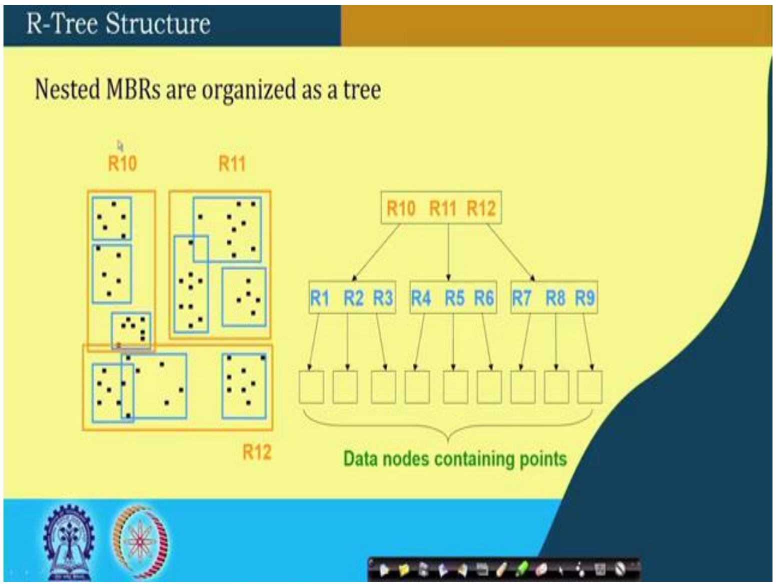
Spatial Informatics

Prof. Soumya K. Ghosh
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 20
Spatial Indexing - IV

Hello. So, we will continue our discussion on Spatial Indexing under our that course on Spatial Informatics and we seen several indexing structure, we will see few more examples or few more aspects of the spatial indexing specially with R-tree and type of stuff and try to see that how it benefits the overall structure right.

(Refer Slide Time: 00:52)



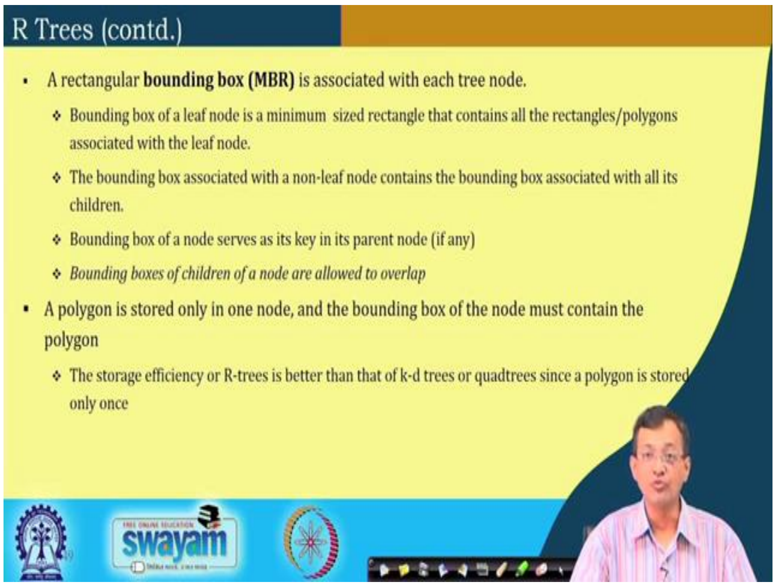
R-Tree Structure

Nested MBRs are organized as a tree

So, little bit some couple of slides, which are overlapping so, to say with our previous slides one is that if this is my overall region of interest. So, there are nested MBRs. So, there is R 10, R 11, R 12, are one level. So, below that there are other levels and type of things right. So, I these are all data points where the data nodes containing the points or it is connecting to the underlining databases right.

So, if I want so, these are contained in this. So, these are contained in these and these are contained in this. So, it is organized in a hierarchical tree root and this several non leaf nodes and the leaf nodes which are which contains the data points right. In this case these are all this particular example some data points. It can be a polygon or line or any type of feature, but what we are interested in the MBR of the things right so, that I quickly search to the things and then in my refinement stage we go more on the more specifically on the geometry of that particular object.

(Refer Slide Time: 02:02)



R Trees (contd.)

- A rectangular **bounding box (MBR)** is associated with each tree node.
  - Bounding box of a leaf node is a minimum sized rectangle that contains all the rectangles/polygons associated with the leaf node.
  - The bounding box associated with a non-leaf node contains the bounding box associated with all its children.
  - Bounding box of a node serves as its key in its parent node (if any)
  - *Bounding boxes of children of a node are allowed to overlap*
- A polygon is stored only in one node, and the bounding box of the node must contain the polygon
  - The storage efficiency or R-trees is better than that of k-d trees or quadtrees since a polygon is stored only once

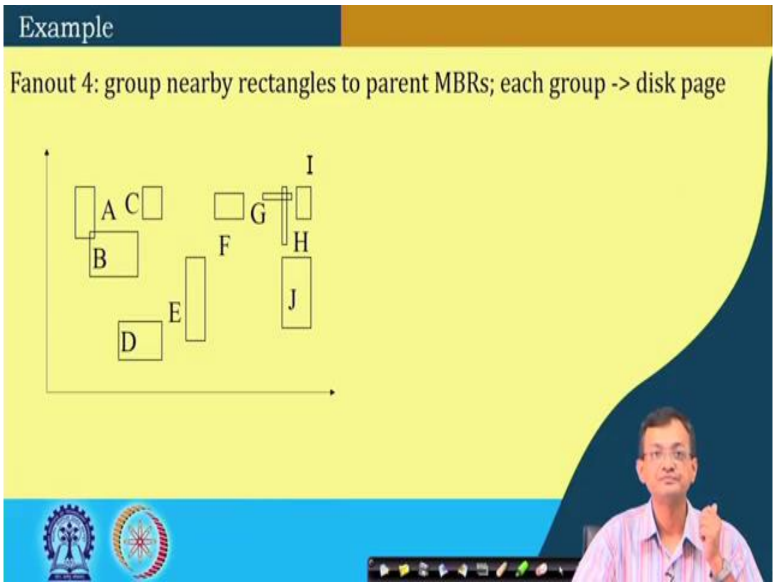
So, also we have seen that MBR is associated with each tree node. So, leaf node minimum size rectangle that contain all the rectangles, all the polygon rectangles or polygon associated with the leaf node alright. The bounding box associated with a non leaf node contains the bounding box associated with all its children so; that means, it is a superset of that bounding box of a nodes serve as a key to the parent node and bounding box of the children allowed to the overlap.

So, children bounding box are allowed to overlap the polygon is stored in only one node. So, while storing if you remember then the I think in our last discussing so, it is stored in one of the node. So, there is a possibility desire from the one upper node we can search more than one node but nevertheless it is stored in one node.

The storage or a R-trees a better than k-d tree or quadtree, since the polygon is stored in one place right. So, if you remember that when we partitioned the quad tree then, that my homogeneous thing is a one of the major aspects right. In doing so long I am not getting a homogeneous quadrant I go on partitioning. So, in doing so the polygon or the my object of interest spatial object of interest may contain in more than one nodes right.

So, then that creates a problem right, that creates a problem in the sense I need to access the thing. So, the efficiency you lose out on the efficiency. Whereas, in this case it is only one in the things right, even if we have seen there are plus tree so this is there, but any of this node will help right any of the node will connect to the things. So, this is a much more so to say, efficient in case of with respect to k-d tree or quad tree type of structure.

(Refer Slide Time: 04:04)

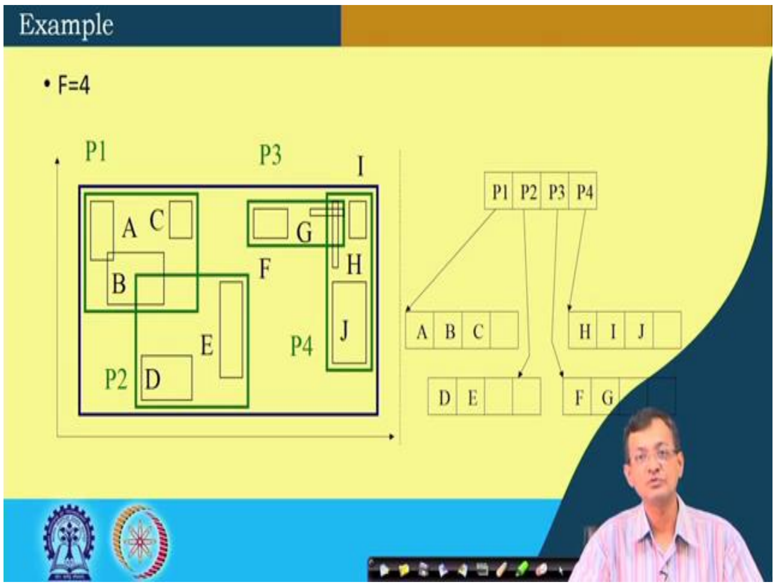


Example

Fanout 4: group nearby rectangles to parent MBRs; each group -> disk page

So, again we have seen that if you have a rectangle to parent MBRs. So, which group map to a disk page.

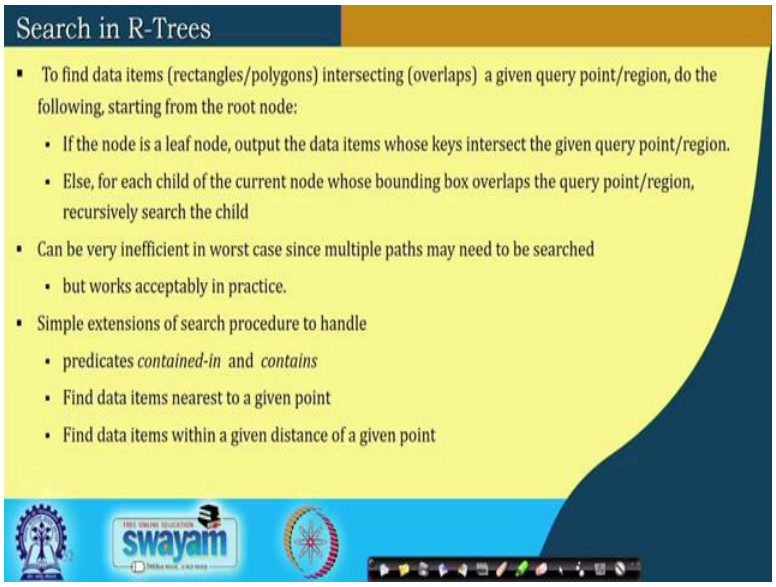
(Refer Slide Time: 04:14)


Example

- F=4

And then we have pan out of four it is stored either I have P 1, P 2, P 3, P 4. So, four here there are four sub node or children; and then P 1 contained A, B, C P 2 D E; P 3 F G and P 4 H I J right. So, this way this is structure. Now, if I search for any node. So, I go to that particular track and search it right. So, that is a advantage a advantage of this sort of structure.

(Refer Slide Time: 04:59)


Search in R-Trees

- To find data items (rectangles/polygons) intersecting (overlaps) a given query point/region, do the following, starting from the root node:
  - If the node is a leaf node, output the data items whose keys intersect the given query point/region.
  - Else, for each child of the current node whose bounding box overlaps the query point/region, recursively search the child
- Can be very inefficient in worst case since multiple paths may need to be searched
  - but works acceptably in practice.
- Simple extensions of search procedure to handle
  - predicates *contained-in* and *contains*
  - Find data items nearest to a given point
  - Find data items within a given distance of a given point

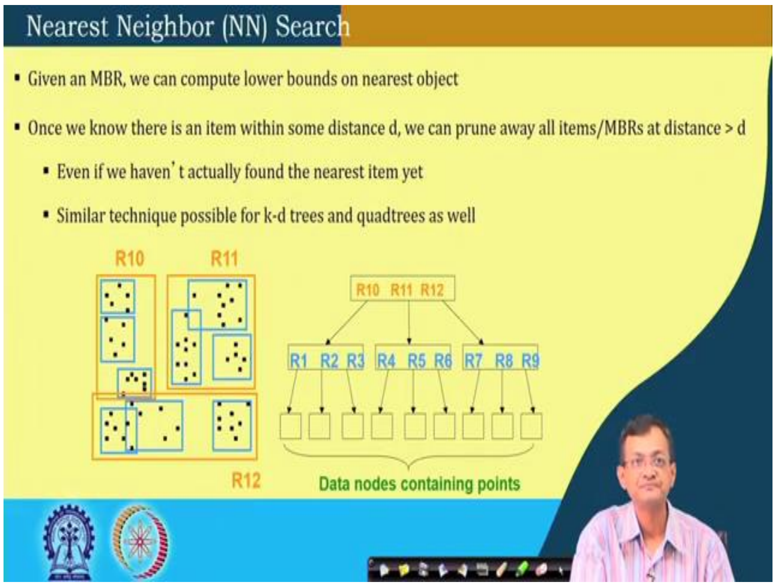
Now, we just looked into the searching to find a data item. So, when we search in our tree. So, to find a data item polygon or rectangle like inserting intersecting overlapping a given point or region we do the following things. If the node is in the leaf node, the output of the data whose keys intersects the query item region right. So, output that data which key is intersect with the things if easily.

If it is a leaf node else for each children in the current node, whose bounding box overlap with the query region recursively search the child. So, one it is; if it is a leaf node you take the key and get that particular region. If it is a non leaf node or where the it is matches in the bounding it is bounding rectangle. Then you go down and search to the recursively search the child so that is the approach right, can be very inefficient in worst-case since multiple paths may need to be searched.

So, it is it will be very inefficient in what is case because multiple paths need to be searched, but works in acceptable in practice right. But usually, it is not like that that it is a worst-case in the sense that at a higher level you have number of branches and then you have to search lot of branches but it is in fact, that it is seen that it is much more efficient then these our other standard structure like quad tree or k-d tree type of things.

Simple extension of the search procedure handle, predicate containing and contains right. So, these sort of things find data item nearest to a given points a finding a data item to a nearest point find data items within given distance of a point right. I have to find a data item which is a given distance from a d distance from a point like searching the number of hospitals from or means health care centre or health medical centers or say bookstore for a from a particular region for a particular region or a particular point within that particular radius or buffer. This type of things can be handled by this right with some simple extension.

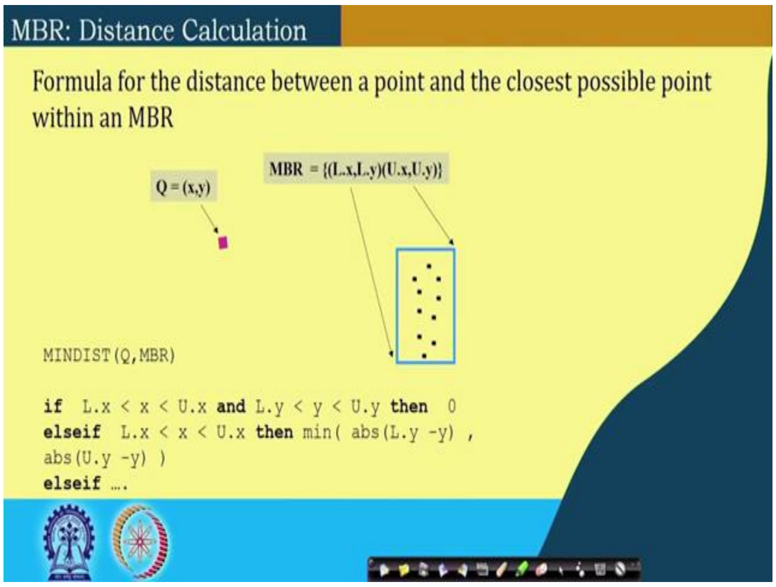
(Refer Slide Time: 07:34)


Nearest Neighbor (NN) Search

- Given an MBR, we can compute lower bounds on nearest object
- Once we know there is an item within some distance d, we can prune away all items/MBRs at distance > d
    - Even if we haven't actually found the nearest item yet
    - Similar technique possible for k-d trees and quadtrees as well

So, like nearest neighbour search, given an MBR we can compute the lower bound of the nearest object right. Once we know that there is there is an item within some distance d; we can prune away all other items or in we are distance greater than d so; that means, I can prune away the items with is there. So, even if you cannot actually found the nearest item say it similar technique possible for k-d tree we can also have type of things right. Here this is my data node what we have what we have shown.

(Refer Slide Time: 08:17)


MBR: Distance Calculation

Formula for the distance between a point and the closest possible point within an MBR

```
MINDIST(Q,MBR)

if  L.x < x < U.x and L.y < y < U.y then  0
elseif  L.x < x < U.x then min( abs(L.y -y) ,
abs(U.y -y) )
elseif ....
```

So, like here we have a we can have an approach type of thing to distance between a point and the closest possible point within the MBR. So, MBR is containing some set of points, either these are the point items or it may be a polygon. So, what is my thing is that find the closest point given a point Q equal to x y find the closest point from a particular MBR or for a from a from possible points within the MBR right.

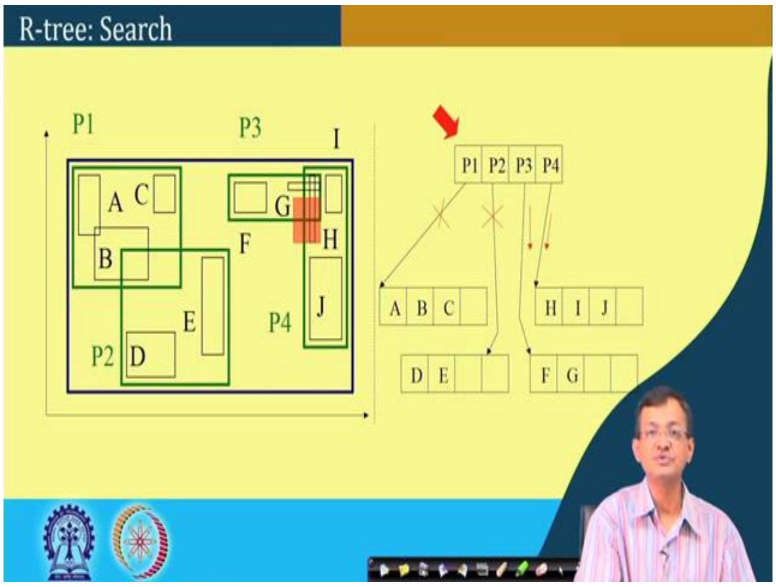
So, these are my possible points and the MBR has been constructed right. Rather, I should say the MBR could have been more finely drawn so, it is it will be ideally this and these type of things so that it is just the minimal that bounding rectangle should be there right. Now, one is that if L L dot x that MBR is L dot x, L dot y and U dot y. So, lower left corner of upper right corner.

So, if we say that L dot x is less than x and U dot x and L dot y is less than y and then it is a distance is 0. In other since I have what I what I mean to say that if this is satisfied is within the MBR it says. So, we say that the there is within the things elseif if this is true then minimal only this is true.

Then, we the minimum of this absolute value of L dot y and minus y and absolute value of y y dot y is this is within the thing right. So, that if it is within the x range, then I just find out this absolute value either from the these or from the top and find out there what is the nearest point right; else if there are other condition if it is a wide range then I go for the things fine.

So, this is a simple we can say formula simple approach to find out that distance between a point and the closest possible point within the MBR right. Now again if you if we recollect or remember so, this is what we are trying to do is mostly the my filtering stage right actual calculation. So, I can boil down that this is the set of point on this is the candidate point then I can have that actual calculation to the things right. So, this helps.

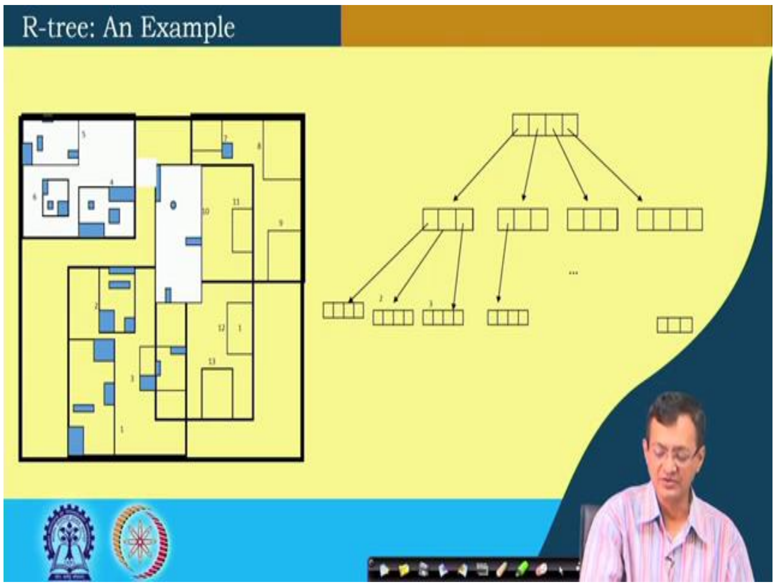
(Refer Slide Time: 10:57)


R-tree: Search

So, there R-tree search. So, if I search item is that particular this box is orange color box then I can have so, it is a candidate where for P 3 F G it is one of the candidate and P 4 F G I there can be candidates, but never the in any case considering this MBRs P 1 and P 2 is nowhere in the things right.

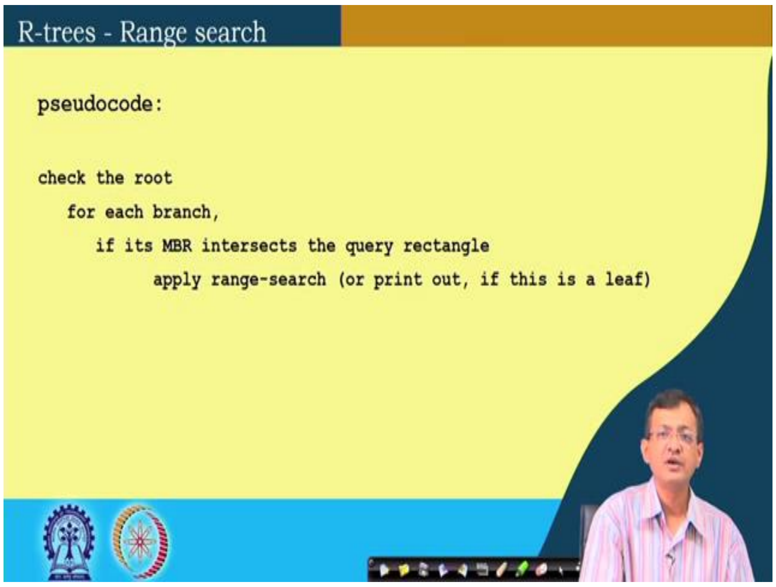
So, from my tree R-tree structure we can find out that these prospective candidates are P 3 and P 2. Rather, in this case if you see we have more or less reduce that 50 percent of the search space right. So, we are considering this P 3 and P 4 and then try to look into the those data points which can be within the search space.

(Refer Slide Time: 12:07)


R-tree: An Example

So, this is another example scenario, where we have that R-tree and the type of structure.

(Refer Slide Time: 12:17)


R-trees - Range search

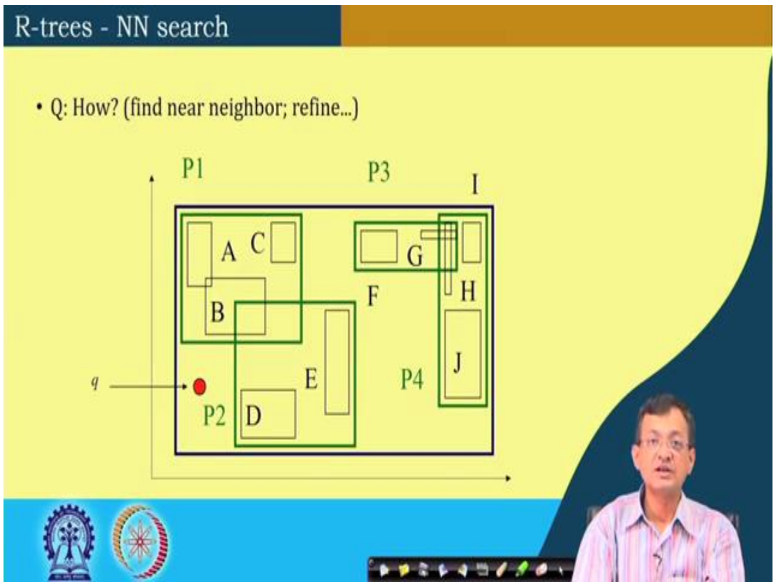
pseudocode:

```
check the root
   for each branch,
      if its MBR intersects the query rectangle
            apply range-search (or print out, if this is a leaf)
```

So, in case of a range search we have code like check the root for each branch, if its MBR intersects with the query rectangle apply range search or print out, this is a if this is a leaf node right. So, that can be a very simple heuristic or approach like we check the root for each of its branch if MBR intersects with the query rectangles. So, each of the branch if it is a MBR rectangles MBR intersects with the rectangles then apply a range search right.

So, again go for that things that which are the other if it has a subtree and a; if it is a non leaf tree it none leaf node then go for there is again subsequently down the hierarchy or if it is a leaf node apply range search right or print out if this is a leaf right. So, apply a range search right. So, this way it goes on looking for those particular range search.

(Refer Slide Time: 13:31)

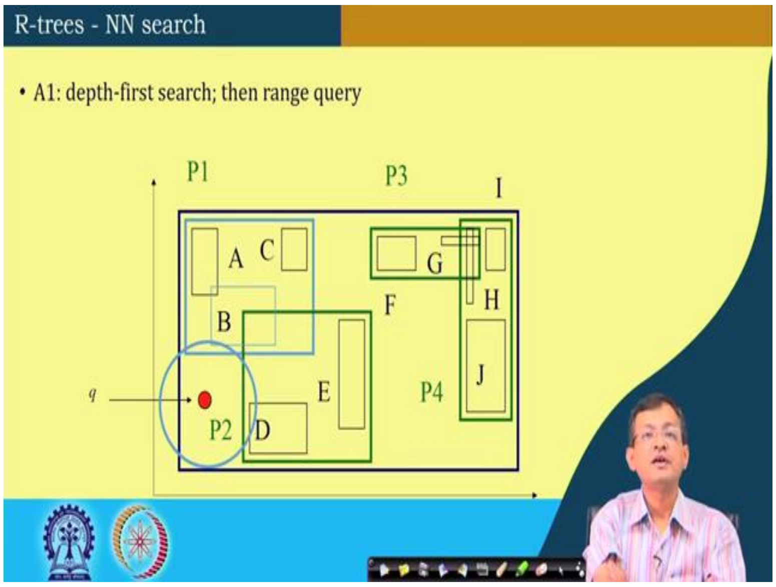


R-trees - NN search

- Q: How? (find near neighbor; refine...)

So, nearest neighbour search like this query point is this q; which is red and I have to find out that this which is the nearest point search. So, find near nearest neighbour or in the refinement stage right.

(Refer Slide Time: 13:55)

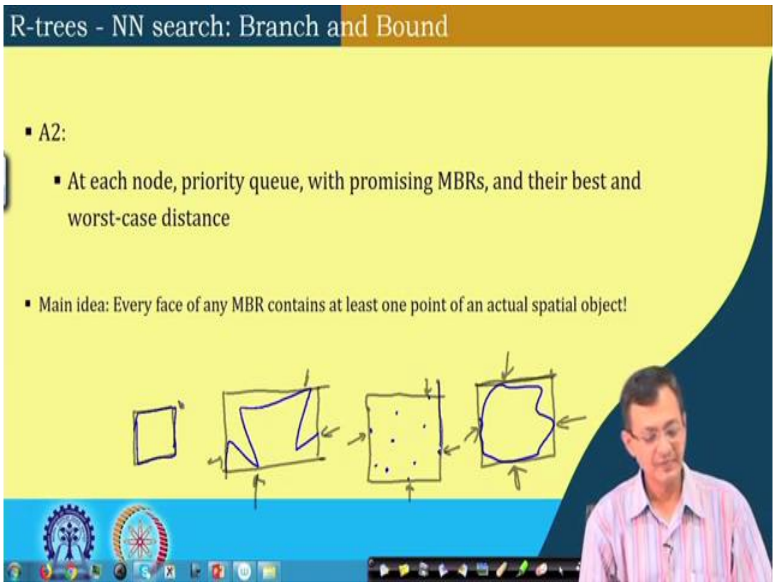


R-trees - NN search

- A1: depth-first search; then range query

So, we have we can have a buffering type of operations and try to see that which are the MBR which overlaps right. So, in this case, the candidate nodes it candidate nodes or the non leaf nodes at P 1 and P 2; and we can thrown out P 3 and P 4 because this is not contributing in the node right.

(Refer Slide Time: 14:23)



R-trees - NN search: Branch and Bound

- A2:
  - At each node, priority queue, with promising MBRs, and their best and worst-case distance
- Main idea: Every face of any MBR contains at least one point of an actual spatial object!

So, we have branch and bound a two at each node priority queue with promising MBRs and best with their best and worst distances right. We can have a priory we can have a calculation done for each node then we can maintain a priority queue with promising MBRs and their best and worst distances.

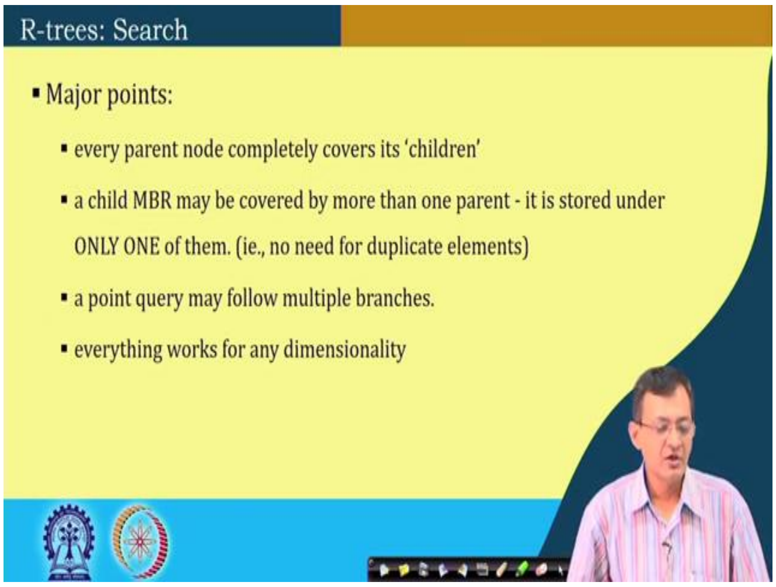
So, the basic idea is that that every face of any MBR contains at least one point of the actual special objects right. See, MBR has been constructed to finding out this minimum rectangle right or bounding rectangle which where. So, every face contain at least one point of that actual object, whether it is a point or line or thing.

So, that it is constructed like this right like because, say I have a say a polygon like this right. So, if I construct that MBR drawing may not be very good sorry, it is touch this right. So, you see any of this point is touching right, otherwise even if it is a sorry if this is a set of even set of points then also if I construct a MBR then it will be also like this sorry right this is the MBR ok.

So, here also at least you see at least some points like this point, this point, this point, this point at least one of the things that at least one point in the actual. Here these, these, these, these or one it may be more than one. Even if you if you consider a any irregular a any polyline, then also we have sorry this is may be little right.

Here also if you see these, these, these, these, these are the points which are touching one or more points are touching so; that means, it is actually any of the surfaces are having at least one if not more. It can be a whole rectangle per se right and then, if you construct the MBR then it will be right same thing. So, it can be like this right.

(Refer Slide Time: 17:48)



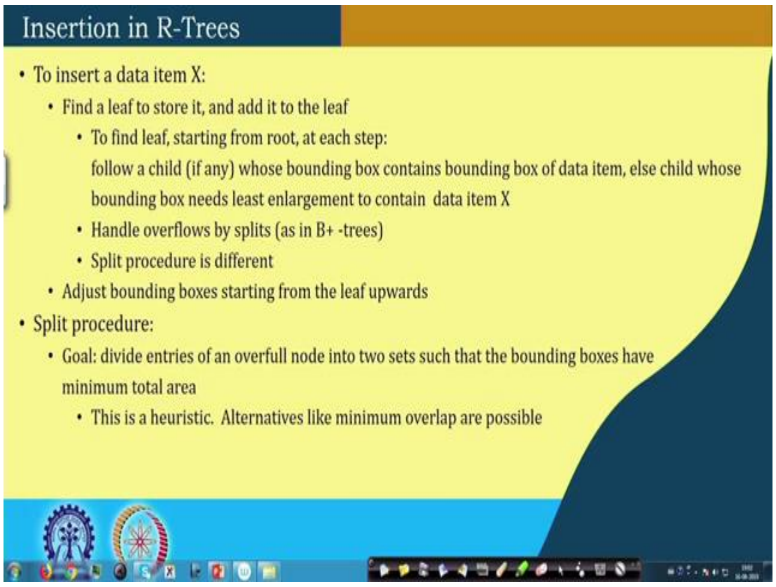
R-trees: Search

- Major points:
  - every parent node completely covers its 'children'
  - a child MBR may be covered by more than one parent - it is stored under ONLY ONE of them. (ie., no need for duplicate elements)
  - a point query may follow multiple branches.
  - everything works for any dimensionality

So, if you look at the salient points, every parent node completely covers its children right this one a child MBR may be covered by more than one parents. So, it is stored under only one of them right, it maybe more than one parents, but stored under one of them in under R-tree construction. So, there is no need to duplicate elements otherwise you have to duplicate element.

That may if it is being some efficiency in the searching because one of the branch you could have gone and got that thing. But in this case it is the duplication is avoided a point in the query may follow multiple branches right. A everything works for a any this works for any dimensionality of the data set.

(Refer Slide Time: 18:37)



Insertion in R-Trees

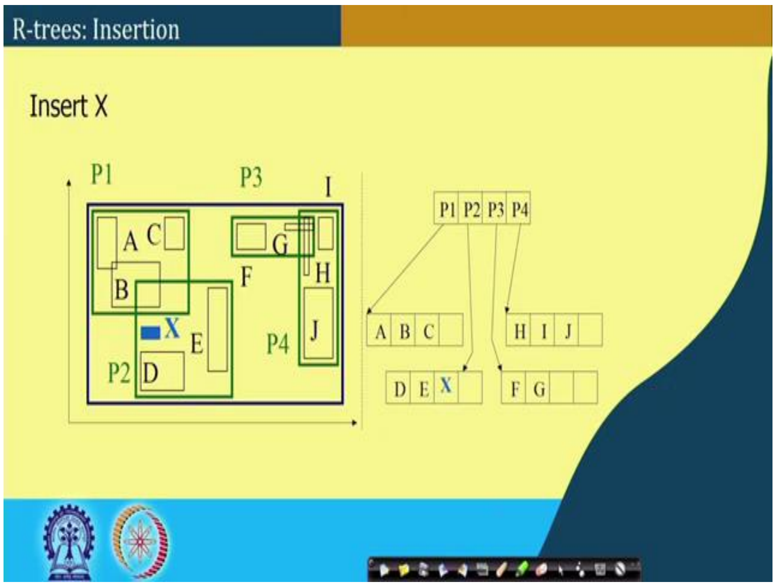
- To insert a data item X:
  - Find a leaf to store it, and add it to the leaf
    - To find leaf, starting from root, at each step:
      follow a child (if any) whose bounding box contains bounding box of data item, else child whose bounding box needs least enlargement to contain data item X
    - Handle overflows by splits (as in B+ -trees)
    - Split procedure is different
  - Adjust bounding boxes starting from the leaf upwards
- Split procedure:
  - Goal: divide entries of an overfull node into two sets such that the bounding boxes have minimum total area
    - This is a heuristic. Alternatives like minimum overlap are possible

So, if we look at a insertion of a node in R-tree. So, to insert a data node item in say X; find the leaf to store and add it to the leaf right, to find a leaf start from the root node start from the root node and look at the 1 minute. So, start from the to find a leaf starting from the root node for each step follow the child, if the bounding box contains the thing fine otherwise it needs to enlargement of the item X right.

Handle overflow by split as in case of a B plus tree right, if you remember in the B plus tree also if it is overflow of the things then we split it. So, here the split of operation is little different, but nevertheless, we can split the things we are not going to that much nitty gritty of the things, but we can split as we can do any case of a tree. So, then adjust the bounding boxes starting from the leaf node right.

So, split procedure goal divide the entries of an over full node, into two set such that the bounding boxes have minimal total area. So over full node into two sets, so that the bounding boxes as the minimal total area right so that I can split in. So, there is a definitely it is a heuristic alternatively, like minimum overlap are possible and type of things.

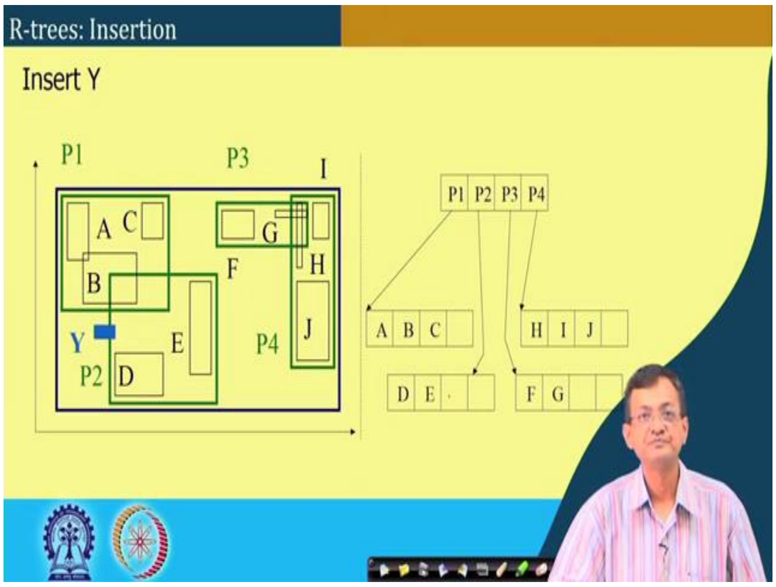
(Refer Slide Time: 20:17)


R-trees: Insertion

Insert X

Like here if I want to insert X right so, this will go under P 2 right it is a clear case. So, I go it goes under P 2 right. So, particular these insertion so, that in the P 2, we have this extra node thing extra that here that insert node that X is inserted right.

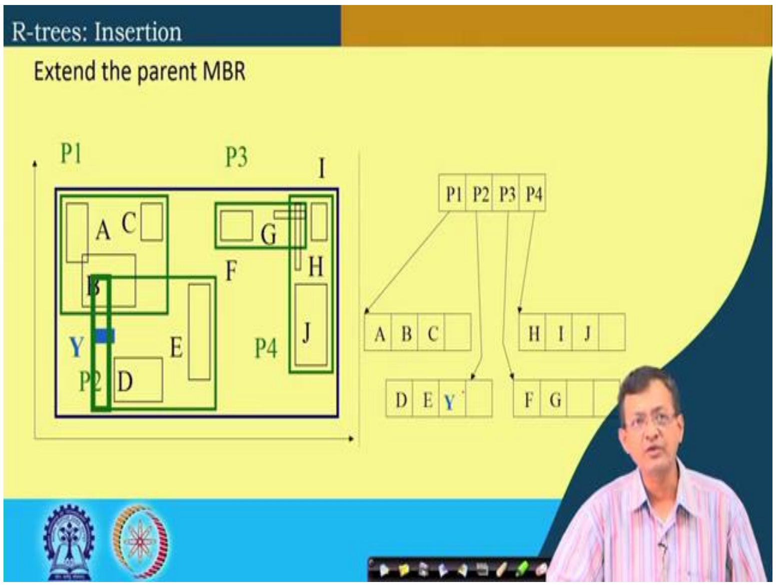
(Refer Slide Time: 20:46)


R-trees: Insertion

Insert Y

So, if it is this sort of scenario where it is going out of P 2 there is Y.

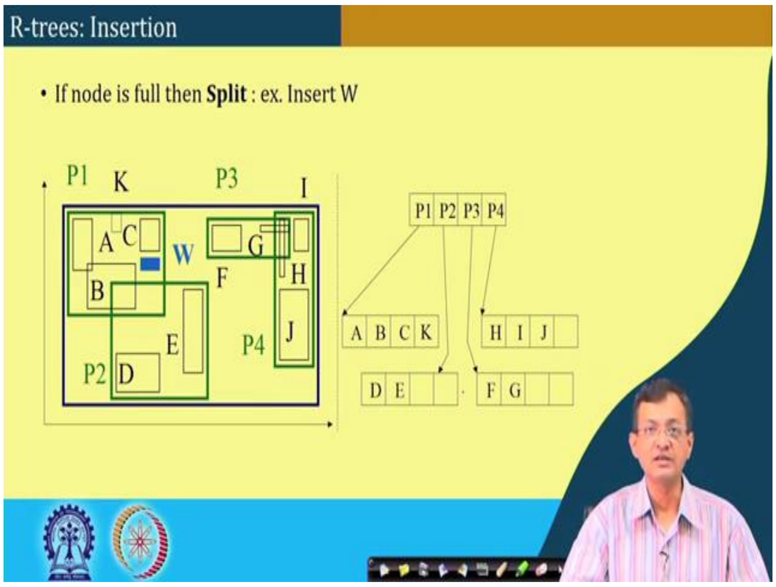
(Refer Slide Time: 20:58)


R-trees: Insertion

Extend the parent MBR

Then, then we can put this inside the P 2 and we can extend this MBR to accommodate Y right. So, extend the parent MBR to accommodate this new inserted things right.

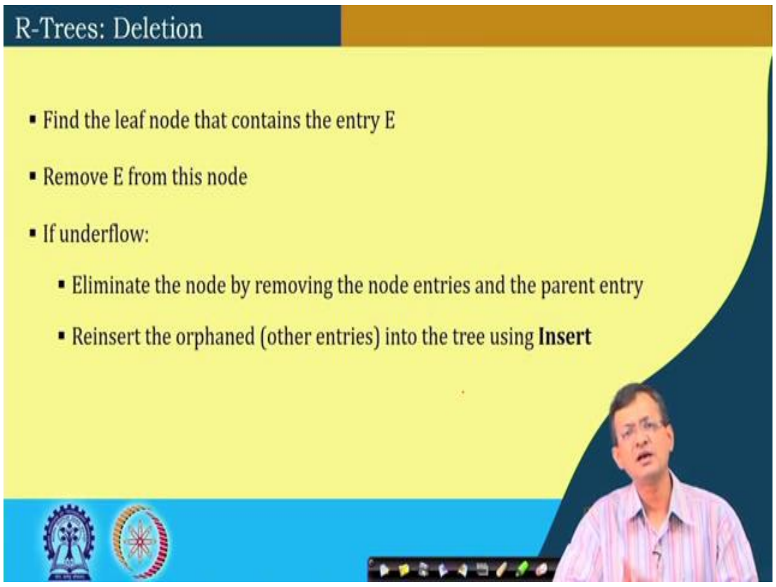
(Refer Slide Time: 21:24)


R-trees: Insertion

- If node is full then **Split** : ex. Insert W

So, if it is totally outside then I can increase any of the MBR based on the based on some heuristic and type of thing. If the if the node is full like if this is ABC and then the new node comes into the thing; and if the k is there then the new node W, it is full then we may have to split the node right. So, that split the node into to another structure. So, I eventually I can I can go on splitting those nodes into structure they in to different thing as we done in the B plus tree.

(Refer Slide Time: 21:59)



R-Trees: Deletion

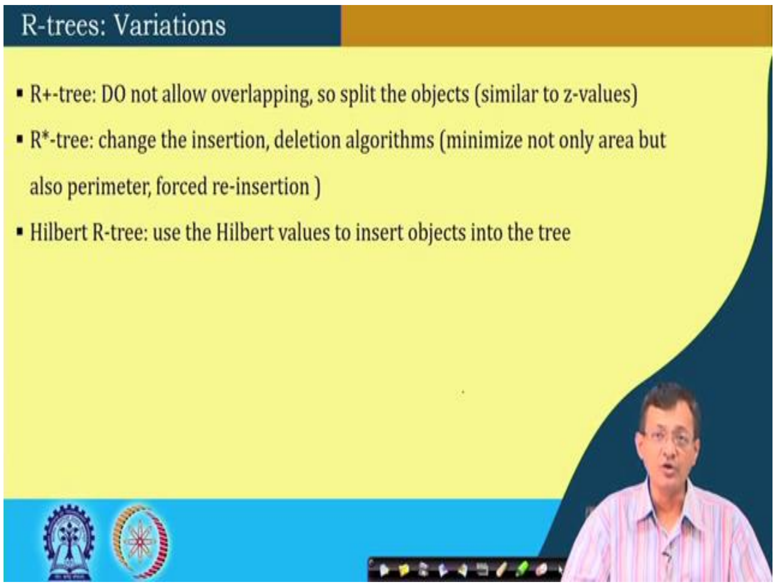
- Find the leaf node that contains the entry E
- Remove E from this node
- If underflow:
  - Eliminate the node by removing the node entries and the parent entry
  - Reinsert the orphaned (other entries) into the tree using **Insert**

So, this is one of insertion one is that deletion, find a node if the content entry are if I want to delete sorry. If I want to delete a entry E, remove E from this node if it is under full elimination of the node by removing the node entries of the parent entry reinsert the orphan or other entries into the tree using insert right so; that means, if the node is only 1.

Then if it is under full right over full in case of insert then I have to reorganized the thing in other thing that I can call the insert and that who have no parent no those nodes can be inserted again. So, these are typically features of R-tree which helps in better indexing definitely and also what we are what we can see that it gives a better efficiency in searching appearance.

Because one of the basic goal is that every object that every polygon or any other objects it is under in under one node only. So, that way that way there is like in quadtree etcetera there is multiple nodes I do not have to look into those things. But it may happen that the search may go in different directions and finally, where end up in one node.

(Refer Slide Time: 23:31)



R-trees: Variations

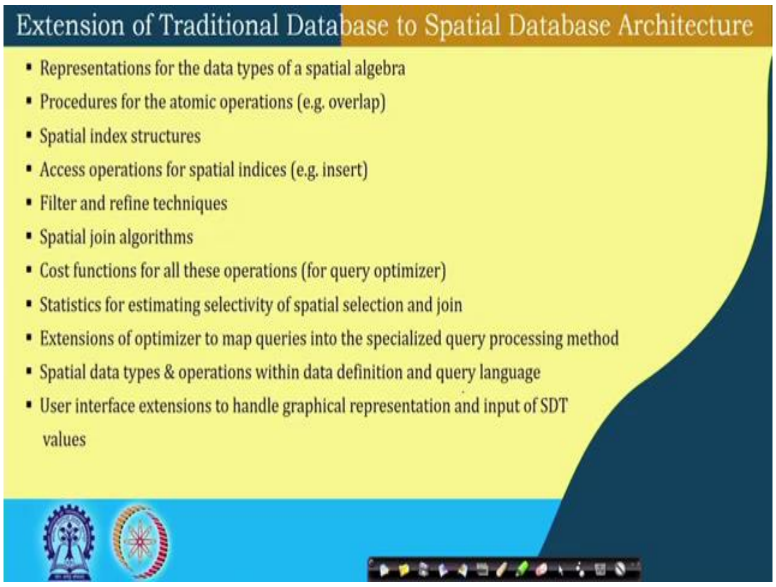
- R+-tree: DO not allow overlapping, so split the objects (similar to z-values)
- R*-tree: change the insertion, deletion algorithms (minimize not only area but also perimeter, forced re-insertion)
- Hilbert R-tree: use the Hilbert values to insert objects into the tree

So, there are various at one various and we have seen that R plus tree do not allow overlapping. So, split the object similar to z-values type of things we have seen that while starting this are and R plus tree we just in the last lecture. We have seen there is there is a thing not overlapping things but it allows exist in multiple leaf nodes.

In other sense it can be it can be duplicated, but nevertheless the that helps in only one access out of those it will access rate is there but it is a variant of the things. R star tree change the inserts and deletion algorithm, minimize not only the area but also the perimeter force reinsertion and type of things.

So, there is a variation of the things there is a Hilbert R-tree using Hilbert value to insert objects into the tree. So, while inserting objects it takes the Hilbert value right, if you remember Hilbert curve was a space filling curve taking those index value while inserting into the objects into the thing so, that is considered as the values to be means for constructing the tree.

(Refer Slide Time: 24:59)



Extension of Traditional Database to Spatial Database Architecture

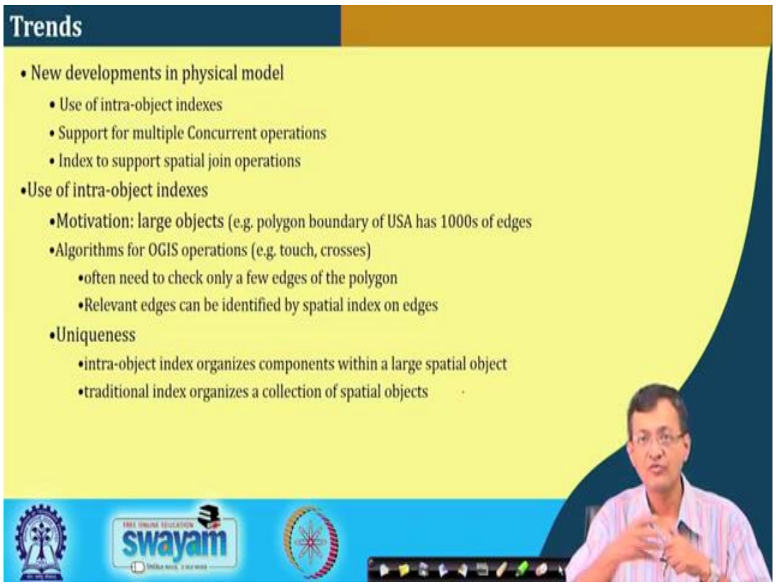
- Representations for the data types of a spatial algebra
- Procedures for the atomic operations (e.g. overlap)
- Spatial index structures
- Access operations for spatial indices (e.g. insert)
- Filter and refine techniques
- Spatial join algorithms
- Cost functions for all these operations (for query optimizer)
- Statistics for estimating selectivity of spatial selection and join
- Extensions of optimizer to map queries into the specialized query processing method
- Spatial data types & operations within data definition and query language
- User interface extensions to handle graphical representation and input of SDT values

So, few just few more aspects what we try to look at ah, one is this extensive list additional database two spatial database architecture is one of our one of the goal. Because, we cannot say we do not want to change the whole we cannot change the whole database is a established thing.

So, representation of other data types of a spatial algebra is one of the aspects. Procedure for atomic operations for like operations like overlap intersects and type of things. Spatial index structure what we discussed, access operations for spatial index is like insert operation and type of things filter and refined techniques. So, boiling down the search space I means what we say that reducing the search space.

And then refined actual objects there are several spatial joint applications cost function for the spatial operations statistics for estimating selective data selection and join and so and so forth. So, one of the major aspects what we see that, a good indexing structure may help in increasing or achieving better efficiency and R-tree is one of the major such index.

(Refer Slide Time: 26:13)



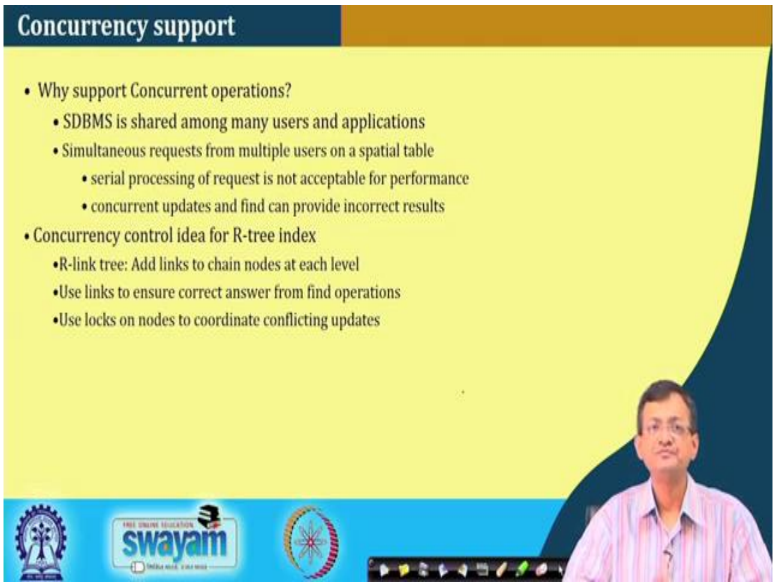
Trends

- New developments in physical model
  - Use of intra-object indexes
  - Support for multiple Concurrent operations
  - Index to support spatial join operations
- Use of intra-object indexes
  - Motivation: large objects (e.g. polygon boundary of USA has 1000s of edges
  - Algorithms for OGIS operations (e.g. touch, crosses)
    - often need to check only a few edges of the polygon
    - Relevant edges can be identified by spatial index on edges
  - Uniqueness
    - intra-object index organizes components within a large spatial object
    - traditional index organizes a collection of spatial objects

So, there are few trends new development in physical models. Use of ultra object in intra-object in indexing within the object indexing things; support of multiple concurrent operation. So, number of things are concurrently access those supports index to support spatial join operation so, that is another index to support spatial join operation.

Use of intra-object indexes; so, motivation large objects that is polygon boundary of USA say there are 10000 edges so, intra-object indexes are required because, that is a huge number of things. So, there are specification of algorithms for open geospatial or operations like touch cross. So, uniqueness intra-object index organizes components into large spatial object traditional index organizes a collection of spatial objects right. So, there are way of handling the things.

(Refer Slide Time: 27:21)



Concurrency support

- Why support Concurrent operations?
  - SDBMS is shared among many users and applications
  - Simultaneous requests from multiple users on a spatial table
    - serial processing of request is not acceptable for performance
    - concurrent updates and find can provide incorrect results
- Concurrency control idea for R-tree index
  - R-link tree: Add links to chain nodes at each level
  - Use links to ensure correct answer from find operations
  - Use locks on nodes to coordinate conflicting updates

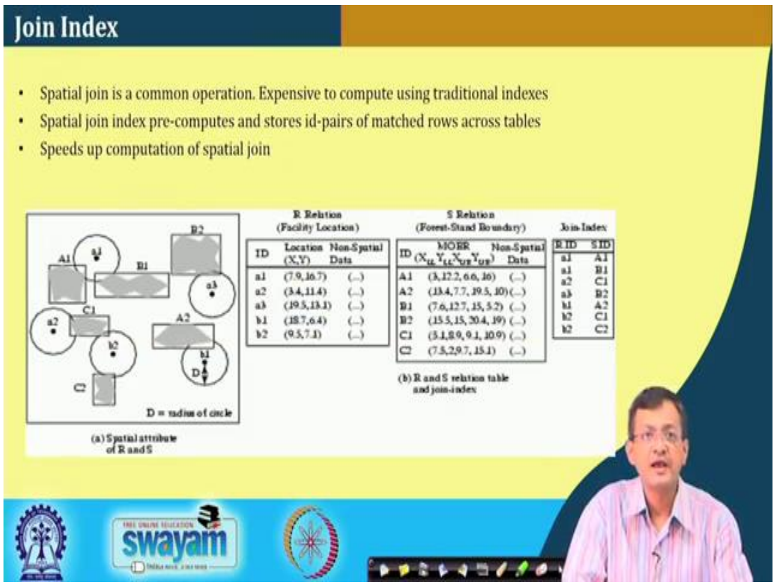
Concurrency support is a one another major feature, that is that concurrent axis of the things concurrent updates of the things right. So, there is read is fine, but whenever you have a read write type of conflict then there are concurrency control or concurrency control mechanism should be put in place put into place. This type of mechanism are pretty standardized and operationally in case of traditional database systems.

So, we are we could do we have a I means those things are very standardized. So, why support concurrent operation spatial DBMS is shared among many users and applications right we several time we discussed. So, it is a data set which is being shared by among various users and applications simultaneous requests from multiple users on the spatial table right serial processing of the request is not acceptable for performance concurrent updates finds provides incorrect results right.

So, there can be a concurrent updates of the things and that may end up in incorrect result or inconsistency in the things. So, concurrency control for R-tree indexes. So, R-tree link add links to chain nodes in each level. So, there is a what we say sort of a horizontal movement at the thing. Use link to ensure correct answer from the find operation use locks to node contribute no coordinate for conflict updates.

So, if there is a conflicting update then the lock operation is there. So, if there is a find operation this links this horizontal links can be used to have better efficiency right or finding this correctness of the things right. So, this is something on the concurrency control aspects.

(Refer Slide Time: 29:35)



Join Index

- Spatial join is a common operation. Expensive to compute using traditional indexes
- Spatial join index pre-computes and stores id-pairs of matched rows across tables
- Speeds up computation of spatial join

(a) Spatial attribute of R and S

R Relation
(Facility Location)

ID	Location (X,Y)	Non-Spatial Data
a1	(7.9,16.7)	(...)
a2	(3.4,11.4)	(...)
a3	(19.5,13.1)	(...)
b1	(18.7,6.4)	(...)
b2	(9.5,7.1)	(...)

S Relation
(Forest-Stand Boundary)

ID	MOBR $(X_{LL}, Y_{LL}, X_{UR}, Y_{UR})$	Non-Spatial Data
A1	(3,12.2,6.6,16)	(...)
A2	(13.4,7.7,19.5,10)	(...)
B1	(7.6,12.7,15,5.2)	(...)
B2	(15.5,15,20.4,19)	(...)
C1	(5.1,8.9,9.1,10.9)	(...)
C2	(7.5,2,9.7,15.1)	(...)

Join-Index

R ID	S ID
a1	A1
a1	B1
a2	C1
a3	B2
b1	A2
b2	C1
b2	C2

(b) R and S relation table and join-index

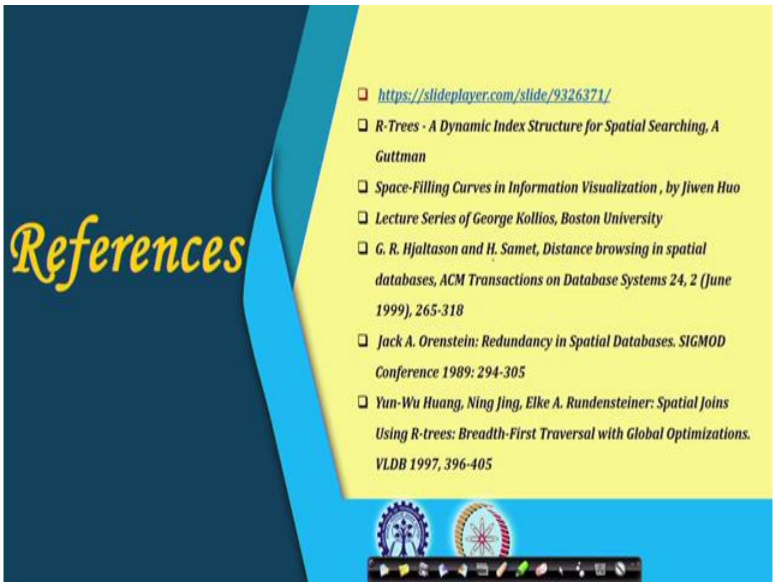
And finally, we have a challenge of join index right. So, several operations looks for joining to separate tables like we have seen in case of traditional databases. Here also tables with spatial context need to be joined together. So, spatial join is a common operation several queries requires these join expensive to compute using traditional indexes right.

We cannot compute with the traditional indexes right spatial join pre-computes and stored id-pairs of matched rows along the tables right. So, speed up operations of spatial join. So, what happened in number of cases as the data sets are there if we have a priori, some knowledge about that what sort of data queries etcetera are spatial joins will be there or I can have multiple join indexes.

So, I can create a join indexes right a 1, A 1; a 1; B 1 and type of things. So, this can be done pre compute. So, I pre metalize of pre compute these things. So, whenever this type of join requests come it will consult that particular table handles. So, this type of things I can have several such instances of these things right. So, based on the join request we can have different spatial join things right.

So, this is one aspects of the things which need to be look because is a first of all it is a costly affair costly in that computationally costly affair and secondly, there is a much needs and when multiple different queries need to execute this join operation. So, there is a need other things so, this is a spatial joins can help.

(Refer Slide Time: 31:26)



References

- https://slideplayer.com/slide/9326371/
- R-Trees - A Dynamic Index Structure for Spatial Searching, A Guttman
- Space-Filling Curves in Information Visualization , by Jiwen Huo
- Lecture Series of George Kollios, Boston University
- G. R. Hjaltason and H. Samet, Distance browsing in spatial databases, ACM Transactions on Database Systems 24, 2 (June 1999), 265-318
- Jack A. Orenstein: Redundancy in Spatial Databases. SIGMOD Conference 1989: 294-305
- Yun-Wu Huang, Ning Jing, Elke A. Rundensteiner: Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations. VLDB 1997, 396-405

Let us conclude before that I should mention that these are the set of references, what we constructed; what we have consulted here along with book of browser searching cycle which is which is also used for these several things. So, what we have seen in this particular indexing structure. First of all index is a one of the major aspects of any database operations.

Like a proper index in helping inefficient things whether it is a traditional our non spatial data or even spatial data sets. The spatial data by nature of the things is difficult to is it cannot be put in the traditional way I cannot put b people are still into the things. So, what we have seen there are several approaches? One is the space filling crafts what we are trying to looked at there are approaches of k-d tree or mostly this quad tree and type of approaches.

Also we have seen approaches for things like that tree R-tree and variation of R-trees, which help in better or efficient processing of the queries. So, with this let us conclude our discussion today; we will continue with different other aspects of spatial informatics in our subsequent lectures.

Thank you.