# Quad Trees

December 4, 2021

**Aditya Aggarwal (2020CSB1066)** ,
**Anubhav Kataria (2020CSB1073)** ,
**Prakhar Saxena (2020CSB1111)**

---

**Instructor:**
Dr. Anil Shukla

**Teaching Assistant:**
Dr. Ravi Bhatt

**Summary:** Quadtrees are hierarchical spatial tree data structures that are based on the principle of recursive decomposition of space. The term quadtree originated from representation of two dimensional data by recursive decomposition of space using separators parallel to the coordinate axis. The resulting split of a region into four regions corresponding to southwest, northwest, southeast and northeast quadrants is represented as four children of the node corresponding to the region, hence the term "quad" tree.

---

## 1. Introduction

A Quad Tree is a data structure in which each node has exactly four children. Quad trees are most useful in indexing and partitioning any two dimensional space by recursively subdividing it into four regions. The type of quad tress which has been explored and implemented in this project is called a region quad tree.

Region Quad Tree - By breaking the region into four equal quadrants, sub quadrants, and so on, the region quad tree represents a two-dimensional partition of space, with each leaf node carrying data pertaining to a specific sub region. Every node in the tree either has four children or none at all (a leaf node). The spatial distribution of areas in the space being decomposed is sensitive to and dependent on the height of quad trees that follow this decomposition technique (i.e. subdividing sub quadrants as long as there is relevant data in the sub quadrant for which greater refinement is required).
Naturally, this data structure proves useful for image processing wherein each leaf node represents pixel value ranging between 0-255. The root node represents the whole image.

Quad Tree is a simple yet powerful data structure. Some of its uses are image compression (done in this project for gray scale images), mesh generation, spatial indexing (done in this project), and storing sparse data.

## 2. Point Quad Trees & Region Quad Trees

This section discusses in detail about Point Quad Trees & Region Quad Trees.

### 2.1. Point Quad Trees

The point quadtree is a natural generalization of the binary search tree data structure to multiple dimensions. For convenience, first consider the two dimensional case. Start with a square region that contains all of the input points. Each node in the point quadtree corresponds to an input point. To construct the tree, pick an arbitrary point and make it the root of the tree. Using lines parallel to the coordinate axis that intersect at the selected point, (see Figure 1) divide the region into four subregions corresponding to the southwest, northwest, southeast and northeast quadrants, respectively. Each of the subregions is recursively decomposed in a similar manner to yield the point quadtree. For points that lie at the boundary of two adjacent regions, a convention can be adopted to treat the points as belonging to one of the regions. For instance, points lying on the left and bottom edges of a region may be considered included in the region, while points lying on the top and right edges are not. When a region corresponding to a node in the tree contains a single point, it is considered a leaf node. Note that point quadtrees are not unique and their structure depends on the selection of points used in region subdivisions. Irrespective of the choices made, the resulting tree will have n nodes, one corresponding to each input point.[3]

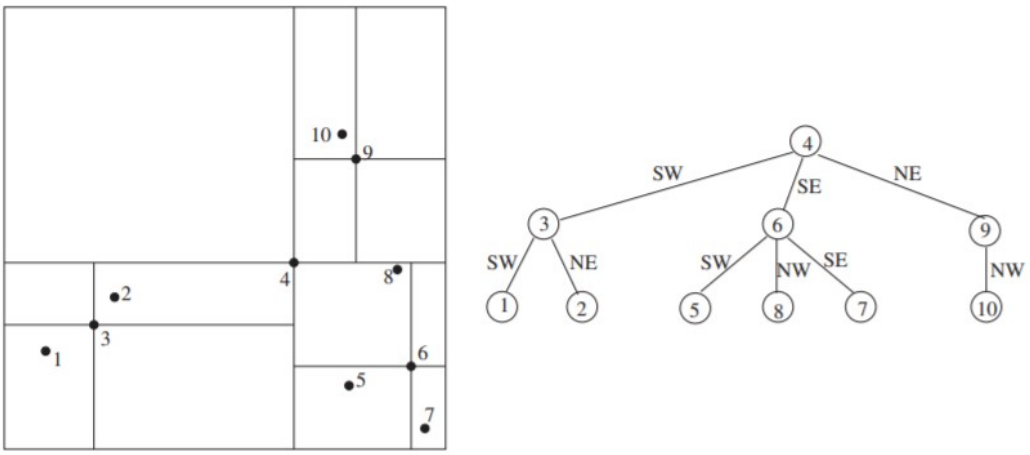



Figure 1: A two dimensional set of points and a corresponding point quadtree

### 2.2. Region Quad Tree

The region quadtree represents a partition of space in two dimensions by decomposing the region into four equal quadrants, subquadrants, and so on with each leaf node containing data corresponding to a specific subregion. Each node in the tree either has exactly four children, or has no children (a leaf node). A region quadtree with a depth of n may be used to represent an image consisting of $2^n * 2^n pixels$, where each pixel value is 0 or 1. The root node represents the entire image region. In this application, each leaf node represents a block of pixels that are all 0s or all 1s. Note the potential savings in terms of space when these trees are used for storing images; images often have many regions of considerable size that have the same colour value throughout. Rather than store a big 2-D array of every pixel in the image, a quadtree can capture the same information potentially many divisive levels higher than the pixel-resolution sized cells that we would otherwise require. The tree resolution and overall size is bounded by the pixel and image sizes.

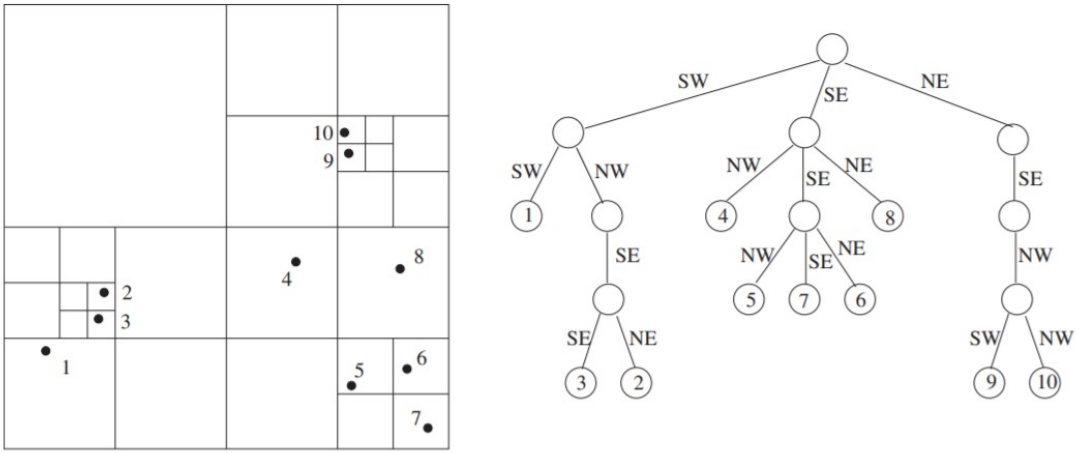



Figure 2: A two dimensional set of points and a corresponding region quadtree

## 3. Equations

This section deals with the mathematical part involved int the project. In this we will see the time complexity and space complexity of various functions like insert, search etc. and the compression of images using Quadtrees.

The Time Complexity of Insert Function in Quadtree is **O(logN)** ;where n is the size of the distance. During insert we have to traverse through the tree to find the appropriate location of the node. We traverse through the tree like in search and insert the node at it's location if it is not already present.

The Time Complexity of Search Function in Quadtree is **O(logN)** ;where n is the size of the distance.As we traverse through the tree with each traversal the number of nodes to be checked decreases by a factor of 4 by classifying the nodes's x-coordinate(& y-coordinate) either greater than or less than the desired node location.

The Space Complexity of the Quadtree is **O(klogN)** ; where k is the count of points in the space and space is of dimension **N x M, N >= M**.

As for finding the percentage of image compression we use the formula

$$(OS - CS) * 100/OS$$

where OS = Original Size of the Image
CS = Compressed Size of the Image

# 4. Figures, Tables and Algorithms

This section in detail covers various examples of compression of Images, %age calculation of compression achieved using our scheme and analysis of same .

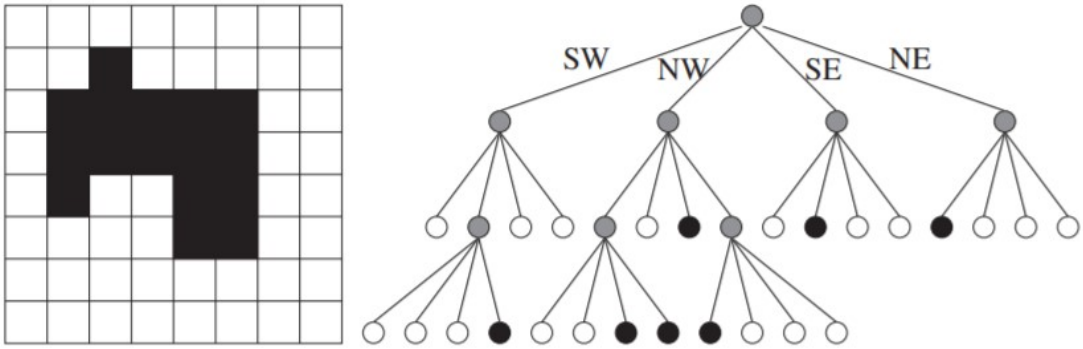



Figure 3: A two dimensional array of pixels, and the corresponding region quadtree

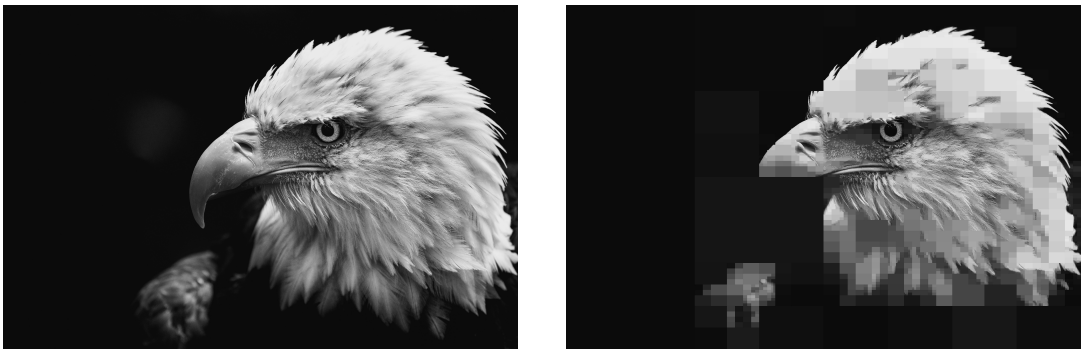
## 4.1. Demonstration of Image Compression

(a) Original Image. (b) Compressed Image

Figure 4: Compression Using QuadTrees-Example-1

(a) Original Image. (b) Compressed Image

Figure 5: Compression Using QuadTrees-Example-2



(a) Original Image. (b) Compressed Image

Figure 6: Compression Using QuadTrees-Example-3

## 4.2. %age Compression Calculations

**Calculations**

| | Original Image Size | Compressed Image Size | %age Compression |
|---|---|---|---|
| **Figure 4** | 214.7KB | 74.6KB | 65.25% |
| **Figure 5** | 94.9KB | 48.6KB | 48.78% |
| **Figure 6** | 2.7MB | 1.9MB | 29.62% |

Table 1: Compression %age achieved from our code

Images with less detail i.e. images that have similar grey scale intensities in adjoining pixels achieve a higher rate of compression as compared to images with higher detail.[2]

### 4.3. Algorithms

There are two main algorithms implemented in this project for spatial indexing. Insert node inserts a node at its correct location according to its coordinates. Search node locates a node based on given coordinates in the quad tree.

Algorithm 1 Insert node (Recursion)

```
Node is given as initial input.
if node is null then
    return
end if
if outside − boundary then
    return
end if
if area of quad unit <= 1 then
    insert at this location
    return
end if

//Figure out which quadrant node belongs to
if node − x <= (left − x + right − x)/2 then
    if node − y >= (bottom − y + top − y)/2 then
        insert at top left tree recursively
    else
        insert at bottom left tree recursively
    end if
else
    if node − y >= (bottom − y + top − y)/2 then
        insert at top right tree recursively
    else
        insert at bottom right tree recursively
    end if
end if
```

Algorithm 2 Search node (Recursion)

```
Coordinates of point are given as initial input.
if outside − boundary then
    return
end if
if area of quad unit = 1 then
    return this pointer
end if

//Figure out which quadrant coordinates belong to
if point.x <= (left − x + right − x)/2 then
    if point.y >= (bottom − y + top − y)/2 then
        if top left tree == null then
            return NULL
        end if
        search in top left tree recursively
    else
        if bottom left tree == null then
            return NULL
        end if
        search in bottom left tree recursively
    end if
else
    if point.y >= (bottom − y + top − y)/2 then
        if top right tree == null then
            return NULL
        end if
        search in top right tree recursively
    else
        if bottom right tree == null then
            return NULL
        end if
        search in bottom right tree recursively
    end if
end if
```

## 5. Practical Real Life implementation of Quadtrees

Quadtrees have various practical implementation which we come across from time to time. Quadtrees main implementation come in digital field. Few of it's implementation that will help us understand how beneficial it is to learn about Quadtrees are as follows:

### 5.1. Spatial Indexing

Spatial indices are used by spatial databases (databases which store information related to objects in space) to optimize spatial queries. Now Quadtrees divide the space into 2-D Plane and optimize it's searching efficiency. We lessen the number of queries by a factor of 4 with each iteration. This can be seen in geolocation.

### 5.2. Scaling Internet Service

Quadtrees are used for scaling an internet service to handle thousands of requests every second. This requires an excellent caching strategy. When geolocation is the main parameter of those requests, conventional caching techniques and procedures fall through. Quadtrees are used to understand high cache hit ratios, while also

keeping the responses relevant, even in intense areas. Here you can see how Quadtrees spatial indexing is used in an amusing way.[1]

### 5.3. Image Compression

Images in large quantity take a lot of space. We often see our device memory running out and space taken by images is a huge factor in doing so. But most forms of image compression help solve these problems by reducing the size of the image and thus reducing the overall pixels and memory they take up. Quadtrees for image compression works by recursively dividing the image into four subspaces with each holding the average RGB color and the error determining that color for its subspaces. The threshold is set based on that error and helps the tree determine if a node should be split further or not.[4]

## 6. Conclusions

We found out that quad trees are a very efficient data structure with applications in variety of fields. We have learnt a lot about Quad Trees in this course project. If we work on this topic further we can scale up our project to various other image processing operations along with research on new topics like compressed quad trees and related octrees and competition between K D Trees and Quad Trees.

## 7. Bibliography and Citations

Here is the list of online and offline sources we consulted from while doing our project.

https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00
https://en.wikipedia.org/wiki/Quadtree
https://www.youtube.com/watch?v=S4z-C-96xfU&list=PLAp0ZhYvW6XbEveYeefGSuLhaPlFML9gP&index=5
https://blog.qburst.com/2014/10/quadtree-simplified/

## Acknowledgements

We would like to thank Dr. Anil Shukla for providing us with the opportunity to work on an interesting topic of our choice and motivating us to continuously work towards our goal. We would also like to acknowledge our Teaching Assistant Dr. Ravi Bhatt for helping us at every stage of project.

## References

[1] Ravi Kanth V Kothuri, Siva Ravada, and Daniel Abugov. Quadtree and r-tree indexes in oracle spatial: a comparison using gis data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 546–557, 2002.

[2] Tassos Markas and John Reif. Quad tree structures for image compression applications. *Information Processing & Management*, 28(6):707–721, 1992.

[3] Hanan Samet. An overview of quadtrees, octrees, and related hierarchical data structures. *Theoretical Foundations of Computer Graphics and CAD*, pages 51–68, 1988.

[4] Eli Shusterman and Meir Feder. Image compression via improved quadtree decomposition algorithms. *IEEE Transactions on Image Processing*, 3(2):207–215, 1994.