

*Geographic Information Technology Training Alliance (GITTA) presents:*

# **Spatial Partitioning and Indexing**

**Responsible persons: Claudia Dolci, Dante Salvini, Michael  
Schrattner, Robert Weibel**



# Content

1. Spatial Partitioning and Indexing .....	2
1.1. Overview .....	3
1.1.1. Spatial Object Approximation .....	3
1.1.2. Spatial Data Access Methods .....	5
1.1.3. Basics of Computer File and Database Structures .....	6
1.1.4. Principles of Spatial Data Access and Search .....	7
1.2. Regular Decomposition .....	9
1.2.1. Regular Grids .....	9
1.2.2. Geometry allocation .....	10
1.2.3. Quadtrees .....	11
1.2.4. Searching Quadtrees .....	13
1.3. Object-oriented Decomposition .....	15
1.3.1. Binary Tree .....	15
1.3.2. R-Trees .....	16
1.4. Summary .....	21
1.5. Bibliography .....	22

# 1. Spatial Partitioning and Indexing

This lesson will explain the basic concepts of spatial partitioning and indexing processes. Firstly, regular decomposition theory is discussed. The different methods are widely described and illustrated with comprehensive examples.

Afterwards, the object-oriented decomposition is explained, which on completion should allow the understanding of important aspects of spatial partitioning and indexing.

## Learning Objectives

- You recognise the meaning and characteristics of spatial object approximation and spatial data access methods.
- You know and understand the advantages and disadvantages of space and data driven indexes.
- You are able to compare different regular decomposition methods (regular grids, quadrees, etc.) and you are familiar to their specific characteristics.
- You can explain the object-oriented B-tree and R-tree decomposition methods.

# 1.1. Overview

## Introduction

If you are looking on a world map and someone asks you where Geneva is, you will probably move your attention to the European continent, in particular to Swizerland (provided you know where Geneva approximately lies). Then you will try to locate successively smaller and smaller regions, where Geneva should and gradually narrowing until you identify the city at lake Geneva in the southwest of Switzerland.

**Only pictures can be viewed in the PDF version! For Flash etc. see online version. Only screenshots of animations will be displayed. [\[link\]](#)**

Whilst searching for Geneva you decomposed the space into sectors. You discarded the ones which aren't pertinent to your task and you subdivided the candidate region to contain the city recursively. You partitioned the map (space), for instance.

Another example: You forgot the phone number of a friend. To find it out you will probably look in the directories. Actually looking for Mr. Jones, you won't begin at the first page of the directories going through until you find it. Surely you will use the index and begin your search by the letter 'j' and then slip through all the pages until you find 'jo' and so on. In other words you make use of the indexing applied to the structured information contained in the directories.

Hierarchical spatial data structures are based on the principle of recursive decomposition. They are attractive because they are compact and depending on the nature of the data, they can save storage space as well as time and also facilitate operations such as search.

This means that spatial data access methods (spatial indexing) in geodatabases, e.g. for spatial searches in geographic information systems, provide fast access to spatial data.

### 1.1.1. Spatial Object Approximation

#### Reproduction

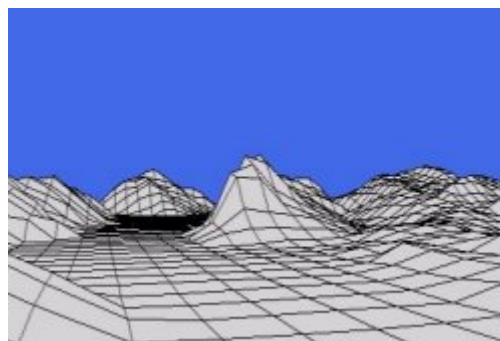
A GIS reproduces items of the real world as objects in its data structure and stores them in its database. The geometry of these objects is normally based on the geometric primitives (point, line, surface), which requires some simplification and approximation of the shapes (discretized space) as they are in reality. A higher resolution of the representation requires more processing time as well as considerable storage space and may therefore not be practical.

**Reality**



*S.Salvatore by Lugano*


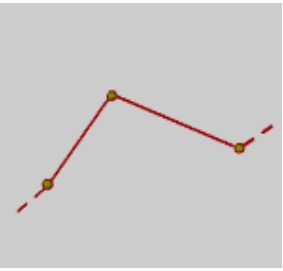
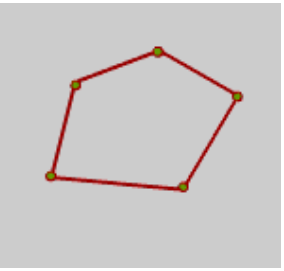

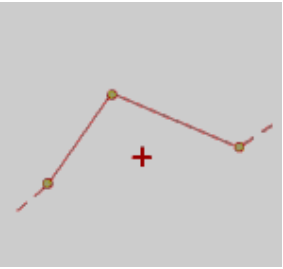
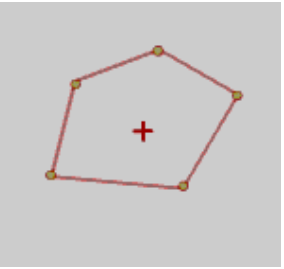

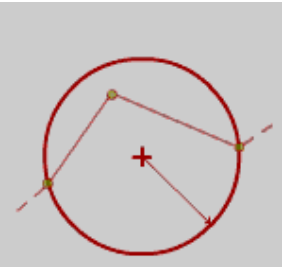
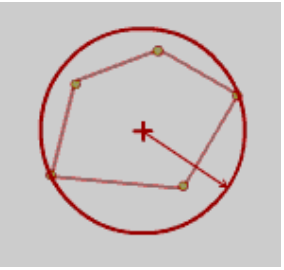
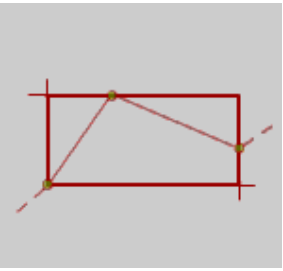
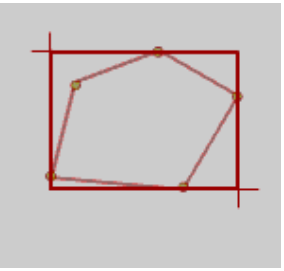
**discretized representation**

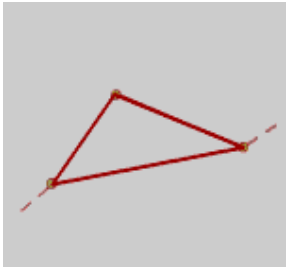
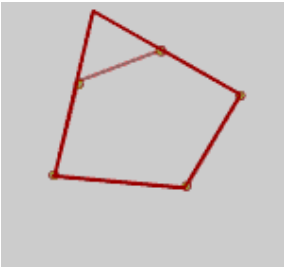


*DTM of the region of Lugano*

### Storage

The approximation of the geometry of spatial objects in GIS is applied for storage purposes as well as for querying. The geometric attribute of each object uses pointers to the corresponding geometric shape and bounding box to store an approximation of its extent. The most elementary approximation of a geometric shape is with its centroid (2 parameters:  $x$  and  $y$ ). The next approximation level adds the radius of a circle that encloses the object (3 parameters). More complex approximation levels are shown in the table below. When for instance the searching algorithm retrieves a set of approximated items corresponding to the query, then those objects are processed using the whole geometric details stored. Through this approximation, the search of complex objects can be done easily.

	Geometry type		
Approximation level	Point	Line	Polygon
			
2 parameters Through the centroid (X,Y)			
3 parameters Through a circle (X,Y,R)			
4 parameters Through a bounding box (Xul,Yul / Xbr,Ybr)			

n parameters Through complex geometries (X1,Y1 / X2,Y2 / ... / Xn,Yn)			
--	--	--	---

*Approximation of spacial objects [After R. Bill, Vol. 1, 1999]*

### Object approximation by tessellation

The word "tessellate" means to form or arrange small squares in a checkered or mosaic pattern. A regular tessellation means a tessellation made up of congruent regular polygons. With a tessellation of squares, we can easily approximate the geometric representation of an object as shown below:

**Only pictures can be viewed in the PDF version! For Flash etc. see online version. Only screenshots of animations will be displayed. [\[link\]](#)**

## 1.1.2. Spatial Data Access Methods

### Introduction

Fundamental to all information systems is the need to search through a large quantity of data, in order to find a subset that satisfies the user's query. The distinguishing characteristics of geographical data retrieval is that it is expressed in terms of spatial locations and spatial relationships.

As shown before in general in the introduction, spatial queries may be either location-based (geometry-based) or phenomenon-based (attribute-based), or a combination of the two (see B-AN: Spatial Queries).

### Attribute-based queries (phenomenon-based)

This type of request selects features or records geographic features that satisfy a statement expressing a set of conditions that forms the basis for the retrieval. The expression considers only conditions for the attributes describing the features. In this case the required result may be generated from the intersection of several layers corresponding to particular thematically specific phenomena.

**Only pictures can be viewed in the PDF version! For Flash etc. see online version. Only screenshots of animations will be displayed. [\[link\]](#)**

### Geometry-based queries (location-based)

The need to access data specified in terms of geometry (points, lines and polygons) and of spatial relationships between them, has introduced the requirement for specialized storage and data-search procedures. This is because, in such a case we need to be able to retrieve records based on some spatial properties, which are not stored explicitly in the database.

In a relatively simple case, known as a range search, the query may request all data or particular classes that

are inside a rectangular spatial window defined by ranges of coordinates in two dimensions. Stored geometric objects may actually lie within the ranges, i.e. be entirely inside, in which case they can be retrieved as a whole. Alternatively they may overlap the range, in which case the overlapping objects may need to be clipped at the boundary of the ranges of the search region to find the part that are inside.

**Only pictures can be viewed in the PDF version! For Flash etc. see online version. Only screenshots of animations will be displayed. [\[link\]](#)**

### Topological relationships

Queries that include topological relationships between phenomena may make use of stored topological relations. Commonly used procedures are those to test whether a point, a line or polygon is located inside a specified polygon. Other related procedures test whether geometric objects are coincident or adjacent with each other.

**Only pictures can be viewed in the PDF version! For Flash etc. see online version. Only screenshots of animations will be displayed. [\[link\]](#)**

## 1.1.3. Basics of Computer File and Database Structures

### Introduction

A goal of GIS is to represent and store the graphic entities of mapped information along with relevant attributes in such a way to make all the data easily retrievable and manipulable. This is done by taking advantage of the ways computers handle data in a logical fashion through file and database structures. A brief overview of the ways computers can handle data is offered here.

### Simple List

In this file structure, there really is no absolute ordering of the data. The data occur in the file in essentially the same way in which they were originally entered. Simple lists may start out in a logical fashion but whenever modifications are made, they rapidly get out of order because new data are appended to the end of the list.

### Ordered sequential files

They can be thought of as a rolodex (rolodex = rolling index: rotating file device used to store business contact information) in which we keep everything in alpha-numeric order. As new data are added, the file is restructured (sorted) to maintain that order.

### Indexed file structures

These provide pointers to more efficiently search data. The most efficient system is to develop an index that is based on a commonly searched attribute in the database, as shown in the following figure. The search is performed on the index field. Once the record is found, the corresponding complete information is accessed through the pointer.

**Only pictures can be viewed in the PDF version! For Flash etc. see online version. Only screenshots of animations will be displayed. [\[link\]](#)**



The software used for management and manipulation of databases is called a database management system (DBMS). The principles of data storage using DBMS are dealt with in the lesson “Data Management” in the basic level.

### 1.1.4. Principles of Spatial Data Access and Search

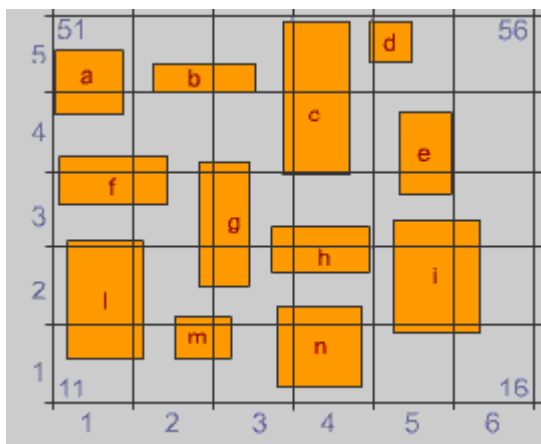
#### Introduction

One of the few important principles governing the searching algorithm is the partitioning of the search space into regions that are usually, but not necessarily, rectangular in shape. Considered simply, this consists of placing data into uniquely identifiable boxes or cells. These methods are characterized as employing spatial indexing because with each block the only information that is stored is whether or not the block is occupied by the spatial object or part of the object.

Jones (1997) distinguishes two types of space decomposition or partitioning: regular decomposition and object-directed decomposition. Here you will find just a short explanation. They will be discussed in an exhaustive manner in the following units.

#### Regular decomposition (space driven indexes)

The space is partitioned in a regular or semi-regular manner that is only indirectly related to the objects in the space (“space primary”). The idea of superimposing a regular pattern of cells over the geometric data to be stored has much in common with the raster model of data storage. The main difference between a regular grid and a raster is that rather than the cells being uniform, equivalent to a pixel, they are compartments capable of storing geometric objects.



Regular decomposition

geom_id	coordinate
a	x1,y1 x2,y2 x3,y3 ...
b	... ..
c	... ..
...	...

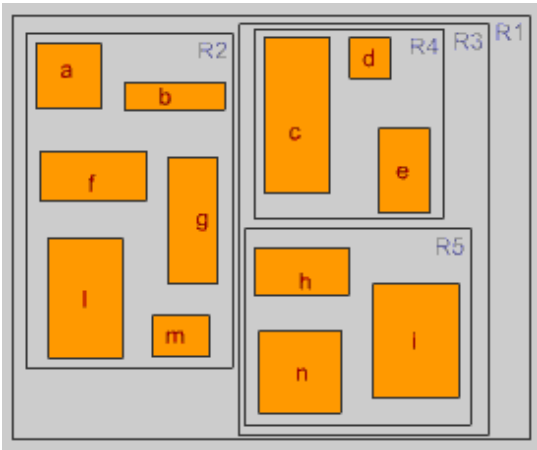
cell_id	geom_ids
11	l
12	m
13	m,n
...	... ..

Regular decomposition

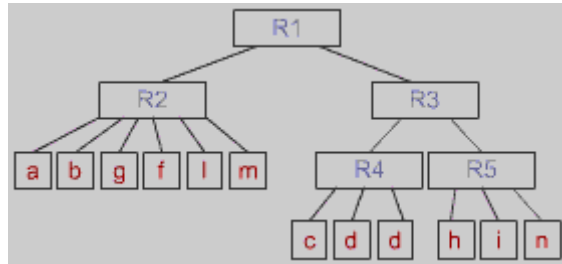
#### Object-directed decomposition (data driven indexes)

The partitioning of the index space is determined directly by the objects (“object primary”). This technique partitions space by means of the coordinates of individual data points or of the extents or bounding rectangles or geometric objects which are to be stored. There is a multiplicity of object-directed decomposition search methods. The most common are:

- Binary tree
- R-tree



*Object-directed decomposition*



*Object-directed decomposition*

# 1.2. Regular Decomposition

## Introduction

Applying the regular decomposition methods, the data space is divided in a regular or semi-regular way. The subdivision of space should be specified and afterwards the object will be addressed in the new structure. The geometry of the object is hence distributed between several adjacent cells (or regions). The objects descriptions are generally kept intact, while the spatial index cells store references to the database locations of the complete objects that intersect them. The data associated with each cell will normally be stored in one or more records, the address of which is given in terms of the coordinates of the lower corner of the cell.

For the regular decomposition of space, cells mainly have three different shapes:

- **Triangle:** convenient for representing approximately spherical surfaces. Triangles have the advantage that they can be regularly subdivided any number of times.
- **Rectangle:** most suitable because its edges can be aligned with the axis of a coordinate system. Rectangles simplify inclusion analysis within rectangular search window.
- **Hexagon:** useful for mapping statistical properties since their neighboring centers are equidistant in all six directions.

### 1.2.1. Regular Grids

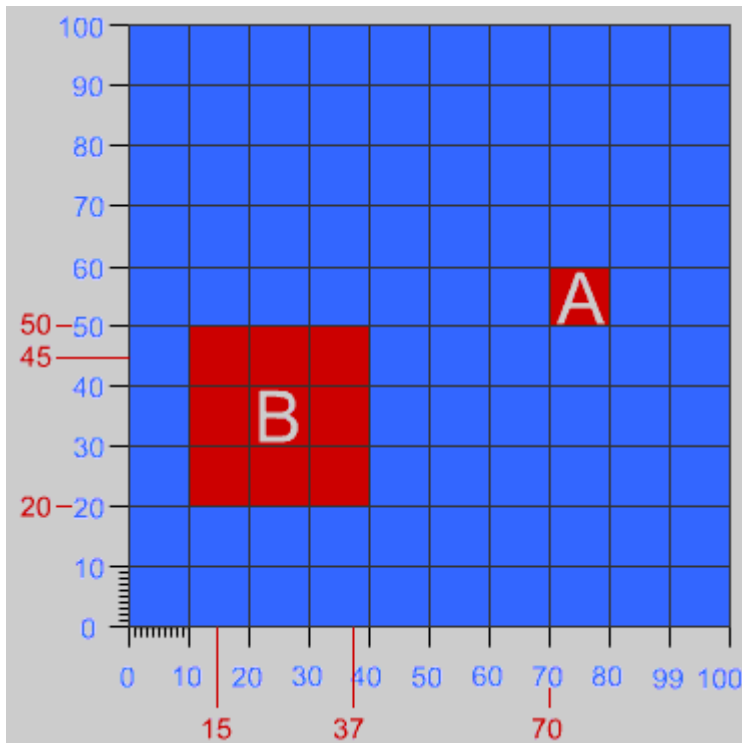
#### Description

In the regular grids decomposition method, the pattern to place on top of our object is a regular grid. Assuming that the x and y coordinates are merged into a single 'composite' number, this could be used as the key for a hashed index or be translated directly into a relative record address of a direct access file.

The choice of the cell size is an important issue when defining the manner to discretize the continuous domain of interest into a regular scheme grid. The content of each cell is stored in one or more records of a file. To avoid wasted space within the record, it is useful to match the quantity of data in the cells to the size of the record or vice versa.

#### An example

We consider a grid extend from 0 to 100 units in each direction and each cell of 10 units square.



Grid 100x100 units

The key  $K$  for a cell  $A$  with coordinates:

- $x = 70$
- $y = 50$

could be:  $K = 75$

Note that the last digit of each value is redundant since cells are 10 units apart.

To retrieve data from a rectangular spatial window ( $B$ ), it is only necessary to derive the address of all cells covering the window.

For the window:

- $x_{\min} = 15$   $x_{\max} = 37$
- $y_{\min} = 20$   $y_{\max} = 45$

The corresponding range of cell addresses would be all those keys whose  $x$  components lay between 10 and 30 and whose  $y$  components were between 20 and 40 (inclusive), the most south-westerly key being the corresponding range of cell addresses would be all those keys whose  $x$  components lay between 10 and 30 and whose  $y$  components were between 20 and 40 (inclusive), the most south-westerly key being  $K = 12$ .

### 1.2.2. Geometry allocation

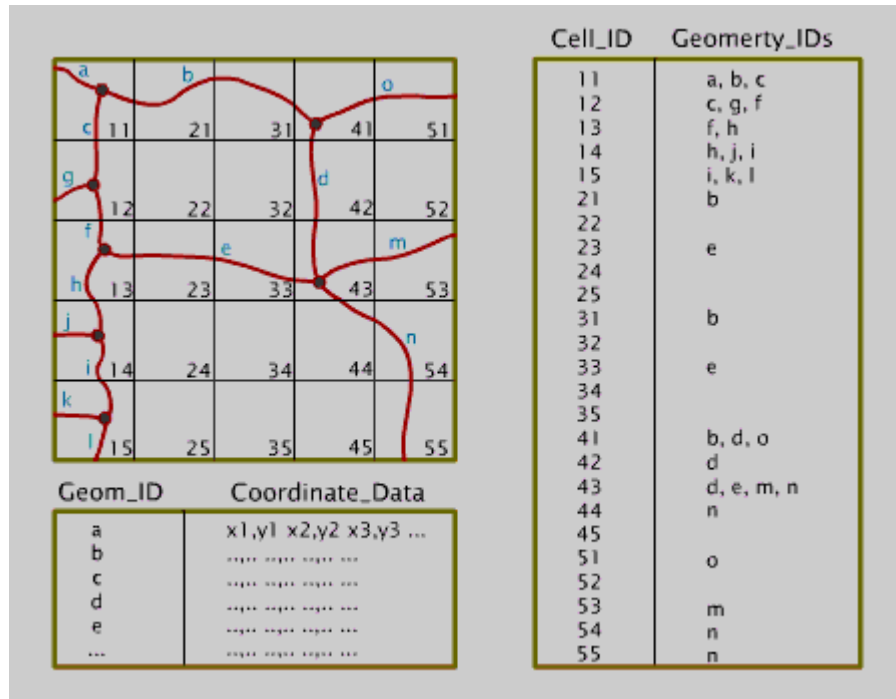
#### Points

Ambiguity exists if a regular grid is used for storing single points, because a point can occupy just a cell. A problem can occur when the point is placed on the boundary of more than one cell. In this case, a rule should be formulated, e.g. doubtful points are placed in the cell immediately above or to the right of the border.

#### Linear geometry

The situation is not so straightforward for allocating linear geometry to grid cells, since lines can frequently cross cell boundaries. One solution to this problem is to cut the line at the cell boundary and to store the

resulting boundary point twice, in both of the cells that share the boundary. This is not in general a satisfactory solution, as it tends to degrade the quality of the data by introducing points that ought to be collinear but are not, due to numerical imprecision of the computer. If linear and polygonal data are not cut at cell boundaries in a regular grid, the data stored in the cells may be references (or pointers) to the storage location of the complete geometric objects (Jones 1997).

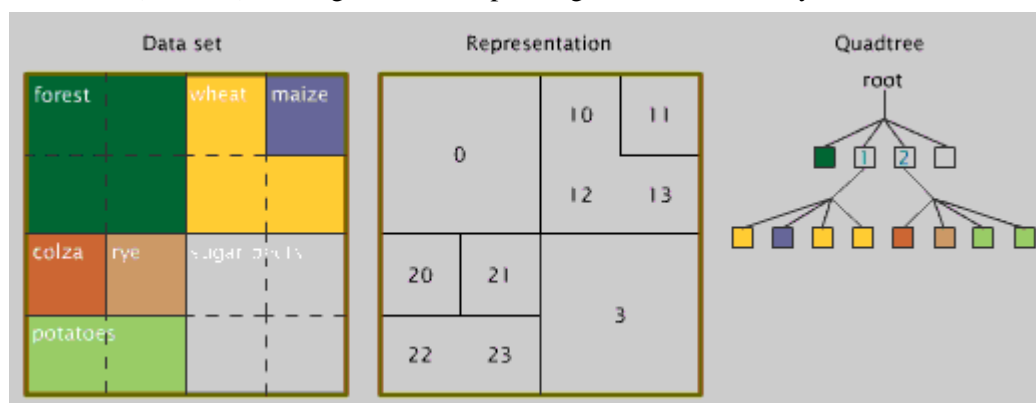


Index table

### 1.2.3. Quadrees

#### Defintion

Quadrees represent a partition of space in two dimensions by decomposing the region into four equal quadrants, subquadrants and so on until the contents of the cells meet some criterion of data occupancy. The resolution (cell size) of the grid varies depending on the data density.

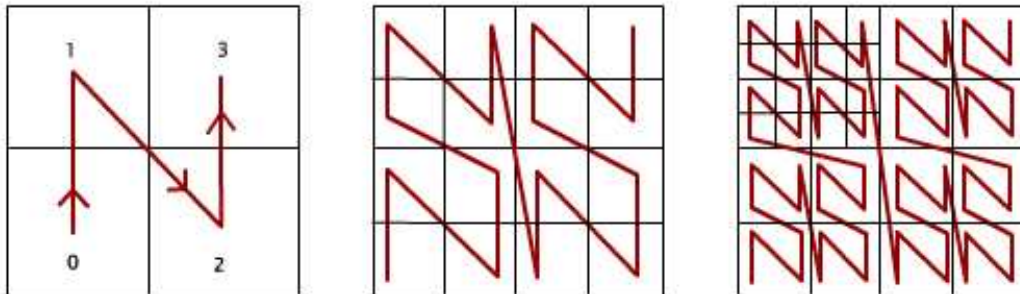


Quadtree indexing

On one side, the quadtree solution is widely used to solve spatial indexing problems. As shown in the next lesson (Data Compression), another application field of this technique is the data compression. The quadtree

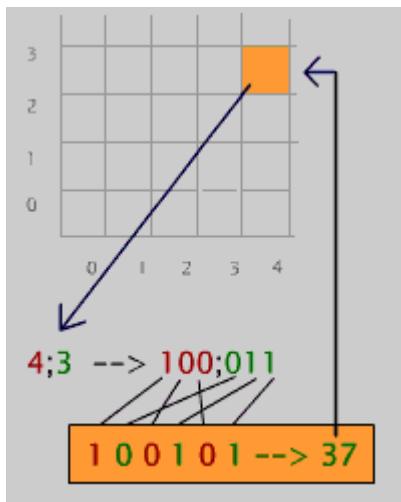
compression technique is the most common compression method applied to raster data.

The pattern of the linear quadtree numbering sequence is that of a **Peano curve**, which is one of a variety of space-filling curves that may be of interest for indexing spatial data, whereby cells that are adjacent in space are more likely to have similar spatial index addresses than in column or row ordering schema. Hence, data that are close in space are close in the storage system.



*Peano curve*

The first use of this particular numbering sequence for spatial indexing is usually attributed to Morton (1966), and it is sometimes referred to variously as Morton sequence, Morton matrix or Morton numbering, while individual addresses may be called **Morton number**. An important property of Morton number is that they can be generated by alternating successive bits of each of the binary representations of the x and y coordinates of the lower left corner of the cell to which they refer. The process is called **bit interleaving** (Jones 1997).



*Bit interleaving process*

In the example above the quadtree address 37 is obtained by two steps:

1. Conversion of the decimal coordinates (4,3) to binary (100, 011).
2. The bit-interleaving process produces the binary number (100101), which converted to decimal is 37.

### An example

Considering a picture as a matrix  $A$  whose dimension is a power of 2, say  $2^n$ , this can be subdivided into four square matrices  $A_0, A_1, A_2, A_3$ , whose dimensions are half of  $A$ . This process can be repeated recursively  $n$  times, until the pixels within a quadrant are all of the same value (homogeneity criterion). The levels can be

numbered, starting with zero for the whole picture, down to  $n$  for the single pixel. A particular square may be labeled with one of the symbols 0, 1, 2, or 3, concatenated to the label of its predecessor square. In this way, single pixels will have labels that are  $n$  characters long. We can express this arrangement as a tree, whose nodes correspond to the squares. Nodes are connected if one of the corresponding squares immediately contains the other. The root of the tree corresponds to the whole picture, the leaves to the single pixels, and all other nodes have down degree 4.

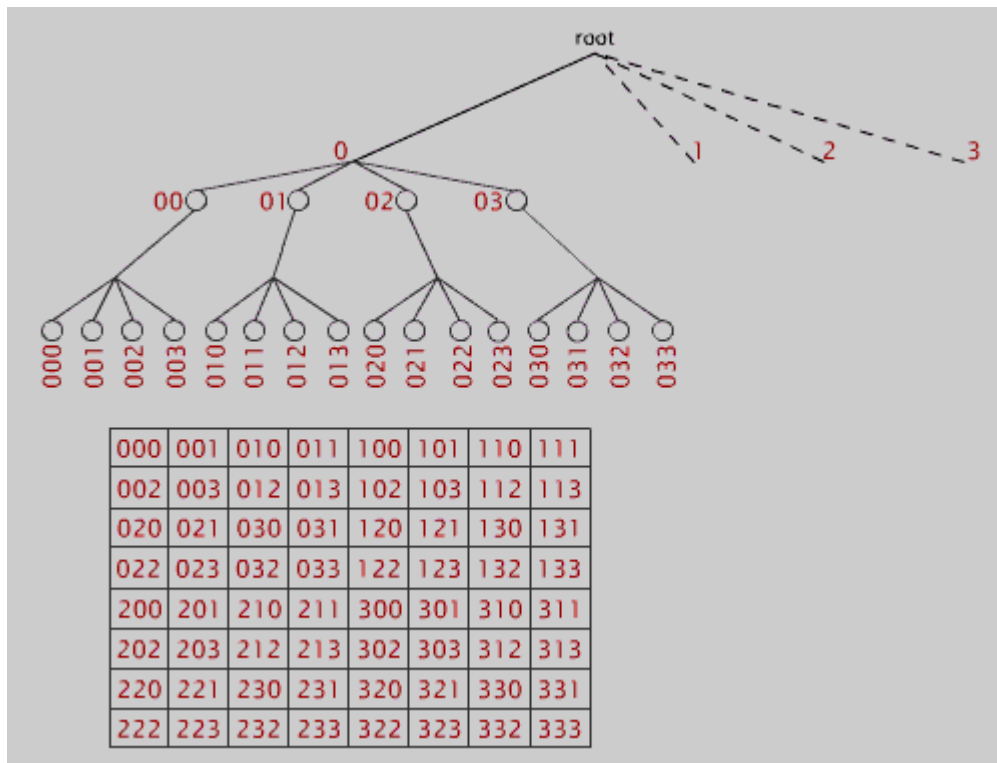
**Only pictures can be viewed in the PDF version! For Flash etc. see online version. Only screenshots of animations will be displayed. [\[link\]](#)**

Since the  $k^{\text{th}}$ -level contains  $4^k$  squares, the tree has a total of:

$$N = \sum_{k=0}^n 4^k = \frac{4^{n+1} - 1}{3} \approx \frac{4}{3} 4^n$$

nodes. Therefore, there are approximately 33% more nodes than pixels.

The following figure shows the addressing notation for a 8x8 picture:



Adressing notation

### 1.2.4. Searching Quadtrees

To search a linear quadtree index, in order to find stored data inside a search window, the window itself may be described in the form of a list of quadtree cells that cover it. It is not necessary for this search list to correspond exactly with the window, provided it covers it entirely. Once stored data cells are found that overlap the search cells, precise comparison can be performed with an exact (vector format) geometric definition of the search window.

### **Excursus: Grid files**

In a grid file the space, of whatever dimension, is divided in a slightly less regular manner, but like a quadtree, adapts to the spatial variation in data density. The cells of a 2D grid are referenced by a 2D grid array, the elements of which store the address of other data records (called buckets) storing the geometry that is inside or intersects the cell. The geographical dimensions of the grid (in 2D) are defined by a set of vertical and horizontal partition lines. The relationship between real-world grid coordinates of the cells and the grid array elements is maintained by 1D arrays called linear scales. The coordinate values of the x-direction grid lines are stored in one 1D array while those of y-direction are stored in another.

A characteristic of the grid file is that a bucket is assumed to be able to store several items of data (actual geometric data, or references to the storage of relevant geometric data) and that several directory cells may reference the same bucket (Nievergelt 1984).



# 1.3. Object-oriented Decomposition

## Introduction

In data driven indexing methods (object-directed decomposition) the objects determine the partitioning of space (e.g. the 2D space containing the lines) into regions called buckets. They are also commonly known as bucketing methods. There are some principal approaches to decomposing the space from which the data is drawn. In one possible approach to object-oriented decomposition, partitioning is achieved by applying divide-and-conquer strategies whereby individual data points or lines may be selected to subdivide the data space into successively smaller half-spaces (Binary-tree). Another approach buckets the data based on the concept of a minimum bounding (or enclosing) rectangle (R-tree).

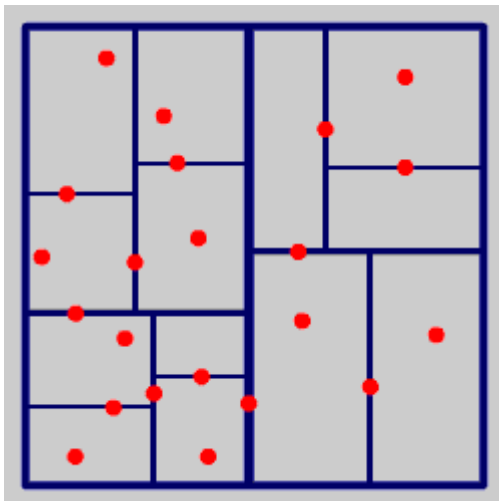
Such strategies generate hierarchical or tree data structures, in which descending down each branch of the tree should result in reducing the relevant volume of data at each stage. The branching factor defines the number of branches at each fork and the number of leaves at the end of each branch. The height of a binary tree is the number of levels within the tree.

In the following section, two main approaches of object-directed decomposition will be presented in order to clarify the principles that drive these methods.

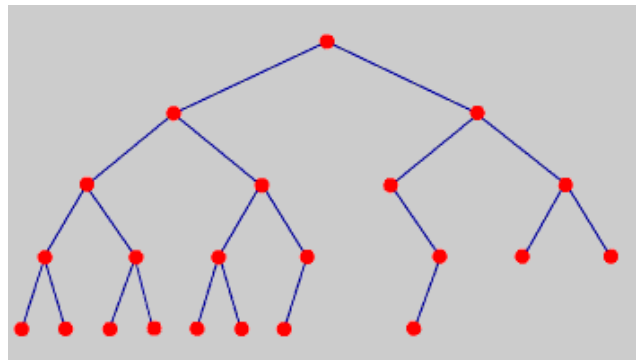
### 1.3.1. Binary Tree

#### Two dimensions



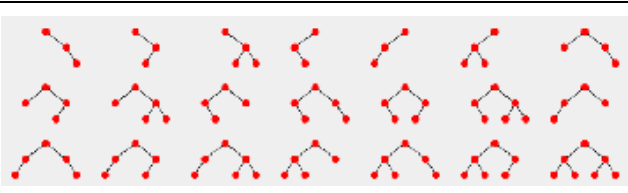
This best known technique applies the principle of divide-and-conquer to the organization and search for point data. This range search technique makes use of a binary tree to order the data with respect to their x and y coordinates. Initially a feature located approximately centrally within the range of x coordinates is chosen to partition the data set vertically into two halves. In each half another feature is chosen in similar manner to partition the halves along horizontal lines passing through these features. The process of splitting stops whenever a new subregion contains no other points.



*k-D-tree*



The branching factor of binary trees is at most 2. In fact every fork has none, one or max. two branches. The numbers of binary trees of height  $h = 1, 2, \dots$  are 1, 3, 21, 651, 457653, ...

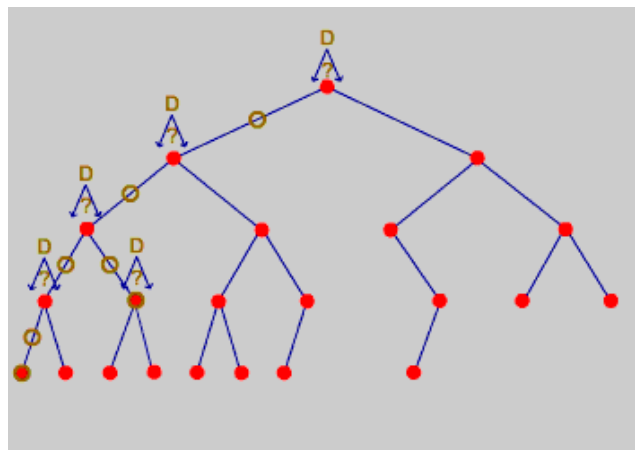
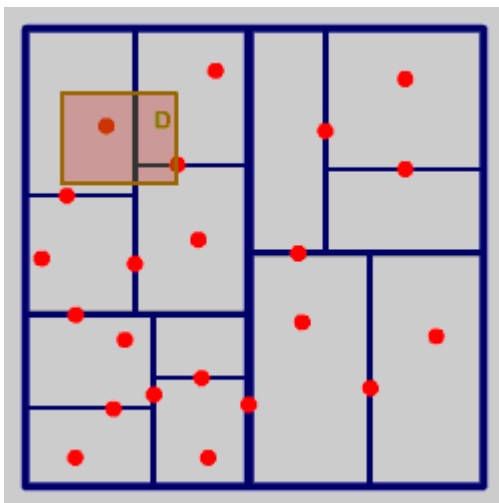
	H = 1 level -> 1 tree
	H = 2 levels -> 3 different trees
	H = 3 levels -> 21 possible trees

### Higher dimensions

The range search approach can be extended into higher dimensionality by considering planes or hyperplanes, which partition the k-dimensional space into two. As the tree is descended, splitting will take place for each dimension in turn. The general tree structure is called k-D tree, standing for k-dimensional binary tree (Jones 1997).

### Binary search scheme

The tree resulting from the recursive splitting of the data space can be searched to determine points that lie inside a search rectangle, for instance named D. Starting at the root, a test is performed to determine whether D lies in one or other of the two regions separated by the point stored in the root node. If D does lie entirely within one side or the other, the corresponding branch of the tree is descended and a similar test is performed with the point in that node. If, however, D is found to cross the partitioning line, a test is performed to find whether the point in the node lies inside the window. If it does, it is saved. The search then continues down the branches of the tree before applying the same logic to each of the two branch nodes. The search terminates at individual nodes when the node is a leaf, i.e. there are no branches to descend.

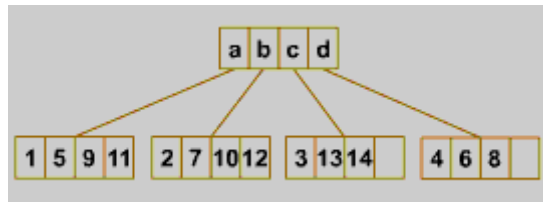
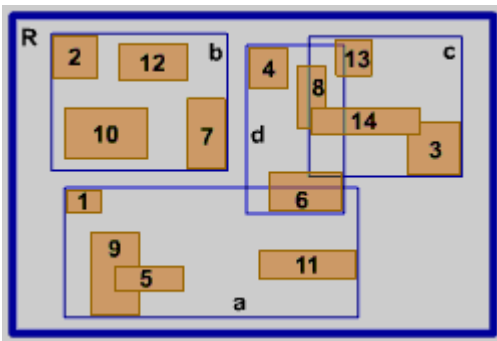


### 1.3.2. R-Trees

#### Description

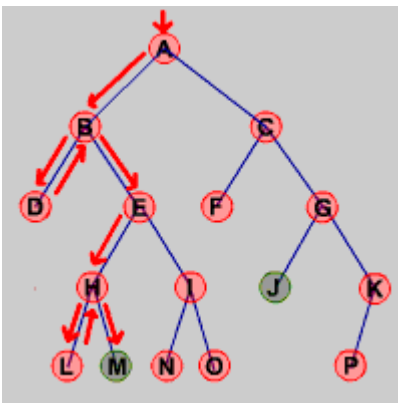
The R-tree is intended for indexing two (and higher) dimensional objects in terms of their minimum bounding rectangles (MBR). Nodes of the tree store MBRs of objects or collections of objects. The leaf nodes of the R-tree store the exact MBRs or bounding boxes of the individual geometric objects, along with

a pointer to the storage location of the contained geometry. All non-leaf nodes store references to several bounding boxes for each of which is a pointer to a lower level node. The tree is constructed hierarchically by grouping the leaf boxes into larger, higher level boxes which may themselves be grouped into even larger boxes at the next higher level. Since the original boxes are never subdivided, a consequence of this approach is that the non-leaf node 'covering boxes' can be expected to overlap each other. Another drawback of this method is that it does result in a disjointed decomposition of space. The problem is that an object is only associated with one bounding rectangle. In the worst case, this means that when we wish to determine which object is associated with a particular point in the two-dimensional space from which the objects are drawn, we may have to search the entire database.



### Searching the R-tree

It consists of comparing the search window with the boxes in each node, starting at the root, and following the child pointers of those boxes that are included in or overlap the ranges of the search window. The procedure is continued, possibly down several branches, until reaching the leaf nodes, the contents of which are then tested against the extent of the search window.



Depth-first searching

A depth-first search (DFS) explores a path all the way to a leaf before backtracking and exploring another path.

For example, after searching A, then B, then D, the search backtracks and tries another path from B.

Node are explored in the order A B D E H L M N I O P C F G J K Q.

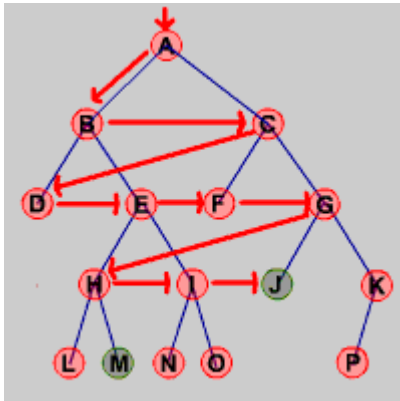
M will be found before J.

A breadth-first search (BFS) explores nodes nearest the root before exploring nodes further away.

For example, after searching A, then B, then C, the search proceeds with D, E, F, G.

Node are explored in the order A B C D E F G H I J K L M N O P Q.

J will be found before M.



Breadth-first searching

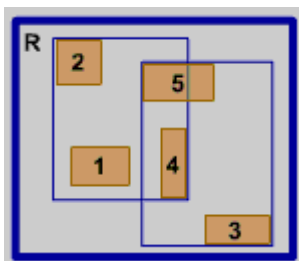
### Variations of R-tree

To cope with the overlapping boxes this method was improved, decomposing the space into disjoint cells, which are mapped into buckets. This method based on disjointness partitions the objects into arbitrary disjoint subobjects and then groups the subobjects in another structure. This data structure is called **R+-tree**. Overlapping of rectangles is avoided by clipping them against each other, creating additional, smaller rectangles. Each object is associated with all the bounding rectangles that it intersects. All bounding rectangles in the tree are non-overlapping (with the exception of the bounding rectangles for the objects at the leaf nodes). The result is that there may be several paths starting at the root to the same object. This may lead to an increase in the height of the tree. However, retrieval time is sped up.

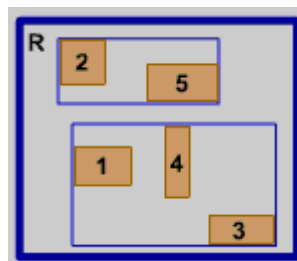
The **R\*-tree** is a variant of the R-tree which makes use of the most complex of the node splitting algorithms. The algorithm differs from the other algorithms as it attempts to reduce both overlap and coverage. In particular, the primary focus is on reducing overlap with ties broken by favoring the splits that reduce the coverage by using the splits that minimize the perimeter of the bounding boxes of the resulting nodes. In addition, when a node A overflows, instead of immediately splitting A, an attempt is made first to see if some of the objects in A could possibly be more suited to being in another node. This is achieved by reinserting a fraction (30% has been found to yield good performance) of these objects in the tree (termed forced reinsertion). The node is only split if it has been found to overflow after reinsertion has taken place. This method is quite complex.

The insertion algorithm has following advantages:

- Minimize node overlap
- Minimize area covered by nodes
- Minimize perimeters of the rectangles at leaf nodes



R-Tree split

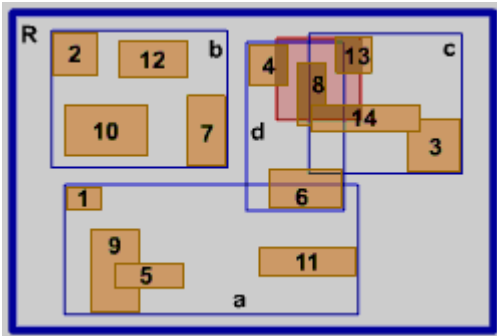


R\*-Tree split

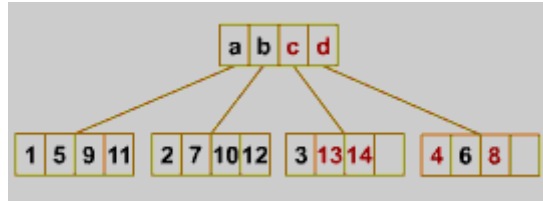
### Operation with R+-Tree

### Searching operation

The idea is to first decompose the search space into disjoint sub-regions and for each of those descend the tree until the actual data objects are found in the leaves.



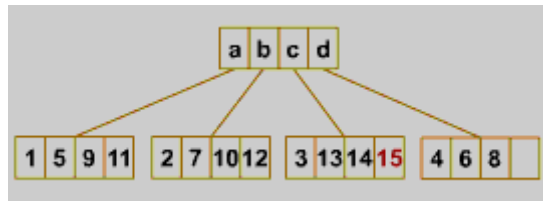
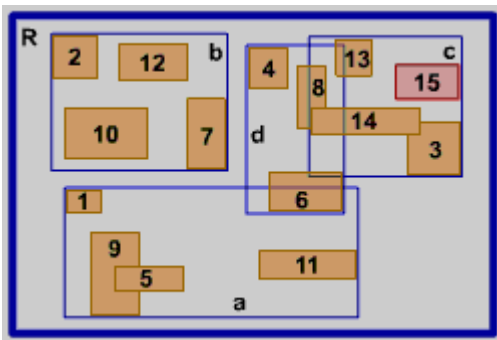
Find all objects in range



Result: objects 13, 14, 4, 8

### Insertion operation

Insertion of a new rectangle in an R+ - tree is done by searching the tree and adding the rectangle in leaf nodes. The difference from R-tree is that the input rectangle may be added to more than one leaf node, the reason is that it may be broken to sub-rectangles along existing partitions of the space.



Add object 15 to leaf C

1. Traverse tree top-down, finding all nodes whose directory root contain object's MBR
2. Node is chosen so that enlargement of rectangles is minimal
3. Repeat until leaf is reached
4. If leaf is not full then add and adjust
5. If leaf is full then a new leaf is created and the objects are split (Split Algorithms)

### Deletion operation

Deletion of a rectangle from an R+ - tree is done as in R-trees by first locating the rectangle(s) that must be deleted and then removing it (them) from the leaf nodes. The reason that more than one rectangle may have to be removed from leaf nodes is that the insertion routine outlined above may introduce more than one copy for a newly inserted rectangle.

### Node Splitting operation

Splitting algorithm is needed to produce two new nodes. We require that the two sub-nodes cover mutually disjoint areas; we search for a "good" partition that will decompose the space into two sub-regions. Contrary to R-tree, downward propagation of the split may be necessary. This is due to that a rectangle R should not

be found in a subtree rooted at a node  $A$  unless the rectangle associated with  $A$  covers  $R$  completely. Hence, nodes intersected by the partitions must be split recursively.

### 1.4. Summary

In this lesson the most often used spatial partitioning and indexing methods were discussed. Using spatial indexing in geodatabases provides fast access to spatial data because only the required part of data is taken into account for spatial analyzing tasks (e.g. give me the result set of buildings which lie inside a specific geometry). The first part of the lesson gave a general remark on spatial object approximation and data access methods. The central point of the module dealt with different aspects of regular decomposition (space driven indexing) and object-oriented decomposition (data driven indexing). Some of the most widely used indexing methods (quadrees, B-tree, R-tree, etc.) were explained.

### 1.5. Bibliography

- **BILL, R.**, 1999. *Grundlagen der Geoinformationssysteme - Band 1*. 4th. Wichmann Verlag.
- **JONES, C.**, 1997. *Geographical Information Systems and Computer Cartography*. Prentice-Hall.
- **NIEVERGELT, J.**, 1984. *The Grid file: an adaptable symmetric multi-key file structure*. Berlin/Heidelberg: Springer.