R-TREE IMPLEMENTATION OF IMAGE DATABASES

Chandresh Pratap Singh

Department of Computer Science & Engineering, JUIT, Waknaghat, H.P, INDIA

chandresh_pratap_singh@hotmail.com

ABSTRACT

With the onslaught of multimedia in the near past, there has been a tremendous increase in the uses of images. A very good example of which is the web on which most of the documents contain images. Other than this the images are being used in other applications like weather forecasting, medical diagnosis, police department. In R-Tree implementation of image database, images are made available to the program which are then stores in the database. The image database is presented using R-tree and the database is stored in separate file .This R-tree implementation results in both update as well as efficient retrieval of images from hard disk [1][2][4]. We use the similarity based retrieval feature to retrieve the required number of similar images being inquired by the user [3][5][6]. Distance matrix approach is used to find similarity of images [7]. Sobel edge detection algorithm is used to form sketches. If sketch of image is entered for similarity based retrieval, then sketches of stored images are formed and these sketches are compared with input image (sketch) using distance matrix approach[8][9].

KEYWORDS

R-Tree, Image Database, Distance Matrix, Sobel Edge Detection.

1. INTRODUCTION

Spatial data objects often cover areas in multi-dimensional spaces and are not well represented by point locations For example, map objects like counties, census tracts etc occupy regions of non-zero size in two dimensions A common operation on spatial data is a search for all objects in an area, for example to find all counties that have land within 20 miles of a particular point This kind of spatial search occurs frequently in computer aided design (CAD) and geo-data applications, and therefore it is important to be able to retrieve objects efficiently according to their spatial location.

An index based on object's spatial locations is desirable, but classical one dimensional database indexing structures are not appropriate to multi-dimensional spatial searching Structures based on exact matching of values, such as hash tables, are not useful because a range search is required [17]. Structures using one dimensional ordering of key values, such as B-trees and ISAM indexes, do not work because the search space is multi dimensional.
Day by day, with increase in number of applications of images, we are confronted with a greater need to manage such large number of images efficiently. This is where the concept of image database comes in where the database consists of two dimensional objects rather than one dimensional object as in normal text databases.

To manage multidimensional data in database, various techniques have been proposed like k-d trees, k-d-b trees and variations like Quad tree, which are used in image databases [7].

DOI : 10.5121/sipij.2011.2408

In this paper R-tree based databases are used for efficient image retrieval and distance metric based approach is used for similarity based image retrieval. For this approach the two images are separated by distance metric. Lesser the value of distance metric, closer the two images are.

2. REVIEW AND GENERAL CONSIDERATIONS

2.1. Image Representations

The images can be stored in two ways, either storing the image properties as that is it could be in simple bitmap format or it can be stored as compressed image to save space [12]. The contents of an image consist of all objects in the image that may be of some interest from the point of an application. The objects in an image could have many properties associated with them few of them are the following:

Shape descriptor: It describes the portion of the image within which the object of interest may be located inside the given image. It could be of any shape rectangular, circular, and elliptical or any polygon of any arbitrary shape depending on the ease of processing over that region.

Property descriptor: It describes the properties of the individual pixels in the given image. Thus every image may be associated with lesser resolution (h1 x h2) which is less than the original image (m x n) (Fig. 1). Another way to say is that image takes the form of a grid of rectangular cells. This (h1 x h2) resolution is called grid resolution of the image. This divides the image into equal sized cells (h1 x h2), called the image grid, with each cell in the given gridded image consisting of a collection of pixels.

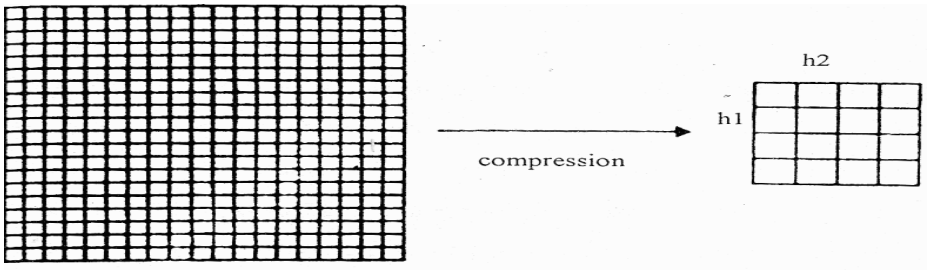


Figure 1. Compressed representation of an image.

The image compression consists of two main parts. The first is that of size selection wherein the size (c, r) of the compressed image is selected by the image database developer. The second and most important part consists that of selecting the compression algorithm. The two of the better known transformations are the DFT and DCT [13][14]. Discrete Fourier transform is represented as,

$$DFT(x, y) = (1/mn) \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} (I(a, b) \times e\ xp(-j(2\pi xa/m) + (2\pi yb/n)))$$

One of the important properties of DFT is that one can get back the original image from compressed representation. Inverse DFT is applied to compressed image and original image is obtained.

DCT forms the basis of jpeg compression of images. This is also invertible and equation is given by

$$DCT(i,j) = (2/\sqrt{(m \times n)}) * C(i) \times C(j) \sum_{r=0}^{m-1} \sum_{s=0}^{n-1} \left(\cos((2r+1)*i\pi/2r) \times \cos((2s+1)*j\pi/2s) \right)$$

Where

$C(i), C(j) = \{1/\sqrt{2}$ When i, j=0

Some advantage of DCT is that it improves quality of compression though both take same amount of time in computation.

2.2. Representations of Image Databases

Image database consists of images that are two dimensional in space. Traditional index structures, such as hash indices and B-trees, are not suitable, since they only deal with one dimensional data whereas image data are of two dimensions. To represent 2-dimensional data, techniques like k-d trees and Quad trees were proposed

k-d Tree structures

The k-d tree is used to store k-dimensional point data. A 2-d tree (i.e. k = 2) stores 2-D point data, a 3-D trees stores 3-D point data end so on. Images consist of 2-d point data. In k-d tree each level of a k-d tree partitions the space into two. The partitioning is done along one dimension at the node at the top level of the tree, along another dimension in nodes at the next level, and so on cycling through the dimensions. The partitioning is done such that, at each node, approximately one-half of the points stored in the sub tree fall on one side and one- half fall on the other. Partitioning stops when a node has less than a given maximum number of points. The numbering of lines is according to the level of the tree at which the corresponding node appears.

Point Quad tree structures

Point quad trees are also used to represent point data in 2-dimensional spaces. Unlike 2-d tree, point quad trees always split regions, into four parts. The set of points are the same. Each node of a quad tree is associated with a rectangular region of space. The top node is associated with the entire target space. Each non leaf node in a quad tree divides its region into four equal-sized quadrants, and correspondingly each such node has four child nodes corresponding to the four quadrants. Leaf nodes have between zero and some fixed maximum number of points. If the region corresponding to a node has more than the maximum number of points, child nodes are created for that node.

3. DESIGN OF IMAGE DATABASES

R-Tree Representations of Image Database

R-Tree is a data structure that may be used to store rectangular regions of an image or map .R-tree are particularly useful in storing very large amounts of data on disk [4]. Because disk accesses are often very slow, R-tree provides a convenient way of minimizing the number of disk accesses made.

Each R-tree has an associated order, which is an integer K. Each non-leaf R-tree node contains a set of at most K rectangles and at least K/2 rectangles. This feature makes R-tree appropriate for disk-based retrieval because each disk access brigs back a page containing several (at least K/2)

rectangles. Additionally, storing many rectangles per page, the height of the R-tree used to store a collection of rectangles is usually quiet small, thus diminishing the number of disk accesses required.

In R-tree leaf node contains one rectangle per node, while non leaf node contains group rectangles [1]. When inserting a new rectangle into a R–tree, the following way is preceded:

1) We check to see which of rectangles associated with the root needs to be expanded the least(in terms of area) in order to accommodate the rectangles being inserted.

2) Following the link associated with the root the rectangle is inserted into available slot.

3) In case the slots are full, the rectangle is inserted in the group in which the total area of the group rectangles is smallest.

The R-tree implementation involves the following steps:

1) First ask the user whether he wants to manipulate image database. For that GetRectangle () will be used. It gets objectid, imageid and object coordinates from the users and stores them in database.

2) With the available images, create the R-tree that stores all rectangles. The grouping is done so as to maintain the maximum and minimum limits on the order of the tree.

3) Those rectangles which results in minimum expansion of area are included in one group and in case of underflow and overflow, other groups are used.

4) Each rectangle has an associated set of fields that specify the properties of rectangle. These fields contain information about the content of the rectangles.

ENTRIES

An n-dimensional spatial database consists of an amount of tuples where each of them has a unique identifier, by which the tuple can be retrieved [2]. Every leaf node contains a tuple like (I, leaf-identifier).In this case leaf-identifier points to the place where the object is stored in the spatial database, and I is an n-dimensional rectangle I = (I0, I1, ..., In-1).

Every Ik represents a closed bounded interval [ak, bk] (see figure 2.), which means the starting and ending point of the rectangle in the k-th dimension. If one or two end points of the interval Ik are equal to the infinity, it is meant that the described object extends outward indefinitely in the k-th dimension. Non-leaf nodes contain entries (I, child-pointer) where child-pointer points to the child node in the R-Tree and I covers all rectangles of the child node.

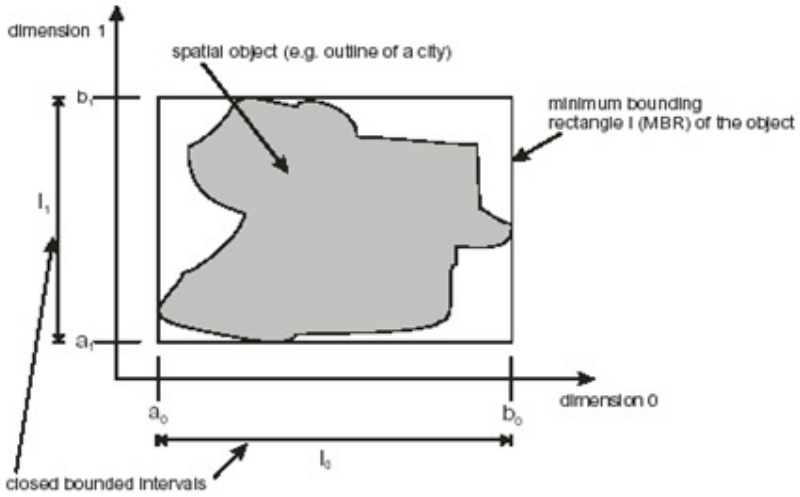


Figure 2. Starting and Ending point of the rectangle in the k-th dimension

PROPERTIES

If M (depends on the machine's disk page size and the number of dimensions n) is the maximum number of entries, that will fit in one node and m □specifies the minimum number of entries in a node, then a R-Tree must fulfill the following properties:

1) Every leaf node, which is not the root, contains between m and M index records.

2) For each index record (I, leaf-identifier) in a leaf, I is the smallest rectangle that contains the n-dimensional data object represented by this tuple.

3) Every non-leaf node, which is not the root, has between m and M children.

4) For each entry (I, child-pointer) in a non-leaf node, I is the smallest rectangle that contains the child node.

5) The root node has at least two children unless it is a leaf.

6) All leaves appear on the same level.

7) The height of an R-Tree, with N index records is □□□□1 - N logmin the best case because each node contains at least m child nodes.

OVERFLOW and UNDERFLOW

An overflow could happen if the m (minimum number of entries in a node) is set too high (near to M), so that the node is filled very densely. If one or more additional entries will be written into this node, the maximum number of entries M will be exceeded and the node overflows (see figure 3.).

Equivalent to the overflow the node underflow appears also when m is set too large and one or more entries are removed so that the number of entries will remain under m (see figure 4.).

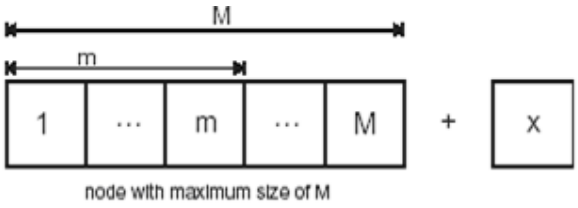


Figure 3.

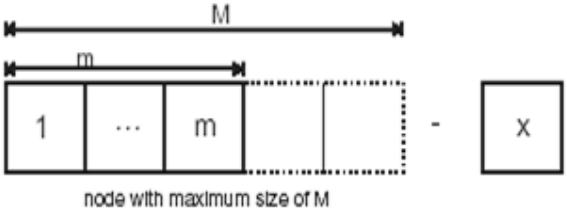


Figure 4.

ALGORITHMS

Guttman gave the algorithms for the elementary operations on the R-Tree. These methods of the R-Tree are leant on the ones from the B-Tree, only the handling of overflow and underflow are different owing to the spatial location of the data.

In this database (see figure 5, 6,7) name is the name of the student, semester tells us in which semester the student is, and credits is the sum of all achieved credits in the university.

name	semester	credits
A	8	100
B	4	10
C	6	35
D	1	10
E	6	40
F	5	45
G	7	85
H	3	20
I	10	70
J	2	30
K	8	50
L	4	50

Figure 5. The Example Database

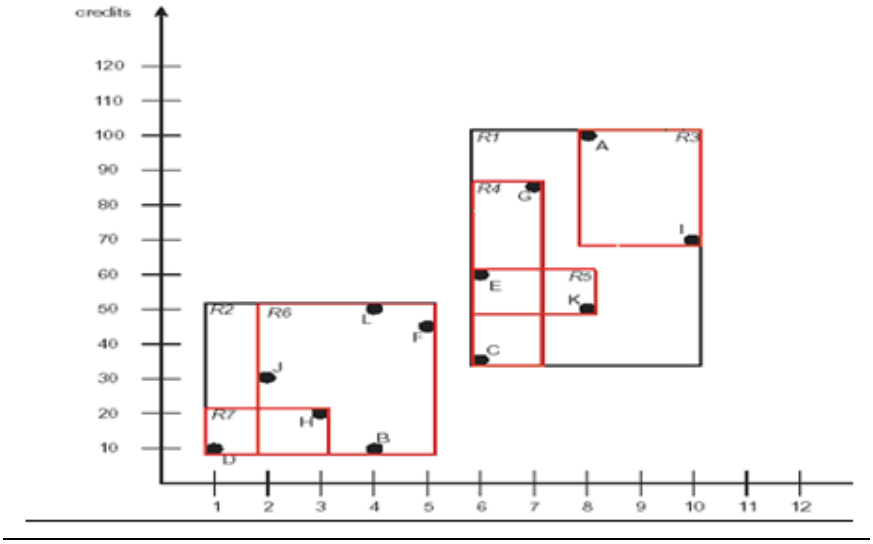


Figure 6. The Example Database as R-Tree Structure

SEARCHING

The searching in an R-Tree works similar to the search in a B-Tree; the tree will be descended from the root. But since it could happen that in an R-Tree - in distinction to the B-Tree - several rectangles, which have to be searched, are overlapping (see fig. 7).

Algorithm: ***Search***

Let T be the root of an R-Tree. We search all index records whose rectangles overlap a specified search rectangle S [3].

If T is not a leaf, then apply Search on every child whose root is pointed by child-pointer and that overlaps with S.

If T is a leaf, return all entries which overlap with S as the result set.
An example for search according to the example database

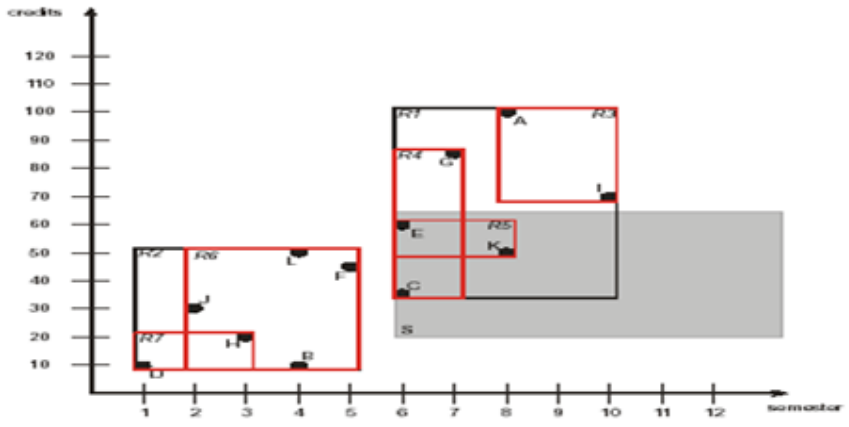


Figure 7. The Search Rectangle S over graphical display of example database.

INSERTION

If a new entry has to be inserted into a database, a new index record has to be added to the R-Tree. This is also the only chance for the R-Tree to grow in height, namely if there is a node overflow, the node has to be split. In the case that the split reaches the root, the height will grow

Algorithm: **Insert**

1) Let E be the new entry.

2) Use ChooseLeaf to find the leaf node L where E has to be placed in.

3) If there is enough space in L (no node overflow), add E to it. Else apply SplitNode to L, which will return L and L' containing E and all the former elements of L.

4) Apply AdjustTree on L, if there was a split before do this also on L'.

5) If the split reaches the root and it has to be split, create a new root whose children are the two nodes which the split of the root returned.

Algorithm: ChooseLeaf

1) Select a suitable leaf node to place the new entry E in.

2) Let N be the root node.

3) If N is a leaf, return it.
4) If N is not a leaf, find the entry Fk in N, whose rectangle has to be enlarged least to include the rectangle of E.
5) For the case that several entries Fk are found, choose the smallest one.

6) Apply ChooseLeaf on the chosen Fk until a leaf is reached.

Algorithm: **AdjustTree**

1) Climb from a leaf node L to root, while adjusting the rectangles and doing node splits.

2) Set N=L. If L was split formerly set N' as L'.

3) If N is the root, end.

4) Let P be the parent of N. Adjust N's entry in P so that is covers all rectangles in N.

5) If a splitting has occurred, add a new entry to P that points to N'. If the parent node P overflows, invoke SplitNode on it.

SPLITING a NODE

When adding a new entry to a full node (a node containing M entries), it is necessary to split these M+1 entries up in two new nodes. When thinking about how to divide them, the goal should be to minimize the size of the two resulting rectangles because the smaller they are the better is

the chance for the search algorithm to visit only the nodes that are necessary to visit [5]. It's because the smaller the rectangles are the smaller is the possibility that they overlap with others.

In figure.8a , you can see an example for a good split and in figure. 8b one for a bad split

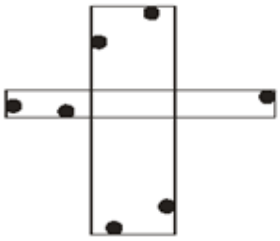


Figure 8a

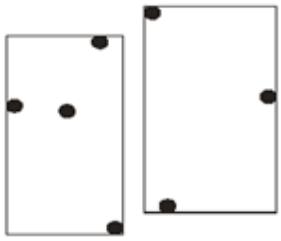


Figure 8b

There are three algorithms for SplitNode - Exhaustive splitnode, Quadratic cost, linear cost algorithms, offered by Guttman, to divide M+ 1 entry in two nodes N and N'. Their names tell us how their CPU usage increases relative to M. But i used Linear cost algorithm for splitting Facenodes and Treenodes .

Linear cost algorithm*:*

This one chooses for each dimension i {0, …, k-1} the two entries that are the most distant ones from each other in this dimension i and puts them into different nodes.After that has been done for every dimension, the left nodes are randomly distributed [6].The dissimilarity to the quadratic cost algorithm is localized in the methods PickSeeds and PickNext.

For the reason that this algorithm is very speedy, it can outweigh its bad search performance. I have used linear cost algorithm for splitting Facenodes and Treenodes.

4. EDGE DETECTION

Edge detection is to mark the points in a digital image at which the luminous intensity changes sharply. It takes an input image of size 256 × 256. Sharp changes in image properties usually reflect important events and changes in properties of the world. These include (i) discontinuities in depth, (ii) discontinuities in surface orientation, (iii) changes in material properties and (iv) variations in scene illumination [9].

Edge detection of an image reduces significantly the amount of data and filters out information that may be regarded as less relevant, preserving the important structural properties of an image. There are many methods for edge detection, but most of them can be grouped into two categories, search-based and zero-crossing based. To implement edge detection I followed Sobel Edge detection Algorithm.

EDGE ELEMENTS EXTRACTION BY THRESHOLD

Most of the edge detection techniques have two steps :
Finding the rate of change of change of gray levels, i.e. the gradient of the image
Extracting the edge elements where gradient exceeds a predefined threshold.
Now, we like have an image whose value e(r, c) at the pixel (r ,c) is obtained as

$$e(r,c) = \begin{cases} 1 & \text{if } g'(r,c) > t(r,c) \\ 0 & \text{if } g'(r,c) \le t(r,c) \end{cases}$$

So, e(r, c) is a binary image where the image subset *S*e contains only edge elements of g(r, c). Here t(r, c) is the threshold at the pixel (r, c) and can be found out using relation :

$$t(r, c) = \Phi''_{th}(r, c, Q_p(r, c))$$

where Q_p (r ,c) denotes the set of features at pixel (r, c). depending on the variable to determine t(r, c) the threshold is called *global* (or space invariant), local (or space variant) nad dynamic. In brief, the edge detection technique then reduces the selection of threshold that transforms the gradient image onto the edge image, i.e.

$$T_{th}: \; g'(r, c) \rightarrow e(r, c)$$

The threshold that extracts edges between the regions can be fond in three ways :
1. If the shape of the ideal edge in the image is known (that means, the shape or type of the regions whose edges are to be extracted are known) a priori, then t(r, c) can be chosen in an attractive way so that the subset S_e of edge image represents the nearest approach to the ideal edge.

2. If the threshold t(r, c) (or simply, t) is known) a priori, then the image subset S_e, obtained by the shareholding operation, gives the edges of the objects whatever they might be. This approach is used when the threshold is estimated based on a large number of training images with proper ground-truth.

3. If neither the shape of S_e nor the threshold t(r, c) is known a priori, then the threshold chosen such that S_e satisfies the following criteria:
$\{t_i, i = 1,2,3,\ldots., n1\} \leftarrow$ A non-empty set of threshold values
$\{P_j , j = 1,2,3,\ldots., n2\} \leftarrow$ A non-empty set of properties those an edge should possess
$\epsilon_{pj}(t_i) \leftarrow$ An error function characterizing how loosely S_e satisfies the property P_j for the threshold t_i .

$\epsilon_{pj}(t_i) \leftarrow$ Some convex combination of $\epsilon_{p1}(t_i)$, $\epsilon_{p2}(t_i)$,……, $\epsilon_{pn2}(t_i)$
Hence t_0 can be taken as optimum threshold if $\epsilon(t_0) = \min\{\epsilon(t_i)\}$ and the image subset S_e is accepted as an optimum edge.. Various types of errors introduced during edge extraction are shown in figure 9.

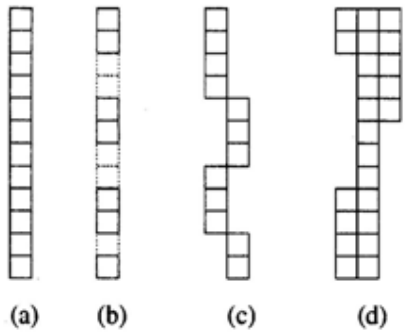


Figure 9. Various types of defects that occur in extracted edges: a) ideal
b) fragmented, c) offset, and d) smeared

SOBEL EDGE DETECTION

The Sobel operator performs a 2-D spatial gradient measurement on an image. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image. A convolution mask is usually much smaller than the actual image [8]. As a result, the mask is slid over the image, manipulating a square of pixels at a time. The actual Sobel masks are shown below:

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 10. Sobel Masks.

The magnitude of the gradient is then calculated using the formula:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

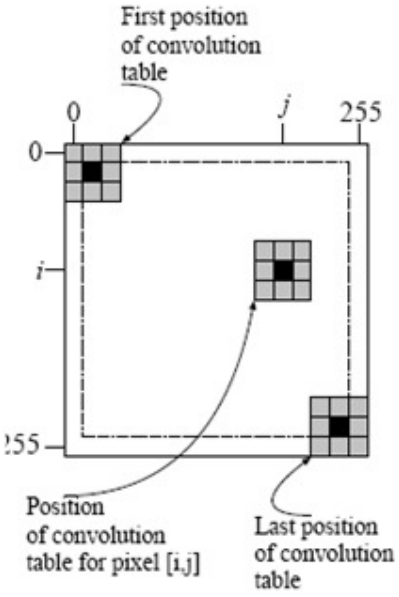
An approximate magnitude can be calculated using:

|G| = |Gx| + |Gy|

SOBEL ALGORITHM

The Sobel edge detection algorithm uses a 3 x 3 table of pixels to store a pixel and its neighbors while calculating the derivatives [11]. The 3 x 3 table of pixels is called a *convolution table*, because it moves across the image in a convolution-style algorithm.

Figure 11. shows the convolution table at three different locations of an image: the first position (calculating whether the pixel at [1, 1] is on an edge), the last position (calculating whether the pixel at [254,254] is on an edge, and at the position to calculate whether the pixel at [*i,j*] is on an edge.



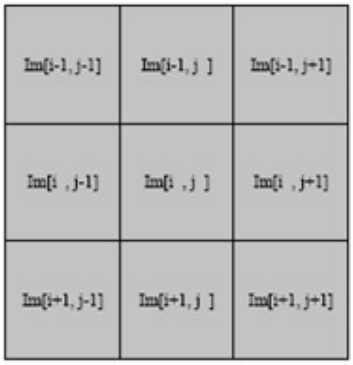


Figure 3: Contents of convolution table to detect edge at coordinate $[i, j]$

Figure 11. 256 x 256 image with 3 x 3 neighborhoods of pixels

[0,0]	[0,1]	[0,2]
[1,0]	[1,1]	[1,2]
[2,0]	[2,1]	[2,2]

Figure 12. Coordinates of 3 x 3 convolution table

Figure 12. shows a convolution table containing the pixel located at coordinate [*i,j*] and its eight neighbours. The table is moved across the image, pixel by pixel. For a 256 x 256 pixel image, the convolution table will move through 64516 (254 x 254) different locations. The algorithm shows how to move the 3 x 3 convolution table over a 256 x 256 image. The lower and upper bounds of the loops for *i* and *j* are 1 and 254, rather than 0 and 255, because we cannot calculate the derivative for pixels on the perimeter of the image.

The Sobel edge detection algorithm identifies both the presence of an edge and the direction of the edge (Figure 13.). There are eight possible directions: north, northeast, east, southeast, south, southwest, west, and northwest [10].

For each direction, Figure 6, shows an image sample, a convolution table, and the encoding of the direction. In the image sample, the edge is drawn in white and direction is shown with a black arrow. The eight directions are grouped into four orientations: NE SW, N S, E W, and NW SE.

For a convolution table, calculating the presence and direction of an edge and is done in three major steps:

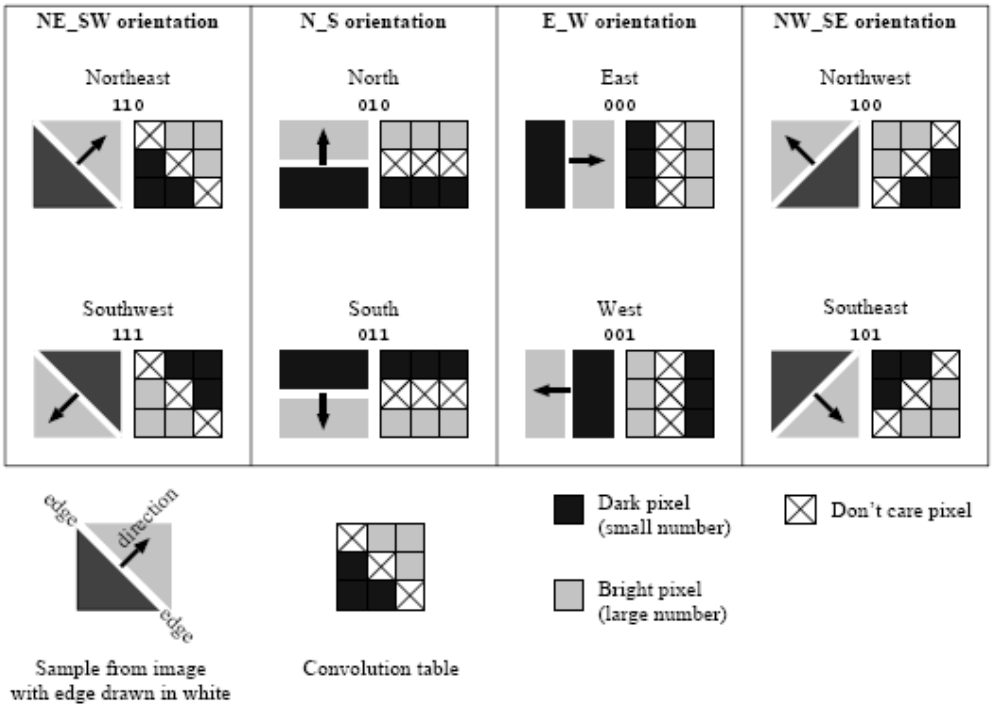


Figure 13. Four orientations and eight directions

1. Calculate the derivative along each of the four orientations. The equations for the derivatives are written in terms of elements of a 3×3 table, as shown in Figure 5.4

Deriv_NE_SW = (table[0,1]+2 × table[0,2]+table[1,2]) - (table[1,0]+2 × table[2,0]+table[2,1]);
Deriv_N_S = (table[0,0]+2 × table[0,1]+ table[0,2]) - (table[2,0]+2 × table[2,1]+table[2,2]);
Derive_E_W = (table[0,2]+2 × table[1,2]+table[2,2]) - (table[0,0]+2 × table[1,0]+table[2,0]);
Derive_NW_SE = (table[1,0]+2 × table[0,0]+table[0,1]) - (table[2,1]+2 × table[2,2]+table[1,2]);

2. Find the value and direction of the maximum derivative, and the absolute value of the derivative that is perpendicular to the maximum derivative.
EdgeMax = Maximum of absolute values of four derivatives
DirMax = Direction of EdgeMax
EdgePerp = Absolute value of derivative of direction perpendicular to DirMax
Note: *The following priority order determines which direction gets picked if more than one derivative have the same magnitude.*
(a) Deriv N S
(b) Deriv E W
(c) Deriv NE SW
(d) Deriv NW SE
This means that if, for instance, Deriv N S and Deriv E W are equal, Deriv N S must be picked.

3. Check if the maximum derivative is above the threshold. When comparing the maximum derivative to the threshold, the Sobel algorithm takes into account both the maximum derivative and the derivative in the perpendicular direction.
if EdgeMax + EdgePerp/8 >= threshold value then
Edge = true

Dir = DirMax
else
Edge = false
Dir = 000

5. IMPLEMENTATION

The implementation of R-TREE (region tree) consist of three modules, which are given below:-

A) **Insertion module**: This block is used for creation of R-TREE and insertion of rectangles into R-TREE.

```
struct treenode * InsertRtree(treenode *rtree ,char objname[],char imagename[],int xlb
,int ylb,int xub,int yub);
```

This is called inside GetRectangles() function as given below:

```
void GetRectangles(treenode *rtree)
{
rtree=InsertRtree(rtree,objname,imagename,xlb,ylb,xub,yub);
}
```

InsertRtree() function consists of following four modules:

choseleaf module: This block traverse tree from route and return facenode(leaf), where rectangle to be inserted. We check to see which of child associated with the root needs to be expanded the least(in terms of area) in order to accommodate the rectangles being inserted. If this is Treenode(non-leaf), perform choseleaf() on Treenode until Facenode is found. If child is Facenode(leaf), perform adjusttree() on Facenode and return.

adjusttree module: This module insert rectangle in facenode and adjust tree after insertion. If facenode return by choseleaf() has slot for new rectangle, then put new node in vacant slot and modify region covered by Facenode and all its parent nodes. If slot is not available, perform splitnode() on Facenode.

Adjusttree() function consists of two modules:

splitnode module: This function splits Facenode into two Facenode(N and N1) and modifies area covered by these Facenodes and all its Parent nodes. Linear cost algorithm is used to split Facenodes. Two most distant rectangles are put into N and N1. New rectangle is put into N. All others rectangles are distributed into N and N1.

splittreenode module: Linear cost algorithm is used to split Treenodes. Two most distant Facenodes are put into T and T1 Treenodes. New Facenode (N1) is put into T. All others Facenodes are distributed into T and T1.

B) **process_bmp()**: This function store regions of image in database. This function stores selected region (rectangle) of input image on files in hard disk.

C) **Search module**: This module is used to search rectangle for given query and display images on to the monitor. All Facenodes are visited and return to find_most _similar(), which overlaps specified search rectangle.

search(treenode *rtree,struct facenode**cinsert,struct treenode * parinsert,char objname[],char imagename[],int xlb,int ylb,int xub,int yub)

D) find_most_similar(): This function will search similar images from database. This function performs similarity based retrieval on search image and rectangles of all Facenodes return by search module using distance matrix approach.
Sobel edge detector module: This function performs edge detection on all rectangles of all Facenodes return by search module. The sketches of these rectangles are compared with **sketch** of search image.

FILE * sobel (char objname [], FILE *f, int xlr, int ylr, int xhr, int yhr)

I have used following structures:

```
struct infotype
{
        char objname[12];
        struct infotype  *next;
};
struct objnode
{
        char objid[12];
        char imageid[12];
        struct infotype *data;
        struct objnode *next;
};
struct facenode
{
        struct rectangle rect[ORDER];
        int xl,yl,xh,yh;
};
struct treenode
{
        int xl,yl,xh,yh;
        struct cell cells[ORDER];
        struct treenode *backptr;
 };
```

6. EXPERIMENTAL RESULTS

INPUT FILE SUCCESSFULLY OPENED FOR READING...
OFFSET VALUE : 72
ORDER OF BYTES : II
TIFF VERSION : 42
IFD NO : 1 AT OFFSET : 72
NO OF ENTRIES/TAGS IN THIS IFD : 17
PRESS ANY KEY TO CONTINUE.....

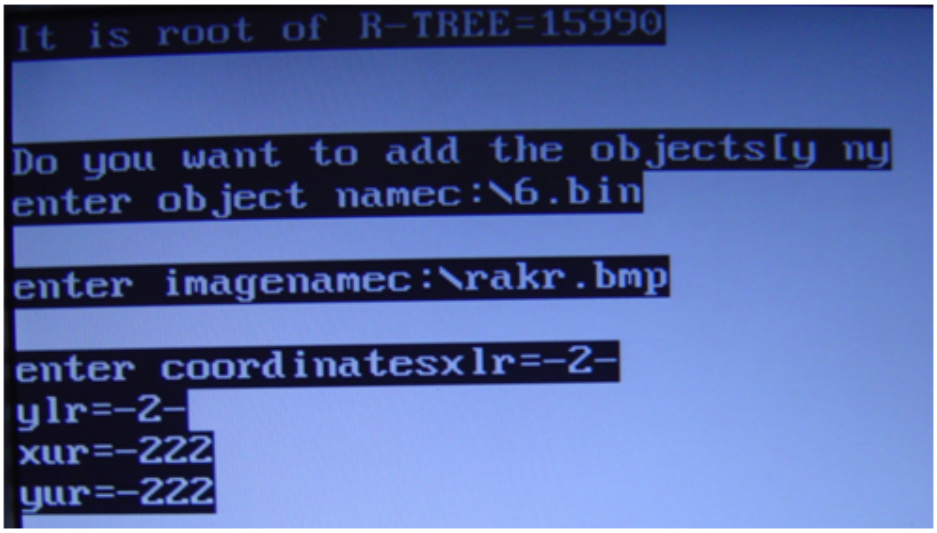


Figure 14. Requesting object and image name with their x and y coordinates.

In figure 14, User is requested to add the object. if user wants to add object then prompts for object and image name. The user is requested to enter the coordinates of the input image

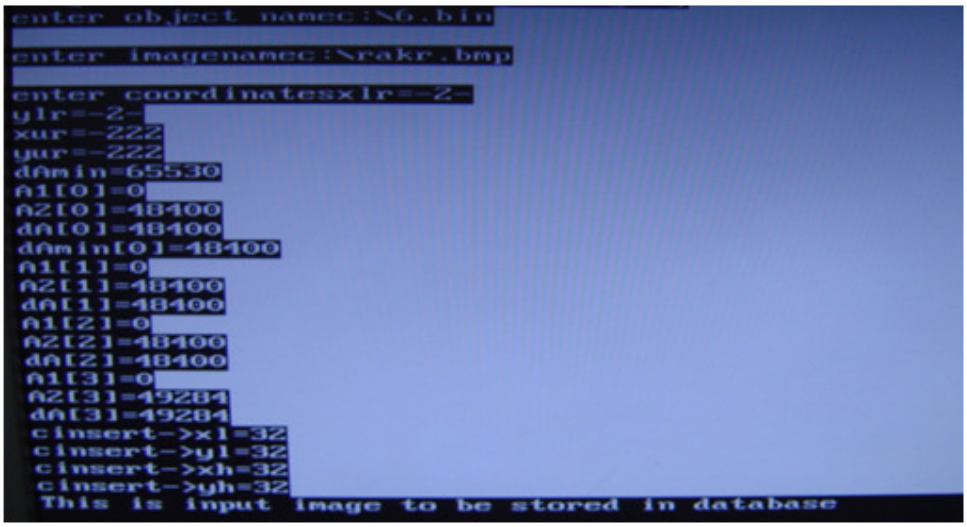


Figure 15. Displaying image stored in database .

Figure 16. Displaying the input image to be stored in database.

In figure 16, the input image is displaying and stored in the database, which is further used for the similarity based image retrieval.

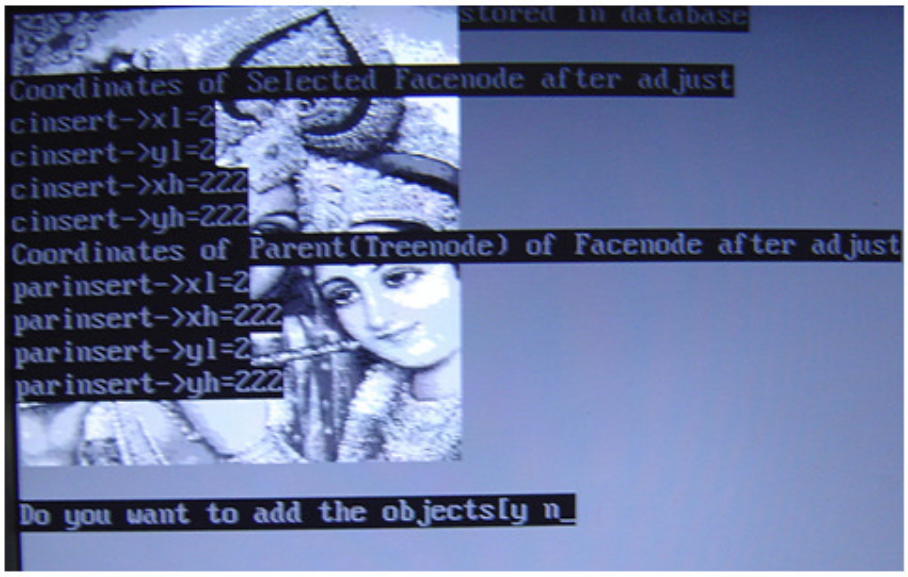


Figure 17. Displaying the coordinates of Facenode after adjust the image.

The figure 17, shows the coordinates of the selected facenode of the input image after adjusting it. It presents the values of the facenode and the tree node after inserting the coordinates of the input image. It asks the user ,do he wants to add the objects.

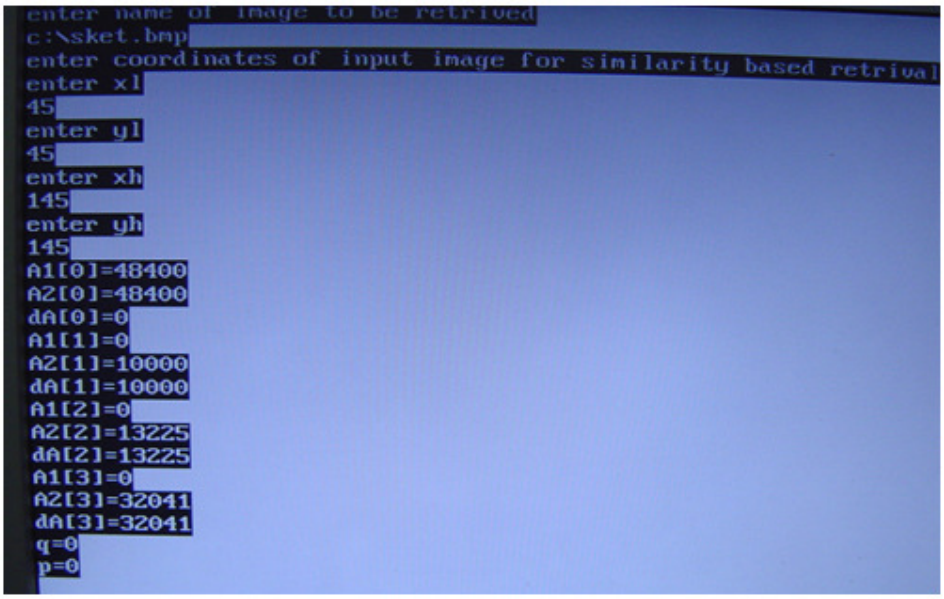


Figure: 18, Requesting the coordinates of input imagefor similarity based retrieval.

In figure: 18, the screen prompts the user to enter the name of the image which is to be retrieved. Then enter the coordinates of the image to be shown on the screen.

Figure 19. Displaying the sobel edge detected image as on the basis of the similarity.

The figure: 19, shows the similar image retrieved on the basis of the input image entered. The program search the similar images as the input image from the database and determine the number of images which are similar to the input image.

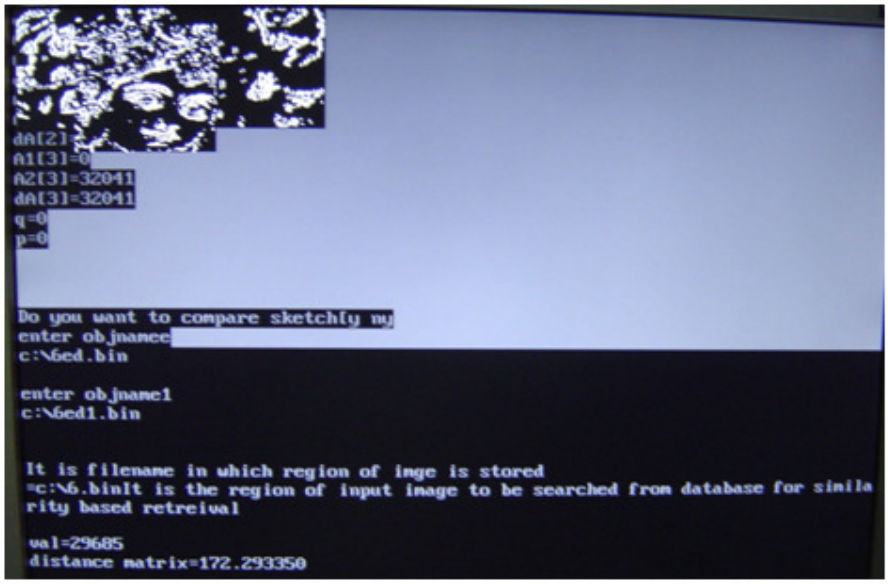


Figure 20. Displaying the distance matrix value 172.293350.

The figure: 20 shows the region of the input image to be searched from the database for similarity based retrieval. it uses the distance matrix value ie.172.293350. the distance matrix approach is used to find the similarity between the given images.

7. CONCLUSIONS

R-tree image database is successfully implemented. I have been able to search similar images for given part of image .I have been able to develop a workable image database, manipulate it and retrieve the given number of similar images from database. This R-tree is able to store large no. of images. I have used linear method of node splitting. Only regions of images are stored in image database.

There is no 1 set threshold value which can give optimum results for all images. Intensity level variations play a major role on determining the best threshold value. If there are intense variations in pixel intensities, the gradient values calculated by the Sobel Edge Detector will be high. In such cases the threshold value needs to be set at a high level to give a better output. If threshold is kept low, Smearing effects will be dominant, as many unwanted intensity variations will also be detected. This will result in high noise levels at the output.

FUTURE WORK

To achieve major improvement in database, a more efficient tree structure need to be found so there will be no need to fully traverse the tree and area of expansion can be found quickly. As far as similarity based retrieval is concerned, there is need to tackle the challenges of recognizing the shape similarity .more feature need to be added for much better shape retrieval. Future approach should be to extract feature of object as eye perceive.

REFERENCES

[1] Weihua Lin, "An Improvement of Index Method and Structure Based on R-Tree", International Conference on computer science and software engineering. Issue Date : 12-14 Dec. 2008.

[2] Hui Li, Shiguang Ju, Weihe Chen, "Design and Implementation of Generalized R-Tree", International Symposium on Computer Science and Computational Technology -2008 ,Volume 2.

[3] Augustine, K.V. Huang Dongjun, "Image similarity for rotation invariants image retrieval system",International Conference on Multimedia Computing and Systems, 2009. ICMCS '09.

[4] R-Tree implementation ,www.rtreeportal.org/code.html.

[5] J.F. Omhover, "A Region-Similarity-Based Image Retrieval System".

[6] Cheng Chang "Image retrieval based on region shape similarity".

[7] Distance matrix methods, www.hum.utah.edu/~mhaber/Documents/.../FelsensteinCh11.pdf.

[8] O. R. Vincent,, "A Descriptive Algorithm for Sobel Image Edge Detection" Proceedings of Informing Science & IT Education Conference (InSITE) 2009.

[9] James Matthews, "An introduction to Edge Detection :The Sobel Edge Detector", an article.

[10] Wenshuo Gao, "An improved Sobel edge detection", 3rd IEEE International Conference on Computer Science and information technology(ICCSIT),2010,Volume: 5 .

[11] Hongyan Sun, "Image retrieval based on blocked histogram And sobel edge detection algorithm" International Conference on Computer Science and Service System (CSSS), 2011.

[12] B.FURHT, "Multimedia System: an Overview", IEEE Multimedia, Vol. 1, No. 1, spring 1994, pp. 47-59.

[13] Adam, "Applications", IEEE Spectrum, Vol. 30, No. 3, spring 1994, pp. 24-31.

[14] J.L Bentley, "Multidimensional Binary search Trees used for Associative Searching", Communication of the ACM, Vol. 18, No. 9, sept. 1975, pp. 509-517.

[15] N Gudivada and V.V Raghvan, "Content-based Image and video Content: The QBIC System", Computer, Sept. 1995, pp. 23-32.

[16] J.P. Eakins, "Similarity Retrieval of Trade Mark Images", IEEE Multimedia, Vol. 5, No. 2.

[17] D.Philips, "Image Processing in C", bpb Publication, New Delhi, 1990.

Author:

Chandresh Pratap Singh received his B.Tech degree in Information Technology from Krishna Institute of Engineering & Technology (U.P.T.U), Ghaziabad, U.P, India in 2009. He is currently pursuing M.TECH degree in Computer Science & Engineering at Jaypee University of Information Technology, Waknaghat, Solan, H.P, India. His research interest includes Image processing, Data mining & Web Crawling.