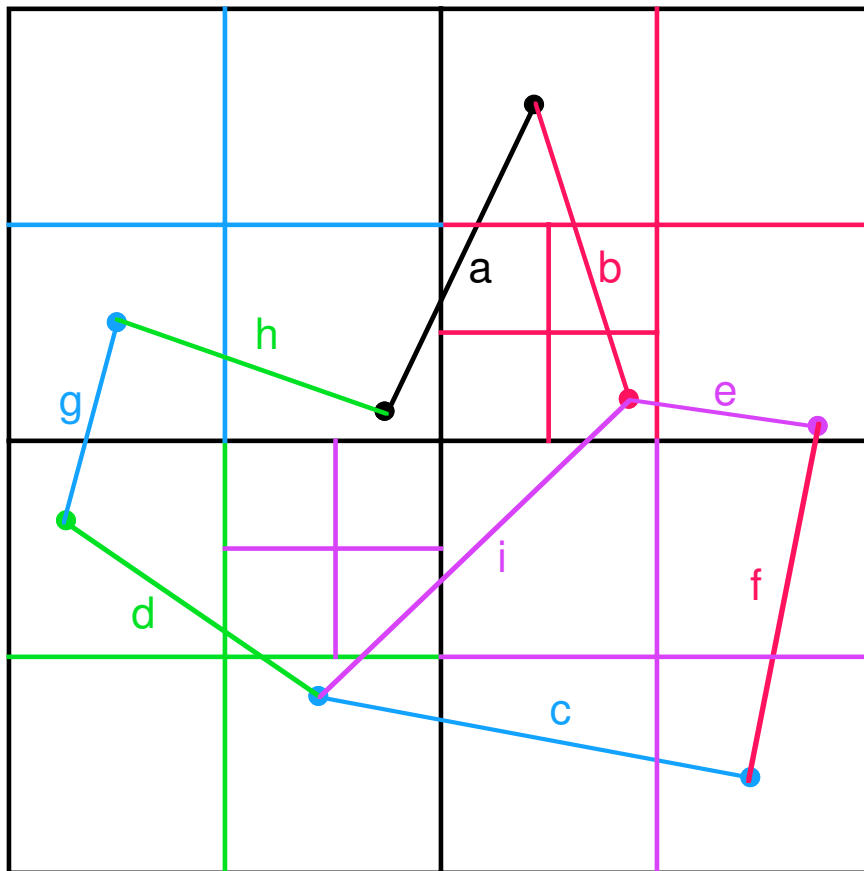
PM1 QUADTREE

9	8	7	6	5	4	3	2	1
v	g	z	r	v	g	z	r	b

cd32



- 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



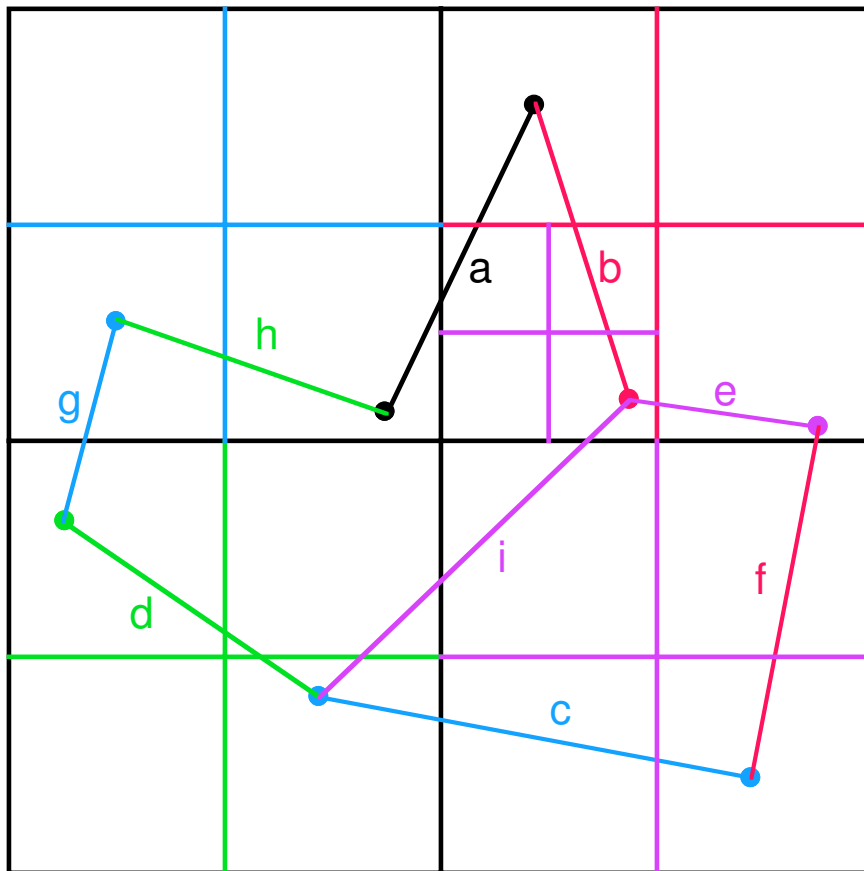
PM2 QUADTREE

9	8	7	6	5	4	3	2	1
v	g	z	r	v	g	z	r	b

cd33



- Vertex-based (one vertex per block)



DECOMPOSITION RULE:

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

- Shape independent of order of insertion

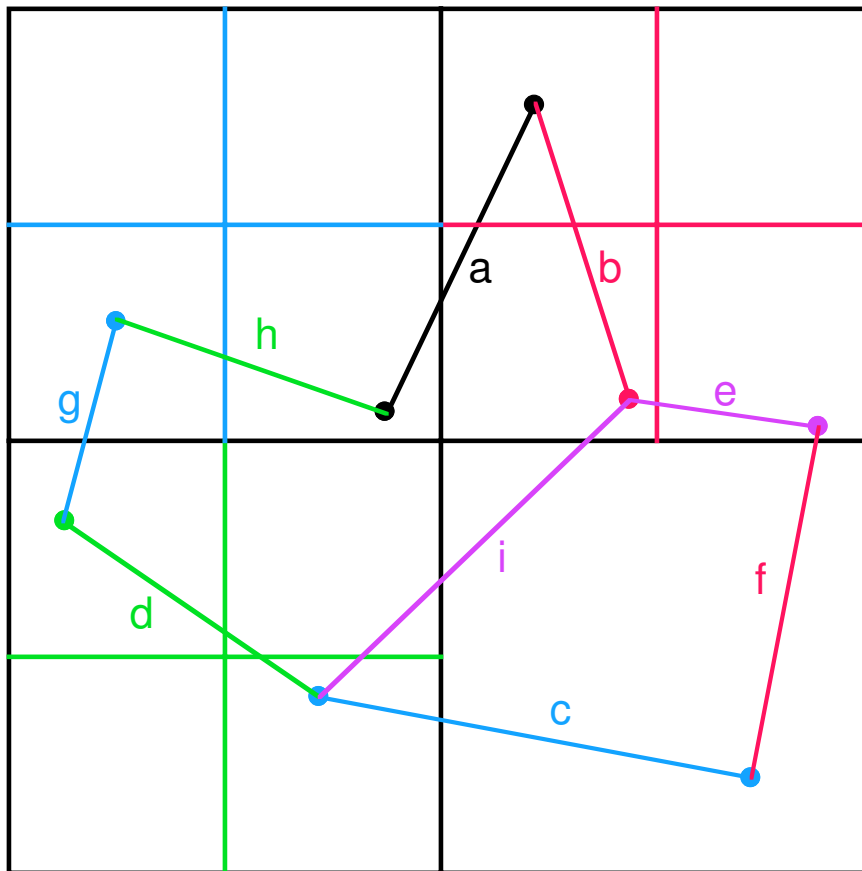


PM3 QUADTREE

9	8	7	6	5	4	3	2	1
v	g	z	r	v	g	z	r	b

cd34

- 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

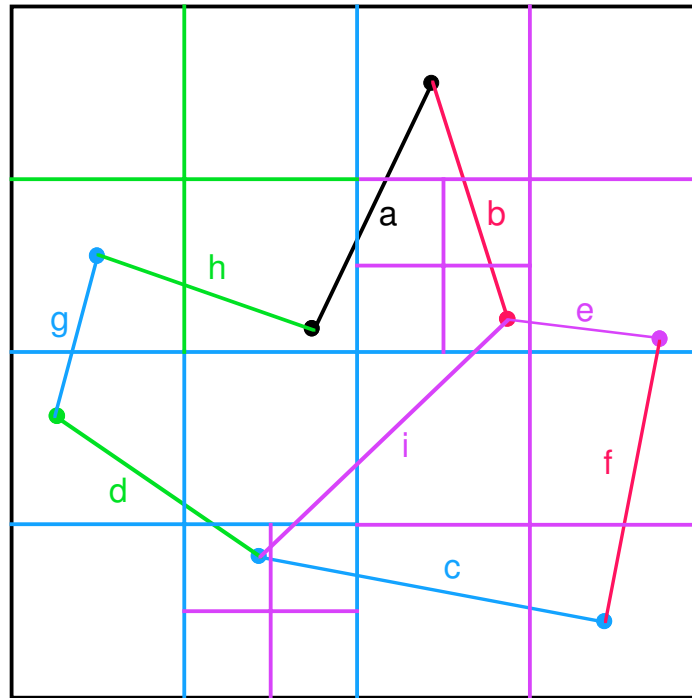
9	8	7	6	5	4	3	2	1
v	g	z	r	v	g	z	r	b

cd35



- 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:

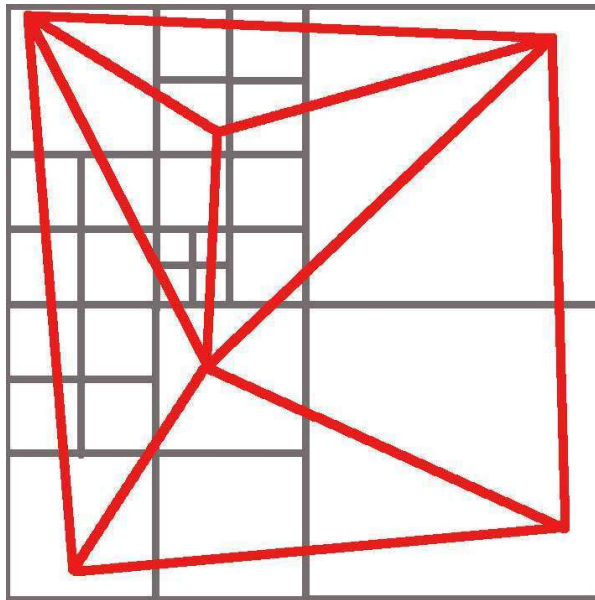
Split a block *once* if upon insertion the number of segments intersecting a block exceeds N

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

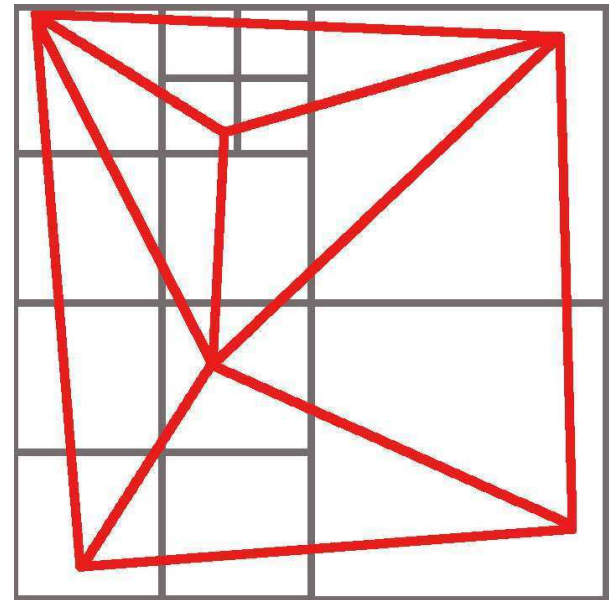
- Merges can be performed more than once
- Does not guarantee that each block will contain at most N line segments
- Splitting threshold is not the same as bucket capacity
- Shape depends on order of insertion

Triangulations

- PM_2 quadtree is quite useful vis-a-vis PM_1 quadtree
- Given a triangle table, only need to store at most a single vertex with each cell and can reconstruct mesh with the aid of clipping
- Example triangular mesh



PM_1 quadtree



PM_2 quadtree

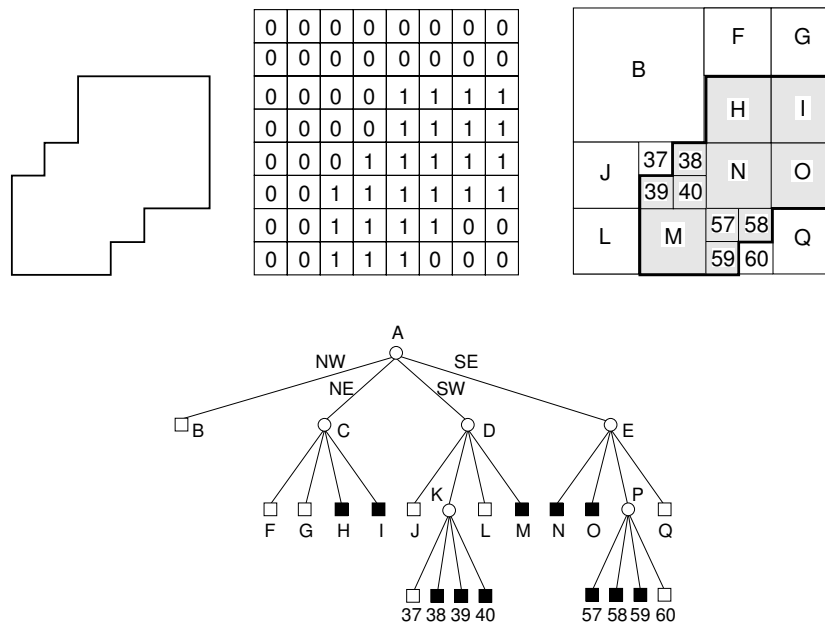
- Can also formulate a PM-triangle quadtree variant

Outline

1. Introduction
2. Points
3. Lines
4. Regions, Volumes, and Surfaces
5. Bounding Box Hierarchies
6. Rectangles
7. Surfaces and Volumes
8. Metric Data
9. Operations
10. Example system

REGION QUADTREE

- Repeatedly subdivide until obtain homogeneous region
- For a binary image (BLACK \equiv 1 and WHITE \equiv 0)
- Can also use for multicolored data (e.g., a landuse class map associating colors with crops)
- Can also define data structure for grayscale images
- A collection of maximal blocks of size power of two and placed at predetermined positions
 1. could implement as a list of blocks each of which has a unique pair of numbers:
 - concatenate sequence of 2 bit codes corresponding to the path from the root to the block's node
 - the level of the block's node
 2. does not have to be implemented as a tree
 - tree good for logarithmic access
- A variable resolution data structure in contrast to a pyramid (i.e., a complete quadtree) which is a multiresolution data structure

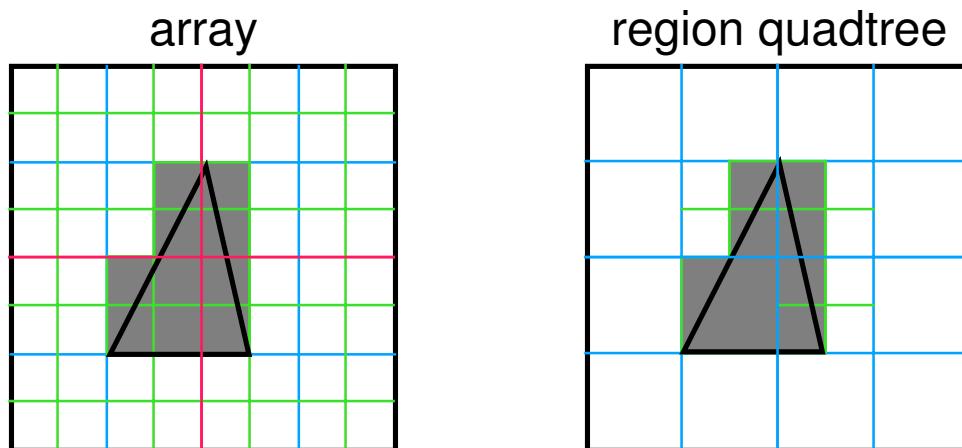


SPACE REQUIREMENTS

1. Rationale for using quadtrees/octrees is not so much for saving space but for saving execution time
2. Execution time of standard image processing algorithms that are based on traversing the entire image and performing a computation at each image element is proportional to the number of blocks in the decomposition of the image rather than their size
 - aggregation of space leads directly to execution time savings as the aggregate (i.e., block) is visited just once instead of once for each image element (i.e., pixel, voxel) in the aggregate (e.g., connected component labeling)
3. If want to save space, then, in general, statistical image compression methods are superior
 - drawback: statistical methods are not progressive as need to transmit the entire image whereas quadtrees lend themselves to progressive approximation
 - quadtrees, though, do achieve compression as a result of use of common subexpression elimination techniques
 - a. e.g., checkerboard image
 - b. see also vector quantization
4. Sensitive to positioning of the origin of the decomposition
 - for an $n \times n$ image, the optimal positioning requires an $O(n^2 \log_2 n)$ dynamic programming algorithm (Li, Grosky, and Jain)

DIMENSION REDUCTION

- Number of blocks necessary to store a simple polygon as a region quadtree is proportional to its perimeter (Hunter)
 - implies that many quadtree algorithms execute in $O(\text{perimeter})$ time as they are tree traversals
 - the region quadtree is a dimension reducing device as perimeter (ignoring fractal effects) is a one-dimensional measure and we are starting with two-dimensional data
 - generalizes to higher dimensions
 - region octree takes $O(\text{surface area})$ time and space (Meagher)
 - d -dimensional data take time and space proportional to a $O(d-1)$ -dimensional quantity (Walsh)
- Alternatively, for a region quadtree, the space requirements double as the resolution doubles
 - in contrast with quadrupling in the array representation
 - for a region octree the space requirements quadruple as the resolution doubles
 - ex.



- easy to see dependence on perimeter as decomposition only takes place on the boundary as the resolution increases



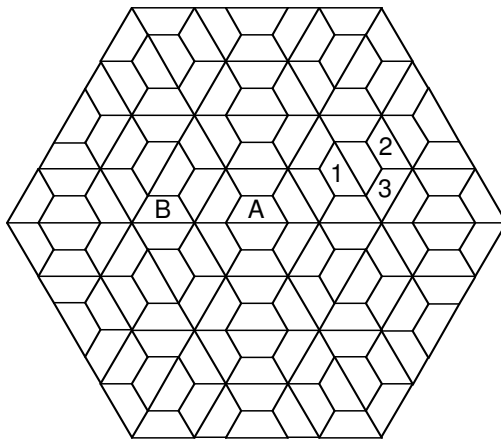
tl1



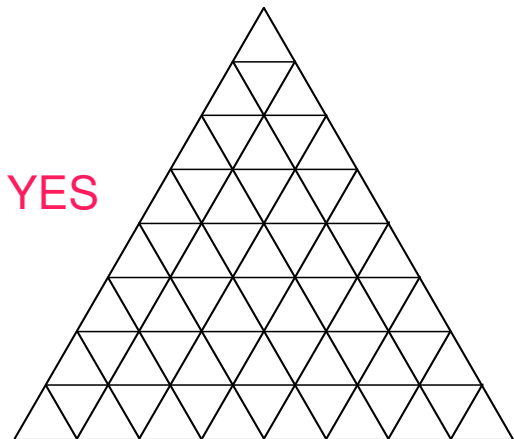
ALTERNATIVE DECOMPOSITION METHODS

- A planar decomposition for image representation should be:
 1. infinitely repetitive
 2. infinitely decomposable into successively finer patterns
- Classification of tilings (Bell, Diaz, Holroyd, and Jackson)
 1. isohedral — all tiles are equivalent under the symmetry group of the tiling (i.e., when stand in one tile and look around, the view is independent of the tile)

NO

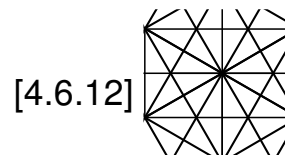
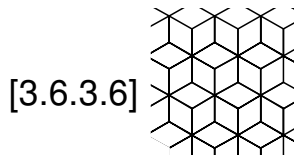
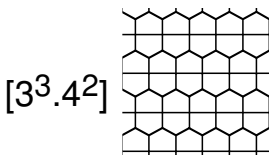
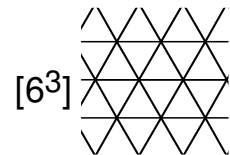
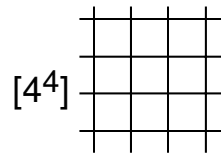
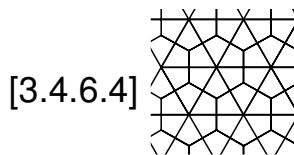
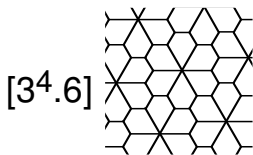
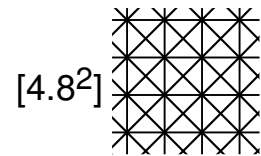
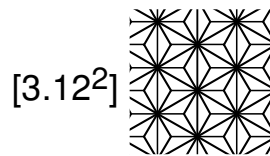
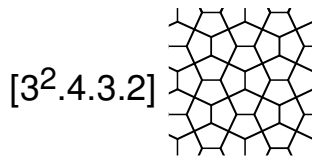
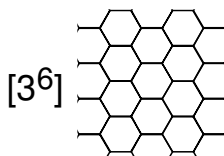


YES



2. regular — each tile is a regular polygon

- There are 81 types if classify by their symmetry groups
- Only 11 types if classify by their adjacency structure



- $[3.12^2]$ means 3 edges at the first vertex of the polygonal tile followed by 12 edges at the next two vertices

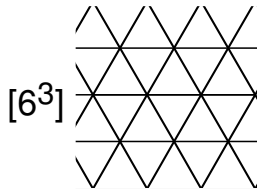


tl2

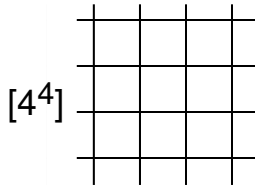


PROPERTIES OF TILINGS — SIMILARITY

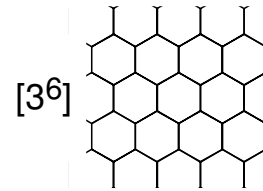
- Similarity — a tile at level k has the same shape as a tile at level 0 (basic tile shape)



YES

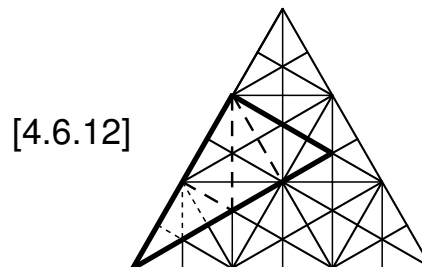
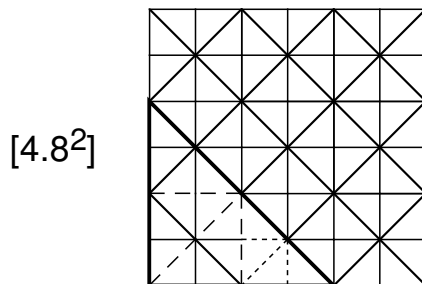


YES

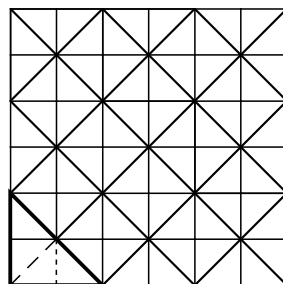
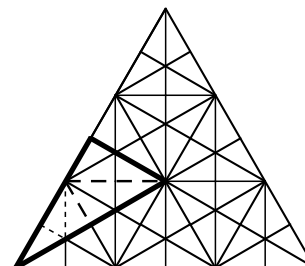


NO

- Limited \equiv NOT similar (i.e., cannot be decomposed infinitely into smaller tiles of the same shape)
- Unlimited: each edge of each tile lies on an infinitely straight line composed entirely of edges
- Only 4 unlimited tilings $[4^4]$, $[6^3]$, $[4.8^2]$, and $[4.6.12]$



- Two additional hierarchies:

rotation of 135° between levels

reflection between levels

Note: $[4.8^2]$ and $[4.6.12]$ are not regular



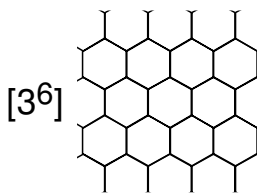
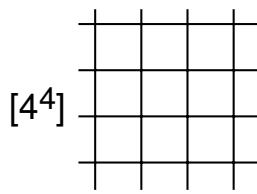
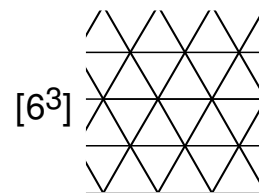
PROPERTIES OF TILINGS — ADJACENCY



tl3



- Adjacency — two tiles are neighbors if they are adjacent along an edge or at a vertex
- Uniform adjacency \equiv distances between the centroid of one tile and the centroids of all its neighbors are the same
- Adjacency number of a tiling (A) \equiv number of different adjacency distances

 $A=1$  $A=2$  $A=3$

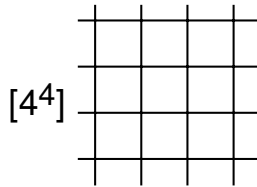


tl4

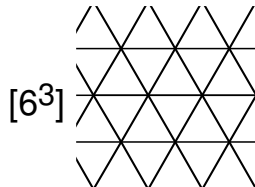


PROPERTIES OF TILINGS — UNIFORM ORIENTATION

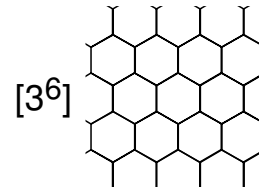
- Uniform orientation
- All tiles with the same orientation can be mapped into each other by translations of the plane which do not involve rotation or reflection



YES



NO



YES

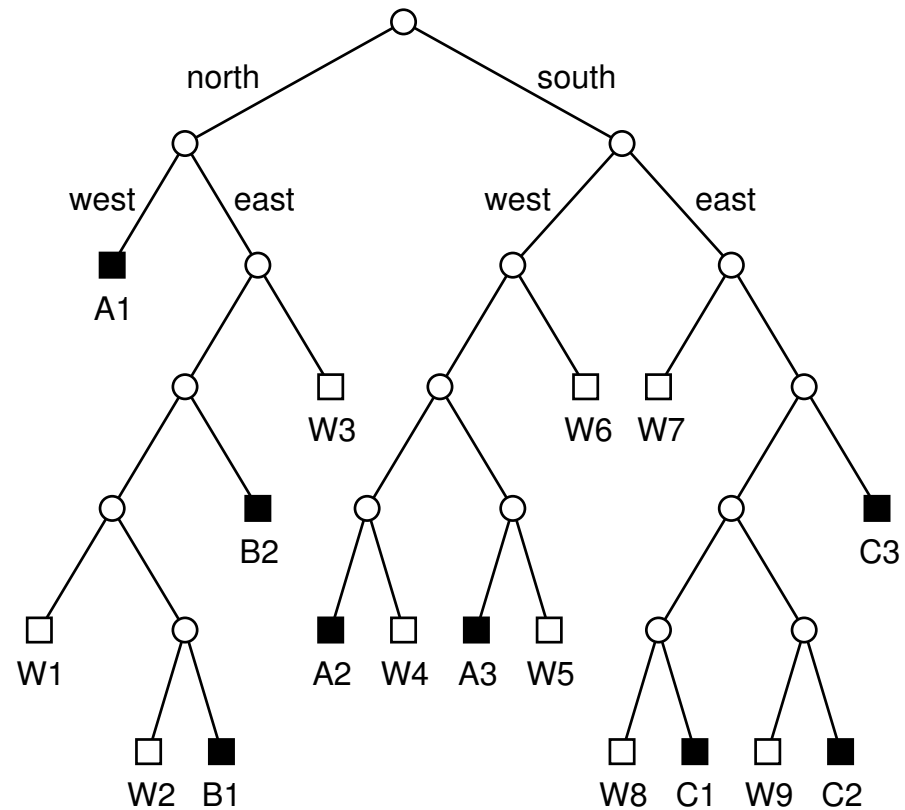
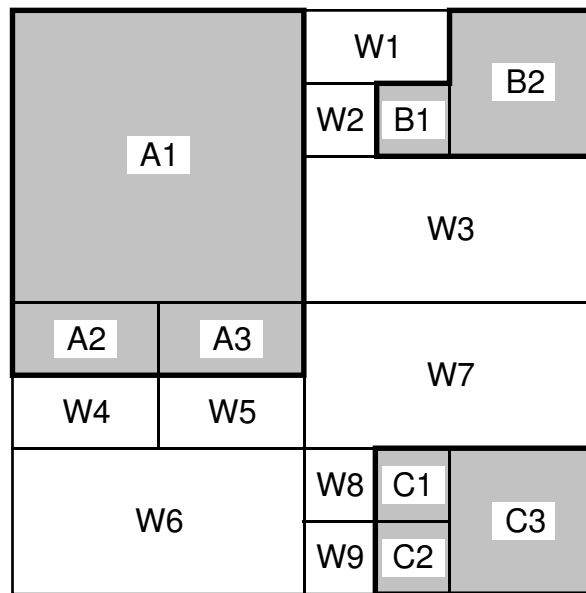
Conclusion:

- $[4^4]$ has a lower adjacency number than $[6^3]$
- $[4^4]$ has a uniform orientation while $[6^3]$ does not
- $[4^4]$ is unlimited while $[3^6]$ is limited

Use $[4^4]$!

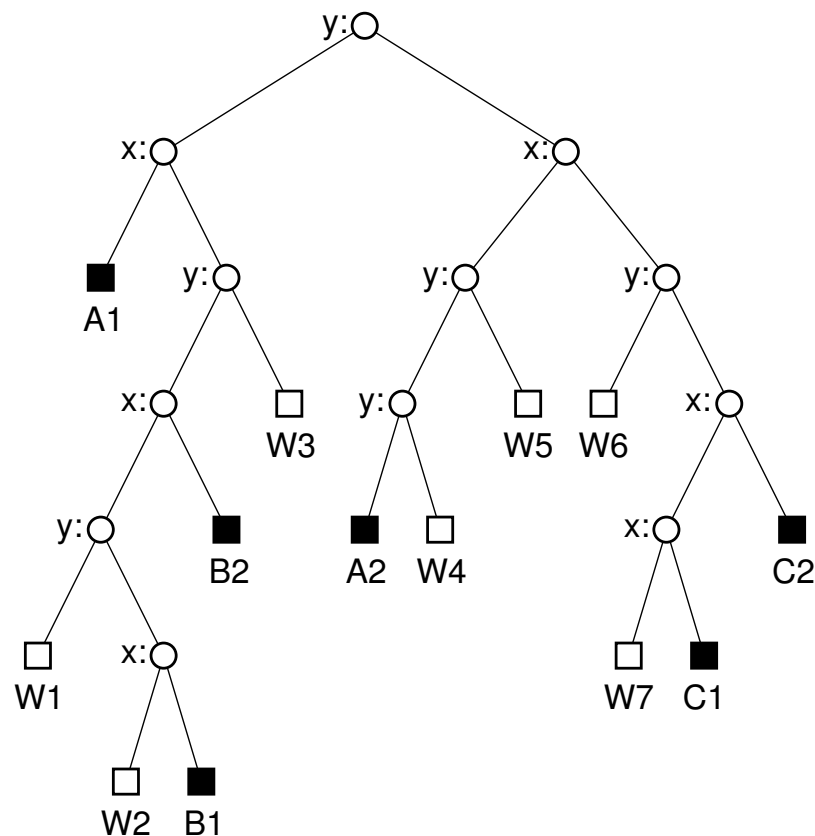
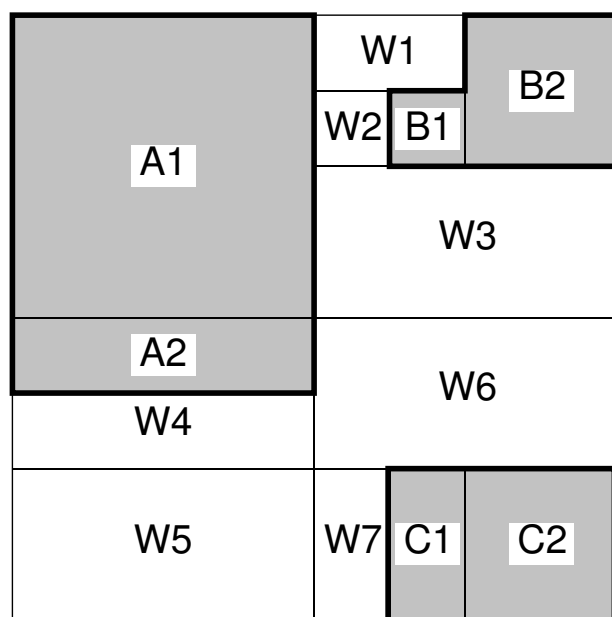
Bintree

- Regular decomposition k-d tree
- Cycle through attributes



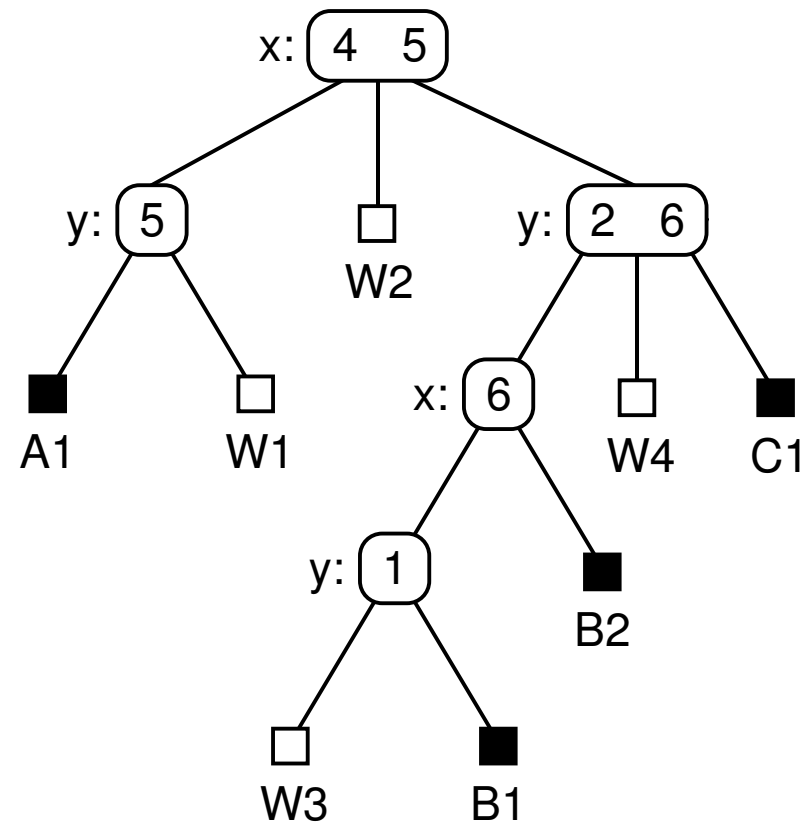
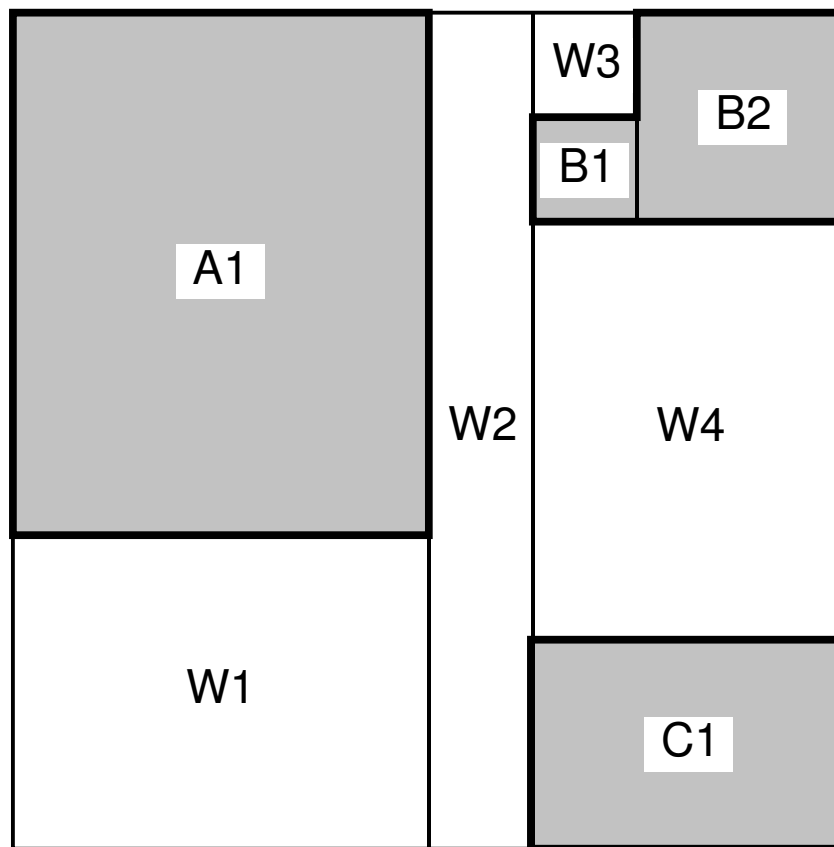
Generalized Bintree

- Regular decomposition k-d tree but no need to cycle through attributes
- Need to record identity of partition axis at each nonleaf node



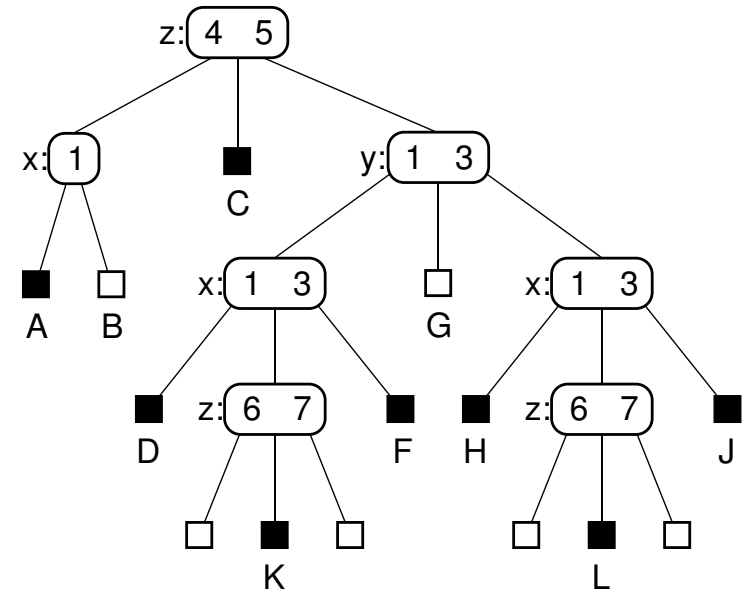
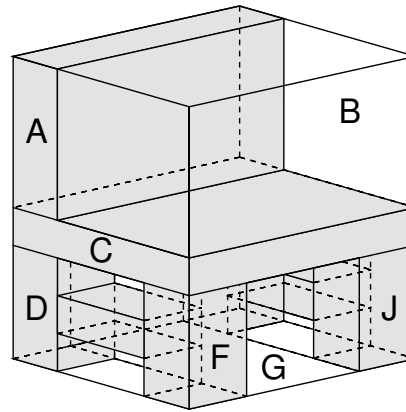
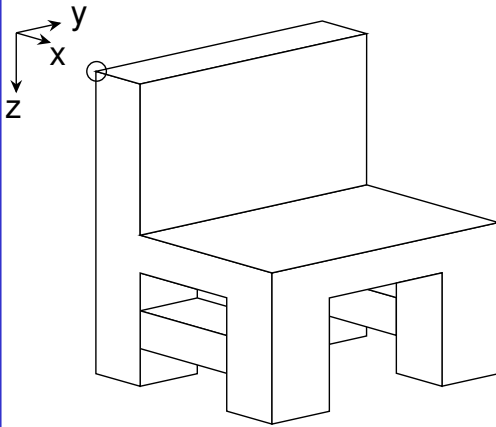
X-Y Tree, Treemap, and Puzzletree

- Split into two or more parts at each partition step
- Implies no two successive partitions along the same attribute as they are combined
- Implies cycle through attributes in two dimensions



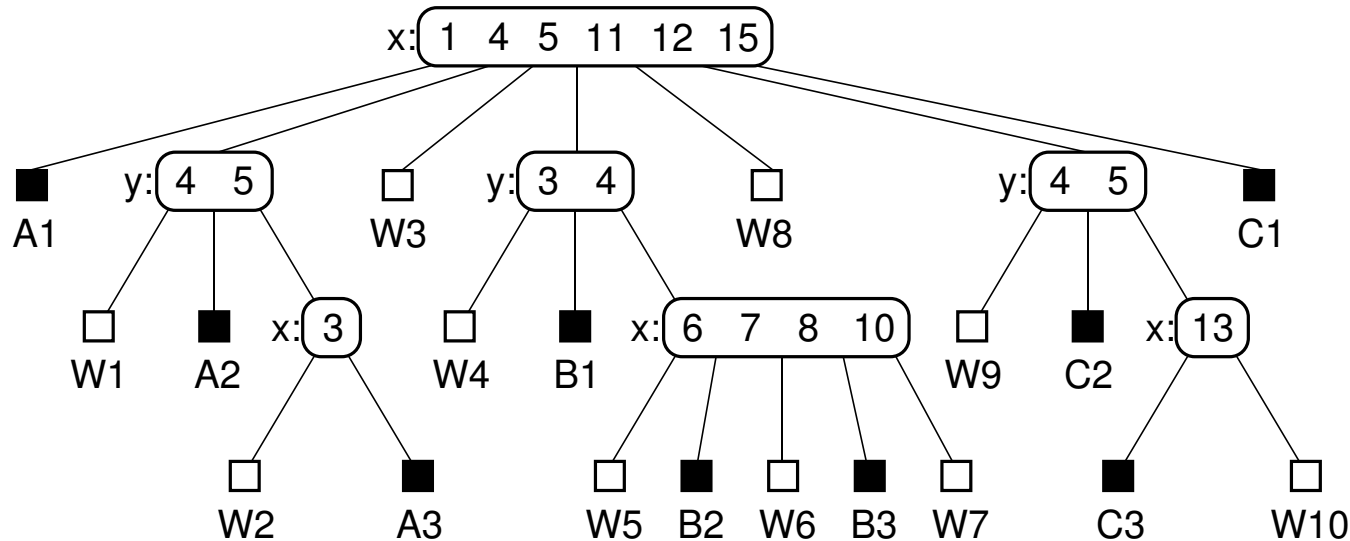
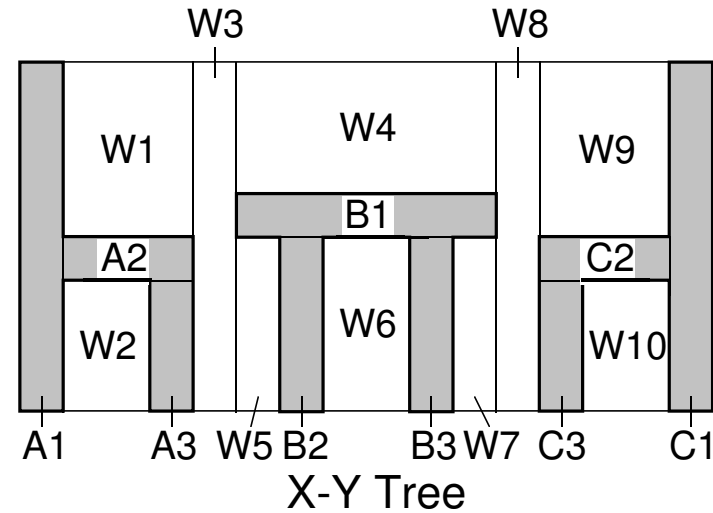
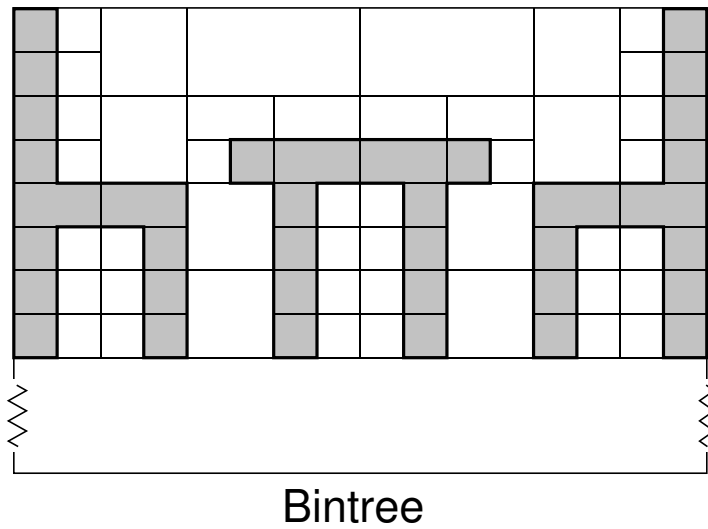
Three-Dimensional X-Y Tree, Treemap, and Puzzletree

- No longer require cycling through dimensions as this results in losing some perceptually appealing block combinations



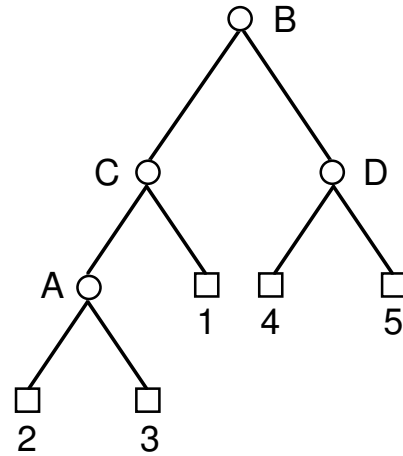
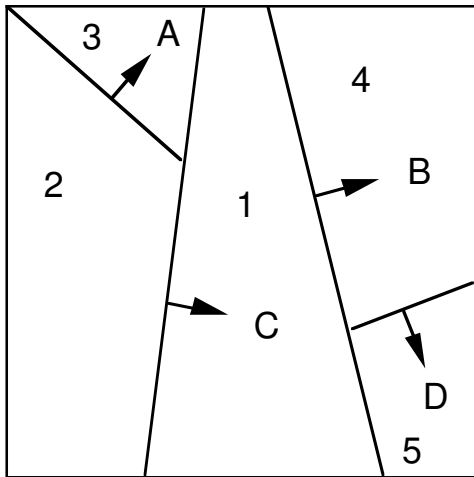
Bintree compared with X-Y Tree, Treemap, Puzzletree

- Much more decomposition in bintree



BSP TREES (Fuchs, Kedem, Naylor)

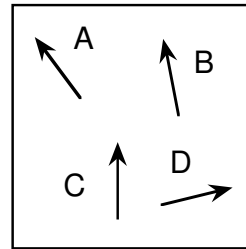
- Like a bintree except that the decomposition lines are at arbitrary orientations (i.e., they need not be parallel or orthogonal)
- For data of arbitrary dimensions
- In 2D (3D), partition along the edges (faces) of a polygon (polyhedron)
- Ex: arrows indicate direction of positive area



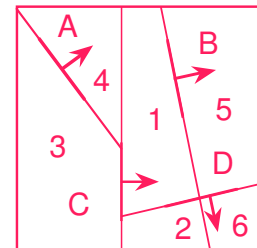
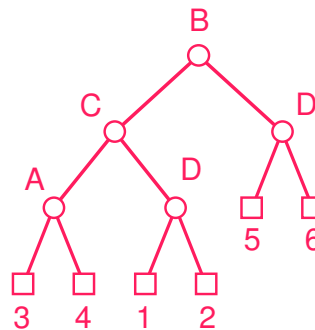
- Usually used for hidden-surface elimination
 1. domain is a set of polygons in three dimensions
 2. position of viewpoint determines the order in which the BSP tree is traversed
- A polygon's plane is extended infinitely to partition the entire space

DRAWBACKS OF BSP TREES

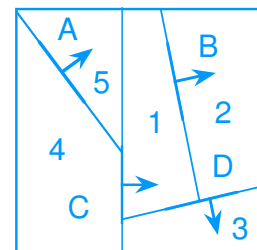
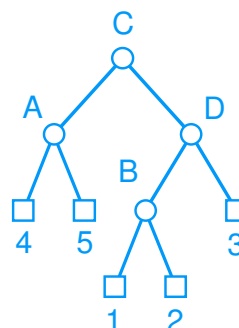
- A polygon may be included in both the left and right subtrees of node
- Same issues of duplicate reporting as in representations based on a disjoint decomposition of the underlying space
- Shape of the BSP tree depends on the order in which the polygons are processed and on the polygons chosen to serve as the partitioning plane
- Not based on a regular decomposition thereby complicating the performance of set-theoretic operations
- Ex: use line segments in two dimensions



1. partition induced by choosing B as the root



2. partition induced by choosing C as the root

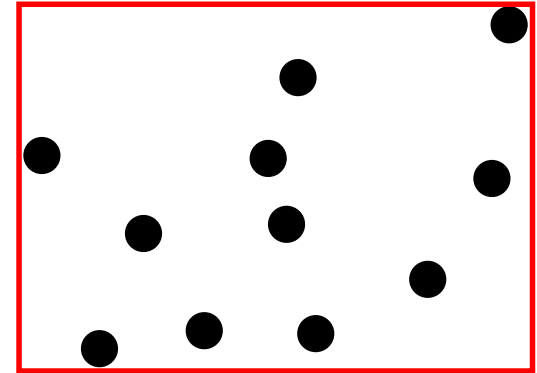


Outline

1. Introduction
2. Points
3. Lines
4. Regions, Volumes, and Surfaces
5. Bounding Box Hierarchies
6. Rectangles
7. Surfaces and Volumes
8. Metric Data
9. Operations
10. Example system

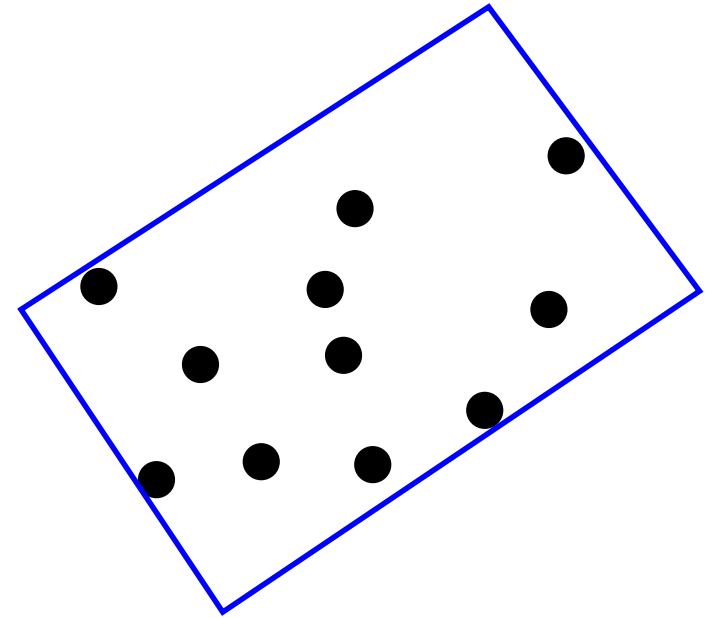
Bounding Box Hierarchies

1. Axis-aligned bounding boxes (AABB)
2. Oriented bounding boxes (OBB)
 - Arbitrary orientation for bounding hyperrectangles
3. Minimum bounding hyperspheres (sphere tree, SS-tree)
4. Combination of hyperspheres and hyperrectangles (SR-tree)
5. 3-dimensional pie slices (BOXTREE)
6. Truncated tetrahedra (prism tree)



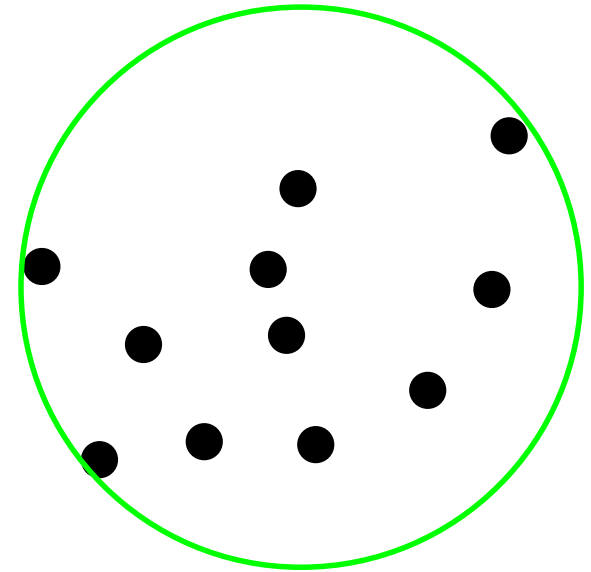
Bounding Box Hierarchies

1. Axis-aligned bounding boxes (AABB)
2. Oriented bounding boxes (OBB)
 - Arbitrary orientation for bounding hyperrectangles
3. Minimum bounding hyperspheres (sphere tree, SS-tree)
4. Combination of hyperspheres and hyperrectangles (SR-tree)
5. 3-dimensional pie slices (BOXTREE)
6. Truncated tetrahedra (prism tree)



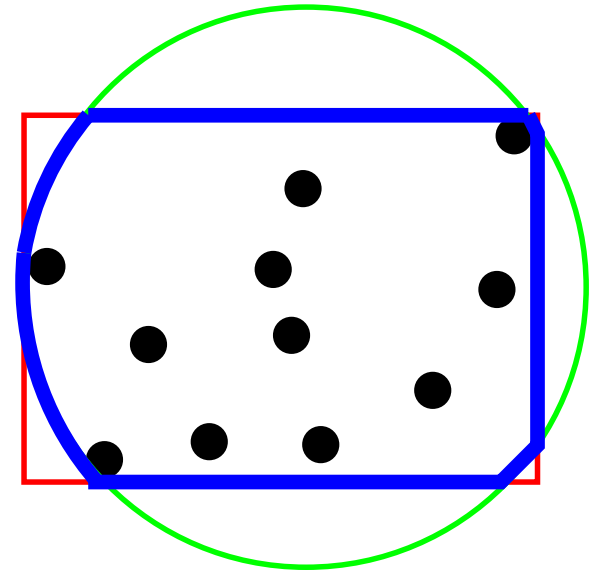
Bounding Box Hierarchies

1. Axis-aligned bounding boxes (AABB)
2. Oriented bounding boxes (OBB)
 - Arbitrary orientation for bounding hyperrectangles
3. Minimum bounding hyperspheres (sphere tree, SS-tree)
4. Combination of hyperspheres and hyperrectangles (SR-tree)
5. 3-dimensional pie slices (BOXTREE)
6. Truncated tetrahedra (prism tree)



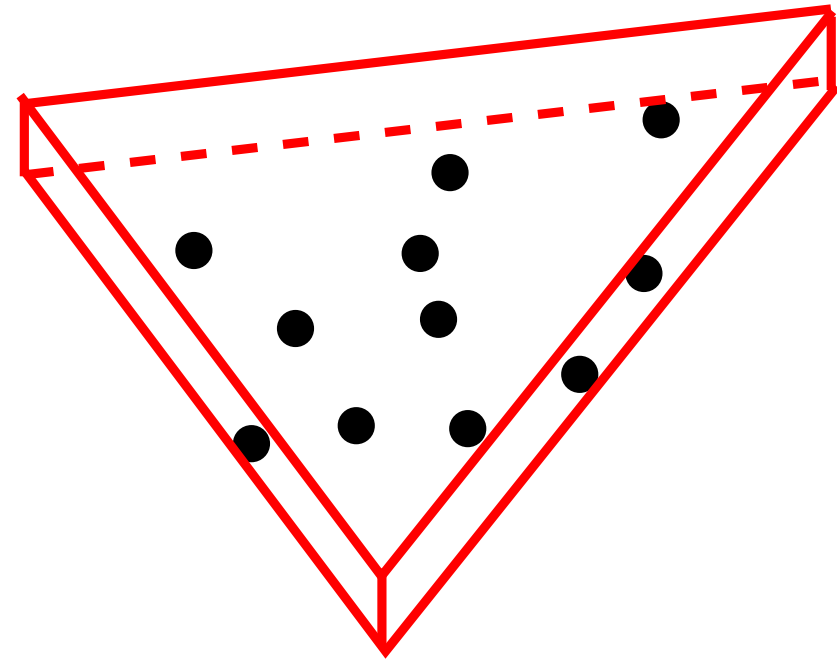
Bounding Box Hierarchies

1. Axis-aligned bounding boxes (AABB)
2. Oriented bounding boxes (OBB)
 - Arbitrary orientation for bounding hyperrectangles
3. Minimum bounding hyperspheres (sphere tree, SS-tree)
4. Combination of hyperspheres and hyperrectangles (SR-tree)
5. 3-dimensional pie slices (BOXTREE)
6. Truncated tetrahedra (prism tree)



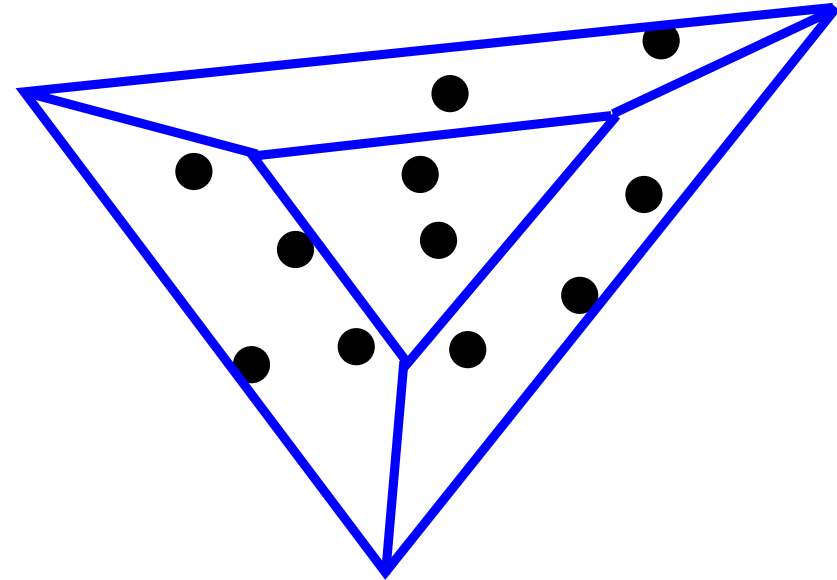
Bounding Box Hierarchies

1. Axis-aligned bounding boxes (AABB)
2. Oriented bounding boxes (OBB)
 - Arbitrary orientation for bounding hyperrectangles
3. Minimum bounding hyperspheres (sphere tree, SS-tree)
4. Combination of hyperspheres and hyperrectangles (SR-tree)
5. 3-dimensional pie slices (BOXTREE)
6. Truncated tetrahedra (prism tree)



Bounding Box Hierarchies

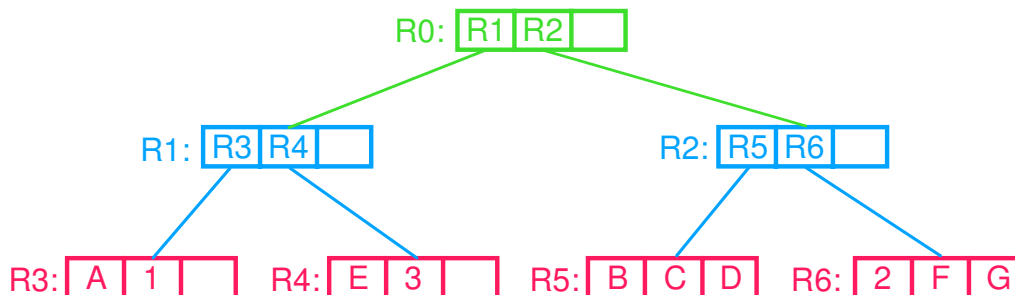
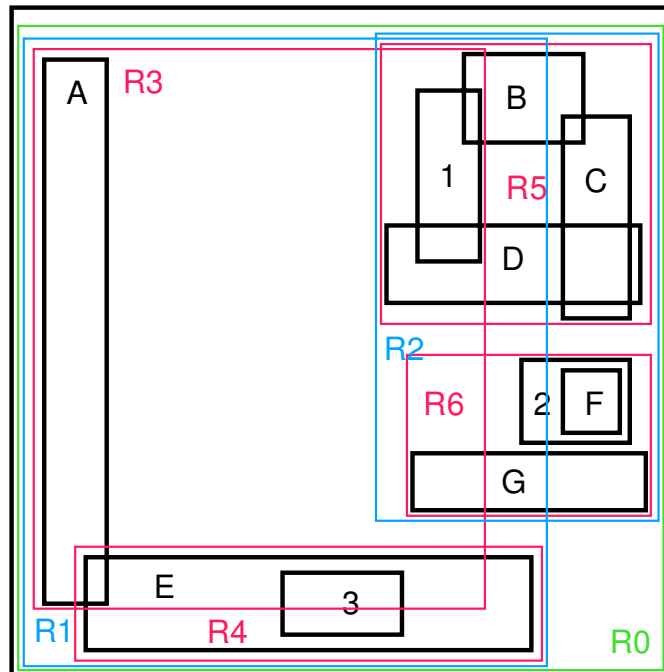
1. Axis-aligned bounding boxes (AABB)
2. Oriented bounding boxes (OBB)
 - Arbitrary orientation for bounding hyperrectangles
3. Minimum bounding hyperspheres (sphere tree, SS-tree)
4. Combination of hyperspheres and hyperrectangles (SR-tree)
5. 3-dimensional pie slices (BOXTREE)
6. Truncated tetrahedra (prism tree)



MINIMUM BOUNDING RECTANGLES

4 3 2 1 rc13
g z r b

- Rectangles grouped into hierarchies, stored in another structure such as a B-tree
- Drawback: not a disjoint decomposition of space
- Rectangle has single bounding rectangle, yet area it spans may be included in several bounding rectangles
- May have to visit several rectangles to determine the presence/absence of a rectangle
- Order (m, M) R-tree
 - between $m \leq \lceil M/2 \rceil$ and M entries in each node except root
 - at least 2 entries in root unless a leaf node
- Ex: order $(2,3)$ R-tree

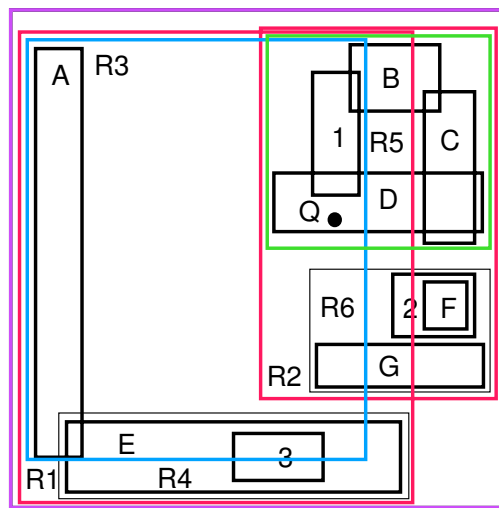
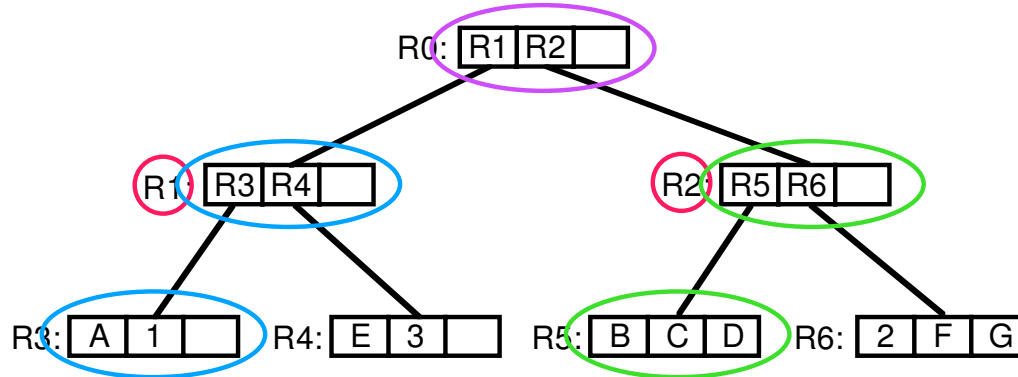


SEARCHING FOR A RECTANGLE CONTAINING A POINT IN AN R-TREE

5 4 3 2 1 rc15
g z r v b

- Drawback is that may have to examine many nodes since a rectangle can be contained in the covering rectangles of many nodes yet its record is contained in only one leaf node (e.g., D in R0, R1, R2, R3, and R5)

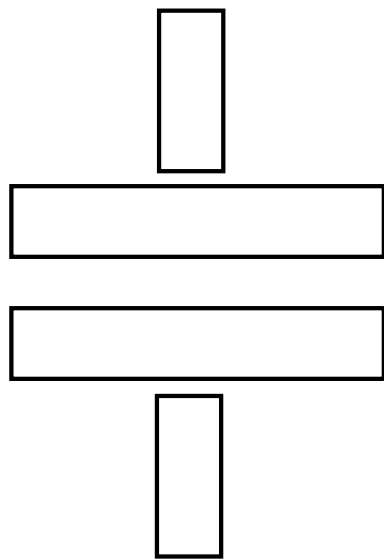
Ex: Search for the rectangle containing point Q



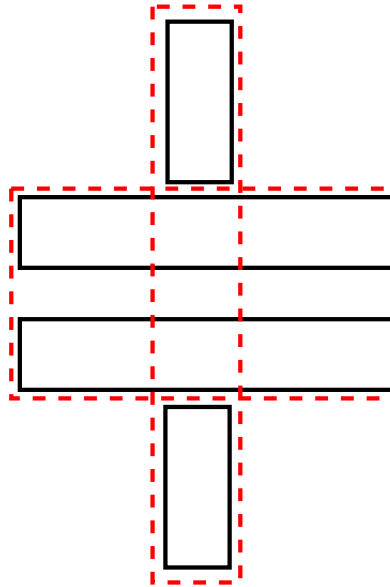
- Q is in R0
- Q can be in both R1 and R2
- Searching R1 first means that R3 is searched but this leads to failure even though Q is in a part of D which is in R3
- Searching R2 finds that Q can only be in R5

Dynamic R-Tree Construction

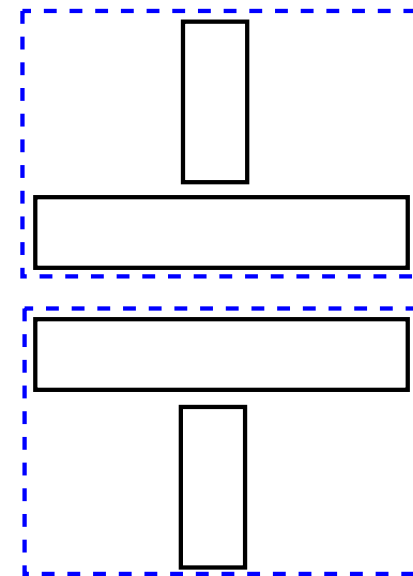
- Differ by how to split overflowing node p upon insertion
- Conflicting goals
 1. Reduce likelihood that each node q is visited by the search
 - achieve by minimizing total area spanned by bounding box of q (coverage)
 2. minimize number of children of p that must be visited by search operations
 - achieve by minimizing area common to children so that the area that they span is not visited a multiple number of times (overlap)



Rectangles



Goal 1



Goal 2

EXAMPLE DYNAMIC SPLITTING METHODS

1. Methods based on reducing coverage:
 - exhaustive search
 - quadratic
 - linear
2. R*-tree
 - minimize overlap in leaf nodes
 - Minimize coverage in nonleaf nodes
 - also reduces coverage by minimizing perimeter of bounding boxes of resulting nodes when effect on coverage is the same
 - when node overflows, first see if can avoid problem by reinserting a fraction of the nodes (e.g., 30%)
3. Ang/Tan: linear with focus on reduction of overlap
4. Packed methods that make use of an ordering
 - usually order centroids of bounding boxes of objects and build a B+-tree
 - a. Hilbert packed R-tree: Peano-Hilbert order
 - b. Morton packed R-tree: Morton order
 - node overflow
 - a. goals of minimizing coverage or overlap are not part of the splitting process
 - b. do not make use of spatial extent of bounding boxes in determining how to split a node

R-TREE OVERFLOW NODE SPLITTING POLICIES

- Could use exhaustive search to look at all possible partitions
- Usually two stages:
 1. pick a pair of bounding boxes to serve as seeds for resulting nodes ('seed-picking')
 2. redistribute remaining nodes with goal of minimizing the growth of the total area ('seed-growing')
- Different algorithms of varying time complexity
 1. quadratic:
 - find two boxes j and k that would waste the most area if they were in the same node
 - for each remaining box i , determine the increase in area d_{ij} and d_{ik} of the bounding boxes of j and k resulting from the addition of i and add the box r for which $|d_{rj} - d_{rk}|$ is a maximum to the node with the smallest increase in area
 - rationale: find box with most preference for one of j, k
 2. linear:
 - find two boxes with greatest normalized separation along all of the dimensions
 - add remaining boxes in arbitrary order to box whose area is increased the least by the addition
 3. linear (Ang/Tan)
 - minimizes overlap
 - for each dimension, associate each box with the closest face of the box of the overflowing node
 - pick partition that has most even distribution
 - a. if a tie, minimize overlap
 - b. if a tie, minimize coverage

R*-TREE

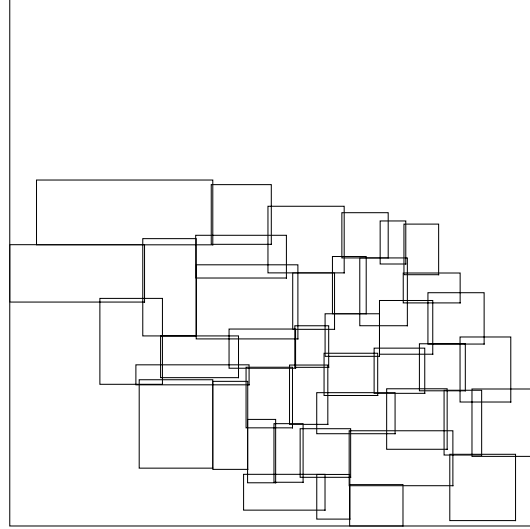
- Tries to minimize overlap in case of leaf nodes and minimize increase in area for nonleaf nodes
- Changes from R-tree:
 1. insert into leaf node p for which the resulting bounding box has minimum increase in overlap with bounding boxes of p 's brothers
 - compare with R-tree where insert into leaf node for which increase in area is a minimum (minimizes coverage)
 2. in case of overflow in p , instead of splitting p as in R-tree, reinsert a fraction of objects in p
 - known as 'forced reinsertion' and similar to 'deferred splitting' or 'rotation' in B-trees
 - how do we pick objects to be reinserted? possibly sort by distance from center of p and reinsert furthest ones
 3. in case of true overflow, use a two-stage process
 - determine the axis along which the split takes place
 - a. sort bounding boxes for each axis to get d lists
 - b. choose the axis having the split value for which the sum of the perimeters of the bounding boxes of the resulting nodes is the smallest while still satisfying the capacity constraints (reduces coverage)
 - determine the position of the split
 - a. position where overlap between two nodes is minimized
 - b. resolve ties by minimizing total area of bounding boxes (reduces coverage)
- Works very well but takes time due to reinsertion

EXAMPLE OF R-TREE NODE SPLITTING POLICIES

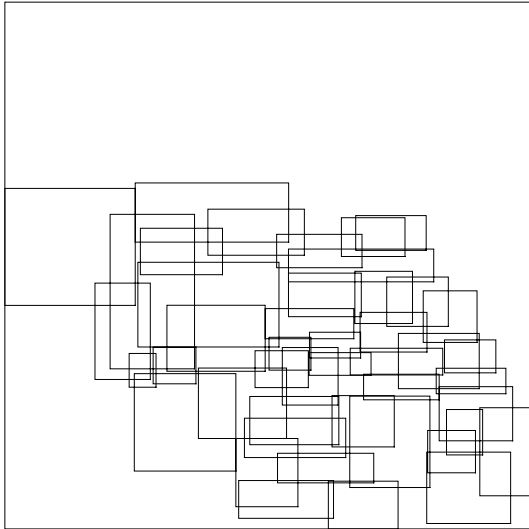
- Sample collection of 1700 lines using $m=20$ and $M=50$



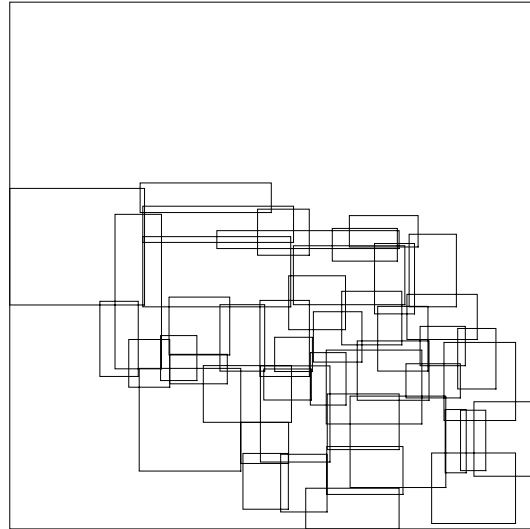
Collection of lines



R*-tree



Linear



Quadratic

Outline

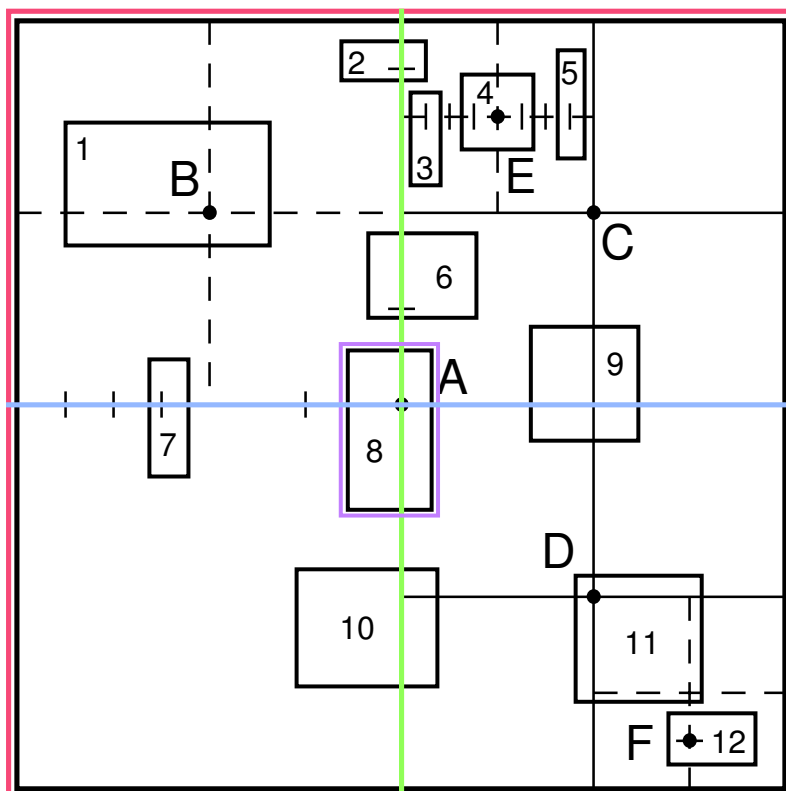
1. Introduction
2. Points
3. Lines
4. Regions, Volumes, and Surfaces
5. Bounding Box Hierarchies
6. Rectangles
7. Surfaces and Volumes
8. Metric Data
9. Operations
10. Example system

MX-CIF QUADTREE (Kedem)

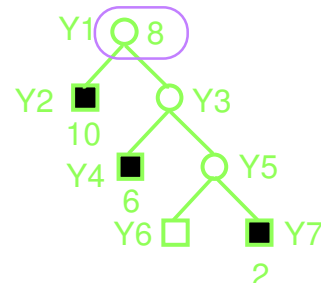
5 4 3 2 1
z v g r b

hp14

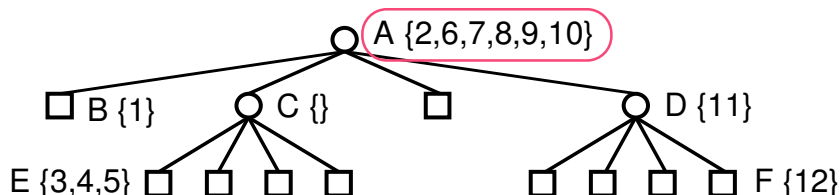
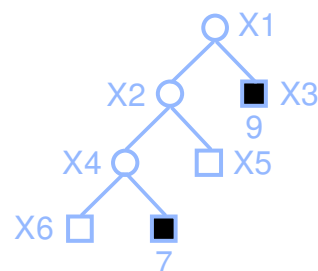
1. Collections of small rectangles for VLSI applications
2. Each rectangle is associated with its minimum enclosing quadtree block
3. Like hashing: quadtree blocks serve as hash buckets
4. Collision = more than one rectangle in a block
 - resolve by using two one-dimensional MX-CIF trees to store the rectangle intersecting the lines passing through each subdivision point
 - one for y-axis
 - if a rectangle intersects both x and y axes, then associate it with the y axis
 - one for x-axis



Binary tree for y-axis through A



Binary tree for x-axis through A



Loose Quadtree (Octree)/Cover Fieldtree

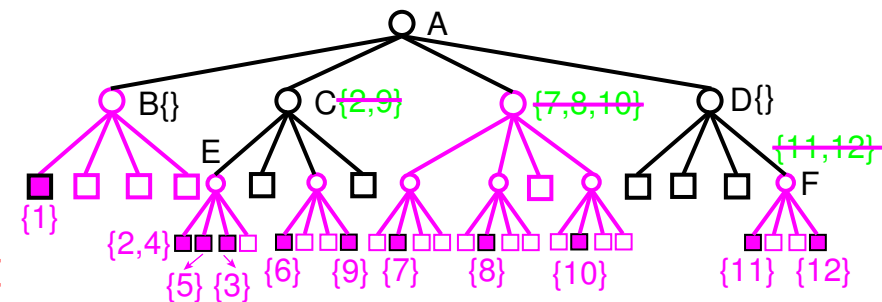
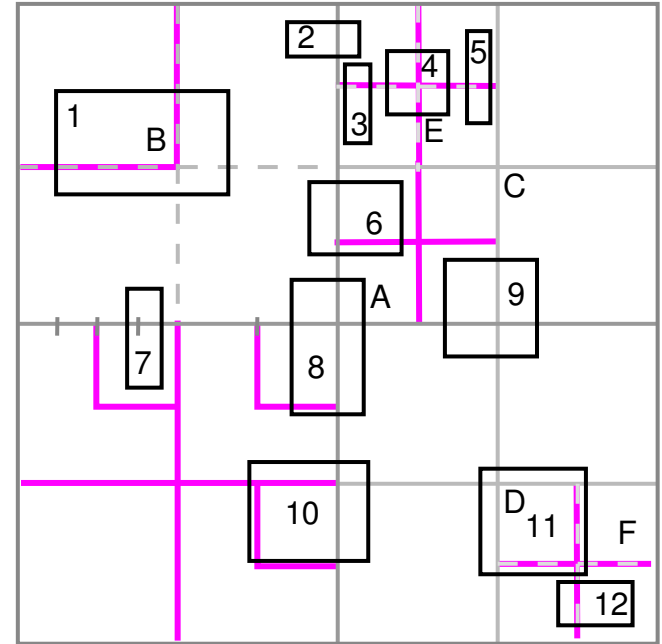
- Overcomes drawback of MX-CIF quadtree that the width w of the minimum enclosing quadtree block of a rectangle o is not a function of the size of o
- Instead, it depends on the position of the centroid of o and often considerably larger than o

- Solution: expand size of space spanned by each quadtree block of width w by expansion factor p ($p > 0$) so expanded block is of width $(1 + p)w$

1. $p = 0.3$
2. $p = 1.0$

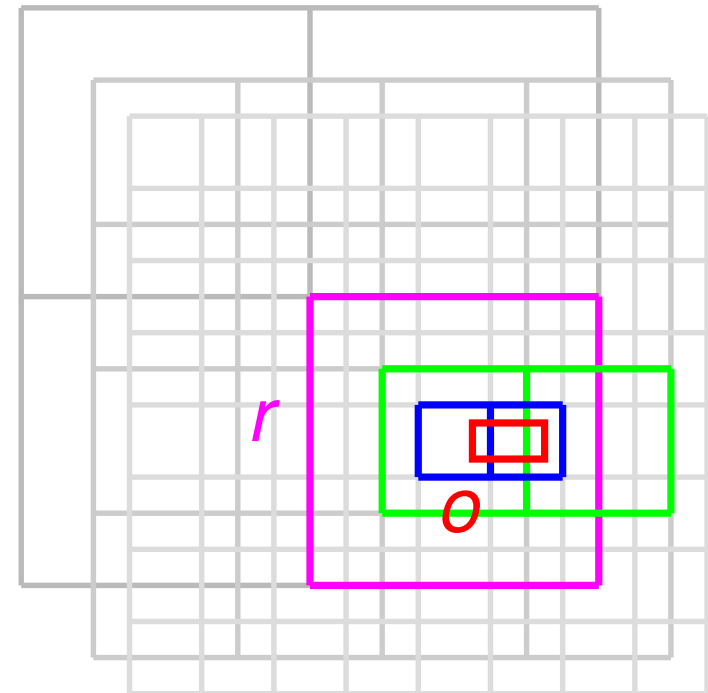
- Maximum w (i.e., minimum depth of minimum enclosing quadtree block) is a function of p and radius r of o and independent of position of centroid of o

1. Range of possible ratios $w/2r$:
 $1/(1 + p) \cdot w/2r < 2/p$
2. For $p \geq 1$, restricting w and r to powers of 2, $w/2r$ takes on at most 2 values and usually just 1



Partition Fieldtree

- Alternative to loose quadtree (octree)/cover fieldtree at overcoming drawback of MX-CIF quadtree that the width w of the minimum enclosing quadtree block of a rectangle o is not a function of the size of o
- Achieves similar result by shifting positions of the centroid of quadtree blocks at successive levels of the subdivision by one half of the width of the block that is being subdivided
- Subdivision rule guarantees that width of minimum enclosing quadtree block for rectangle o is bounded by 8 times the maximum extent r of o
- Same ratio is obtained for the loose quadtree (octree)/cover fieldtree when $p = 1/4$, and thus partition fieldtree is superior to the cover fieldtree when $p < 1/4$
- Summary: cover fieldtree expands the width of the quadtree blocks while the partition fieldtree shifts the positions of their centroids

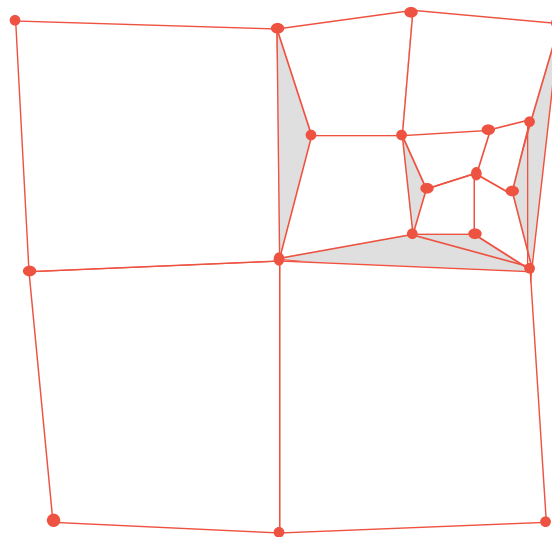


Outline

1. Introduction
2. Points
3. Lines
4. Regions, Volumes, and Surfaces
5. Bounding Box Hierarchies
6. Rectangles
7. Surfaces and Volumes
8. Metric Data
9. Operations
10. Example system

HIERARCHICAL RECTANGULAR DECOMPOSITION

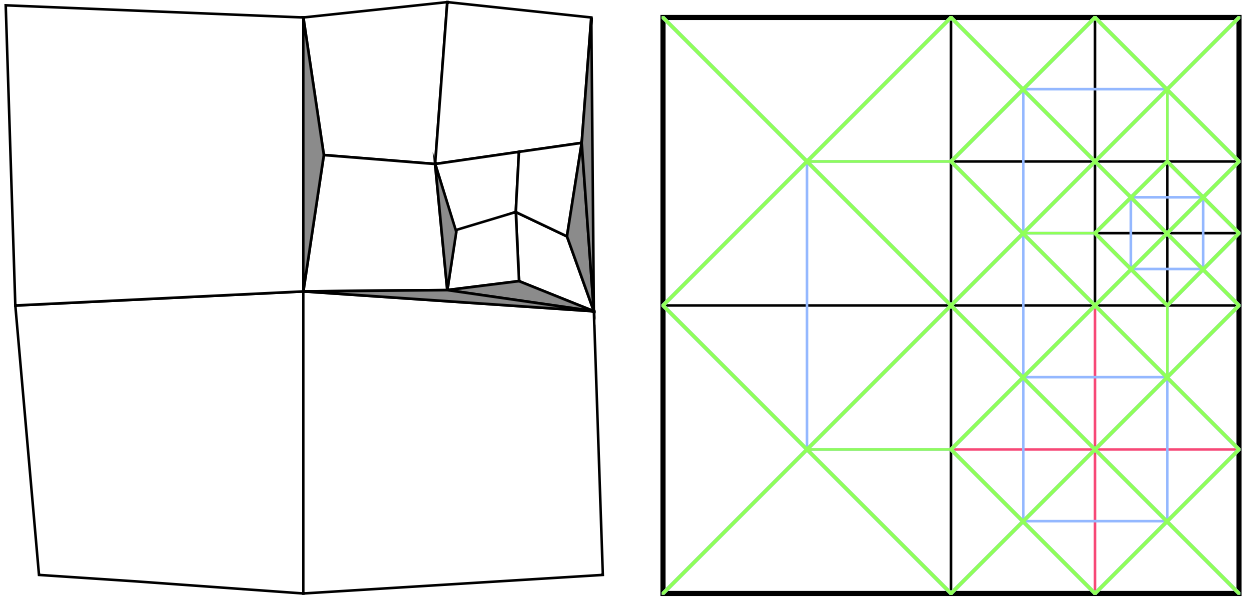
- Similar to triangular decomposition
- Good when data points are the vertices of a rectangular grid
- Drawback is absence of continuity between adjacent patches of unequal width (termed the *alignment problem*)



- Overcoming the presence of cracks
 1. use the interpolated point instead of the true point (Barrera and Hinjosa)
 2. triangulate the squares (Von Herzen and Barr)
 - can split into 2, 4, or 8 triangles depending on how many lines are drawn through the midpoint
 - if split into 2 triangles, then cracks still remain
 - no cracks if split into 4 or 8 triangles

RESTRICTED QUADTREE (VON HERZEN/BARR)

- All 4-adjacent blocks are either of equal size or of ratio 2:1
 Note: also used in finite element analysis to adaptively refine an element as well as to achieve element compatibility (termed *h-refinement* by Kela, Perucchio, and Voelcker)



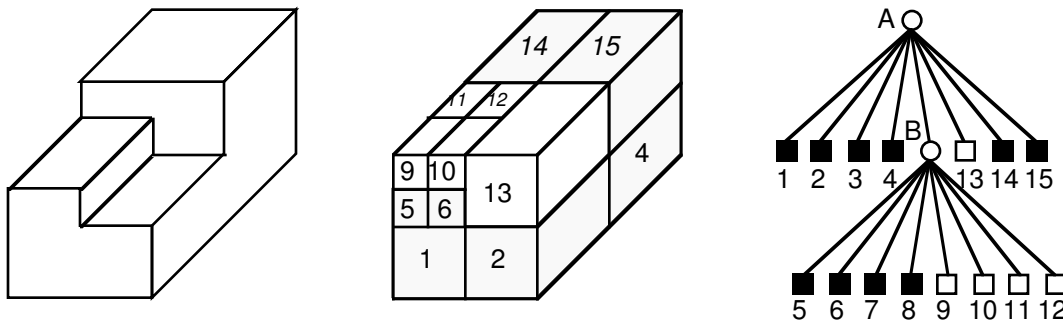
- 8-triangle decomposition rule
 1. decompose each block into 8 triangles (i.e., 2 triangles per edge)
 2. unless the edge is shared by a larger block
 3. in which case only 1 triangle is formed
- 4-triangle decomposition rule
 1. decompose each block into 4 triangles (i.e., 1 triangle per edge)
 2. unless the edge is shared by a smaller block
 3. in which case 2 triangles are formed along the edge
- Prefer 8-triangle rule as it is better for display applications (shading)

OCTREES

1. Interior (voxels)

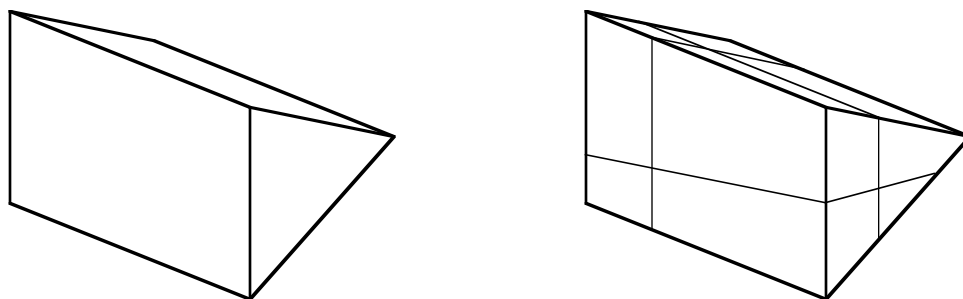
- analogous to region quadtree
- approximate object by aggregating similar voxels
- good for medical images but not for objects with planar faces

Ex:



2. Boundary (PM octrees)

- adaptation of PM quadtree to three-dimensional data
- decompose until each block contains
 - a. one face
 - b. more than one face but all meet at same edge
 - c. more than one edge but all meet at same vertex
- impose spatial index on a boundary model (BRep)



Outline

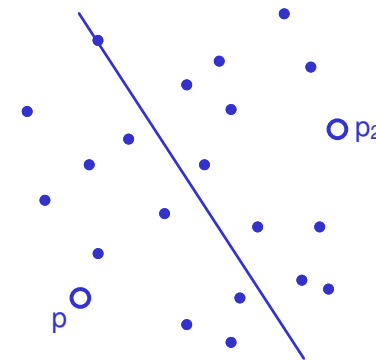
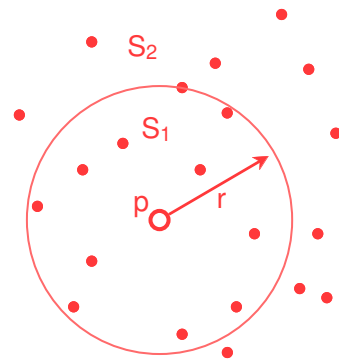
1. Introduction
2. Points
3. Lines
4. Regions, Volumes, and Surfaces
5. Bounding Box Hierarchies
6. Rectangles
7. Surfaces and Volumes
8. Metric Data
9. Operations
10. Example system

Basic Definitions

1. Often only information available is a distance function indicating degree of similarity (or dis-similarity) between all pairs of N data objects
2. Distance metric d : objects must reside in finite metric space (S, d) where for o_1, o_2, o_3 in S , d must satisfy
 - $d(o_1, o_2) = d(o_2, o_1)$ (symmetry)
 - $d(o_1, o_2) \geq 0$, $d(o_1, o_2) = 0$ iff $o_1 = o_2$ (non-negativity)
 - $d(o_1, o_3) \leq d(o_1, o_2) + d(o_2, o_3)$ (triangle inequality)
3. Triangle inequality is a key property for pruning search space
 - Computing distance is expensive
4. Non-negativity property enables ignoring negative values in derivations

Pivots

- Identify a distinguished object or subset of the objects termed *pivots* or *vantage points*
 1. sort remaining objects based on
 - a. distances from the pivots, or
 - b. which pivot is the closest
 2. and build index
 3. use index to achieve pruning of other objects during search
- Given pivot $p \in S$, for all objects $o \in S' \subseteq S$, we know:
 1. exact value of $d(p, o)$,
 2. $d(p, o)$ lies within range $[r_{lo}, r_{hi}]$ of values (ball partitioning) or
 - drawback is asymmetry of partition as outer shell is usually narrow
 3. o is closer to p than to some other object $p_2 \in S$ (generalized hyperplane partitioning)
- Distances from pivots are useful in pruning the search



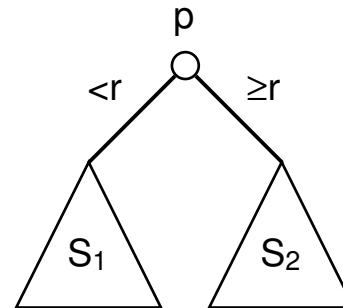
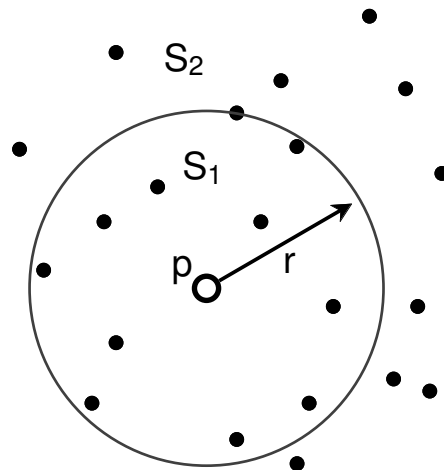
vp-Tree (Metric Tree; Uhlmann|Yianilos)

- Ball partitioning method
- Pick p from S and let r be median of distances of other objects from p
- Partition S into two sets S_1 and S_2 where:

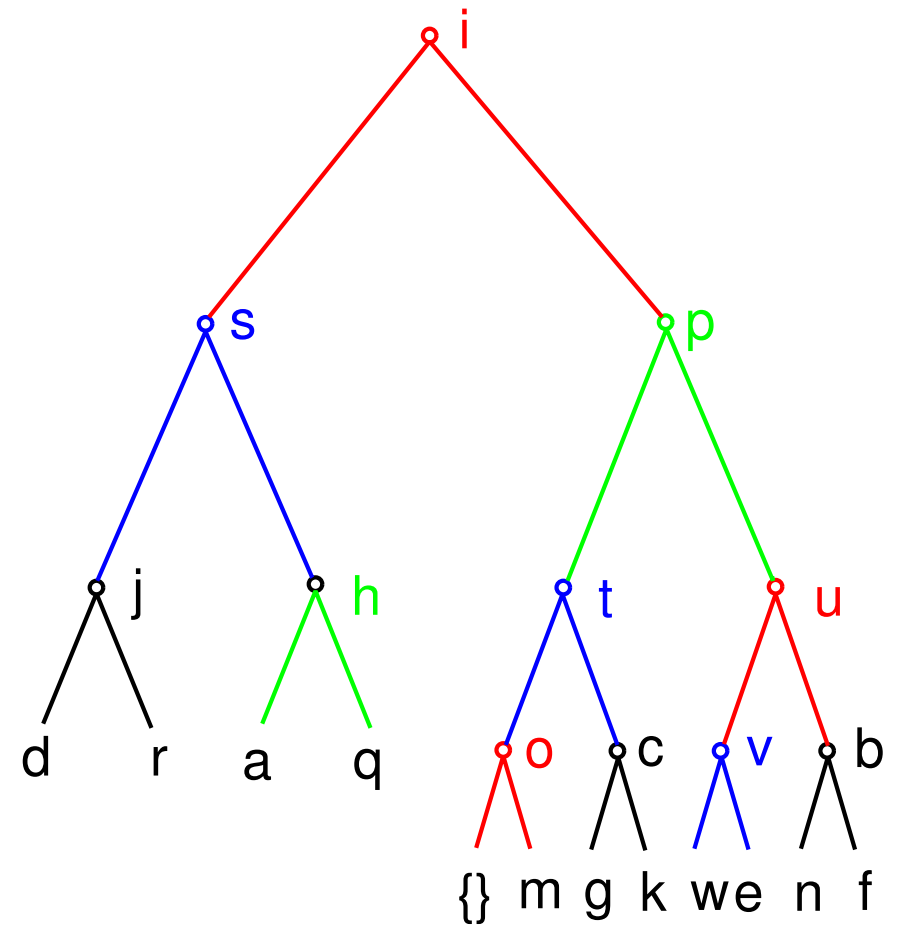
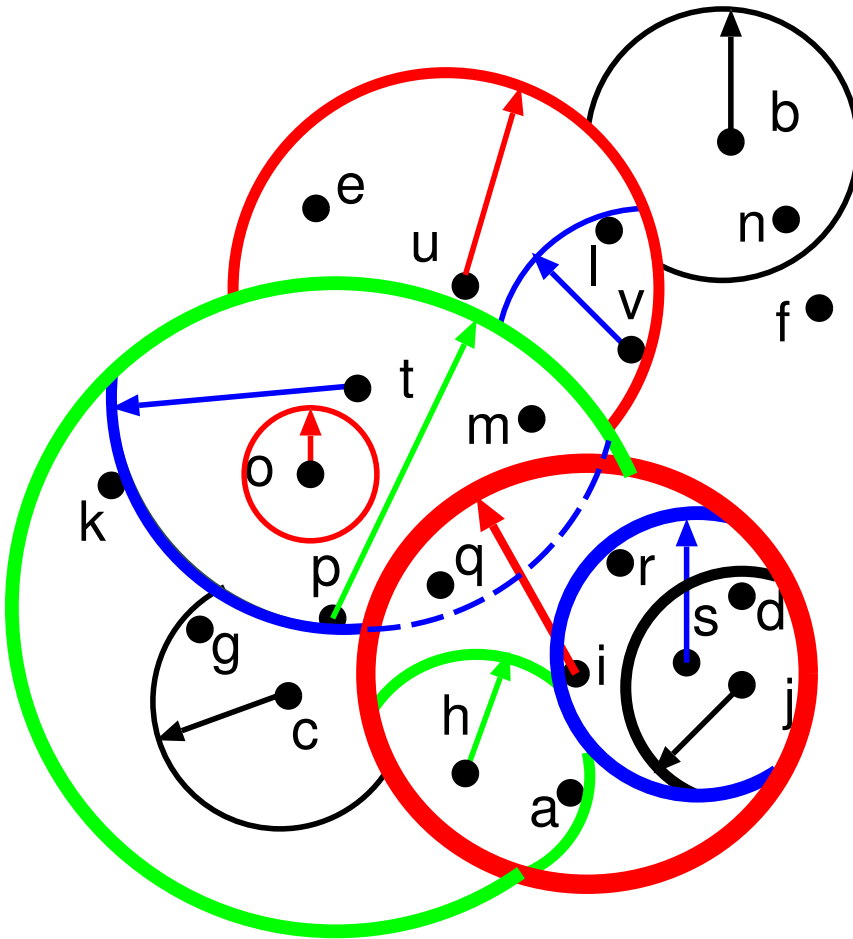
$$S_1 = \{o \in S \setminus \{p\} \mid d(p, o) < r\}$$

$$S_2 = \{o \in S \setminus \{p\} \mid d(p, o) \geq r\}$$

- Apply recursively, yielding a binary tree with pivot and radius values at internal nodes
- Choosing pivots
 1. simplest is to pick at random
 2. choose a random sample and then select median



vp-Tree Example



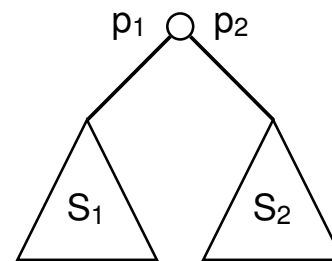
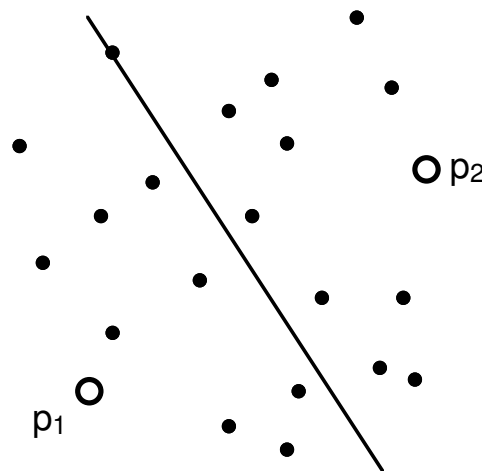
gh-Tree (Metric Tree; Uhlmann)

- Generalized hyperplane partitioning method
- Pick p_1 and p_2 from S and partition S into two sets S_1 and S_2 where:

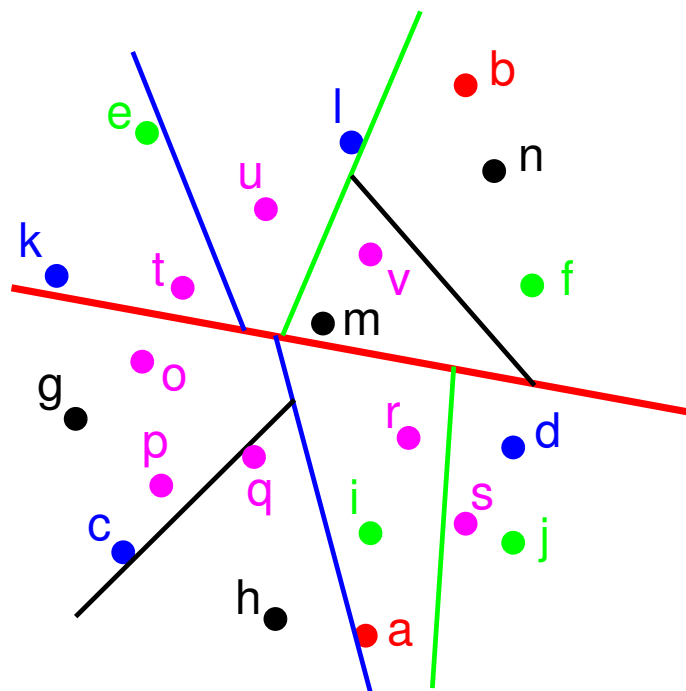
$$S_1 = \{o \in S \setminus \{p_1, p_2\} \mid d(p_1, o) \leq d(p_2, o)\}$$

$$S_2 = \{o \in S \setminus \{p_1, p_2\} \mid d(p_2, o) < d(p_1, o)\}$$

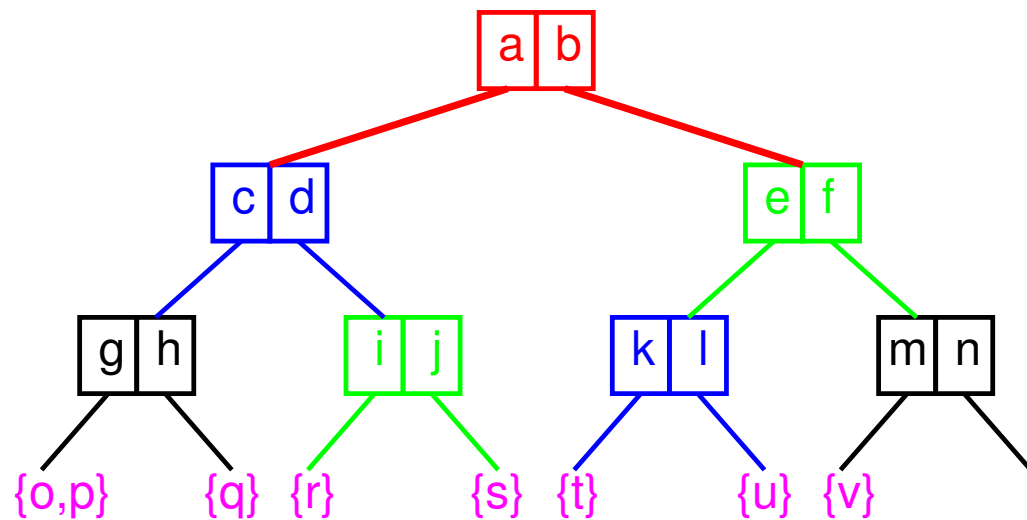
- Objects in S_1 are closer to p_1 than to p_2 (or equidistant from both), and objects in S_2 are closer to p_2 than to p_1
 - hyperplane corresponds to all points o satisfying $d(p_1, o) = d(p_2, o)$
 - can also “move” hyperplane, by using $d(p_1, o) = d(p_2, o) + m$
- Apply recursively, yielding a binary tree with two pivots at internal nodes



gh-Tree Example



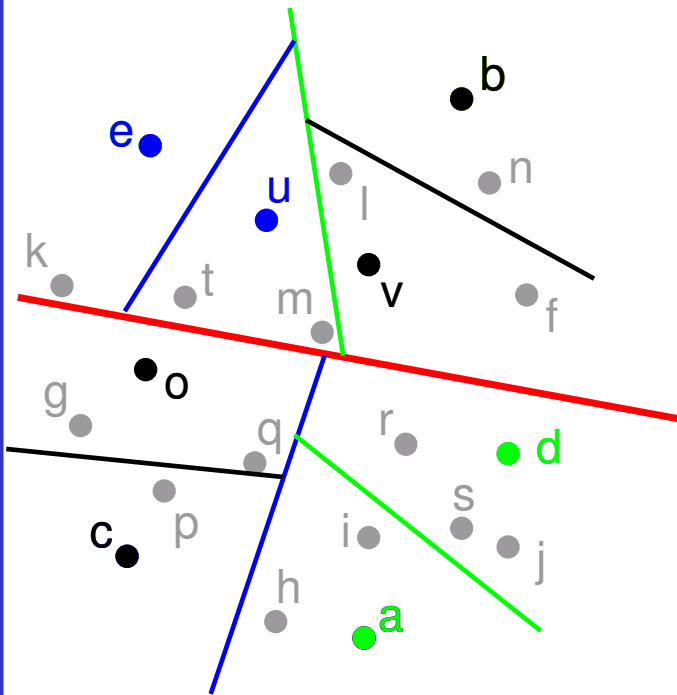
(a)



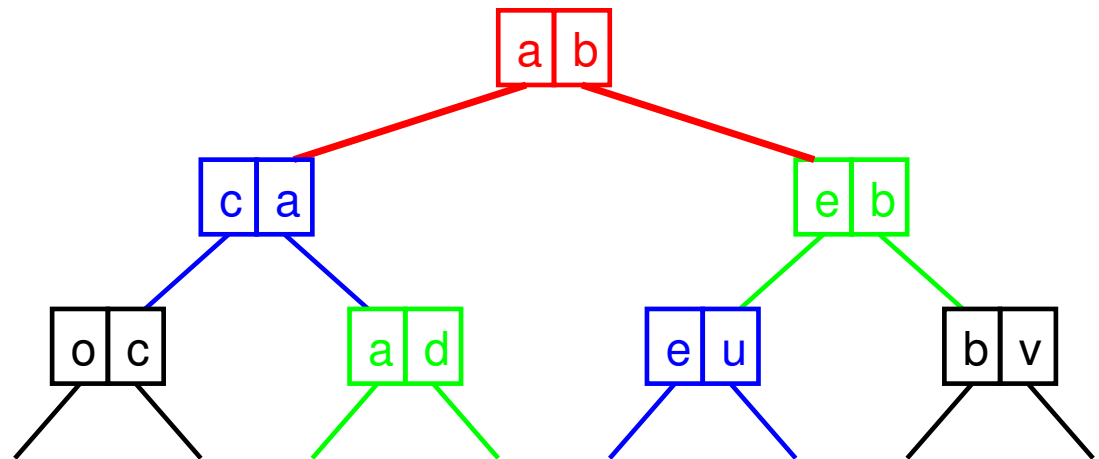
(b)

mb-Tree (Dehne/Noltemeier)

1. Inherit one pivot from ancestor node
2. Fewer pivots and fewer distance computations but perhaps deeper tree
3. Like bucket (k) PR k-d tree as split whenever region has $k > 1$ objects but region partitions are implicit (defined by pivot objects) instead of explicit



(a)

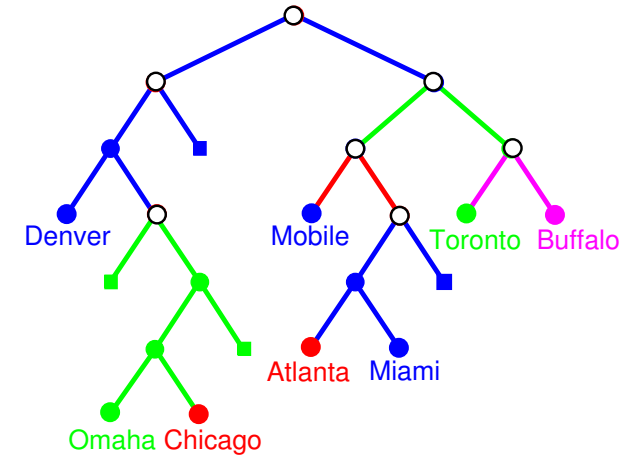
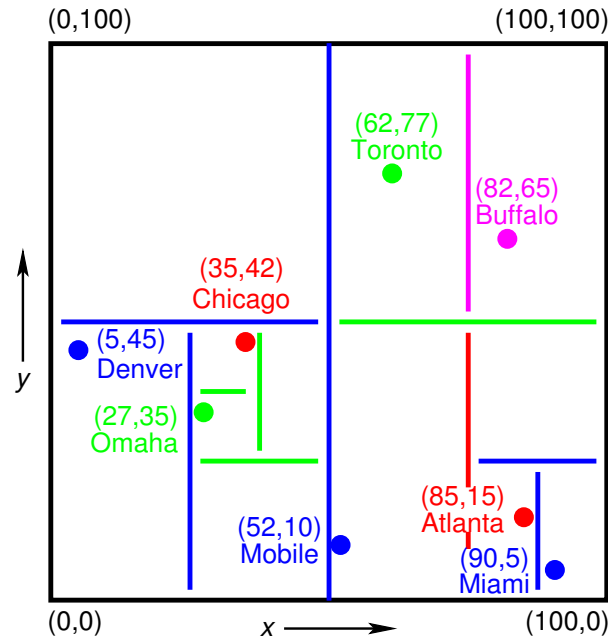


(b)

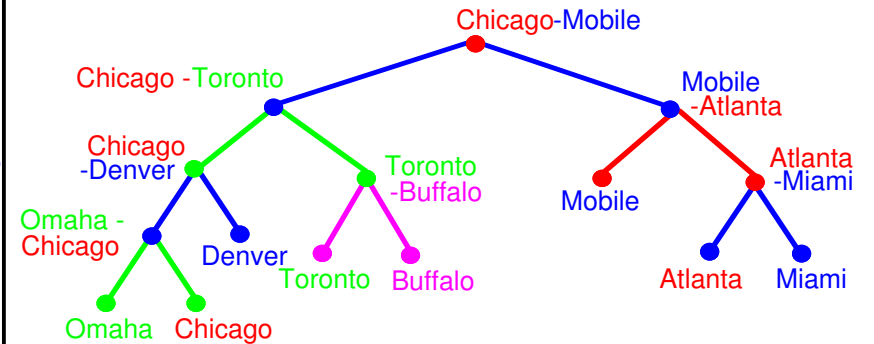
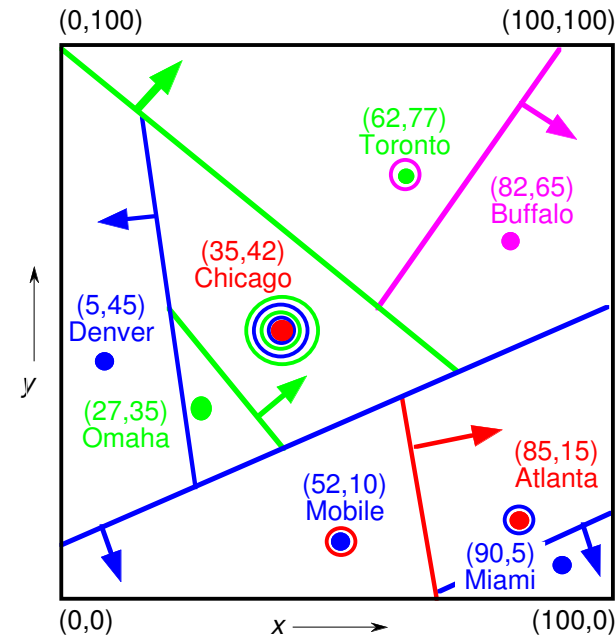
Comparison of mb-tree (BSP tree) and PR k-d tree

PR k-d tree

- Partition of underlying space analogous to that of BSP tree for points



mb-tree



Outline

1. Introduction
2. Points
3. Lines
4. Regions, Volumes, and Surfaces
5. Bounding Box Hierarchies
6. Rectangles
7. Surfaces and Volumes
8. Metric Data
9. Operations
10. Example system

Incremental Nearest Neighbors (Hjaltason/Samet)

■ Motivation

1. often don't know in advance how many neighbors will need
2. e.g., want nearest city to Chicago with population > 1 million

■ Several approaches

1. guess some area range around Chicago and check populations of cities in range
 - if find a city with population > 1 million, must make sure that there are no other cities that are closer with population > 1 million
 - inefficient as have to guess size of area to search
 - problem with guessing is we may choose too small a region or too large a region
 - a. if size too small, area may not contain any cities with right population and need to expand the search region
 - b. if size too large, may be examining many cities needlessly
2. sort all the cities by distance from Chicago
 - impractical as we need to re-sort them each time pose a similar query with respect to another city
 - also sorting is overkill when only need first few neighbors
3. find k closest neighbors and check population condition

Mechanics of Incremental Nearest Neighbor Algorithm

- Make use of a search hierarchy (e.g., tree) where
 1. objects at lowest level
 2. object approximations are at next level (e.g., bounding boxes in an R-tree)
 3. nonleaf nodes in a tree-based index
- Traverse search hierarchy in a “best-first” manner similar to A*-algorithm instead of more traditional depth-first or breadth-first manners
 1. at each step, visit element with smallest distance from query object among all unvisited elements in the search hierarchy
 - i.e., all unvisited elements whose parents have been visited
 2. use a global list of elements, organized by their distance from query object
 - use a priority queue as it supports necessary insert and delete minimum operations
 - ties in distance: priority to lower type numbers
 - if still tied, priority to elements deeper in search hierarchy

Incremental Nearest Neighbor Algorithm

Algorithm:

INCNEAREST(q, S, T)

```

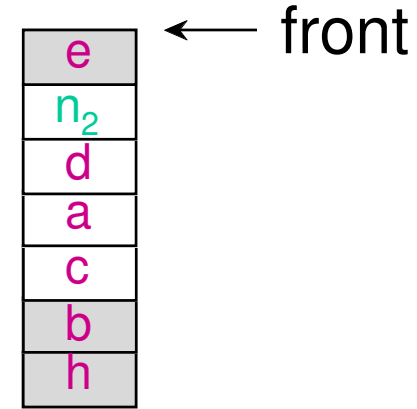
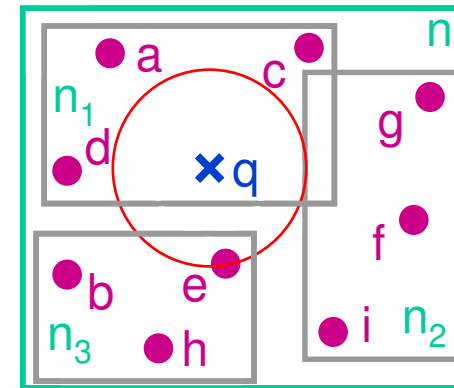
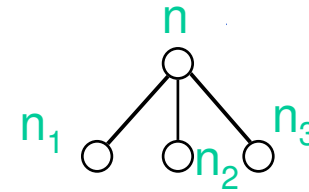
1   $Q \leftarrow \text{NEWPRIORITYQUEUE}()$ 
2   $e_t \leftarrow$  root of the search hierarchy induced by  $q, S$ , and  $T$ 
3  ENQUEUE( $Q, e_t, 0$ )
4  while not ISEMPTY( $Q$ ) do
5     $e_t \leftarrow \text{DEQUEUE}(Q)$ 
6    if  $t = 0$  then /*  $e_t$  is an object */
7      Report  $e_t$  as the next nearest object
8    else
9      for each child element  $e_{t'}$  of  $e_t$  do
10        ENQUEUE( $Q, e_{t'}, d_{t'}(q, e_{t'})$ )

```

1. Lines 1-3 initialize priority queue with root
2. In main loop take element e_t closest to q off the queue
 - report e_t as next nearest object if e_t is an object
 - otherwise, insert child elements of e_t into priority queue

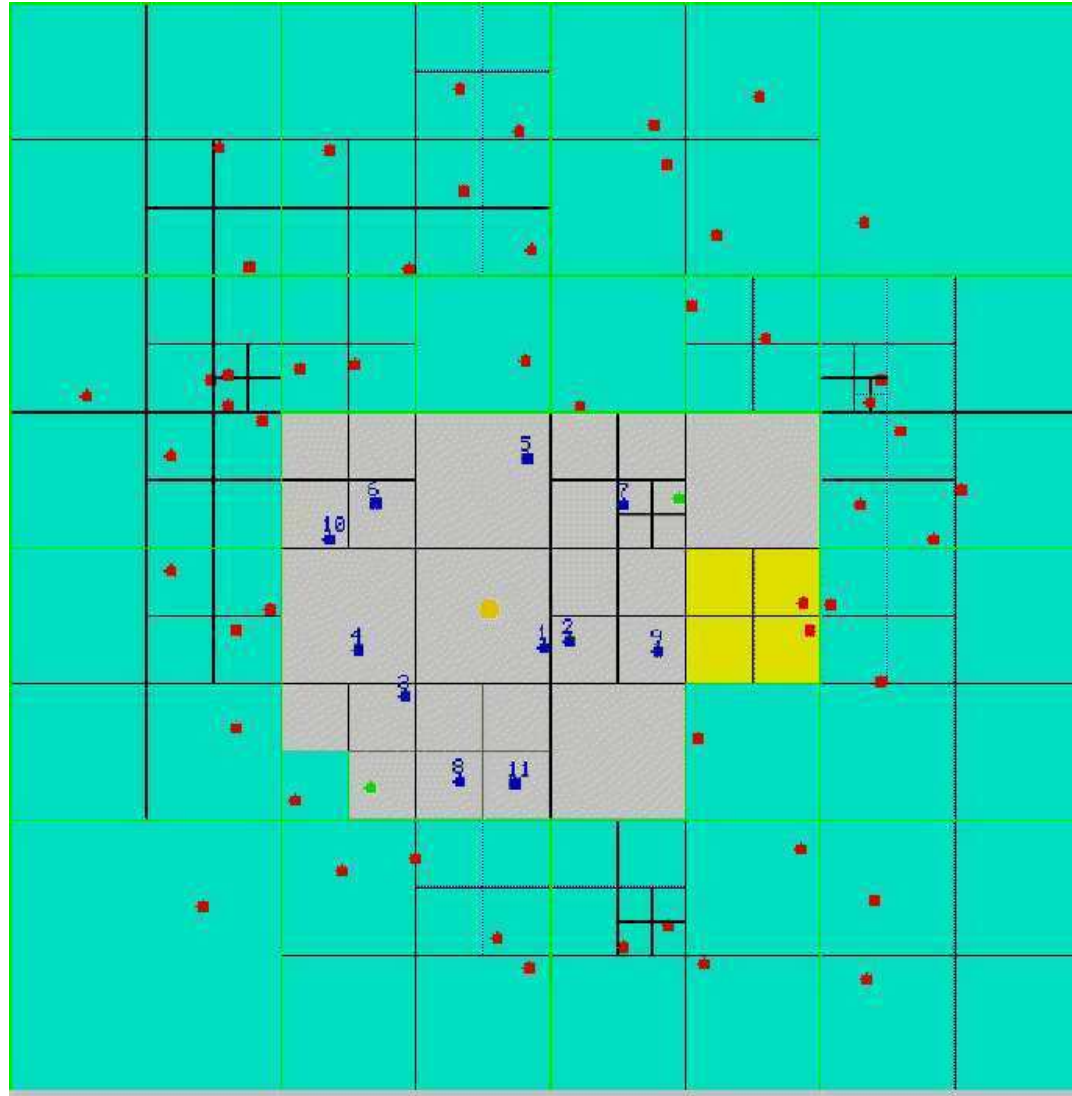
Example of INCNEAREST

- Initially, algorithm descends tree to leaf node containing q
 - expand n
 - expand n_1
- Start growing **search region**
 - expand n_3
 - report e as nearest neighbor



queue

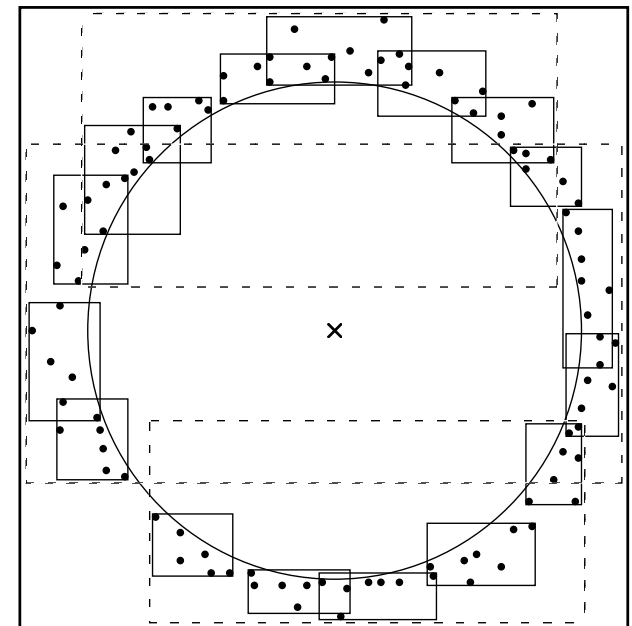
VASCO Spatial Applet



<http://www.cs.umd.edu/~hjs/quadtree/index.html>

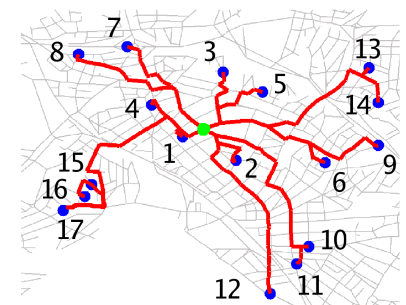
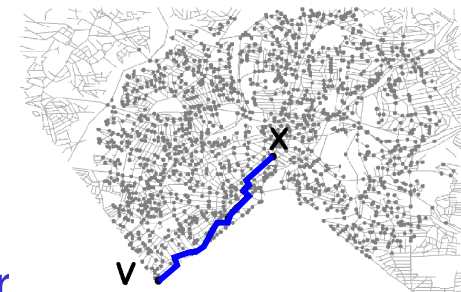
Complexity Analysis

- Algorithm is I/O optimal
 - no nodes outside search region are accessed
 - better pruning than branch and bound algorithm
- Observations for finding k nearest neighbors for uniformly-distributed two-dimensional points
 - expected # of points on priority queue: $O(\sqrt{k})$
 - expected # of leaf nodes intersecting search region: $O(k + \sqrt{k})$
- In worst case, priority queue will be as large as entire data set
 - e.g., when data objects are all nearly equidistant from query object
 - probability of worst case very low, as it depends on a particular configuration of both the data objects and the query object (but: curse of dimensionality!)

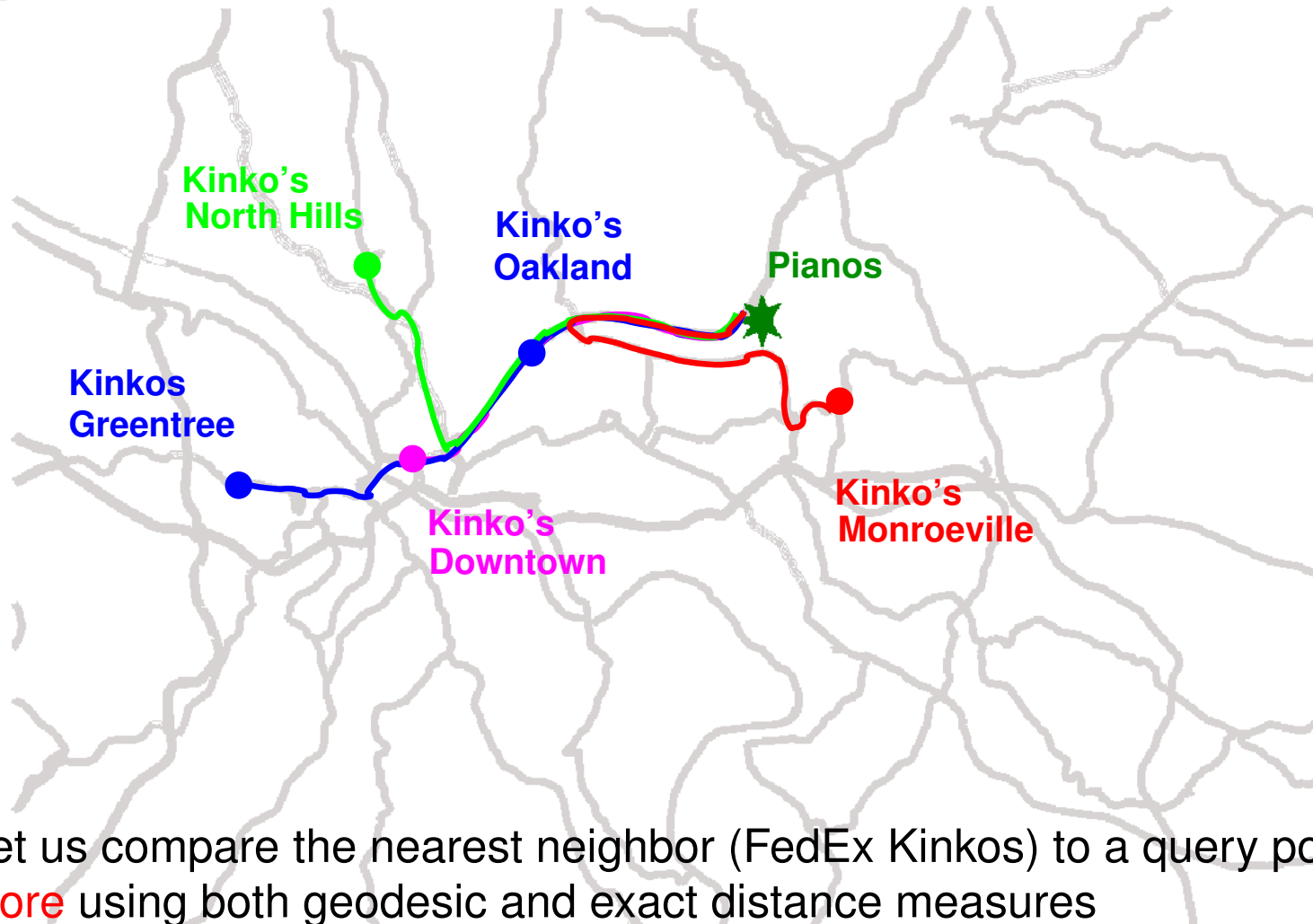


Key to Nearest Neighbor Finding in Spatial Networks

1. Use distance along a graph rather than “as the crow flies”
2. Precompute and store shortest paths between all vertices in network
 - Reduce cost of storing shortest paths between all pairs of N vertices from $O(N^3)$ to $O(N^{1.5})$ using path coherence of destination vertices
 - Can reduce to $O(N)$ by also using path coherence of source vertices
3. Decouple domain S of query objects (q) and objects from which neighbors are drawn from domain V of vertices of network
 - Implies no need to recompute shortest paths each time q or S change
4. Avoids Dijkstra's algorithm which visits too many vertices
 - Ex: Dijkstra's algorithm visits 3191 out of the 4233 vertices in network to identify a 76 edge path from X to V
5. Instead, only visit vertices on shortest paths to nearest neighbors



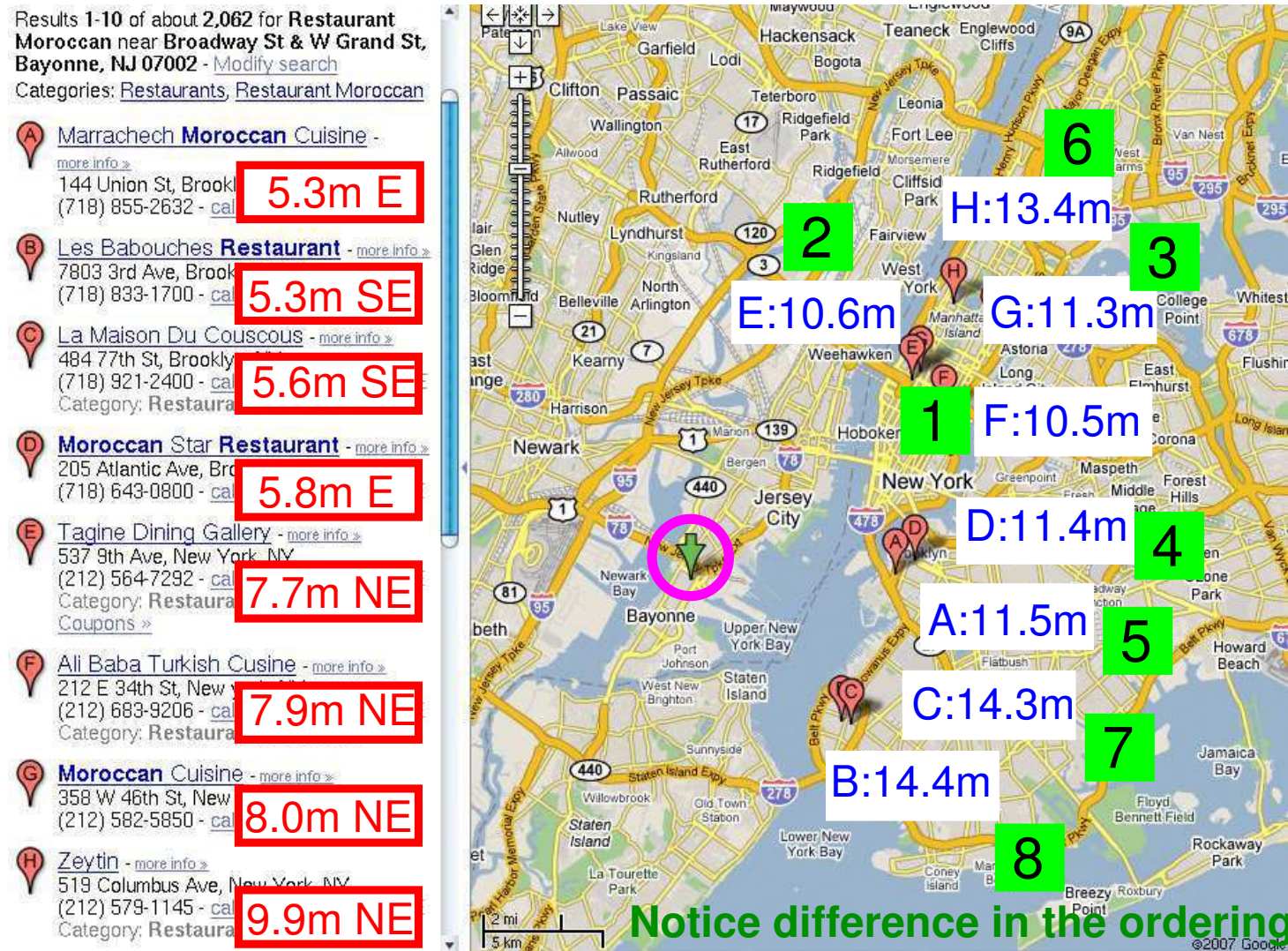
Application – Find the closest Kinko's



- Let us compare the nearest neighbor (FedEx Kinkos) to a query point **Piano store** using both geodesic and exact distance measures
 - geodesic ordering **M O N D G**
 - network distance ordering **O D N M G** (Error: +26 miles)
 - trafficability ordering **O D G N M** (Error: +32 minutes)
- Challenge: Real time + exact queries

Proximity Search on “Google Local”

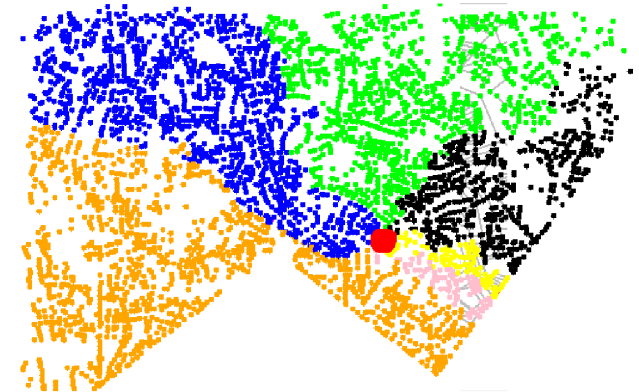
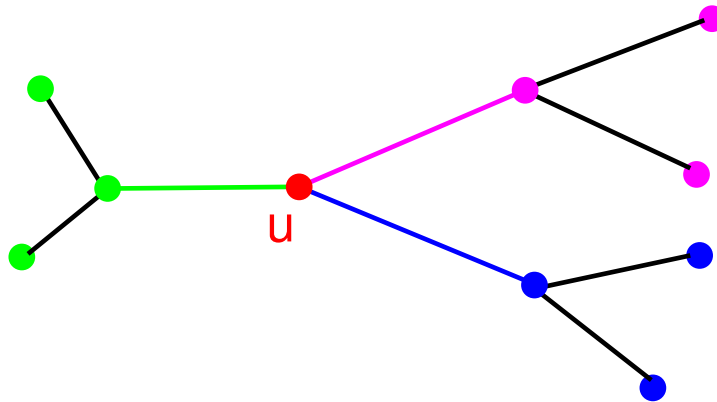
- Let us examine the errors between ordering by the **spatial** distance (“as the crow flies” used by Google) and by the **network** distance (used by us)



- Goal: Instant answers as well as accurate answers

SILC: Using Path Coherence to Encode Shortest Paths

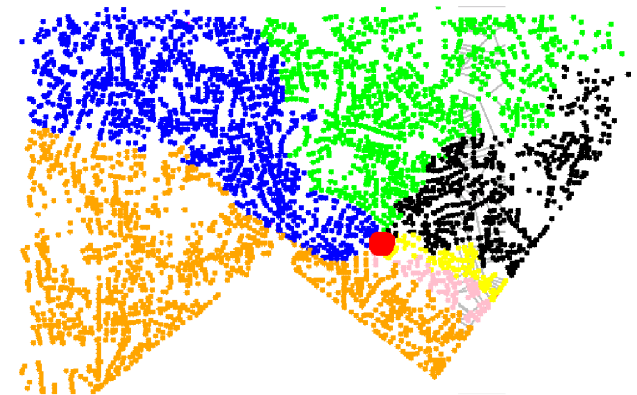
- The SILC path encoding takes advantage of the path coherence
- **How?** Use a *coloring* algorithm



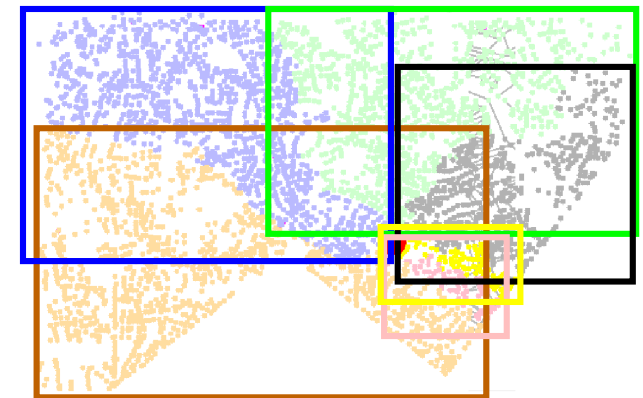
- Source vertex u in a spatial network
- Assign colors to the outgoing edges of u
- Color vertex based on the first edge on the shortest path from u
- Source vertex u in the spatial network of Silver Spring, MD
- Color remaining vertices based on which of the six adjacent vertices of u is the first link in the shortest path from u
- Resulting representation is termed the *shortest-path map* of u
- Assuming planar spatial network graphs means that the coloring results in spatially contiguous colored regions due to path coherence

How to Store Colored Regions?

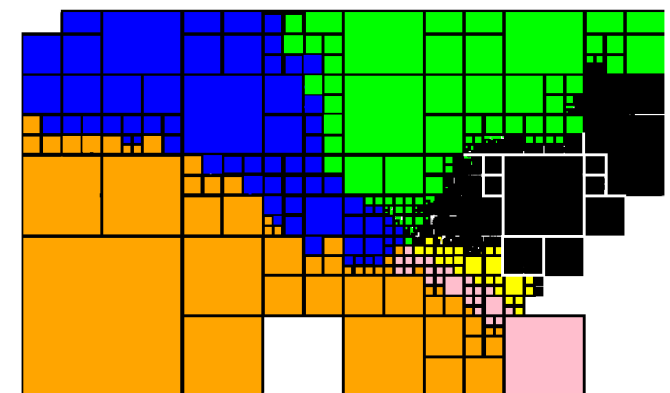
- Minimum bounding boxes (e.g., R-tree) [Wagn03]
 - overlapping boxes imply identity of next vertex cannot be uniquely determined causing the shortest path algorithm to possibly degenerate to Dijkstra's algorithm
- Disjoint decomposition: shortest-path quadtree
 - Decompose until all vertices in block have the same color
- Shortest-path quadtree stored as a collection of Morton blocks
 - Note: no need to store identity of vertices in the blocks
- Proposed encoding leverages the dimensionality reduction property of MX and region quadtrees
 - Required storage cost to represent a region R in a region and MX quadtree is $O(p)$, where p is the perimeter of R



Shortest-path Map



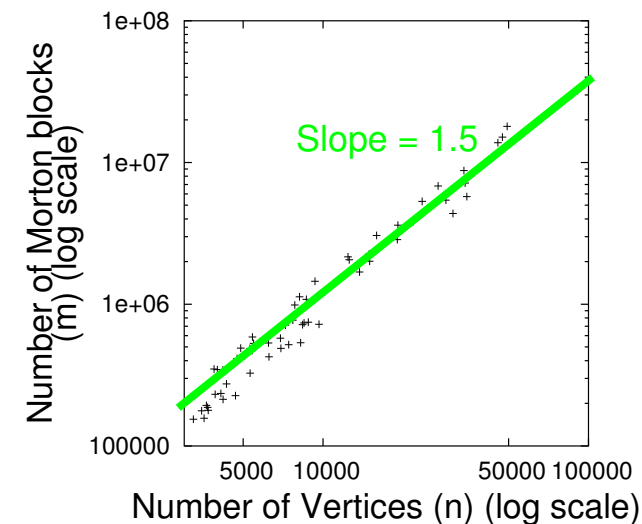
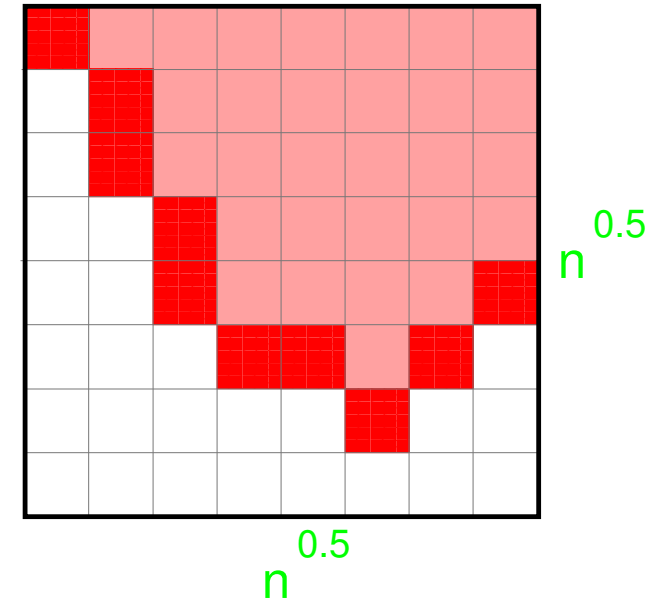
R-tree



Shortest-Path Quadtree

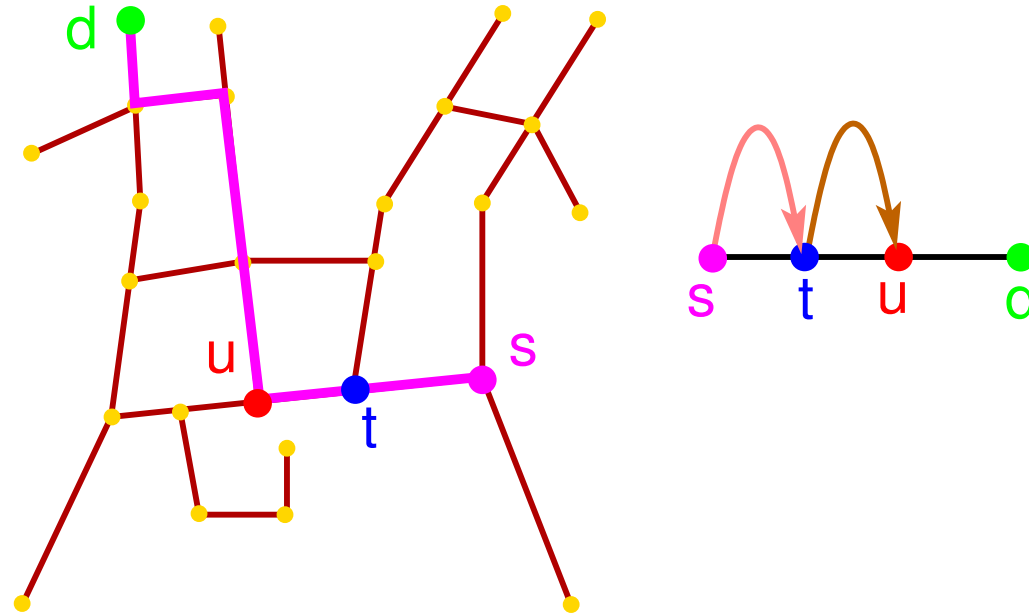
Space Complexity Analysis of Shortest-Path Quadtrees

- Consider a spatial network containing N vertices in a square grid of size $N^{0.5} \times N^{0.5}$ and embed it
- Perimeter of a region with monotonic boundary on one of its coordinates is of size $O(N^{0.5})$
- Perimeter of a region with a non-monotonic boundary can be of size $O(N)$
- Assumption: Regions of the shortest-path quadtree have *monotonic* boundaries
- Size of a shortest-path quadtree of a vertex u is $c\sqrt{N}$, where c is a function of the outdegree of u
- Total storage complexity of the SILC framework is $O(N\sqrt{N})$; closely follows empirical results
- Contribution: A mechanism to capture shortest paths in spatial networks based solely on geometry and independent of topology or connectivity



Path Retrieval

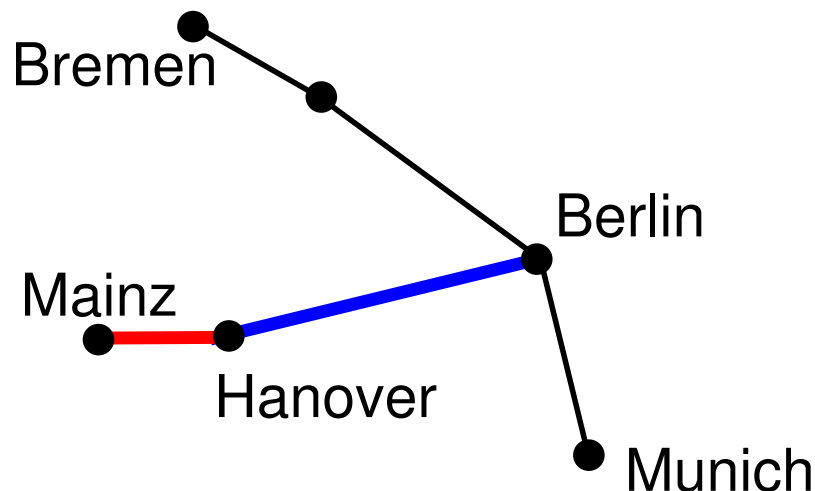
- Problem: How to retrieve the shortest path from a source s to a destination d ?



- Retrieve the shortest-path quadtree Q_s corresponding to s
- Find the colored region that contains d in Q_s
- Retrieve the vertex t connected to s in the region containing d in Q_s
- Retrieve the shortest-path quadtree Q_t corresponding to t
- Find the colored region that contains d in Q_t
- Retrieve the vertex u connected to t in the region containing d in Q_t
- Entire shortest path can be retrieved in size-of-path steps
- Network distance between s and d is immediately obtained from shortest path

Progressive Refinement of Distances

- Avoid full shortest path retrievals using progressive refinement
- Idea: Use distance intervals instead of the exact distance
- Progressive refinement: *Improve* interval if query cannot be answered
 - Associate Min/Max distance information with each Morton block
 - Refinement involves finding the next link in the shortest path
 - Worst case: retrieve entire shortest path to answer query
 - Many queries require distance comparison primitives
- Example: Is Munich closer to Mainz than Bremen?



	Munich	Bremen
Mainz	[10,20]	[15,30]
Hanover	[12,18]	[17,20]
Berlin	[13,15]	[18,19]

- Munich is closer as distance interval via Berlin does not intersect distance interval to Bremen via Berlin

Properties of a Non-Incremental kNN Algorithm

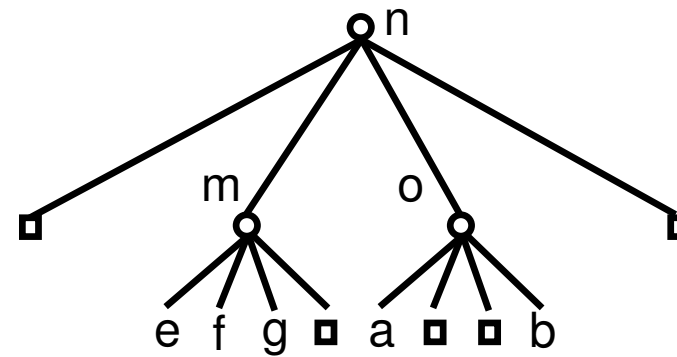
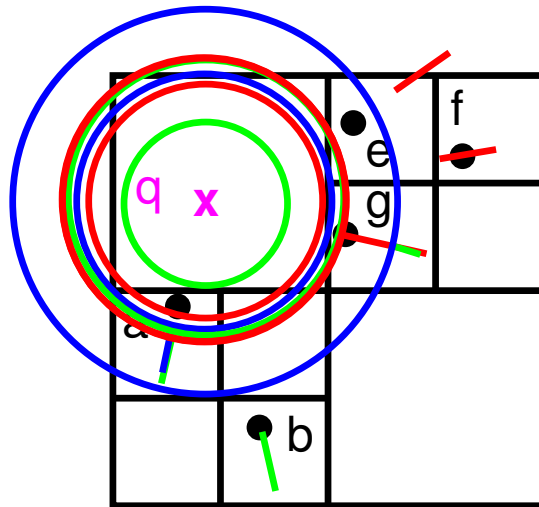
- Neighbors produced in increasing order of distance from q
- Use a priority queue Q of objects and blocks
- Q contains network distance interval $[\delta^-, \delta^+]$ of objects from q
- Additional information stored with each object o in Q
 1. An intermediate vertex u in shortest path from q to u
 2. network distance d from q to u
- Uses another priority queue L in addition to Q
 - Stores k objects found so far in increasing order of δ^+
 - D_k is the maximum of the distance interval of the k th element in L
 - **Idea:** Prune elements e from Q such that $\delta_e^- \geq D_k$
- Elements are removed from Q in increasing order of the minimum of their distance interval δ^- from q
 - Objects may be reinserted in Q if $\delta^- < D_k$
 - Terminate when $\delta^- \geq D_k$
- Advantages over Incremental best-first kNN (INN)
 - Smaller size of Q
 - Faster than INN

kNN Algorithm

1. Initialize priority queue Q by inserting the root T
2. Retrieve top element p in Q at each iteration and halt if minimum distance from q is $> D_k$
3. If p is a LEAF block, then replace it with all objects contained within it for which $\delta^- < D_k$ along with their network distance interval from q
 - Also enqueue objects in L if $\delta^+ < D_k$
4. If p is a NONLEAF block, then replace it with all its children blocks for which the minimum distance from q is $< D_k$
5. If p is an OBJECT, then test the distance interval of p for possible collisions with the current top element of Q
 - A collision occurs if the distance interval of p intersects the distance interval of the current top element in Q
 - Collision:
 - Remove p from L if $\delta^+ \leq D_k$
 - Apply refinement to improve distance interval of p and reinsert p in L if $\delta^+ \leq D_k$ and in Q if $\delta^- < D_k$ and go to Step 2
 - No collision: p is already one of k nearest neighbors in L (Theorem 1) and go to Step 2

Example of a Non-incremental k Neighbor Search

$k = 2$



1. Insert n into Queue.
 2. Expand n . Insert o, m into Queue.
 3. Expand o . Insert a, b into Queue, L . Set D_k .
 4. Expand m . Insert g, e, f into Queue and g into L .
Update D_k . Prune f and b from Queue.
 5. Process a . Collision of a with g .
Refine a . Reinsert a into Queue and L .
 6. Process g . Collision of g with a .
Refine and Reinsert g into Queue and L . Update D_k .
 7. Process a . No collision of a with g . No need to refine a further.
 8. Process g . No collision of g with e .
No need to refine g further. Report L .
- Example of a best-first nearest neighbor algorithm.
(Search radius to first element in Queue)

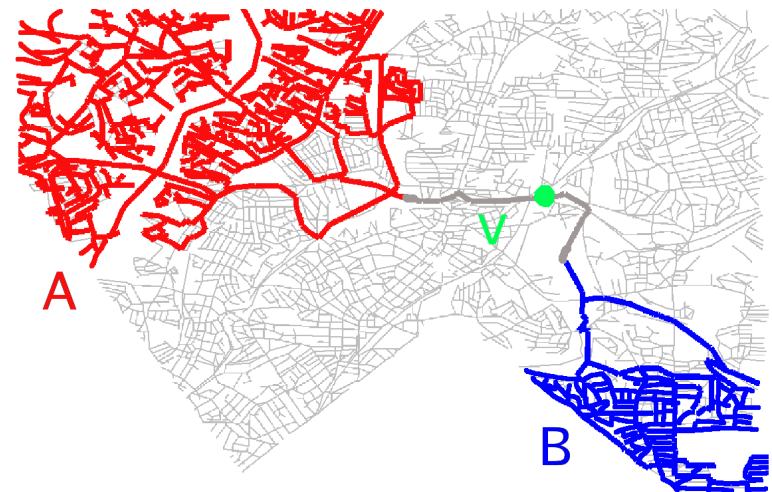
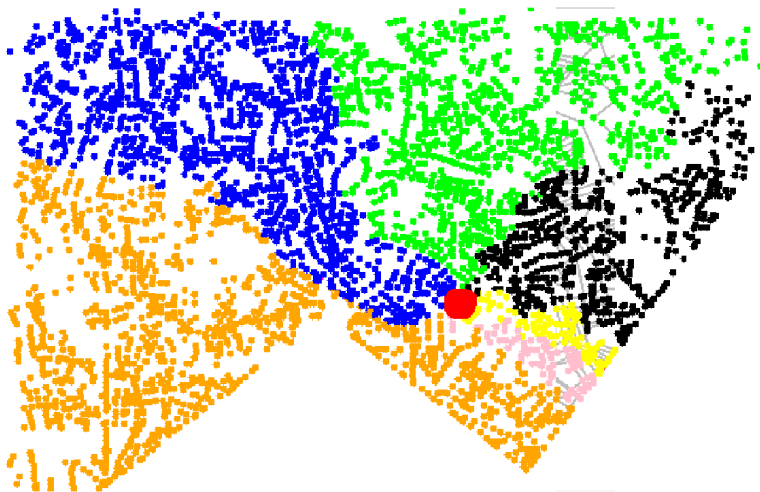
front
L Queue

Musings on How Realistic is the Approach

- How about a system for the whole US?
 - 24 million vertices x 10 seconds (say) per shortest path
 - Single machine = 2777 days
 - Google with 0.5 million machines = 480 seconds
 - Modest Cluster of 2000 machines = 1 day, 10 hours
 - Storage shown to be $cN\sqrt{N}$ Morton Blocks
 - $N = 24$ million vertices, 8 bytes per Morton block, $c = 2$ from empirical analysis = 1.8 TB
 - Easily Parallelizable: data parallelism
 - Mostly a one-time effort (decoupling)
- Open Challenge: Updates!
 - Changes to spatial network (e.g., road closure)
 - Dynamic traffic information
 - Strategy: How to localize changes to minimize recomputation?
- Approximation Strategies: location based services
 - Shortest-path quadtree on proximal vertices only (say, 100 miles around a vertex)
 - Multiresolution spatial networks
 - Full resolution around a source vertex that gets sparse gradually

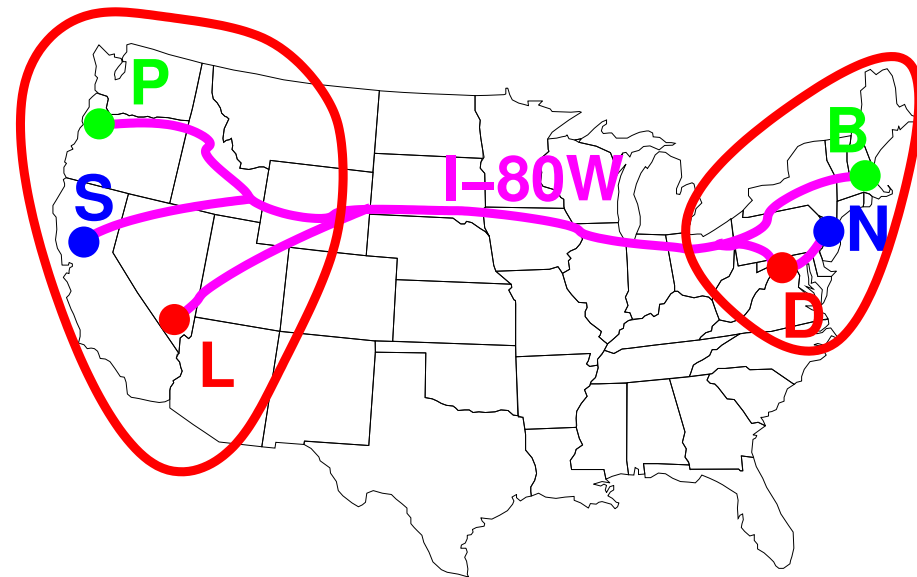
Path Coherence Beyond SILC

- The SILC framework captures the path coherence in the shortest paths
 - Captured: single source vertex to multiple destination vertices
 - Not captured: multiple source vertices to multiple destination vertices
- A new framework: Path Coherent Pairs (PCP)
 - Example of a path coherent pair denoted by: (A, B, v)
 - A is a set of source vertices
 - B is a set of destination vertices
 - v is a common vertex to all pairs of shortest paths



Finding Path Coherent Pairs in Spatial Networks

- Source Vertices: Washington, DC (D) , New York (N) , Boston (B)
- Destination vertices: Las Vegas (L) , Sacramento (S) , Portland (P)
- Anyone driving from “North-East” to “North-West” US uses I-80W
- Capture shortest paths from one million (say) sources in “North-East” to one million (say) destinations in “North-West” using $O(1)$ storage
- Intuition: Sources “sufficiently far” from destinations share common vertices in their shortest paths
- Decompose road network into PCPs:
 - Any vertex pair is contained in exactly one set in the shape of a dumbbell
 - All N^2 shortest paths are captured using $O(s^d N)$ storage where s is a small constant
- Key idea is the analogy to the well-separated pairs in computational geometry





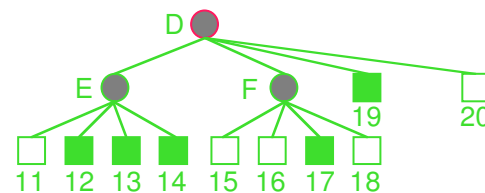
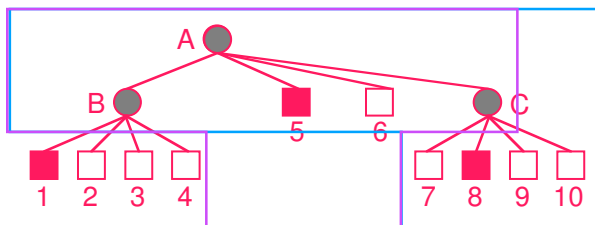
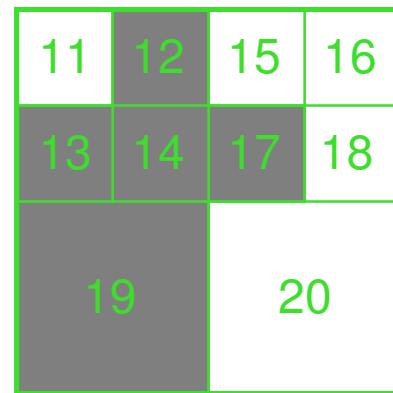
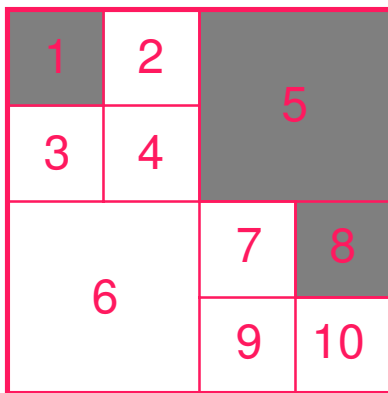
SET OPERATIONS ON QUADTREES

7	6	5	4	3	2	1
z	r	v	z	g	r	b

tf1



- UNION(S,T) : traverse S and T in tandem
 1. GRAY(S) ● :
 - GRAY(T) ● : recursively process subtrees and merge if all resulting sons are BLACK
 - BLACK(T) ● : result is T
 - WHITE(T) ○ : result is S
 2. BLACK(S) ● : result is S
 3. WHITE(S) ○ : result is T



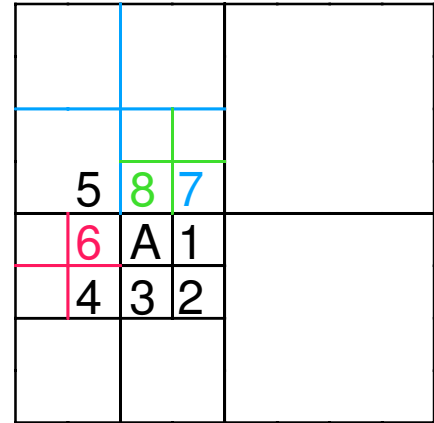
- INTERSECTION: interchange roles of BLACK and WHITE in UNION
- Execution time is bounded by sum of nodes in two input trees but may be less if don't create a new copy as really just the sum of the minimum of the number of nodes at corresponding levels of the two quadtrees
- More efficient than vectors as make use of global data
 1. vectors require a sort for efficiency
 2. region quadtree is already sorted

NEIGHBOR FINDING OPERATIONS USING QUADTREES

- Many image processing operations involve traversing an image and applying an operation to a pixel and some of its neighboring (i.e., adjacent) pixels

- For quadtree/octree representations replace pixel/voxel by block

- Neighbor is defined to be an adjacent block of greater than or equal size

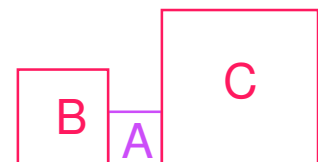


A has ~~5~~ ~~6~~ ~~7~~ 8 neighbors

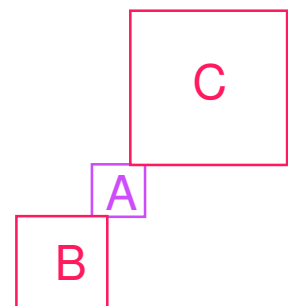
- Desirable to be able to locate neighbors in a manner that
 - is position-independent
 - is size-independent
 - makes no use of additional links to adjacent nodes (e.g., ropes and nets a la Hunter)
 - just uses the structure of the tree or configuration of the blocks

- Some block configurations are impossible, thereby simplifying a number of algorithms

- impossible for a node A to have two larger neighbors B and C on directly opposite sides or touching corners



- partial overlap of two blocks B and C with A is impossible since a quadtree is constructed by recursively splitting blocks into blocks that have side lengths that are powers of 2





7654321
g z r b z r b

nf2

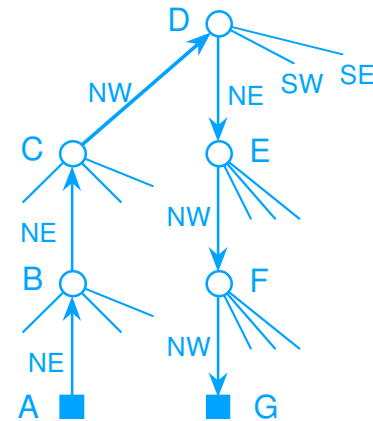
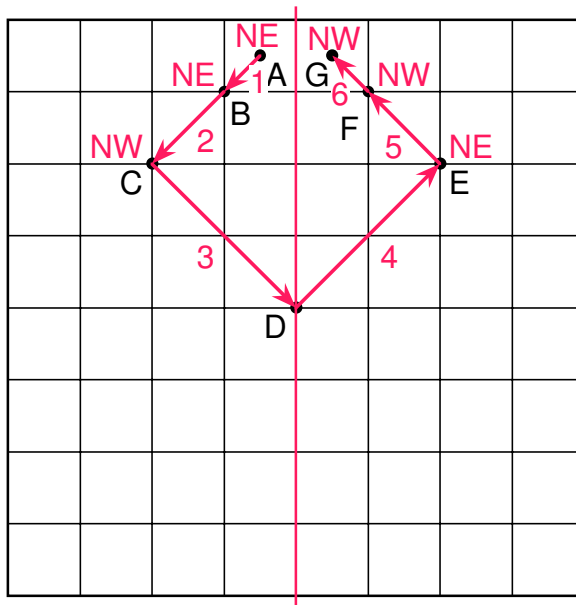


FINDING LATERAL NEIGHBORS OF EQUAL SIZE

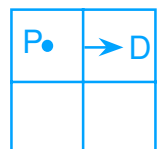
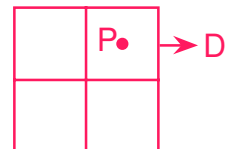
Algorithm: based on finding the nearest common ancestor

1. Ascend the tree if the node is a son of the same type as the direction of the neighbor ($_{ADJ}$)
2. Otherwise, the father F is the nearest common ancestor and retrace the path starting at F making mirror image moves about the edge shared by the neighboring blocks

Ex: E neighbor of A (i.e., G)



```
node procedure EQUAL_LATERAL_NEIGHBOR(P,D);
/* Find = size neighbor of P in direction D */
begin
  value pointer node P;
  value direction D;
  return (SON(if ADJ(D, SONTYPE(P)) then
    EQUAL_LATERAL_NEIGHBOR(FATHER(P),D)
  else FATHER(P),
    REFLECT(D, SONTYPE(P))));
end;
```



ADJ(A,B)

		B			
		NW	NE	SW	SE
A	N	T	T	F	F
	E	F	T	F	T
	S	F	F	T	T
	W	T	F	T	F

		B			
		NW	NE	SW	SE
REFLECT (A, B)	A				
	N	SW	SE	NW	NE
	E	NE	NW	SE	SW
	S	SW	SE	NW	NE
	W	NE	NW	SE	SW

ANALYSIS OF NEIGHBOR FINDING

1. Bottom-up random image model where each pixel has an equal probability of being black or white
 - probability of the existence of a 2x2 block at a particular position is $1/8$
 - OK for a checkerboard image but inappropriate for maps as it means that there is a very low probability of aggregation
 - problem is that such a model assumes independence
 - in contrast, a pixel's value is typically related to that of its neighbors
2. Top-down random image model where the probability of a node being black or white is p and $1-2p$ for being gray
 - model does not make provisions for merging
 - uses a branching process model and analysis is in terms of extinct branching processes
3. Use a model based on positions of the blocks in the decomposition
 - a block is equally likely to be at any position and depth in the tree
 - compute an average case based on all the possible positions of a block of size 1x1, 2x2, 4x4, etc.
 - 1 case at depth 0, 4 cases at depth 1, 16 cases at depth 2, etc.
 - this is not a realizable situation but in practice does model the image accurately



8	7	6	5	4	3	2	1
g	z	b	v	g	z	r	b

nf5



ANALYSIS OF FINDING LATERAL NEIGHBORS

25	9	33	1	41	17	49	
26	10	34	2	42	18	50	
27	11	35	3	43	19	51	
28	12	36	4	44	20	52	
29	13	37	5	45	21	53	
30	14	38	6	46	22	54	
31	15	39	7	47	23	55	
32	16	40	8	48	24	56	

$2^3 \cdot (2^3 - 1)$ neighbor pairs of equal sized nodes in direction E

NCA = nearest common ancestor

1–8 have NCA at level 3

9–24 have NCA at level 2

25–56 have NCA at level 1

Theorem: average number of nodes visited by

EQUAL_LATERAL_NEIGHBOR is ≤ 4

Proof:

- Let node A be at level i (i.e., a $2^i \times 2^i$ block)
- There are $2^{n-i} \cdot (2^{n-i} - 1)$ possible positions for node A such that an equal sized neighbor exists in a given horizontal or vertical direction

2^{n-i} rows

$2^{n-i} - 1$ adjacencies per row

$2^{n-i} \cdot 2^0$ have NCA at level n

$2^{n-i} \cdot 2^1$ have NCA at level $n-1$

...

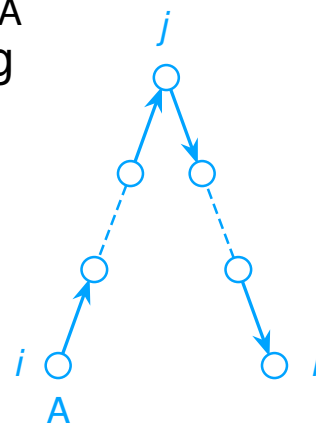
$2^{n-i} \cdot 2^{n-i-1}$ have NCA at level $i+1$

- For node A at level i , direction D, and the NCA at level j , $2 \cdot (j - i)$ nodes are visited in locating an equal-sized neighbor at level i

$$\sum_{i=0}^{n-1} \sum_{j=i+1}^n 2^{n-i} \cdot 2^{n-j} \cdot 2 \cdot (j - i)$$

$$\sum_{i=0}^{n-1} 2^{n-i} \cdot (2^{n-i} - 1)$$

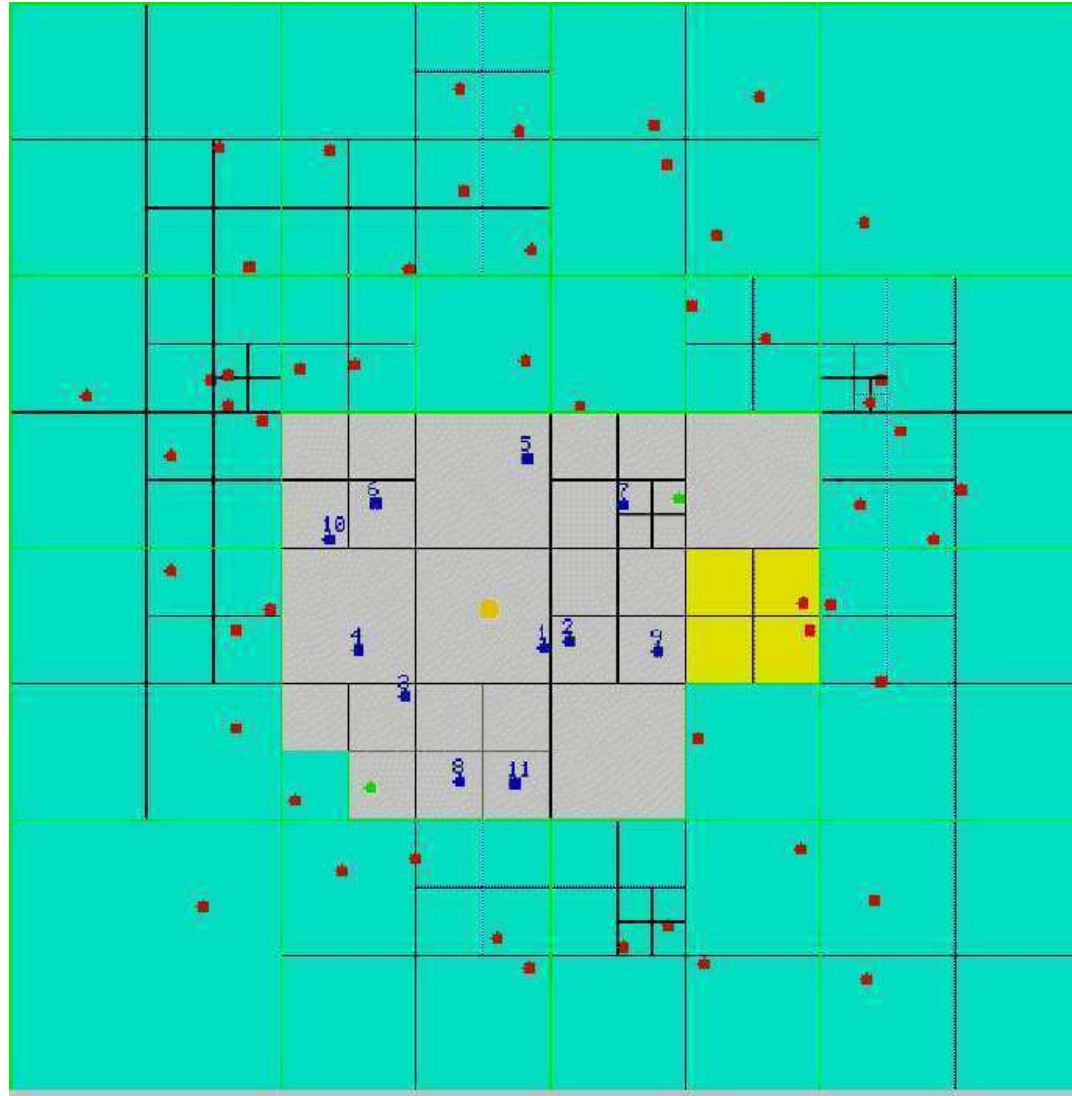
nodes are visited on the average ≤ 4



Outline

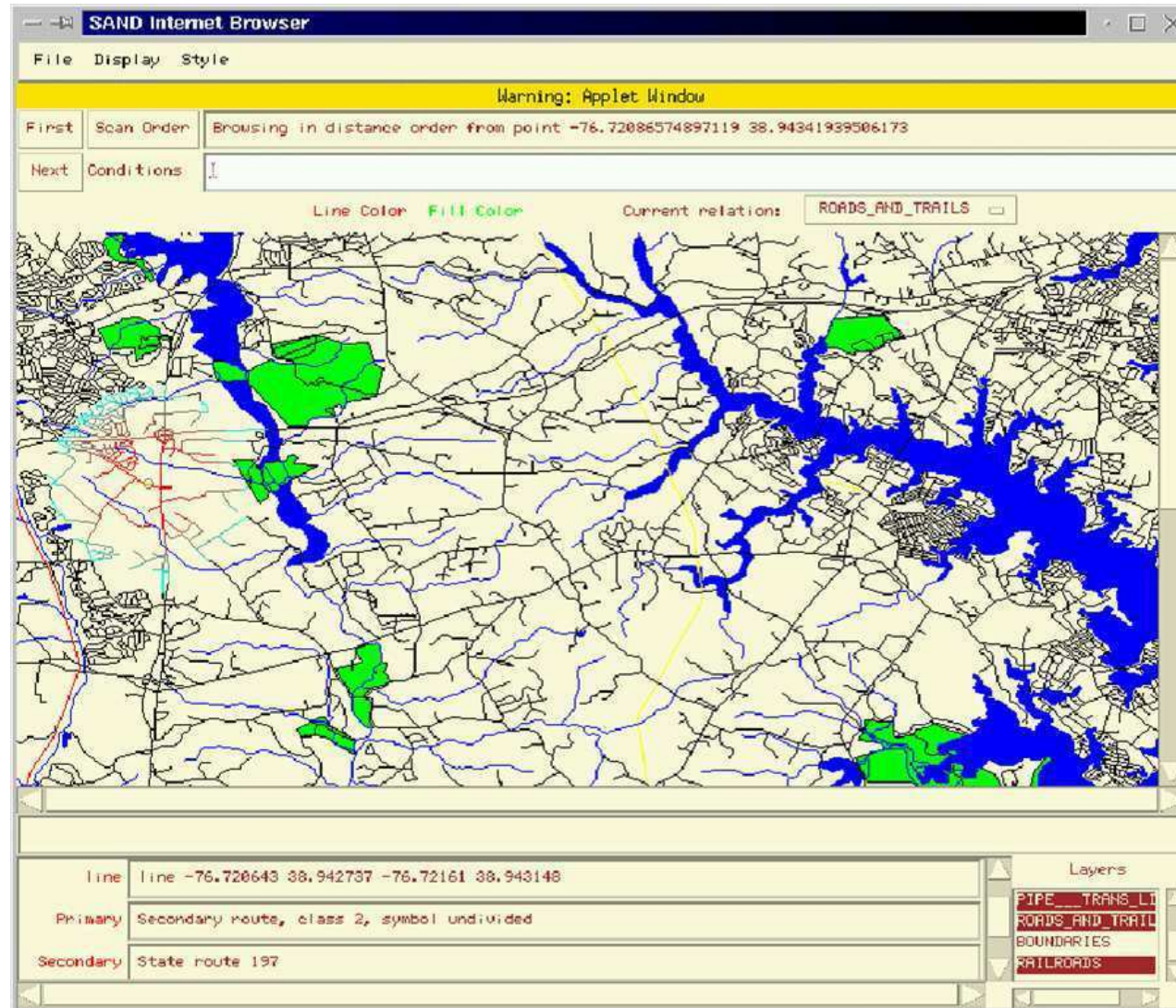
1. Introduction
2. Points
3. Lines
4. Regions, Volumes, and Surfaces
5. Bounding Box Hierarchies
6. Rectangles
7. Surfaces and Volumes
8. Metric Data
9. Operations
10. Example system

VASCO Spatial Applet



<http://www.cs.umd.edu/~hjs/quadtree/index.html>

SAND Internet Browser



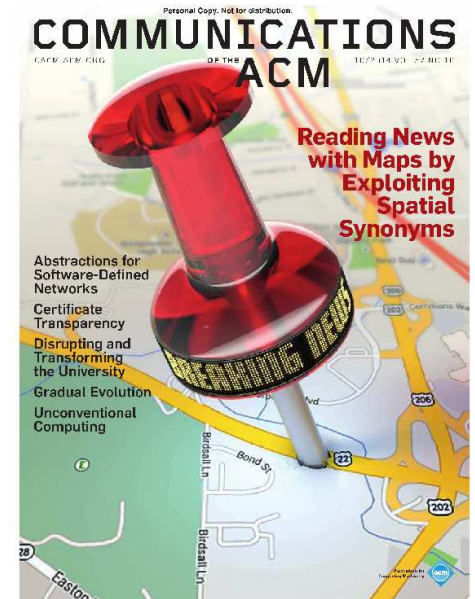
<http://www.cs.umd.edu/~brabec/sandjava/>

Sorting in Words

Hanan Samet*

`hjs@cs.umd.edu`

Department of Computer Science
Institute for Advanced Computer Studies
Center for Automation Research
University of Maryland
College Park, MD 20742, USA



for Video click here or go to <https://vimeo.com/106352925>

* Based on Joint Work with Marco D. Adelfio, Brendan C. Fruin, Jack Lotkowski, Michael D. Lieberman, Daniele Panozzo, Jagan Sankaranarayanan, Jon Sperling, and Ben E. Teitler

NewsStand: Spatio-Textual Aggregation of News and Display

1. Crawls the web looking for news sources and feeds
 - Indexing 10,000 news sources
 - About 50,000 news articles per day
2. Aggregate news articles by both content similarity and location
 - Articles about the same event are grouped into clusters
3. Rank clusters by importance which is based on:
 - Number of articles in cluster
 - Number of unique newspapers in cluster
 - Event's rate of propagation to other newspapers
4. Associate each cluster with its geographic focus or foci
5. Display each cluster at the positions of the geographic foci
6. Other options:
 - Category (e.g., General, Business, SciTech, Entertainment, Health, Sports)
 - Image and video galleries
 - Map stories by disease, brands, people, etc.
 - User-generated news (e.g., Social networks such as Twitter)


NewsStand: Map Mode

The screenshot displays the NewsStand Map Mode interface. At the top, there's a navigation bar with categories: All, General, Business, SciTech, Entertainment, Health, and Sports. A search bar is on the right with options for 'Enter a location...' and 'Keyword Search...'. Below the search bar is a 'Display: 30' slider. The main map area shows a world map with various locations marked. A 'Minimap' is visible in the center, showing a smaller version of the map with a 'Display: 3' slider. A 'Moscow, Idaho, United States' location is highlighted. On the right, a news article titled 'How Obama, Bush, Clinton viewed Russia's Putin over time' is displayed, with a '6 hours ago - usatoday.com' timestamp and a 'category: General' label. Below the article, there's a 'Description' section with text about missile launches. At the bottom, there are several filter panels: 'Languages' (All, Arabic, Chinese, Dutch, English, French, German, Greek, Hebrew, Hindi, Italian, Japanese, Persian), 'Countries' (All, Argentina, Australia, Austria, Bahamas, Belarus, Belgium, Benin, Bolivia, Brazil, Burkina Faso, Canada, Chile, China), 'Sources' (All, 25 25 - يناير Yanayer, A Bola, Algemeen Dagblad, Amar Ujala, Associated Press, Atlanta Journal-Constitution, Avesta, BBC News, Baltische Rundschau, Bangkok Post, Basler Zeitung, Bild, Boston Globe), 'Layers' (Icon Layer, Location Layer, Keyword Layer, People Layer, Disease Layer, Brand Layer), 'Multimedia' (Has 0+ images, Has 1+ images, Has 10+ images, Has 50+ images, Has 0+ videos, Has 1+ videos, Has 5+ videos, Has 10+ videos), and 'Feat. Articles' (Most Reputable, Most Recent, Real-Time). A 'Settings' button is in the bottom right corner.

- NewsStand is at <http://newsstand.umiacs.umd.edu/>
- Query: What is happening at location Y?

NewsStand: Top Stories Mode

How Obama, Bush, Clinton viewed Russia's Putin over time
5 hours ago from [usatoday.com](#)



Presidents Obama, George W. Bush and Bill Clinton held an optimistic view of Russian leader Vladimir Putin during their early interactions.

21042 related documents 8072 Images 277 Videos

Options: ☒ Most Reputable ☐ Most Recent ☐ Real-Time

Questions, answers about college union ruling
2 hours ago from [miamiherald.com](#)

A regional director of the National Labor Relations Board ruled Tuesday that

How Obama, Bush, Clinton viewed Russia's Putin over time
5 hours ago from [usatoday.com](#)

Presidents Obama, George W. Bush and Bill Clinton held an optimistic view

Renewed search for plane debris
2 hours ago from [bbc.co.uk](#)

The search for a missing Malaysian airliner is due to resume in the southern

VIDEO: Why did Facebook spend \$2bn on VR?
6 hours ago from [bbc.co.uk](#)

Facebook has announced it will buy Oculus VR, a Californian company

Two firefighters killed in Boston
1 hour ago from [bbc.co.uk](#)

Two firefighters are killed and 13 injured in the US city of Boston in a wind-

Grace period on 'Obamacare' deadline
12 hours ago from [bbc.co.uk](#)

The Obama administration extends the 31 March deadline for those unable to

Try TwitterStand | PhotoStand

Enter a location...

Keyword Search...

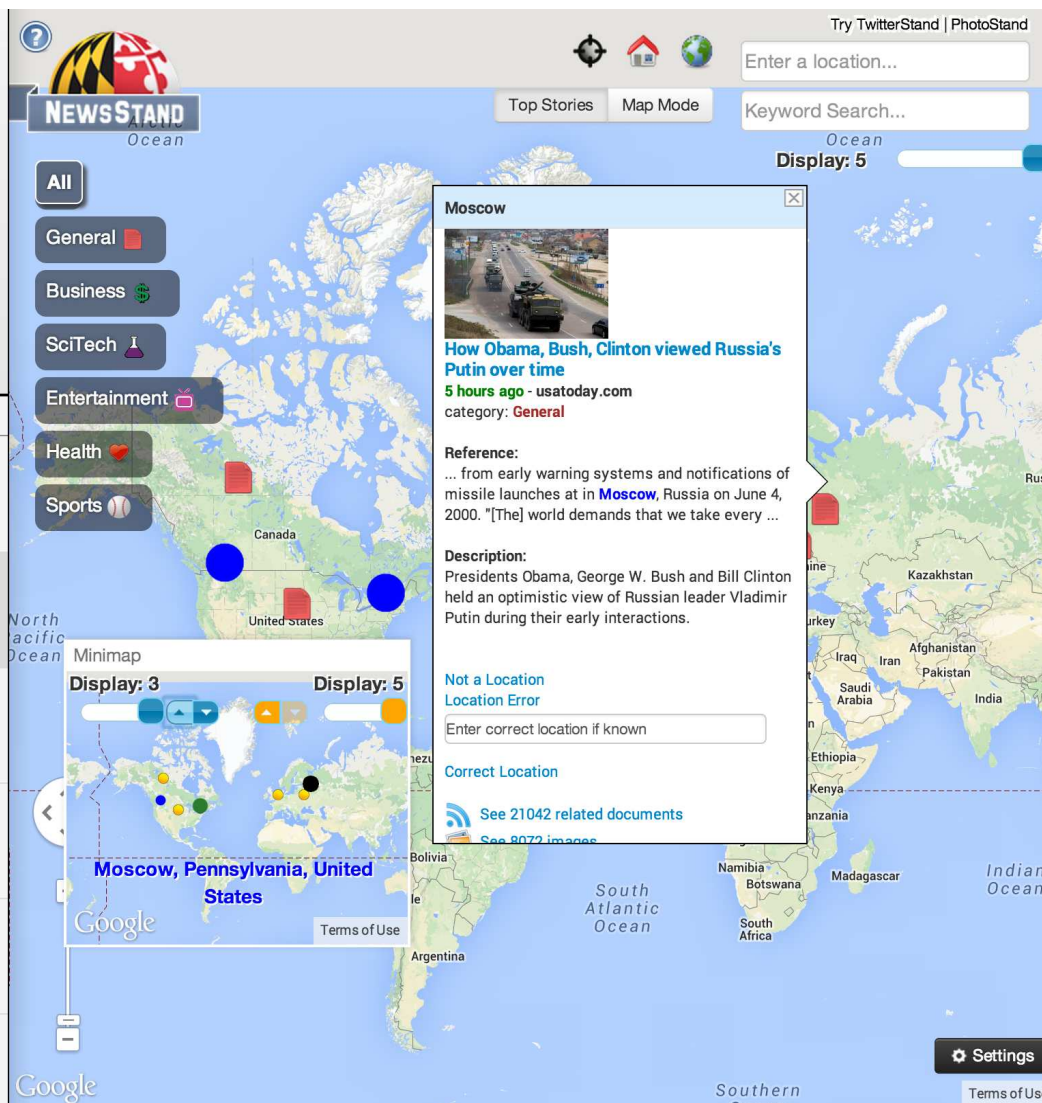
Display: 5

Top Stories Map Mode

NEWSSTAND Ocean

All General Business SciTech Entertainment Health Sports

Moscow



How Obama, Bush, Clinton viewed Russia's Putin over time
5 hours ago from [usatoday.com](#)
category: General

Reference:
... from early warning systems and notifications of missile launches at in **Moscow**, Russia on June 4, 2000. "[The] world demands that we take every ...

Description:
Presidents Obama, George W. Bush and Bill Clinton held an optimistic view of Russian leader Vladimir Putin during their early interactions.

Not a Location
Location Error
Enter correct location if known

Correct Location

See 21042 related documents
See 8072 images

Minimap
Display: 3 Display: 5

Moscow, Pennsylvania, United States

Google

Terms of Use

Settings

Terms of Use

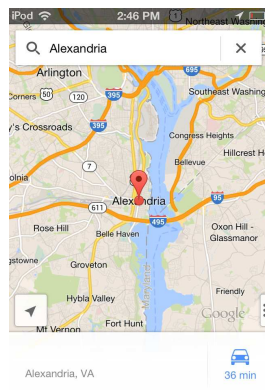
- NewsStand is at <http://newsstand.umiacs.umd.edu/>
- Query: Where is topic X occurring (spatial data mining)?

Geotagging

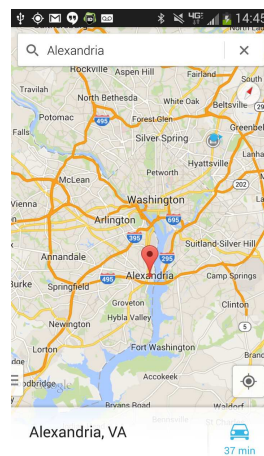
- Geotagging: Understanding textual references to spatial data
 1. Identifying or recognizing
 2. Classifying (is “Michigan” a state or a lake?)
 3. Disambiguating or resolving
 4. Localizing (geocoding to GPS coordinates)
- Context of textual references
 1. Queries - use prior queries and location
 - Ex: Query “Alexandria” when in “College Park, MD”
 2. Underlying data being queried - need context



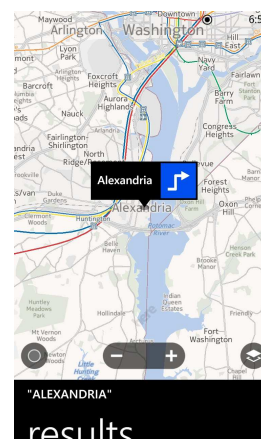
Apple iOS5
Maps by
Google



iOS Maps by
Google



Android
Maps by
Google



Here Maps
on Windows
Phone



Apple iOS6
and iOS7
Maps

Mechanics of Geotagging

1. Goal: high recall in toponym recognition (i.e., not missing toponyms) at expense of precision
 - Rectify by subsequent use of toponym resolution which can (and will) also be used to filter erroneous location interpretations
2. Toponym recognition: 2 stages
 - Finding toponyms
 - Filtering toponyms: postprocessing to remove errors in recognition
3. Toponym resolution
 - Use local lexicons containing locations that can be specified without all of their containers (derived from articles from a particular news source) to determine spatial reader scopes for particular sources
 - E.g., "Dublin" implies "Dublin, Ohio" for readers of a news source in "Columbus, Ohio"
 - Use Wikipedia articles to find concepts related to particular locations so that the presence of these concepts in conjunction with an ambiguous reference to a location can be properly resolved
 - E.g., mention of "White House" in conjunction with "Washington" to provide evidence for resolving as "Washington, D.C."

Local Lexicon Example



Sorting in Space References

1. H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, 2006. (Translated to Chinese ISBN 978-7-302-22784-7).
2. H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.
3. H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.
4. H. Samet. A sorting approach to indexing spatial data. *International Journal on Shape Modeling*, 14(1):15–37, June 2008.
5. H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *Proceedings of the ACM SIGMOD Conference*, pages 43–54, Vancouver, Canada, June 2008. (2008 ACM SIGMOD Best Paper Award).
6. J. Sankaranarayanan and H. Samet. Distance oracles for spatial networks. In *Proceedings of the 25th IEEE International Conference on Data Engineering*, pages 652–663, Shanghai, China, April 2009.
7. J. Sankaranarayanan, H. Samet, and H. Alborzi. Path oracles for spatial networks. *PVLDB*, 2(1):1210–1221, August 2009. Also *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB)*.

Sorting in Space References (Continued)

8. J. Sankaranarayanan and H. Samet. Query processing using distance oracles for spatial networks. *IEEE Transactions on Knowledge and Data Engineering*, 22(8):1158–1175, August 2010. (Best Papers of ICDE 2009 Special Issue).

VASCO References

1. F. Brabec and H. Samet. The VASCO R-tree JAVATM applet. In *Visual Database Systems (VDB4). Proceedings of the IFIP TC2//WG2.6 Fourth Working Conference on Visual Database Systems*, Y. Ioannidis and W. Klas, eds., pages 147–153, Chapman and Hall, L'Aquila, Italy, May 1998.
2. F. Brabec, H. Samet, and C. Yilmaz. VASCO: visualizing and animating spatial constructs and operations. In *Proceedings of the 19th Annual Symposium on Computational Geometry*, pages 374–375, San Diego, CA, June 2003.
3. F. Brabec and H. Samet. Visualizing and animating R-trees and spatial operations in spatial databases on the worldwide web. In *Visual Database Systems (VDB4). Proceedings of the IFIP TC2//WG2.6 Fourth Working Conference on Visual Database Systems*, Y. Ioannidis and W. Klas, eds., pages 123–140, Chapman and Hall, L'Aquila, Italy, May 1998.
4. F. Brabec and H. Samet. Visualizing and animating spatial decompositions and operations in spatial databases on the worldwide web. In *Proceedings of Visual Database Systems*, L'Aquila, Italy, May 1998.

VASCO References (Continued)

5. F. Brabec and H. Samet. Visualizing and animating search operations on quadrees on the worldwide web. In *Proceedings of the 16th European Workshop on Computational Geometry*, K. Kedem and M. Katz, eds., pages 70–76, Eilat, Israel, March 2000.

QUILT References

1. H. Samet, A. Rosenfeld, C. A. Shaffer, and R. E. Webber. A geographic information system using quadrees. *Pattern Recognition*, 17(6):647–656, November/December 1984.
2. H. Samet, A. Rosenfeld, C. A. Shaffer, and R. E. Webber. Quadtree region representation in cartography: experimental results. In *Proceedings of Computer Vision and Pattern Recognition'83*, pages 176–177, Washington, DC, June 1983. Also expanded version in *IEEE Transactions on Systems, Man, and Cybernetics*, 13(6):1148–1154, November/December 1983.
3. H. Samet, C. A. Shaffer, R. C. Nelson, Y.-G. Huang, K. Fujimura, and A. Rosenfeld. Recent developments in linear quadtree-based geographic information systems. *Image and Vision Computing*, 5(3):187–197, August 1987.
4. C. A. Shaffer, H. Samet, and R. C. Nelson. QUILT: a geographic information system based on quadrees. *International Journal of Geographical Information Systems*, 4(2):103–131, April–June 1990. Also University of Maryland Computer Science Technical Report TR–1885.1, July 1987.

SAND Internet Browser References

1. F. Brabec and H. Samet. Client-based spatial browsing on the world wide web. *IEEE Internet Computing*, 11(1):52–59, January/February 2007.
2. C. Esperança and H. Samet. Experience with SAND/Tcl: a scripting tool for spatial databases. *Journal of Visual Languages and Computing*, 13(2):229–255, April 2002.
3. H. Samet, H. Alborzi, F. Brabec, C. Esperança, G. R. Hjaltason, F. Morgan, and E. Tanin. Use of the SAND spatial browser for digital government applications. *Communications of the ACM*, 46(1):63–66, January 2003.
4. H. Samet, A. Phillippy, and J. Sankaranarayanan. Knowledge discovery using the SAND spatial browser. In *Proceedings of the 7th National Conference on Digital Government Research*, pages 284–285, Philadelphia, PA, May 2007.

Sorting on Words Project References

1. A. Abdelrazek, E. Hand, and H. Samet. Brands in NewsStand: Spatio-temporal browsing of business news. In *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, M. Ali, M. Gertz, Y. Huang, M. Renz, and J. Sankaranarayanan, eds., Seattle, WA, November 2015. Article 97.
2. M. D. Adelfio and H. Samet. GeoWhiz: Using common categories for toponym resolution. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, C. A. Knoblock, P. Kröger, J. C. Krumm, M. Schneider, and P. Widmayer, eds., pages 542–545, Orlando, FL, November 2013.
3. M. D. Adelfio and H. Samet. Structured toponym resolution using combined hierarchical place categories. In *Proceedings of 7th ACM SIGSPATIAL Workshop on Geographic Information Retrieval (GIR'13)*, R. Purves and C. Jones, eds., pages 49–56, Orlando, FL, November 2013.
4. M. D. Adelfio and H. Samet. Automated tabular itinerary visualization. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Y. Huang, M. Gertz, J. C. Krumm, J. Sankaranarayanan, and M. Schneider, eds., pages 593–596, Dallas, TX, November 2014.

Sorting on Words Project References (Continued)

5. M. D. Adelfio and H. Samet. Itinerary retrieval: Travelers, like traveling salesmen, prefer efficient routes. In *Proceedings of 8th ACM SIGSPATIAL Workshop on Geographic Information Retrieval (GIR'14)*, R. Purves and C. Jones, eds., pages 1:1–1:8, Dallas, TX, November 2014.
6. B. C. Fruin, H. Samet, and J. Sankaranarayanan. Tweetphoto: photos from news tweets. In *Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, I. Cruz, C. A. Knoblock, P. Kröger, E. Tanin, and P. Widmayer, eds., pages 582–585, Redondo Beach, CA, November 2012.
7. C. Fu, J. Sankaranarayanan, and H. Samet. Weibostand: Capturing Chinese breaking news using Weibo. In *Proceedings of the 6th ACM SIGSPATIAL International Workshop on Location-Based Social Networks (LBSN'14)*, A. Pozdnukhov and S. Xu, eds., pages 41–48, Dallas, TX, November 2014.
8. N. Gramsky and H. Samet. Seeder finder - identifying additional needles in the Twitter haystack. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Location-Based Social Networks (LBSN'13)*, A. Pozdnukhov, ed., pages 44–53, Orlando, FL, November 2013.

Sorting on Words Project References (Continued)

9. S.-S. Ho, M. D. Lieberman, P. Wang, and H. Samet. Mining future spatiotemporal events and their sentiment from online news articles for location-aware recommendation system. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems (MobiGIS 2012)*, pages 25–32, Redondo Beach, CA, November 2012.
10. A. Jackoway, H. Samet, and J. Sankaranarayanan. Identification of live news events using Twitter. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks (LBSN'11)*, Y. Zheng and M. F. Mokbel, eds., pages 25–32, Chicago, November 2011.
11. Y. Kanza and H. Samet. An online marketplace for geosocial data. In *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, M. Ali, M. Gertz, Y. Huang, M. Renz, and J. Sankaranarayanan, eds., Seattle, WA, November 2015. Article 10.
12. R. Lan, M. D. Adelfio, and H. Samet. Spatio-temporal disease tracking using news articles. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on the Use of GIS in Public Health (HealthGIS 2014)*, pages 31–38, Dallas, TX, November 2014.

Sorting on Words Project References (Continued)

13. R. Lan, M. D. Lieberman, and H. Samet. The picture of health: map-based, collaborative spatio-temporal disease tracking. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on the Use of GIS in Public Health (HealthGIS 2012)*, pages 27–35, Redondo Beach, CA, November 2012.
14. J. L. Leidner and M. D. Lieberman. Detecting geographical Project References in the form of place names and associated spatial natural language. *SIGSPATIAL Special*, 3(2):5–11, 2011.
15. H. Li, S. Peng, and H. Samet. Streaming news image summarization. In *Proceedings of the 27th International Conference on Pattern Recognition*, Cancun, Mexico, December 2016.
16. M. D. Lieberman and H. Samet. Multifaceted toponym recognition for streaming news. In *Proceedings of the 34th International Conference on Research and Development in Information Retrieval (SIGIR'11)*, pages 843–852, Beijing, China, July 2011.
17. M. D. Lieberman and H. Samet. Adaptive context features for toponym resolution in streaming news. In *Proceedings of the 35th International Conference on Research and Development in Information Retrieval (SIGIR'12)*, pages 731–740, Portland, OR, August 2012.

Sorting on Words Project References (Continued)

18. M. D. Lieberman and H. Samet. Supporting rapid processing and interactive map-based exploration of streaming news. In *Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, I. Cruz, C. A. Knoblock, P. Kröger, E. Tanin, and P. Widmayer, eds., pages 179–188, Redondo Beach, CA, November 2012.
19. M. D. Lieberman, H. Samet, and J. Sankaranarayanan. Geotagging: Using proximity, sibling, and prominence clues to understand comma groups. In *Proceedings of 6th Workshop on Geographic Information Retrieval*, R. Purves, C. Jones, and P. Clough, eds., Zurich, Switzerland, February 2010. Article 6.
20. M. D. Lieberman, H. Samet, and J. Sankaranarayanan. Geotagging with local lexicons to build indexes for textually-specified spatial data. In *Proceedings of the 26th IEEE International Conference on Data Engineering*, pages 201–212, Long Beach, CA, March 2010.
21. M. D. Lieberman, H. Samet, J. Sankaranarayanan, and J. Sperling. STEWARD: architecture of a spatio-textual search engine. In *Proceedings of the 15th ACM International Symposium on Advances in Geographic Information Systems*, H. Samet, M. Schneider, and C. Shahabi, eds., pages 186–193, Seattle, WA, November 2007.

Sorting on Words Project References (Continued)

22. M. D. Lieberman, H. Samet, J. Sankaranarayanan, and J. Sperling. Spatio-textual spreadsheets: Geotagging via spatial coherence. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, D. Agrawal, W. G. Aref, C.-T. Lu, M. F. Mokbel, P. Scheuermann, C. Shahabi, and O. Wolfson, eds., pages 524–527, Seattle, WA, November 2009.
23. M. D. Lieberman, J. Sankaranarayanan, H. Samet, and J. Sperling. Augmenting spatio-textual search with an infectious disease ontology. In *Proceedings of the Workshop on Information Integration Methods, Architectures, and Systems (IIMAS08) (ICDE Workshops 2008)*, pages 266–269, Cancun, Mexico, April 2008.
24. C. Liu, B. C. Fruin, and H. Samet. Sac: Semantic adaptive caching for spatial mobile applications. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, C. A. Knoblock, P. Kröger, J. C. Krumm, M. Schneider, and P. Widmayer, eds., pages 184–193, Orlando, FL, November 2013.

Sorting on Words Project References (Continued)

25. G. Quercini and H. Samet. Uncovering the spatial relatedness in Wikipedia. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Y. Huang, M. Gertz, J. C. Krumm, J. Sankaranarayanan, and M. Schneider, eds., pages 153–162, Dallas, TX, November 2014.
26. G. Quercini, H. Samet, J. Sankaranarayanan, and M. D. Lieberman. Determining the spatial reader scopes of news sources using local lexicons. In *Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, A. El Abbadi, D. Agrawal, M. Mokbel, and P. Zhang, eds., pages 43–52, San Jose, CA, November 2010.
27. H. Samet. Using minimaps to enable toponym resolution with an effective 100% rate of recall. In *Proceedings of 8th ACM SIGSPATIAL Workshop on Geographic Information Retrieval (GIR'14)*, R. Purves and C. Jones, eds., pages 9:1–9:8, Dallas, TX, November 2014.
28. H. Samet. Location specification and representation in multimedia databases. In *Proceedings of the IEEE International Symposium on Multimedia (ISM2015)*, Miami, FL, December 2015. To appear.

Sorting on Words Project References (Continued)

29. H. Samet, M. D. Adelfio, B. C. Fruin, M. D. Lieberman, and J. Sankaranarayanan. PhotoStand: a map query interface for a database of news photos. *PVLDB*, 6(12):1350–1353, August 2013. Also *Proceedings of the 39th International Conference on Very Large Data Bases (VLDB)*.
30. H. Samet, M. D. Adelfio, B. C. Fruin, M. D. Lieberman, and B. E. Teitler. Porting a web-based mapping application to a smartphone app. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, D. Agrawal, I. Cruz, C. S. Jensen, E. Ofek, and E. Tanin, eds., pages 525–528, Chicago, November 2011.
31. H. Samet, B. C. Fruin, and S. Nutanong. Duking it out at the smartphone mobile app mapping API corral: Apple, Google, and the competition. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems (MobiGIS 2012)*, Redondo Beach, CA, November 2012.
32. H. Samet, B. C. Fruin, and S. Nutanong. Presentation consistency issues in smartphone mapping apps. Technical report, Computer Science Department, University of Maryland, College Park, MD, November 2015.

Sorting on Words Project References (Continued)

33. H. Samet, M. D. Lieberman, J. Sankaranarayanan, and J. Sperling. STEWARD: Demo of spatio-textual extraction on the web aiding the retrieval of documents. In *Proceedings of the 7th National Conference on Digital Government Research*, pages 300–301, Philadelphia, PA, May 2007.
34. H. Samet, S. Nutanong, and B. C. Fruin. Dynamic presentation consistency issues in smartphone mapping apps. *Communications of the ACM*, 59(9):58–67, September 2016.
35. H. Samet, S. Nutanong, and B. C. Fruin. Static presentation consistency issues in smartphone mapping apps. *Communications of the ACM*, 59:88–98, May 2016.
36. H. Samet, J. Sankaranarayanan, M. D. Lieberman, M. D. Adelfio, B. C. Fruin, J. M. Lotkowski, D. Panozzo, J. Sperling, and B. E. Teitler. Reading news with maps by exploiting spatial synonyms. *Communications of the ACM*, 57(10):64–77, October 2014. Also see video at <https://vimeo.com/106352925>.

Sorting on Words Project References (Continued)

37. H. Samet, B. E. Teitler, M. D. Adelfio, and M. D. Lieberman. Adapting a map query interface for a gesturing touch screen interface. In *Proceedings of the Twentieth International Word Wide Web Conference (Companion Volume)*, S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, and R. Kumar, eds., pages 257–260, Hyderabad, India, March-April 2011.
38. J. Sankaranarayanan and H. Samet. Images in news. In *Proceedings of the 24th International Conference on Pattern Recognition*, pages 3240–3243, Istanbul, Turkey, August 2010.
39. J. Sankaranarayanan, H. Samet, B. Teitler, M. D. Lieberman, and J. Sperling. TwitterStand: News in tweets. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, D. Agrawal, W. G. Aref, C.-T. Lu, M. F. Mokbel, P. Scheuermann, C. Shahabi, and O. Wolfson, eds., pages 42–51, Seattle, WA, November 2009.

Sorting on Words Project References (Continued)

40. B. Teitler, M. D. Lieberman, D. Panozzo, J. Sankaranarayanan, H. Samet, and J. Sperling. NewsStand: A new view on news. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, W. G. Aref, M. F. Mokbel, H. Samet, M. Schneider, C. Shahabi, and O. Wolfson, eds., pages 144–153, Irvine, CA, November 2008. (2008 ACM GIS Best Paper Award).
41. B. E. Teitler, J. Sankaranarayanan, H. Samet, and M. D. Adelfio. Online document clustering using GPUs. In *Proceedings of the Second International ADBIS Workshop on GPUs in Databases (GID 2013)*, Genoa, Italy, September 2013. Also University of Maryland Computer Science TR 4970, August 2010.
42. F. Wajid and H. Samet. Crimestand: Spatial tracking of criminal activity. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, M. Ali, S. Newsma, S. Ravada, M. Renz, and G. Trajcevski, eds., San Francisco, November 2016.