Implementation of Tree Concept in Digital Image Processing with Quadtree Decomposition

Jethro Jens Norbert Simatupang - 13523081[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*13523081@std.stei.itb.ac.id*
[2]*jethrojsimatupang@gmail.com*

Abstract—**Quadtree decomposition is a tree-structure based technique that improves the efficiency of digital image processing by dividing an image into hierarchical sub-regions based on pixel homogeneity. This research paper examines the implementation of the quadtree decomposition algorithm, including the recursive decomposition process, data storage in the tree structure, and the efficiency of the algorithm in image processing. The result proves that the quadtree decomposition is effective for processing images containing large homogeneous regions, making it a valuable method for specific image processing applications.**

Keywords—**Digital Image Processing, Image Processing Efficiency, Quadtree Decomposition, Recursive Algorithm**

I. INTRODUCTION

Digital image processing is a branch of computer science that focuses on the analysis, manipulation, and interpretation of visual data in the form of digital images. With the rapid advancement of digital technology, digital image processing has become highly relevant and important in various fields, such as healthcare diagnostics and industrial automation to surveillance systems, autonomous transportation, and even entertainment. The techniques used in digital image processing continue to evolve to meet the demands for more efficient and effective visual data processing.

Digital images, as complex visual representations, contain information that can be processed for various purposes. The field of digital image processing encompasses a wide range of techniques and methodologies aimed at extracting meaningful information from images or transforming them to meet specific objectives. This process typically involves multiple stages, including image enhancement for improved visual quality, segmentation to isolate regions of interest, and pattern analysis for object recognition. Each stage contributes to simplifying and optimizing image data, allowing for more effective processing and decision-making.

Given the increasing complexity of visual data and the demand for more precise results, researchers have continually developed innovative approaches to address various challenges in digital image processing. These challenges include noise reduction, sharpness enhancement, compression, and so on. One technique that can be utilized in improving image processing efficiency is quadtree decomposition, which allows an image to be divided into several homogeneous blocks. Image decomposition using the quadtree concept simplifies and facilitates further analysis of the image since the image processing is not evenly distributed but rather adjusted to the homogeneity of the image.

By organizing image data into a tree structure, quadtree decomposition facilitates efficient storage, processing, and analysis, particularly for large or high-resolution images. This method is widely applicable in tasks such as image segmentation, compression, and spatial analysis. This research paper explores the implementation of the quadtree decomposition technique in digital image processing. It delves into the theoretical foundation of the quadtree approach, its practical applications, and its benefits in addressing contemporary image processing challenges. Through this research, the author aims to highlight the potential of tree-based methods in advancing the efficiency and effectiveness of digital image processing.

II. THEORITICAL BASIS

A. Graph

A graph is a structure that represents discrete objects (vertices) and the relationships between those objects (edges). Graph G is defined as G = (V, E), where V is a non-empty set of vertices, and E is a set of edges connecting pairs of vertices.

Based on the presence of loops or multiple edges, graphs are categorized into two types:

1. Simple graph
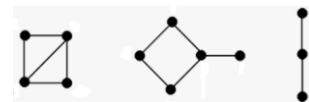   A simple graph is a graph that contains no loops or multiple edges

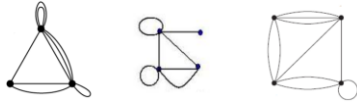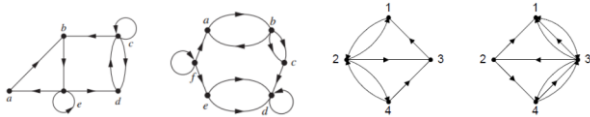**Figure 1.** Simple graph illustration
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

2. Non-simple graph
   A non-simple graph contains either multiple edges or loops. Non-simple graphs are further classified into:
   - Multi-Graph: A graph containing multiple edges.

Figure 2. Multi-graph illustration

- Pseudo-Graph: A graph containing loops.

Figure 2. Pseudo-graph illustration

Based on edge orientation, graphs are divided into two types:

1. Undirected graph

   An undirected graph is a graph where the edges do not have a specific direction.

Figure 3. Undirected-graph illustration

2. Directed graph

   A directed graph is a graph where each edge has a specific orientation or direction.

Figure 4. Directed-graph illustration

There are some graph terminologies that are important in this paper:

1. Adjacent

   Two vertices are said to be adjacent if they are directly connected by an edge.

2. Incident

   For any edge $e = (v_j, v_k)$, e is said to be incident to vertex v_j or vertex v_k.

3. Degree

   The degree of a vertex is the number of edges incident to that vertex.

4. Path

   A path in a graph is a sequence of vertices connected by edges, traversing from one vertex to another.
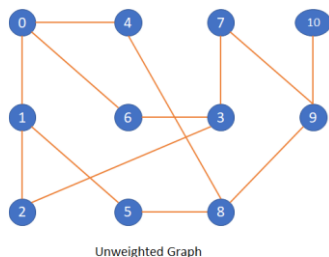

Figure 5. 0, 6, 3, 7, 9, 10 path is the path from vertex 0 to 10

5. Cycle or Circuit

A cycle (or circuit) is a path that starts and ends at the same vertex.
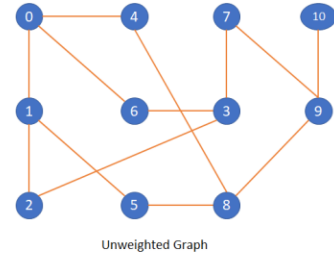
Figure 6. 0, 4, 8, 5, 1, 0 path is a circuit

6. Connected

   Two vertices v_1 and v_2 are said to be connected if there is a path from v_1 to v_2.

7. Connected graph

   A connected graph is a type of graph in which every vertex can be reached from any other vertex through one or more paths.

B. Tree

A tree is an undirected graph that is connected and does not contain any circuits (cycles). A tree where one of its vertices is treated as the root and its edges are directed to form a directed graph is called a rooted tree.
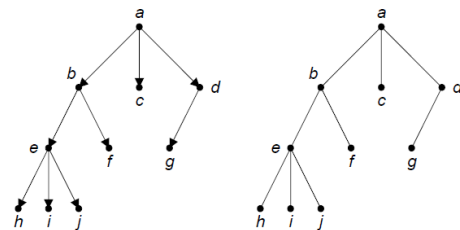

Figure 7. (a) Rooted tree, (b) as an agreement, the arrows on the edges can be discarded

There are some important rooted tree terminologies in this paper:

1. Child

   A child is a vertex directly connected to another vertex as a result of an outgoing branch from that vertex.

2. Parent

   A parent is a vertex directly connected to another vertex through a branch leading to the latter.

3. Siblings

   Siblings are vertices that share the same parent.

4. Subtree

   A subtree is a part of the tree consisting of a specific vertex and all its descendants (children, grandchildren, etc.).

5. Leaf

   A leaf is a vertex in the tree that has no children.

6. Internal Node

   An internal node is any vertex that is not a leaf; it has at least one child.

7. Level

The level of a vertex is its position or depth in the tree, calculated from the root. The root is at level 0, and each subsequent level increases by one.

8. Height or Depth

The height (or depth) of a tree is the number of levels from the root to the deepest leaf.

A rooted tree where each branch vertex has at most n children is called an n-ary tree. An n-ary tree is said to be regular or full if every vertex, except those at the leaf level, has exactly n children.
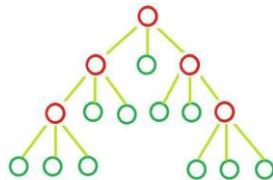

Figure 8. Full 3-ary tree illustration
Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/24-Pohon-Bag2-2024.pdf

C. Digital Image Processing

Digital image processing is a branch of computer science focused on analyzing, manipulating, and interpreting visual data in the form of digital images. A digital image, as a complex visual representation, contains information that can be processed for various purposes. It is represented as an $M \times N$ matrix, where $M \times N$ represents the image resolution and each matrix element represents a pixel.

Digital image processing involves multiple stages, from image enhancement, object segmentation, to pattern analysis and recognition. Over time, various image processing techniques have been developed to address challenges such as noise reduction, sharpness enhancement, compression, and so on. Digital image processing has wide applications in various fields, such as remote sensing, pattern recognition, security, robotics, and the creative industries.

Each pixel in a digital image represents the intensity of light at a specific point, which can be in grayscale (gray levels) for grayscale images or a combination of colors in color models such as RGB for colored images. Additionally, pixel intensity is often represented within a specific range of values, such as 0 to 255 for 8-bit images, indicating the level of brightness or color.

Figure 9. Image noise reduction illustration
Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2019-2020/01-Pengantar-Pengolahan-Citra-Bag1.pdf

D. Quadtree Decomposition

Quadtree decomposition is one of the techniques in image processing that involves dividing an image into blocks that are more homogeneous compared to the image as a whole. At the highest level, if the image is not homogeneous, it is divided into four large blocks. If a block does not meet the homogeneity criteria, it is further subdivided into four smaller blocks. This process continues until each block satisfies the homogeneity criteria or the block size reaches the minimum limit. Image decomposition using the quadtree concept simplifies and facilitates further image analysis since the image processing is not evenly distributed but rather adjusted to the homogeneity of the image.
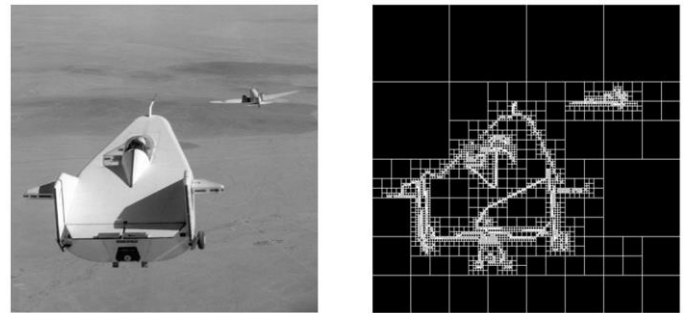

Figure 10. Quadtree decomposition illustration
Source: https://www.mathworks.com/help/images/quadtree-decomposition.html

The quadtree decomposition technique implements the concepts of trees and recursion at a more advanced level. Fundamentally, a quadtree is a full 4-ary tree. The tree data structure is used to divide a two-dimensional space into four sections or quadrants. Each node in the tree represents a smaller area within the two-dimensional space, based on the principle of dividing the space into four parts until homogeneity is achieved.
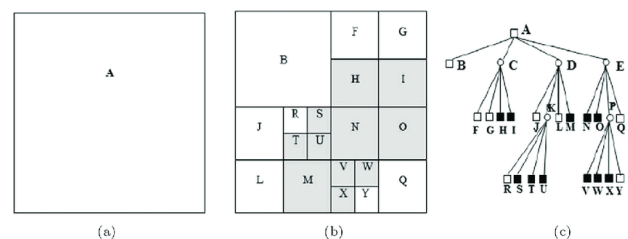

Figure 11. (a) Image, (b) block decomposition of the image, (c) quadtree representation of the blocks in (b).
Source: https://www.researchgate.net/figure/The-Quadtree-decomposition-a-image-b-block-decomposition-of-the-image-and-c_fig3_343874491

III. IMPLEMENTATION

A. Quadtree Decomposition Algorithm

The quadtree decomposition algorithm involves dividing an image into smaller blocks based on the homogeneity of the pixels within those blocks. Below are the general steps of the quadtree decomposition algorithm:

1. Initialization

The digital image is inputted in a two-dimensional matrix format for processing. At this stage, the homogeneity criteria must also be defined, for example, based on the variance of pixel intensity values within a block.

2. Recursive process

The decomposition process begins with the entire image (the highest level) as the initial block. If the initial block is homogeneous, it is not further subdivided. Conversely,

if the block is not homogeneous, it is divided into four sub-blocks (quadrants). This process is repeated recursively for each sub-block until all sub-blocks meet the homogeneity criteria or the block size reaches a predefined minimum limit.

3. Tree reconstruction
   The decomposition result is stored in a quadtree structure, where each node represents one block of the image. Leaf nodes represent homogeneous blocks, while internal nodes have four children representing the sub-blocks.

4. Output
   The algorithm produces a quadtree representation of the image, which can be utilized for various purposes, such as image compression and analysis.

B. Implementation and Testing of the Quadtree Decomposition Algorithm in a Program

Below is the code representing the implementation of quadtree decomposition:

```python
import numpy as np
from skimage import io, color
import matplotlib.pyplot as plt
import os

# Homogenity checking
def is_homogeneous(block, threshold=10):
    return np.var(block) <= threshold

# Decomposition recursion
def quadtree_decomposition(image, x, y, width, height,
threshold, min_size):
    if (width <= min_size or height <= min_size or
          is_homogeneous(image[y:y+height, x:x+width],
threshold)):
        return [(x, y, width, height)]
    half_width = width // 2
    half_height = height // 2
    blocks = []
    # Top left
    blocks += quadtree_decomposition(image, x, y, half_width,
                        half_height, threshold, min_size)
    # Top right
    if x + half_width < image.shape[1]:
        blocks += quadtree_decomposition(image, x +
half_width,
                            y, width - half_width,
                            half_height, threshold,
min_size)
    # Bottom left
    if y + half_height < image.shape[0]:
        blocks += quadtree_decomposition(image, x, y +
half_height,
                            half_width, height -
half_height,
                            threshold, min_size)
    # Bottom right
    if x + half_width < image.shape[1] and y + half_height <
image.shape[0]:
        blocks += quadtree_decomposition(image, x +
half_width,
                            y + half_height, width -
half_width,
                            height - half_height,
                            threshold, min_size)

    return blocks

# Load grayscale image
image = io.imread(r"./test/sample.jpg", as_gray=True) * 255
image = image.astype(np.uint8)

# Quadtree decomposition parameter
threshold = 20
min_size = 3
```

```python
# Algorithm execution
blocks = quadtree_decomposition(image, 0, 0,
image.shape[1],
                    image.shape[0], threshold, min_size)

# Result output
fig, ax = plt.subplots()
ax.imshow(image, cmap='gray')
for x, y, width, height in blocks:
    rect = plt.Rectangle((x, y), width, height,
                edgecolor='red', facecolor='none')
    ax.add_patch(rect)
plt.title('Quadtree Decomposition Result')
plt.show()
```

In the code above, the image is first converted to grayscale using the skimage library. The image needs to be converted to grayscale because the is_homogeneous function utilizes variance (np.var(block)) to evaluate the homogeneity of a block. This variance is calculated based on pixel intensity, which is only relevant for single-channel images such as grayscale. If a color image (RGB) is used without conversion to grayscale, np.var(block) will calculate the variance on a three-dimensional array, which does not align with the expected homogeneity criteria. Once the image is converted to grayscale, it is then converted to an 8-bit data type to simplify decomposition.

Now that the image is converted, the decomposition process is ready to proceed. The main function of this algorithm is quadtree_decomposition, which works recursively to divide the image into four parts if the block is not homogeneous. The homogeneity of the block is checked using the is_homogeneous function, which calculates the variance of pixels within a block and compares it to a predefined threshold. If the variance is below the threshold, the block is considered homogeneous and is not further divided.

In the quadtree_decomposition function, the block division works for both square and rectangular images. For square images, the blocks are divided normally into four square sub-blocks. For rectangular images, block sizes are dynamically adjusted. Each time a block is divided, its dimensions are calculated by halving the previous block's width and height. Additionally, a boundary check ensures that the values of x + half_width do not exceed the image width (image.shape[1]) and y + half_height do not exceed the image height (image.shape[0]). This ensures that the block division remains within the image's dimensions.

After processing the entire image, the resulting divisions are displayed using matplotlib, with each resulting block outlined in red on the image. An example of the quadtree decomposition result using this code is as follows:

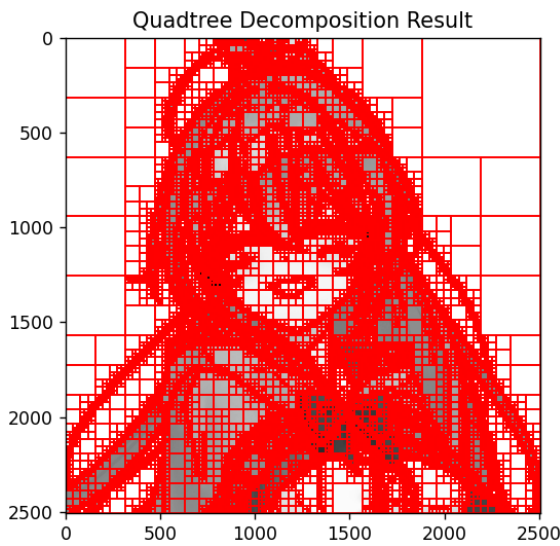**Figure 13.** Sample image
Source: https://x.com/azure_0608_sub

Figure 14. Quadtree decomposition result using the given code

C. Testing Digital Image Processing with the Quadtree Decomposition Technique

To prove that the quadtree decomposition technique can enhance efficiency in digital image processing, testing is required to compares the results of image processing using quadtree decomposition with the results of processing without using the quadtree decomposition. Below is the code used to compare image compression processes with and without the application of the quadtree decomposition technique using the existing quadtree decomposition function:

```
# Compress image with quadtree decomposition
def compress_with_quadtree(image, blocks):
    compressed = np.zeros_like(image)
    for x, y, width, height in blocks:
        mean_value = np.mean(image[y:y+height, x:x+width])
        compressed[y:y+height, x:x+width] = mean_value
    return compressed

# Compress image without quadtree decomposition (simple
downsampling)
def compress_without_quadtree(image, downscale_factor):
    small = cv2.resize(
        image,
        (
            image.shape[1] // downscale_factor,
            image.shape[0] // downscale_factor,
        ),
        interpolation=cv2.INTER_AREA
    )
    return cv2.resize(small, (image.shape[1],
image.shape[0]),
                interpolation=cv2.INTER_AREA)

# Calculate the effective compression ratio of quadtree
def calculate_quadtree_compression_ratio(image, blocks):
    total_blocks = sum([w * h for _, _, w, h in blocks])
    original_pixels = image.size
    return original_pixels / total_blocks

# Load grayscale image
image = io.imread(r"./test/sample.jpg",
            as_gray=True) * 255
image = image.astype(np.uint8)

# Quadtree decomposition parameters
threshold = 20
min_size = 3
```

```
# Execute quadtree decomposition
blocks = quadtree_decomposition(image, 0, 0,
image.shape[1],
                    image.shape[0], threshold, min_size)
compressed_quadtree = compress_with_quadtree(image, blocks)

# Calculate the compression ratio of quadtree
compression_ratio_quadtree =
calculate_quadtree_compression_ratio(image,
                                    blocks)

# Determine downscale factor for similar compression ratio
downscale_factor = int(np.sqrt(image.size / (image.size /
                            compression_ratio_quadtree))
)

# Execute compression without quadtree decomposition
compressed_no_quadtree = compress_without_quadtree(image,
downscale_factor)

# Compute SSIM
ssim_quadtree = ssim(image, compressed_quadtree)
ssim_no_quadtree = ssim(image, compressed_no_quadtree)

# Results output
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Image')
axes[0].axis('off')

axes[1].imshow(compressed_quadtree, cmap='gray')
axes[1].set_title(f'Compressed (Quadtree)\nSSIM:
{ssim_quadtree:.4f}')
axes[1].axis('off')

axes[2].imshow(compressed_no_quadtree, cmap='gray')
axes[2].set_title(f'Compressed (No Quadtree)\nSSIM:
{ssim_no_quadtree:.4f}')
axes[2].axis('off')

plt.tight_layout()
plt.show()

# Save compressed images
io.imsave(r"./test/compressed_quadtree.jpg",
        img_as_ubyte(compressed_quadtree))
io.imsave(r"./test/compressed_no_quadtree.jpg",
        img_as_ubyte(compressed_no_quadtree))
```

The test results below indicate that image compression using the quadtree decomposition algorithm produces smaller file sizes, proving that this algorithm can improve the efficiency of image compression and digital image processing in general.

Figure 16. Image compression results: the original image (left), compression with quadtree decomposition (center), and compression with simple downsampling (right).

Figure 17. Comparison of file sizes for image compression results

However, further testing reveals that quadtree decomposition is less suitable for images with fine details that are evenly distributed.

Figure 18. Comparison of file sizes for different sample image

IV. CONCLUSION

Based on the research conducted, it can be concluded that the application of tree concepts in quadtree decomposition has proven effective in improving efficiency in image processing. The quadtree decomposition technique has been shown to effectively simplify image representation, particularly for images with large homogeneous areas. This technique can also be adapted for various image analysis applications, such as object segmentation and visual data indexing. However, the algorithm is not suitable for processing images with evenly distributed fine details, as it would complicate the image processing algorithm.

V. SUGGESTION

The author's suggestion for future researchers who wish to continue this topic is to explore the development of a hybrid algorithm that combines quadtree decomposition with other methods. Additionally, researchers are encouraged to test the algorithm on larger and more diverse image datasets to evaluate its scalability and generalizability across different applications. Integrating the algorithm with modern technologies, such as artificial intelligence, may also expand its potential for advanced image processing tasks.

VI. APENDIX

The GitHub repository for this paper can be accessed at https://github.com/JethroJNS/Matdis-Digital-Image-Processing-with-Quadtree-Decomposition.git and the explanatory video for this paper can be accessed at https://youtu.be/a7cICCqme4Q

VII. ACKNOWLEDGMENT

The author expresses heartfelt gratitude to our Father in Heaven, whose boundless grace, wisdom, and strength made the completion of this academic research possible. It was through His divine guidance and provision that every step of this work was successfully accomplished. Deep appreciation is also extended to the esteemed lecturer, Dr. Ir. Rinaldi Munir, M.T., whose invaluable guidance, encouragement, and support greatly enhanced the quality of this research. The author further thanks their family for their unwavering love, prayers, and faith, which provided strength and encouragement throughout the making of this paper. Lastly, sincere gratitude goes to friends, whose fellowship and support enriched this meaningful experience.

REFERENCES

[1] Munir, Rinaldi. 2024. "Graf (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf (accessed on 25th December 2024)

[2] Munir, Rinaldi. 2024. "Pohon (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/23-Pohon-Bag1-2024.pdf (accessed on 25th December 2024)

[3] Munir, Rinaldi. 2024. "Pohon (Bagian 2)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/24-Pohon-Bag2-2024.pdf (accessed on 25th December 2024)

[4] Munir, Rinaldi. 2019. "Pengantar Pengolahan Citra (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2019-2020/01-Pengantar-Pengolahan-Citra-Bag1.pdf (accessed on 25th December 2024)

[5] Strobach, Peter "Quadtree-Structured Recursive Plane Decomposition Coding of Images" *IEEE Transactions on Image Processing* 39, no. 6 (July 1991): 1380 – 1397. Available: https://www.researchgate.net/publication/3314487_Quadtree-Structured_Recursive_Plane_Decomposition_Coding_of_Images (accessed on 25th December 2024)

[6] Shusterman, E., and M. Feder. "Image compression via improved quadtree decomposition algorithms." *IEEE Transactions on Image Processing* 3, no. 2 (March 1994): 207–215. Available: https://ieeexplore.ieee.org/abstract/document/277901 (accessed on 25th December 2024)

STATEMENT

Hereby, I declare that this paper I have written is my own work, not an adaptation or translation of someone else's paper, and not a product of plagiarism.

Bandung, 26th December 2024

Jethro Jens Norbert Simatupang
13523081