Image Rotation Using Quad Tree

Aashish Kumar, Lec. ECE Dept. Swami Vivekanand institute of Engineering & Technology, Ramnagar Banur

ABSTRACT
This paper presents a data structure based technique of rotating an image. The data structure used is Quad trees. In a Quad tree the image can be represented using 2-d arrays. This array representation helps to rotate the image by changing the quadrants. The image can be rotated in clockwise and anti-clockwise direction. Firstly, the image is splitted upto a level so that a solid colored block remains at the end. Then, as per requirements the blocks are rotated and join to get the final image.

INTRODUCTION
Image processing is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or, a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it.

a. Image Processing Elements

Why do we need Image Processing?
• Improvement of Pictorial information for human perception
• Image processing for autonomous machine application
• Efficient storage of Image
• Faster & efficient Transmission over communication channels.

In electrical engineering and computer science, image processing is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or, a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it. Image processing usually refers to digital image processing, but optical and analog image processing also are possible. This article is about general techniques that apply to all of them. The acquisition of images (producing the input image in the first place) is referred to as imaging.

TYPICAL OPERATIONS
• Euclidean geometry transformations such as enlargement, reduction, and rotation
• Color corrections such as brightness and contrast adjustments, color mapping, color balancing, quantization, or color translation to a different color space
• Digital compositing or optical compositing (combination of two or more images), which is used in film-making to make a "matte"
• Interpolation, demosaicing, and recovery of a full image from a raw image format using a Bayer filter pattern
• Image registration, the alignment of two or more images
• Image differencing and morphing
• Image recognition, for example, may extract the text from the image using optical character recognition or checkbox and bubble values using optical mark recognition
• Image segmentation
• High dynamic range imaging by combining multiple images
• Geometric hashing for 2-D object recognition with affine invariance
INTRODUCTION TO QUAD TREE

a) Quad tree: A quad tree for representing a picture is a tree in which successively deeper levels represent successively finer subdivisions of picture areas. Each node represents the quadrant of its parent. The tree is filled up by a process of subdividing an image matrix into four quadrants recursively until the quadrant is one solid color. A Quad tree is a tree whose nodes are either leaves or have four children. The children are ordered: 1, 2, 3 and 4.

b) Quad tree for pictures: A quad tree for, or representing, a picture is a quad tree whose leaves represent areas of the picture; each leaf is labeled with the color of the area of the picture. Each node is associated with the square region of the picture. The root is associated with the entire picture. Besides the root, each other node is associated with one of the four quadrants of its parent’s square. The ith child is associated with the ith quadrant of its parent’s square. No parent node may have all its descendants represent the same region of the picture, if all descendants have the same color.

c) Quad Tree Types: Quad trees may be classified according to the type of data they represent, including areas, points, lines and curves. Quadtrees may also be classified by whether the shape of the tree is independent of the order data is processed. Some common types of quadtrees are:

1) The region quadtree: 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). The region quadtree is not strictly a ‘tree’ - as the positions of subdivisions are independent of the data. They are more precisely called ‘tries’. A region quadtree with a depth of n may be used to represent an image consisting of $2^n \times 2^n$ pixels, where each pixel value is 0 or 1. The root node represents the entire image region. If the pixels in any region are not entirely 0s or 1s, it is subdivided. In this application, each leaf node represents a block of pixels that are all 0s or all 1s. A region quadtree may also be used as a variable resolution representation of a data field. For example, the temperatures in an area may be stored as a quadtree, with each leaf node storing the average temperature over the subregion it represents. If a region quadtree is used to represent a set of point data (such as the latitude and longitude of a set of cities), regions are subdivided until each leaf contains at most a single point.

2) Point quadtree: The point quadtree is an adaptation of a binary tree used to represent two dimensional point data. It shares the features of all quadtrees but is a true tree as the center of a subdivision is always on a point. The tree shape depends on the order data is processed. It is often very efficient in comparing two dimensional ordered data points, usually operating in $O(\log n)$ time. A node of a point quadtree is similar to a node of a binary tree, with the major difference being that it has four pointers (one for each quadrant) instead of two ("left" and "right") as in an ordinary binary tree. Also a key is usually decomposed into two parts, referring to x and y coordinates. Therefore a node contains following information:

- 4 Pointers: quad[‘NW’], quad[‘NE’], quad[‘SW’], and quad[‘SE’]
- point; which in turn contains:
  - key; usually expressed as x, y coordinates
  - value; for example a name

3) Edge quad tree: Edge quad trees are specifically used to store lines rather than points. Curves are approximated by subdividing cells to a very fine resolution. This can result in extremely unbalanced trees which may defeat the purpose of indexing.

d) Quad Tree Properties:

1) Image Representation
2) Decompose space into adaptable cells and each cell (or “bucket”) has a maximum capacity (or “threshold”).
3) When max capacity is reached the cell is split.
4) Algorithms for search are generally \( \log_2 N \), where \( d \) is the dimensionality (4 for true quadtree).
5) Related to \( kd \)-trees, space is partitioned in rectilinear subdivisions. Difference is \( kd \)-trees always bisect space, while quadtrees quarter it.
6) Spatial indexing.
7) Efficient collision detection in two dimensions.
8) View frustum culling of terrain data.
9) Storing sparse data, such as a formatting information for a spreadsheet or for some matrix calculations.
10) Solution of multidimensional fields (computational fluid dynamics, electromagnetism).
11) Conway's Game of Life simulation program.
12) State estimation.

**ROTATION**

a) Rotate Image by \( 90^\circ \) in counter-clockwise direction

1) **Initial Image**: Given is the Image to be rotated through given \( \theta \)

   ![Initial Image](Fig. 3.1 Initial Image)

4 children of the root node respectively represent

a) NE (North East)

b) NW (North West)

c) SW (South West)

d) SE (South East)

2) **Split**: Root node Broken into children. The image can be broken into the 4 quadrants as shown below. It is done up to one level only. Now, we will check whether each node is representing only one solid color or not.

   ![Split Image](Fig. 3.2 Split Image)

Here, each node is not representing only one solid color. So, it is to be broken down further.

3) **Split**: Further each node breakdown of each quadrant into children.

   ![Split Image](Fig. 3.3 Split Image)

Here, each child is representing only one solid color.
5) Join: Here, the join is performed recursively to get the final image. Each single color solid pixel is arranged in the prescribed fashion of four quadrants.

6) Rotate: Here, Rotation is performed in the already described fashion and each quadrant is rearranged.

7) Join: The final image after join is as shown. This is the required counter clockwise rotation of the image.

ALGORITHM

1. Recursive Algorithm for Rotation:
   a) If root is a leaf, return.
   b) Rotate the children of the root.
   (NE, NW, SW, SE) \( \rightarrow \) (NW, SW, SE, NE)
   c) Recursively perform the rotation in the subtrees of the root.
   d) Complexity is \( Q \) (size of quad tree).
   e) Quad tree size = \# nodes.
   f) Quad tree size is usually substantially less than matrix size.

2. Rotations Possible:
   a) Clockwise rotations by 180 and 270 degrees are similar.
   b) Counterclockwise rotations by 90, 180, and 270 degrees are similar.
   c) Rotation by 360 degrees is null.

3. Image Representation:
   a) \( n \times n \) matrix.
      1. \( Q (n^2) \) space.
      2. \( O (n^2) \) time for rotation, scaling, union, intersection and so on.
   b) Quad tree.
      1. \( O (n^2) \) space.
      2. \( O (n^2) \) time for rotation, scaling, union, intersection and so on.

For some images, the space and time could be as little as \( O (1) \) (e.g., all white or all black images).

CONCLUSION

This paper has worked over the image rotation using quad trees, a major concern in advanced data structures. Quad trees are very efficient in representing the 2-D images. So, we have used this property of Quad trees to work over a major operation on images i.e. rotation. The images can be rotated only at 90\(^\circ\), 180\(^\circ\), 270\(^\circ\) or 360\(^\circ\). We have observed that image rotation algorithm is very easy to understand for a person and is present with very much less computations and calculations. SO, for simple rotations, we can use this algorithm so as to maintain the soul of the software engineering i.e. complexity. The work done with the quad trees is space as well as time efficient.

For future work, we plan to find the pitfalls of using quad trees to work with image rotations at some other angels that are not multiple of 90\(^\circ\). We will compare the quad tree algorithms with some other image processing algorithms and will try to make quad trees to work on more complex image rotation situations.
We will construct the algorithm to handle such a complex scenario with an acceptable level of accuracy and low computational overhead.