General 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.

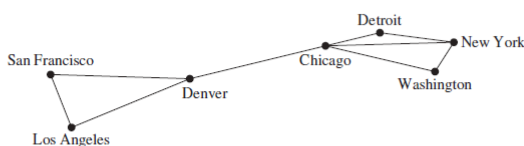
Humans and computers are different in the way of perceiving visual contents, such as images. Humans can directly perceive it as light that is being processed by our brains as colors, as for computers, it can only read numbers as an input. By using an appropriate data structure, such as *quadtrees*, can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

II. Theoretical Basis

A. Graph

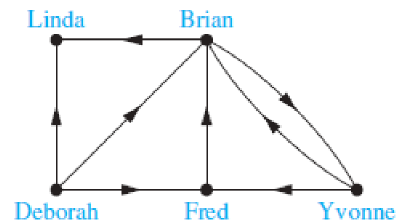
A graph G = (V, E) consists of V, a nonempty set of nodes (sometimes called vertices), and E, a set of edges. For each edge, it has either one or two nodes connected to it, called its endpoints. An edge is said to connect its endpoints.

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. [1]*)*

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 u to node v exists, there may or may not be a path from node v to node u. In undirected graphs, the edge works in both directions.



(Fig 2: Directed graph example. [1]*)*

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

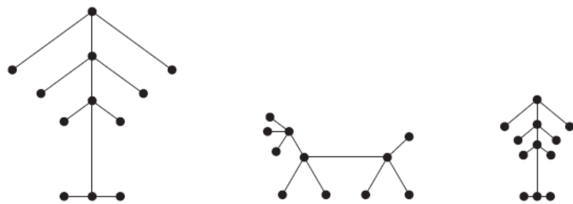
Paths can be created by traversing a graph. A graph traversal has two common modes; *Breadth First Search* (commonly abbreviated as BFS) and *Depth First Search* (commonly abbreviated as DFS). A BFS traversal is usually done by using a queue data structure, while a DFS traversal is usually done by using a stack data structure. The traversal order holds by their own names; BFS will traverse for each existing path, 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 its own use cases and weaknesses; it all depends on how the users theriselves 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.

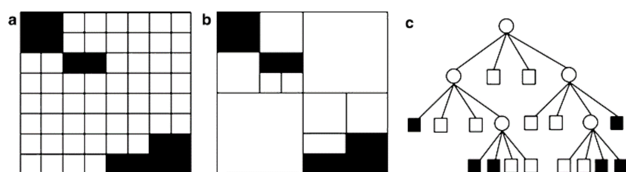


(Fig 3: Examples of possible trees. [1]*)*

A n-ary tree is the is a form of rooted trees, where each internal node has maximum of n 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 ($n = 4$). Formally elaborated, Quadtrees are hierarchical spatial tree data structures that are based on the principle of recursive decomposition of space [2].



(Fig 4: A simple diagram explaining quadtrees. [3] *)*

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 to modify it into our own will. The output wanted may be another modified image or even just a simple output showing what the image holds [4].

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 have 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

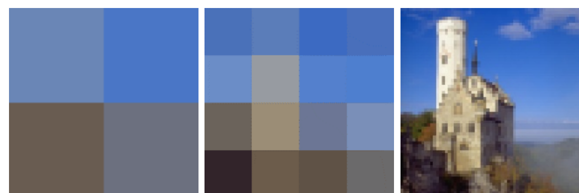
Image compression is a way to reduce the size of an image file with such ways that the image can be perceived almost or perfectly with its uncompressed counterpart. There are two types of image compression: *lossless* and *lossy*. In general, lossless compression doesn't remove or change any data with 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 $2^n.2^n$ 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. Keep in mind; most of these applications are meant to demonstrate how quadtrees store spatial data.

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.

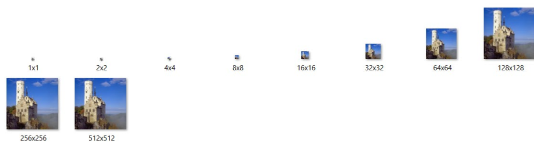


(Fig 5: 2x2, 4x4, and 64x64 scaling. Source: Author)

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.

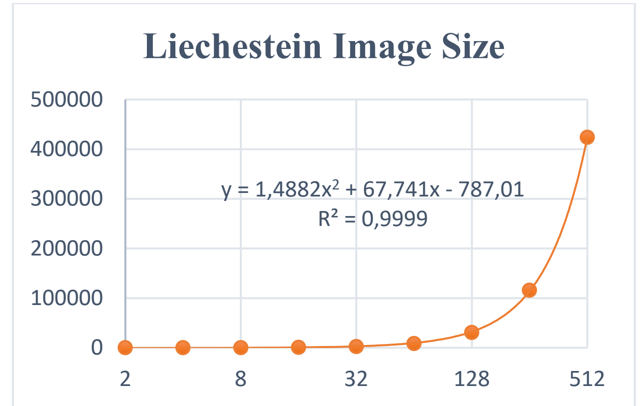
```
# Procedure for leveled scaling
def createLeveledScaling(image, level):
    resultHeight = 1<<level
    resultWidth = 1<<level
    imageResult = np.zeros((resultHeight,resultWidth,3), np.uint8)
    if(level==0):
        return image.mean
    else:
        return setQuadrants(
            imageResult,
            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 via quadtrees actually 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 images, 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. For example, by averaging two colors without weights (meaning: both colors have equal weight; 50%:50%) can be considered image blending. The author chooses to use the airplane and Lenna image, with this outcome below.

(Fig 6: A 50:50 and a 70:30 image blending of Lenna and Airplane. Source: Author)

When changing the weight of the image, the outcome will differ. This image blending actually shows a property of an image, namely *opacity*. A 50:50 blending is actually just blending two images with opacity of 50% into one. As shown above, with a different alpha value (opacity) of image will create a different picture. This property is widely used in many graphical editing applications, such as Adobe Photoshop.

How does one calculate a weighted average color? It is by simply multiplying it by the alpha value. Below is given author's code on the weighted average color.

```python
def imageUnion(image1, image2, level, alpha):
    resultHeight = image1.height
    resultWidth = image2.width
    imageResult = np.zeros((resultHeight, resultWidth, 3)
, np.uint8)

    if(level==0):
        b1, g1, r1 = image1.mean
        b2, g2, r2 = image2.mean
        br, gr, rr = b1*alpha + b2*(1-
alpha), g1*alpha + g2*(1-alpha), r1*alpha + r2*(1-alpha)
# Note that OpenCV outputs color in [B, G, R] format!

        return (br, gr, rr)
    else:
        ... (redacted code, dividing by 4 quadrants)

    return imageResult
```

The highlighted part is actually what matters most; it is how the weighted average color being calculated. The mathematical formula looks like defined below.

$$avgColor = \propto . color1 + (1 - \propto). color2$$

IV. Quadtree Based Lossy Image Compression

A. Introduction to the 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 [5]. 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, as in the default configuration the quality has suffered a lot.

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 is actually pretty good at compressing image into smaller size, but it destroys the image quite bad.

B. Steps of the Algorithm

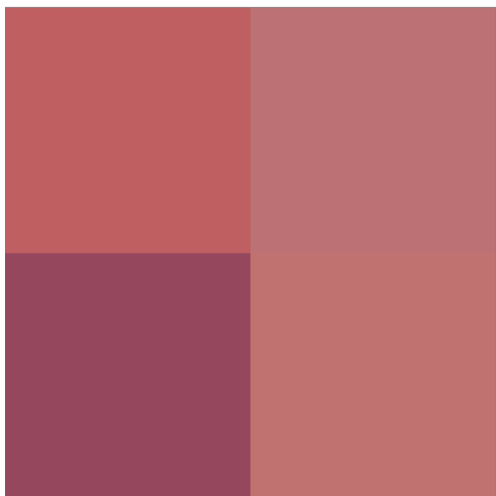
To show how the algorithm works, in this paper the author explains it by showing the images that the algorithm took place. The author used the Lenna image (with size of 512x512 pixels) to show the steps. Before the explanation, here is the truncated code of the BFS based algorithm that the author made.

```python
# Procedure for segementing a quadtree image
def quadtreeSegmentation(filename, limit=7, stdLimit=10.0
, write=False):
   ... (redacted code)

    # Define queue for BFS
    q = []
    coordinateQueue1 = []
    coordinateQueue2 = []
    levelQueue = []

    q.append(Q)
    coordinateQueue1.append((0, 0))
    coordinateQueue2.append((imageInput.shape[0], imageIn
put.shape[1]))
    levelQueue.append(1)

...(redacted code)

    while(len(q)!=0):
        # Pop front of all queues
        currentNode = q.pop(0)
        y1, x1 = coordinateQueue1.pop(0)
        y2, x2 = coordinateQueue2.pop(0)
        currentLevel = levelQueue.pop(0)

        # Find midpoint
        halfX = (x1+x2)//2
        halfY = (y1+y2)//2

        # If the STD is still larger than the stdLimit...
        if(currentNode.std>=stdLimit and currentLevel<=li
mit):
            ...(redacted code, here is where the
segmentation happens; see the GitHub Link for more
information!)

    print("Segmentation complete!")
    showImage(imageResult)
    cv2.waitKey(0)
    cv2.imwrite(resultPath + "lenna" + str(stdLimit) + "_
" + str(limit) + ".png", imageResult)
```

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.

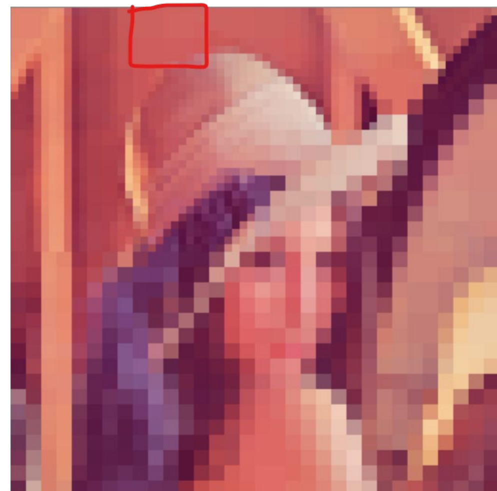
After the quadtree representation has been made, the BFS traversal of the quadtree will begin. The algorithm will push the whole region to the queue. After that, for every region in the front of the queue will be checked; whether it satisfies the required deviation limit and the required level of segmentation.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



(Fig 7: Image of Lenna (512x512), first step of BFS decomposition. Source: Author)

(Fig 8: Image of Lenna (512x512), one of the steps of image segmentation to achieve compression. Source: Author)

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

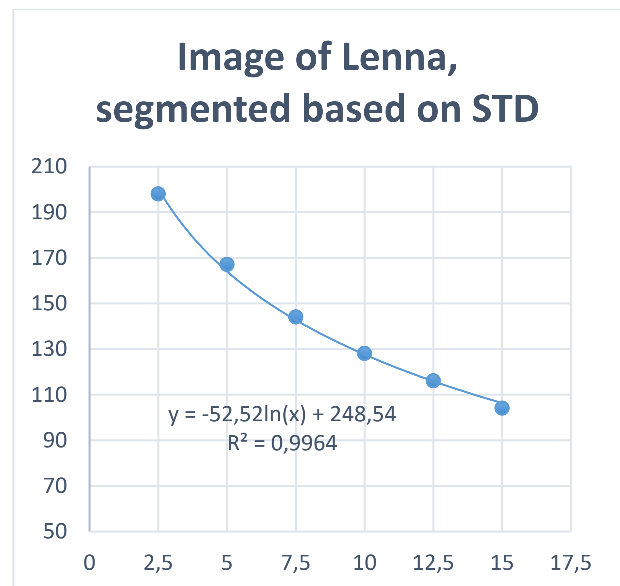


(Fig 9: Image of Lenna (512x512) while segmenting, with a marked region that did not segment further. Source: Author)

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.

C. Compressed Image Size and Their Relationship

How well does the compression work? Well again, it depends on what the aim of the compression is. For creating an "artistic" image (which can be achieved as shown in the reference [5]), the size will tend not to be bigger than going for quality. The author has supplied a plot, explaining the correlation of the image compression size and the deviation given by the user. Again, the image of Lenna will be used here.



(Plot 2: Graph of the size of the image of Lenna when compressed with different STD. Source: Author)

The author can not really explain about how can the deviation and image size correlate with each other. Thus, the author suggested that a lot more test must be taken. As the algorithm's

way of determining a region needed to be segmented further is by its deviation, and the deviation will of course, vary between images. With that reasoning, the author thinks that it is not really possible to find a direct formula to know about how the image will correlate with each other. Thus, the author chose to do the tests statistically based.

With the help of test statistics, the author may or may not find the difference between two images. The author chooses to use two images, which are the Liechtenstein image and the Airplane image. Using a method called *paired t-test*, we can find how two data samples be paired with each other. What the aim is to know how well the two samples align. After crunching some data, the author created this table.

STD Limit	Liechtenstein (KB)	Airplane (KB)	Difference
2,5	148	158	-10
5	127	129	-2
7,5	116	117	-1
10	108	109	-1
12,5	101	102	-1
15	93,3	94,5	-1,2

STD Diff	3,597
Mean Diff	-2,7
T0	-1,83865
N	6
Degree of Freedom	5

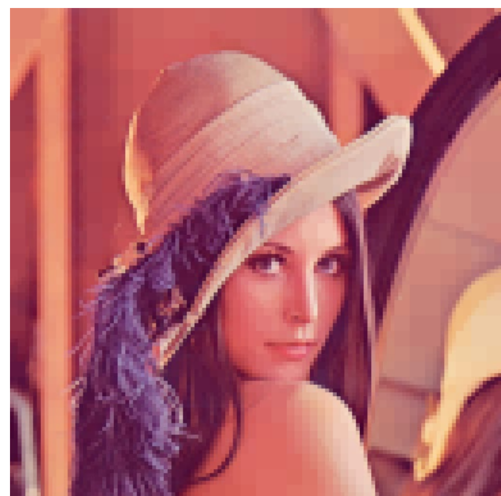
(Table 1: Table for the t-test method. Source: Author)

With a confidence interval of 90%, the author found out the alpha; it is about 0.05 (5%). The author defined two hypotheses: (1) the average difference is equal to 0 (*the H0*), or (2) the average difference is not equal to 0 (*the H1*). By using the *t-table*, because T0 is not in the rejection area, H0 is proven to be failed to reject. Hence, because there is enough evidence of the average difference is equal to 0, it is safe to say that the to sample are alike (with 90% confidence interval).

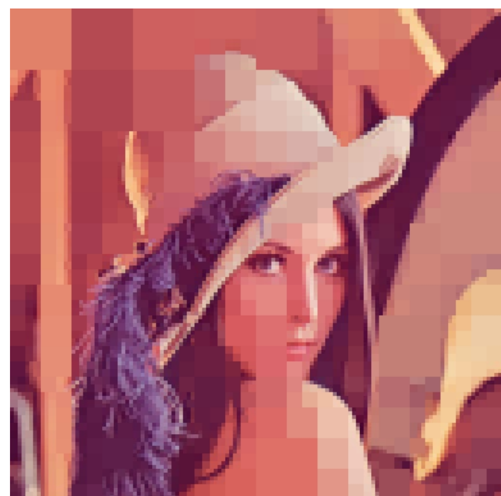
D. Gallery of Examples

As mentioned before, an appropriate value of should be chosen to get a balance between the quality of image and size. Below is shown a gallery of some image with different deviation and depth limit settings.

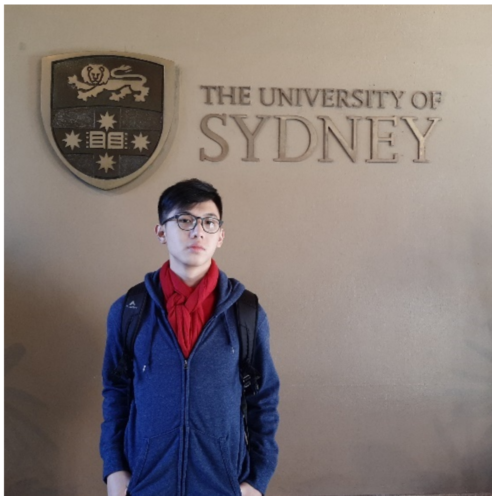
(Fig 10: Initial image of Lenna (512x512). [5]*)*



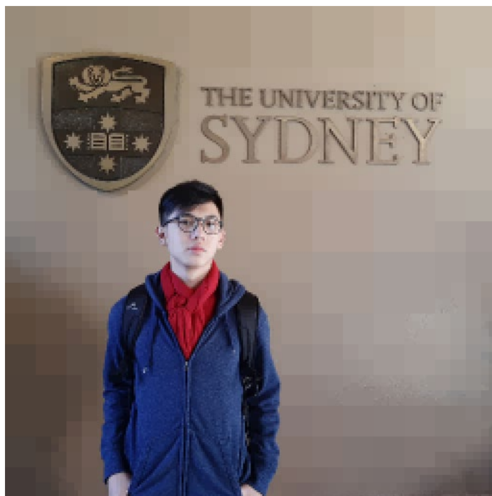
(Fig 11: Image of Lenna (512x512), segmented by STD of 5.0 with depth of 7. Source: Author)



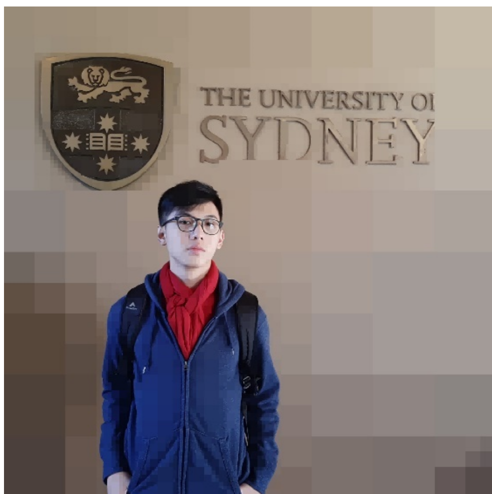
(Fig 12: Image of Lenna (512x512), segmented by STD of 15.0 with depth of 7. Source: Author)



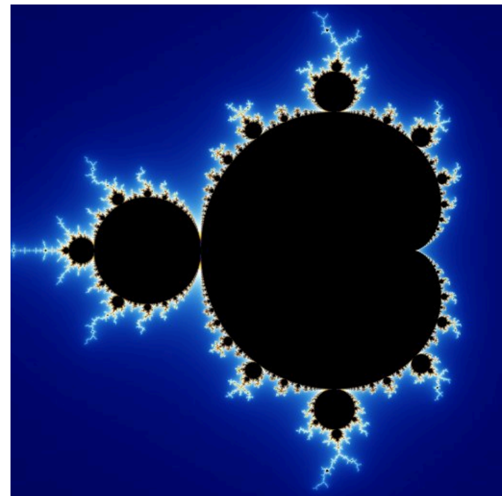
(Fig 13: Initial image of author (512x512 and 2048x2048). Source: Author)



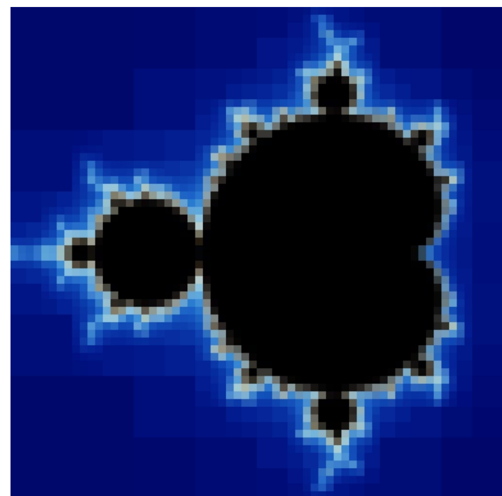
(Fig 14: Image of author(512x512), segmented by STD of 5.0 with depth of 8. Source: Author)



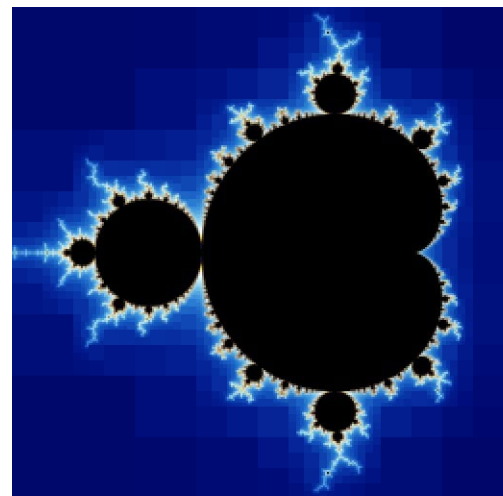
(Fig 15: Image of author(2048x2048), segmented by STD of 10.0 with depth of 10. Source: Author)



(Fig 16: Initial image of the Mandelbrot Set (512x512). [6]*)*



(Fig 17: Image of the Mandelbrot Set (512x512), segmented by STD of 5.0 with depth of 6. Source: Author)



(Fig 18: Image of the Mandelbrot Set (512x512), segmented by STD of 7.5 with depth of 8. Source: Author)

V. Conclusion

Quadtrees hold a really important role in spatial data storing. With these applications shown, such as scaling, blending, and

image compression, it can be shown that quadtrees are really important, especially when the goal is to segment an image. However, the construction method that the author show must be improved further as a bigger image will take a long time to construct the quadtree.

VI. Appendix

The program that the author creates can be access in author's GitHub (https://github.com/mkamadeus/Discrete-Mathematics-Quadtree-Decomposition), and there may be more images to be explored and some of the experimentation that the author made.

VII. Acknowledgment

First and foremost, I wanted to thank God for giving me the passion in writing this paper. I also wanted to thank my parents; without them I will not be able to write this thesis. I am thankful for my lecturers, Dr. Ir. Rinaldi Munir, M. T., Dra. Harlili S., M. Sc., Fariska Zakhralativa Ruskanda, S.T., M.T., for teaching us such wonderful knowledge about the world of discrete mathematics. My sincere appreciation for my friend, Zefania Praventia for helping me in the making of this paper, especially for helping me prove the correlation of the file size and deviation in a statistic manner.

References

[1] K. H. Rosen, Discrete Mathematics and Its Applications, 8 ed., New York: McGraw-Hill Education, 2019.

[2] S. Aluru, "Quadtrees and Octtrees," in *Handbook of Data Structure and Applications*, Florida, CRC Press, 2018, pp. 309-313.

[3] "Semantic Scholar," [Online]. Available: https://www.semanticscholar.org/paper/Constant-time-neighbor-finding-in-quadtrees%3A-An-Aizawa-Motomura/89fe1c4143ab5b2162b5d4caa6c0e863fea42d4b/figure/4. [Accessed 4 December 2019].

[4] University of Tartu, "University of Tartu," [Online]. Available: https://sisu.ut.ee/imageprocessing/book/1. [Accessed 2 December 2019].

[5] "Wikipedia," [Online]. Available: https://upload.wikimedia.org/wikipedia/en/7/7d/Lenna_%28test_image%29.png. [Accessed 21 November 2019].

[6] "Twitter," [Online]. Available: https://pbs.twimg.com/profile_images/1033143908769910786/NyrM5Y4b.jpg. [Accessed 4 December 2019].

[7] fogleman, "GitHub," [Online]. Available: https://github.com/fogleman/Quads.

Statement

With this statement, I hereby declare that this thesis is a product of my hard work, not an adaptation, a translation from other person's work, nor formed by result of plagiarizing.

Bandung, 6 December 2019

Matthew Kevin Amadeus
13518035