Application of Quadtrees in Image Processing

Matthew Kevin Amadeus 13518035

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

mkamadeus.mka@gmail.com 13518035@std.stei.itb.ac.id

*Abstract*—This paper elaborates the practical uses of quadtrees; a data structure that stores information of an image in a tree-like manner. With quadtrees, image can be represented quite efficiently and a lot of algorithms depend on this quadtree model of an image in order for it to work.

*Keywords*—quadtrees, image processing, trees, recursive.

# I. Introduction

Whether you are a person that never touches programs relating to graphic design, you have encountered or will encounter things relating to image processing. Back in the days, analog image processing is preferable, as there are no technologies to do it digitally; yet. In this era, it is very common to use computers as the brains to do that process digitally. Photo editing, 2D or even 3D animations are all created in a computer.

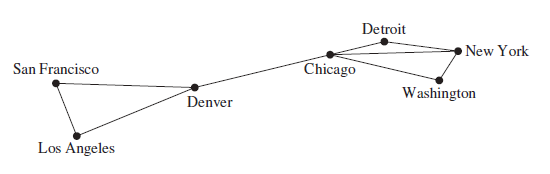
Humans and computers are different in the way of perceiving visual contents, such as images. Humans can directly perceive it as light that is being processed by our brains as colors, as for computers, it can only read numbers as an input. By using an appropriate data structure, such as *quadtrees*, can help computers to access information faster and efficiently.

# II. Theoretical Basis

## A. Graph

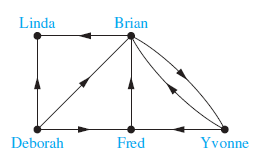
A graph G = (V, E) consists of V, a nonempty set of nodes (sometimes called vertices), and E, a set of edges. For each edge, it has either one or two nodes connected to it, called its endpoints. An edge is said to connect its endpoints.

Based on the connections that the graph makes; graphs are differentiated a *simple graphs* and *multiple graphs*; where for each edge in the graph are connected to two nodes with exception of no two pair of nodes are connected together. In multiple graphs, it is allowed two have more than one edge that connects to the same pair of nodes.



*(Fig 1: Simple graphs example. [XX])*

Based on the direction of the graph; graphs are differentiated into *directed* and *undirected* graphs. A directed graph has directions; if a path from node to node exists, there may or may not be a path from node to node *.* In undirected graphs, the edge works in both directions.



*(Fig 2: Directed graph example. [XX])*

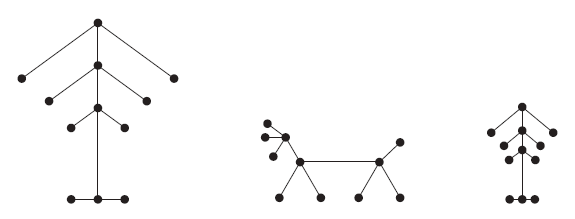
There are some other terminologies that is commonly used in graph. Two nodes and are called *adjacent* if is the endpoint of and vice-versa. Such edge that connects the two edges is to be called *incident* with the nodes and . A *path* is called a sequence of edges that connects the edge and edge . A path that is ends at the starting node is called a *circuit*.

Paths can be created by traversing a graph. A graph traversal has two common modes; *Breadth First Search* (commonly abbreviated as BFS) and *Depth First Search* (commonly abbreviated as DFS). A BFS traversal is usually done by using a queue data structure, while a DFS traversal is usually done by using a stack data structure. The traversal order holds by their own names; BFS will traverse for each existing paths, while DFS will traverse all through until the end.

Data structure wise, a graph is usually created in the form of these: *adjacency list, incidence list, adjacency matrix, incidence matrix,* and *edge list.* Each of the representation mentioned has it own use cases and weaknesses; it all depends on how the user itself wanted to use it.

## B. Tree

A tree is just a special form of a graph. It is basically an undirected graph, with no simple circuit inside of it. An undirected graph is said to be a tree if there exists such unique simple path for each pair of two nodes. Another variation of the tree is called a *rooted tree*, which means a tree with one node selected as the *root*, while every other node is directed further away from its root by the edges. Some terminologies in trees are related to its node. *Internal nodes* are nodes of the tree that have children (descendants), while *leaves* are nodes of the tree that have no children.

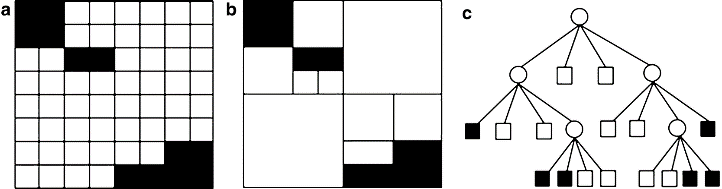


*(Fig 3: Examples of possible trees. [XX])*

*A n-ary tree* is the is a form of rooted trees, where each internal node has maximum of children. Commonly, as the world of computers exists of zeros and ones, a *binary tree* is used. A binary tree is a n-ary tree that consists of maximum two children. It has so many uses in optimizing algorithms and storing data efficiently, for example the BST (Binary Search Tree).

## C. Quadtree

As mentioned before, the most common form of trees that is used in computer science is of course; *the binary tree*. With quadtrees, things are a little bit different. Binary trees, as named for its count of children, has two children. Whereas for *quadtrees*, they have at most *four* children (). Formally elaborated, Quadtrees are hierarchical spatial tree data structures that are based on the principle of recursive decomposition of space [1].



*(Fig 1: A simple diagram explaining quadtrees. [2])*

Quadtrees work in a recursive manner; some quadtrees like shown in Figure 1 stores information of regions. It divides a rather large image into four subsections (northwest, northeast southeast, and southwest). At that example, it divides until a region is homogenous, hence no need of further decomposition. This property of the quadtree makes it rather efficient in storing spatial data.

Quadtrees are a quintessential data structure to store spatial related data. Figure 1 shows a variation of a quadtree, namely the region quadtree. Another variation of the quadtree is to store information of a lot of points in an efficient manner. The principles stay the same; recursively divides the region into four quadrants until there is no need of dividing it, in this case there exists only one point in that region. With that perspective of quadtrees storing spatial data, it can hold so much potential into developing algorithms related to image processing.

## D. Image Processing

By definition, image processing is a way to perform operations to an image, in order to get a certain information from an image, or even modifying it into our own will. The output wanted may be another modified image or even just a simple output showing what the image holds [3].

The term image processing is known before the ages of computers, which was done to photos manually by hand or other methods. Nowadays, image processing is being done in computers using calculations and algorithms that has developed during the years.

The simplest form of image processing is just a simple transformation of an image, such as resizing an image. A term that relates to resizing an image is usually *zooming*; which in practice is just adding more pixels to an image, whether you want it to be zoomed in or out. Another great example is by simply manipulating the pixels that make up the image into something that we want it to be.

## D. Image Compression

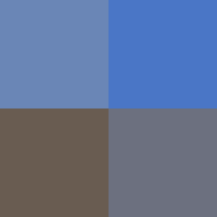
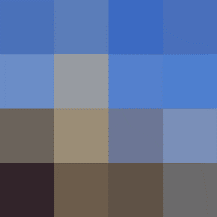
Image compression is a way to reduce the size of an image file with such ways that the image can be perceived almost or perfectly with its uncompressed counterpart. There are two types of image compression: *lossless* and *lossy*. In general, lossless compression doesn’t remove or change any data with a bonus of the size is reduced; thus, no quality is compromised. It is a different story with a lossy image compression. The image may be modified as the compression algorithms want it to be, so the quality may be reduced as well; and there’s no turning back—unlike the lossless image compression as it is reversible.

# III. Image Processing Examples in Quadtrees

Given an image that has been converted into its quadtree form, containing the average color in RGB format and its standard deviation for each quadrant for the image. In this example, as quadtree is scaled by the power of two, we will be using an image of size 2n.2n pixels. It would make things easier for the algorithms explained below to work. If a different sized image is going to be used, a different approach must be made.

## A. Image Scaling (Downscaling)

By how quadtrees work, image scaling by the factor of two is a trivial task. After the quadtrees has been made, by simply returning the image at the wanted level of the tree. For instance, suppose that you’ve constructed the quadtree from a 512x512 image. Keep in mind that each node contains the average color of that region. With an input of level 1 (the root node), the quadtree will return a 1x1 image, with the average color of the whole image. With an input of level 2, the quadtree will return a 2x2 image, consisting of four quadrants with each quadrant’s average color. The pattern may be continued on until the actual image’s original size.

*(Fig 1: 2x2, 4x4, and 64x64 scaling. Source: Author)*

As mentioned before, each node of the quadtree contains both the average color and the standard deviation of the colors. By simply returning the average color from each region, we can get a simple downscaling of the image. The author made a simple script using Python and the OpenCV library to load the image, then manipulating it. Below here is the code.

# Procedure for leveled scaling

def createLeveledScaling(image, level):

    resultHeight = 1<<level

    resultWidth = 1<<level

    imageResult = np.zeros((resultHeight,resultWidth,3), np.uint8)

    if(level==0):

        return image.mean

    else:

        return setQuadrants(

            imageResult,

            createLeveledScaling(image.imageNW, level-1),

            createLeveledScaling(image.imageNE, level-1),

            createLeveledScaling(image.imageSE, level-1),

            createLeveledScaling(image.imageSW, level-1),

        )

# Procedure for exporting to folder

def exportScaling(filename, show=False):

. . . *(redacted code)*

    for i in range(0,10):

        size = 1<<i

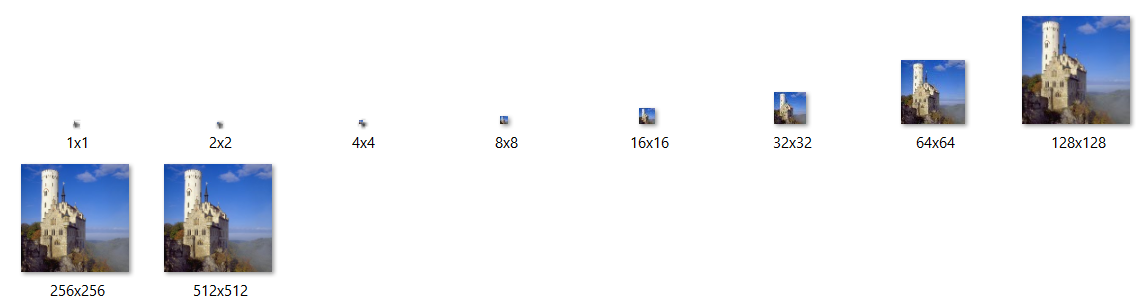
        result = createLeveledScaling(Q, i)

        if(show):

            showImage(result)

. . . *(redacted code)*

The procedure createLeveledScaling is a procedure used to create a scaled image at a certain level of the quadtree. The other procedure, exportScaling is used to output the scaled image into a folder. The author made this procedure in order to show all the possible scaling that quadtree can make.



*(Fig 1: Results shown in a folder. Source: Author)*

Is the scaling useful? Well, it depends on how the image will be used. The user may just adjust the scaling ratio and judge the result by eye, to see which one is the best. The implementation that the author made may not be very suitable for general image scaling, as it blindly takes the average color. The author also made a graph about the image size compared to the dimension of the image.

*(Plot 1: Graph of image size growth compared to its dimension. Source: Author)*

By making the plot, it can be deduced that the growth of the image size can be modelled as *a quadratic equation* (to the height or width). In other words, the image grows size grows *linearly* when compared to the number of pixels. Note that all of the images are exported from the program from OpenCV.

## B. Image Blending

OpenCV itself has its own procedure to blend two image, by using the addWeighted procedure. The author also made an implementation using quadtrees, with the same principles as that procedure. Image blending is simply just averaging two colors together at a certain level of two quadtrees. By averaging two colors without weights (meaning: both colors have equal weight; 50%:50%).

# IV. Quadtree Based Lossy Image Compression

The author implemented a lossy image compression using quadtree. This is merely a concept based on an idea of only returning some colors with a low enough standard deviation. This lossy compression technique uses a breadth first search (BFS) algorithm in order to process each of the region of the quadtree. This lossy image compression was inspired by a quadtree based art on an image [XX]. The lossy compression is made similar to that quadtree art.

The algorithm is highly based on the averaged standard deviation of the RGB values. The author tests for the image of size 512x512, with the same output size. The settings for the default standard deviation limit is set to be 10.0, which the author highly recommends if the purpose is to test for the effectiveness of the lossy compression. If it’s more than 10.0, the image quality is starting to look very degraded.

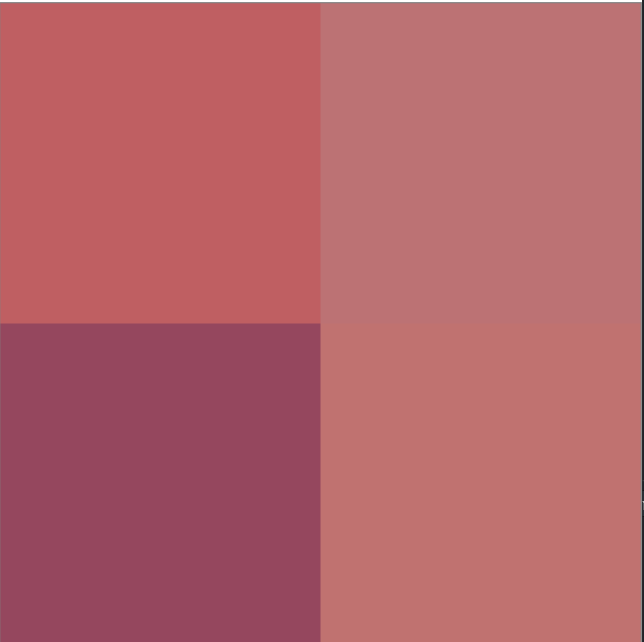
Because OpenCV gives the standard deviation in [B, G, R] format (in the form of list), so the author averages the three standard deviation and crunch it into one number. That number is further used to determine whether the BFS should continue or stop at that point.

This algorithm’s main purpose is generally to just create a concept of a lossy compression based on a quadtree; hence the quality of image may suffer because of that. This compression actually is pretty good at compressing image into smaller size, but it destroys the image quite bad.

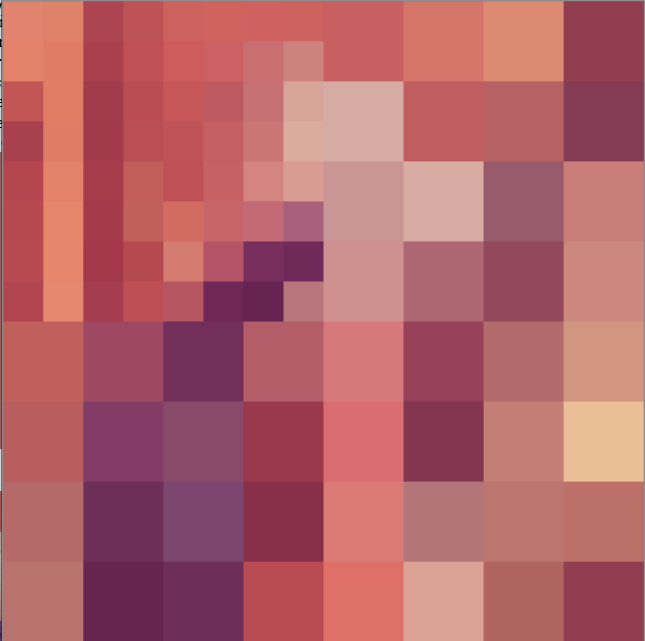
To show how the algorithm works, in this paper the author will be explaining it by showing the images that the algorithm takes place. The author will be using the Lenna image (with size of 512x512 pixels) to show the steps.

First, a quadtree representation of the image will be created. This process will take the longest time, as there will be a lot of regions to be process. This may be a weakness of the author’s implementation, as the author constructed the tree top down; not bottom up. By constructing it top down, there are overlapping regions that are processed multiple times, namely at the process of finding the average color and deviation. Statistically speaking, both of those two components from two different regions can be merged into one by simply averaging it again.

After the quadtree representation has been made, the BFS traversal of the quadtree will begin. The algorithm will push the whole region to the queue. After that, for every region in the front of the queue will be checked; whether it satisfies the required deviation limit and the required level of segmentation. Below is shown the process for segmenting the image, with the minimal deviation of 8 and depth of 5.



*(Fig 1: Image of Lenna (512x512), first step of BFS decomposition. Source: Author)*



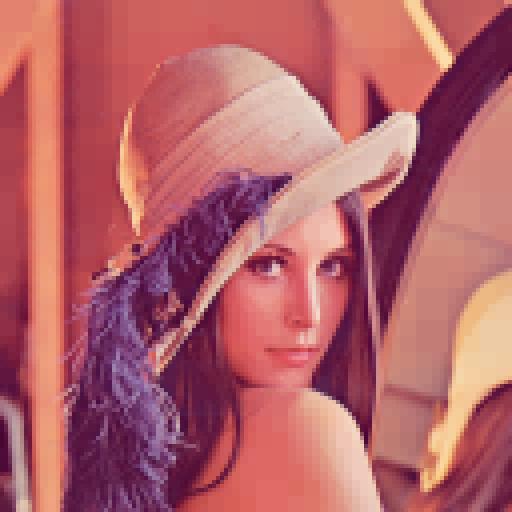
*(Fig 1: Image of Lenna (512x512), one of the steps of image segmentation to achieve compression. Source: Author)*

If observed closely, the segmentation of certain region won’t continue if the deviation already satisfies the required condition, like shown in the figure below.



The BFS traversal of the tree also has a certain pattern to it; the author chooses to do BFS traversal from the NE region, NW region, SW region, and the SE region, respectively. This order of traversal doesn’t really affect the end result of image compression, thus can be ignored.

As mentioned before, an appropriate value of



*(Fig 1: Image of Lenna (512x512), segmented by STD of 5.0. Source: Author)*



*(Fig 1: Image of Lenna (512x512), segmented by STD of 15.0. Source: Author)*

# V. Conclusion

A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

# VI. Appendix

The program that the author creates can be access in this GitHub link :

<https://github.com/mkamadeus/Discrete-Mathematics-Quadtree-Decomposition>

# VII. Acknowledgment

The preferred spelling of the word “acknowledgment” in American English is without an “e” after the “g.” Use the singular heading even if you have many acknowledgments. Avoid expressions such as “One of us (S.B.A.) would like to thank ... .” Instead, write “F. A. Author thanks ... .” Sponsor and financial support acknowledgments are placed in the unnumbered footnote on the first page.

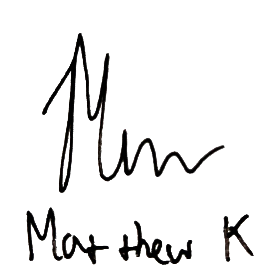
# References

1. S. Aluru, “Handbook of Data Structures and Applications, Second Edition”. Florida: CRC Press, 2018, pp. 309.
2. Vassilakopoulos M., Tzouramanis T. (2018) Quadtrees (and Family). In: Liu L., Özsu M.T. (eds) Encyclopedia of Database Systems. Springer, New York, NY
3. <https://sisu.ut.ee/imageprocessing/book/1>
4. https://github.com/fogleman/Quads
5. W.-K. Chen, *Linear Networks and Systems* (Book style)*.* Belmont, CA: Wadsworth, 1993, pp. 123–135.
6. H. Poor, *An Introduction to Signal Detection and Estimation*. New York: Springer-Verlag, 1985, ch. 4.
7. B. Smith, “An approach to graphs of linear forms (Unpublished work style),” unpublished.
8. E. H. Miller, “A note on reflector arrays (Periodical style—Accepted for publication),” *IEEE Trans. Antennas Propagat.*, to be published.
9. J. Wang, “Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style—Submitted for publication),” *IEEE J. Quantum Electron.*, submitted for publication.
10. C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.

# PeRNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 3 Desember 2017



Matthew Kevin Amadeus

13518035