

**Laporan Tugas Kecil 2 IF 2211**  
**Strategi Algoritma**  
**Kompresi Gambar Dengan Metode Quadtree**



**Oleh:**  
**Bob Kunanda - 13523086**  
**Muhammad Zahran Ramadhan Ardiana - 13523104**

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**JL. GANESA 10, BANDUNG 40132**  
**2025**

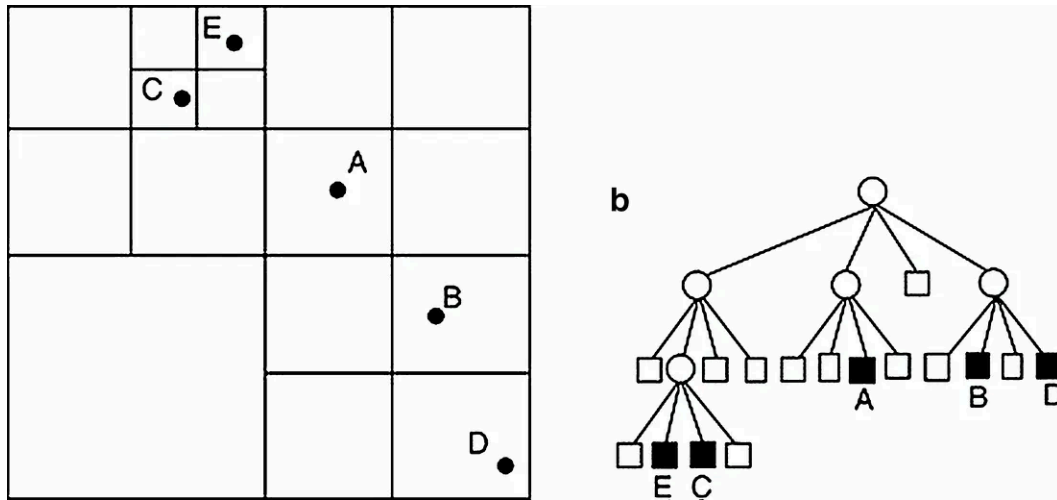
## Daftar Isi

|                                       |           |
|---------------------------------------|-----------|
| <b>BAB 1</b>                          | <b>3</b>  |
| <b>Algoritma Divide and Conquer</b>   | <b>3</b>  |
| 1. Algoritma                          | 3         |
| 2. Pseudocode                         | 4         |
| <b>BAB 2</b>                          | <b>6</b>  |
| <b>Source Program</b>                 | <b>6</b>  |
| <b>BAB 3</b>                          | <b>19</b> |
| <b>Input dan Output</b>               | <b>19</b> |
| <b>BAB 4</b>                          | <b>23</b> |
| <b>Hasil Analisis</b>                 | <b>23</b> |
| 1. Nilai Maksimum Error               | 23        |
| 2. Kompleksitas Waktu                 | 23        |
| <b>BAB 5</b>                          | <b>25</b> |
| <b>BONUS</b>                          | <b>25</b> |
| 1. Target Persentase Kompresi         | 25        |
| 2. Structural Similarity Index (SSIM) | 25        |
| 3. GIF                                | 26        |
| <b>BAB 6</b>                          | <b>27</b> |
| <b>Lampiran</b>                       | <b>27</b> |

# BAB 1

## Algoritma Divide and Conquer

### 1. Algoritma



**Gambar 1.** Struktur Data Quadtree dalam Kompresi Gambar

(Sumber: <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>)

Algoritma divide and conquer yang digunakan pada tugas ini adalah sebagai berikut:

1. Jika luas gambar yang diuji  $\leq$  dari minBlockSize dari input, maka dibiarkan.
2. Jika lusa gambar yang diuji  $>$  dari minBlockSize, maka
  - a. DIVIDE: bagi titik awal menjadi titik baru seperti titik1 =  $(x_0, y_0)$ , titik2 =  $(x_0 + \text{lebar}/2, y_0)$ , titik3 =  $(x_0, y_0 + \text{tinggi}/2)$ , dan titik4 =  $(x_0 + \text{lebar}/2, y_0 + \text{tinggi}/2)$  dan untuk setiap titik ubah ukurannya menjadi lebar/2 x tinggi/2. Titik dan ukuran akan tersimpan sebagai pohon.
  - b. CONQUER: secara rekursif, untuk setiap block masukkan ke fungsi metode pengukuran error untuk mendapatkan angka error yang kemudian akan dibandingkan dengan ambang batas yang sudah dipilih, jika ambang batas melebihi angka error dari block tersebut, maka block tersebut berhenti dibagi.
  - c. COMBINE: setelah semua block diproses, berdasarkan pohon yang ada, buat gambar baru dengan cara untuk setiap simpul ratakan block tersebut menjadi satu warna dan memasukkannya ke gambar baru tersebut berdasarkan lokasi dan ukurannya.

## 2. Pseudocode

**procedure** DivideTree(**input/output** node: Pohon, **input** img: Array of Array of Array of **Real**, **input** method: **integer**, **input** threshold: **real**, **input** minBlockSize: **integer**)

{Masukkan: img tidak kosong, node boleh kosong, Luaran: node yang sudah berisi data yang terkait kompresi img}

Deklarasi

error: **real**

w1, w2, h1, h2: **integer**

Algoritma:

**If** node.width \* node.height <= minBlockSize

**return**

**else**

    case method of

        1 : error ← variance(img, , node.x, node.y, node.width, node.height)

        2 : error ← meanAbsoluteDeviation(img, , node.x, node.y, node.width, node.height)

        3 : error ← maxPixelDifference(img, , node.x, node.y, node.width, node.height)

        4 : error ← entropy(img, , node.x, node.y, node.width, node.height)

        5 : error ← structuralSimilarityIndex(img, , node.x, node.y, node.width, node.height)

**end**

**if** (error <= threshold and method < 5) **or** ( error >= threshold and method = 5)

**return**

w1 ← node.width / 2

w2 ← node.width - w1

h1 ← node.height / 2

h2 ← node.height - w2

node.topLeft ← buildTree(img, node.x, node.y, w1, h1)

node.topRight ← buildTree(img, node.x + w1, node.y, w2, h1)

node.bottomLeft ← buildTree(img, node.x, node.y + h1, w1, h2)

node.bottomRight ← buildTree(img, node.x + w1, node.y + h1, w2, h2)

DivideTree(node.topLeft, img, method, threshold, minBlockSize)

DivideTree(node.topRight, img, method, threshold, minBlockSize)

DivideTree(node.bottomLeft, img, method, threshold, minBlockSize)

DivideTree(node.bottomRight, img, method, threshold, minBlockSize)

**end**

**Procedure** nodeToMatrix(**input** node: Pohon, **input/output** img: Array of Array of Array of **Real**)

{Masukkan: node hasil kompresi quadtree, img adalah matriks gambar asli.

Luaran: img akan dimodifikasi sesuai nilai blok node yang dikompresi.}

Deklarasi:

avgR, avgG, avgB: **real**

i, j: **integer**

Algoritma:

```
if node = null then  
    return  
  
if node.topLeft = null then  
    avgR  $\leftarrow$  0  
    avgG  $\leftarrow$  0  
    avgB  $\leftarrow$  0  
  
    for i  $\leftarrow$  node.y to node.y + node.height - 1 do  
        for j  $\leftarrow$  node.x to node.x + node.width - 1 do  
            avgR  $\leftarrow$  avgR + img[i][j][0]  
            avgG  $\leftarrow$  avgG + img[i][j][1]  
            avgB  $\leftarrow$  avgB + img[i][j][2]  
        end  
    end  
  
    avgR  $\leftarrow$  avgR / (node.width * node.height)  
    avgG  $\leftarrow$  avgG / (node.width * node.height)  
    avgB  $\leftarrow$  avgB / (node.width * node.height)  
  
    for i  $\leftarrow$  node.y to node.y + node.height - 1 do  
        for j  $\leftarrow$  node.x to node.x + node.width - 1 do  
            img[i][j][0]  $\leftarrow$  avgR  
            img[i][j][1]  $\leftarrow$  avgG  
            img[i][j][2]  $\leftarrow$  avgB  
        end  
    end  
  
else  
    nodeToMatrix(node.topLeft, img)  
    nodeToMatrix(node.topRight, img)  
    nodeToMatrix(node.bottomLeft, img)  
    nodeToMatrix(node.bottomRight, img)  
end
```

## BAB 2

### Source Program

| No. | Nama File | Screenshot  |
|-----|-----------|---|
| 1.  | EMM.hpp   | <pre> 1  #ifndef EMM_HPP 2  #define EMM_HPP 3 4  #include &lt;vector&gt; 5  #include &lt;cmath&gt; 6  #include &lt;iostream&gt; 7 8  double average(const std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img, int channel, int x, int y, int width, int height); 9  double standardDeviation(const std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img, int channel, int x, int y, int width, int height); 10 double variance(const std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img, int x, int y, int width, int height); 11 double meanAbsoluteDeviation(const std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img, int x, int y, int width, int height); 12 double maxPixelDifference(const std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img, int x, int y, int width, int height); 13 double entropy(const std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img, int x, int y, int width, int height); 14 double structuralSimilarityIndex(const std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img, int x, int y, int width, int height); 15 #endif 16 </pre>   |
| 2.  | EMM.cpp   | <pre> 1  #include "EMM.hpp" 2 3  using namespace std; 4 5  double average(const vector&lt;vector&lt;vector&lt;double&gt;&gt;&gt;&amp; img, int channel, int x, int y, int width, int height) { 6      int N = width * height; //banyak pixel 7      double sum = 0; 8      for(int i = y; i &lt; y + height; i++) { 9          for(int j = x; j &lt; x + width; j++) { 10             sum += img[i][j][channel]; 11          } 12      } 13      return sum / N; 14  } 15 16  double standardDeviation(const vector&lt;vector&lt;vector&lt;double&gt;&gt;&gt;&amp; img, int channel, int x, int y, int width, int height) { 17      double avg = average(img, channel, x, y, width, height); 18      double sum = 0; 19      for (int i = y; i &lt; y + height; i++) { 20          for (int j = x; j &lt; x + width; j++) { 21             sum += pow(img[i][j][channel] - avg, 2); 22          } 23      } 24      double stdDev = sqrt(sum / ((width * height) - 1)); 25 26      return stdDev; 27  } 28 29  double variance(const vector&lt;vector&lt;vector&lt;double&gt;&gt;&gt;&amp; img, int x, int y, int width, int height) { 30      int N = width * height; 31 32      double sumR = 0, sumG = 0, sumB = 0; 33 34      for (int i = y; i &lt; y + height; i++) { 35          for (int j = x; j &lt; x + width; j++) { 36             sumR += img[i][j][0]; 37 </pre> |

```

38         sumG += img[i][j][1];
39         sumB += img[i][j][2];
40     }
41 }
42
43 double avgR = sumR / N;
44 double avgG = sumG / N;
45 double avgB = sumB / N;
46
47 double varR = 0, varG = 0, varB = 0;
48
49 for (int i = y; i < y + height; i++) {
50     for (int j = x; j < x + width; j++) {
51         varR += pow(img[i][j][0] - avgR, 2);
52         varG += pow(img[i][j][1] - avgG, 2);
53         varB += pow(img[i][j][2] - avgB, 2);
54     }
55 }
56
57 return (varR + varG + varB) / (3 * N);
58 }
59
60 double meanAbsoluteDeviation(const vector<vector<vector<double>>>& img, int x, int y, int width, int height) {
61     int N = width * height;
62
63     double avgR = average(img, 0, x, y, width, height);
64     double avgG = average(img, 1, x, y, width, height);
65     double avgB = average(img, 2, x, y, width, height);
66
67     double sumR = 0, sumG = 0, sumB = 0;
68
69     for (int i = y; i < y + height; i++) {
70         for (int j = x; j < x + width; j++) {
71             sumR += abs(img[i][j][0] - avgR);
72             sumG += abs(img[i][j][1] - avgG);
73             sumB += abs(img[i][j][2] - avgB);
74         }
75     }
76
77     double madR = sumR / N;
78     double madG = sumG / N;
79     double madB = sumB / N;
80
81     return (madR + madG + madB) / 3;
82 }
83
84
85 double maxPixelDifference(const vector<vector<vector<double>>>& img, int x, int y, int width, int height) {
86     double maxR = 0, maxG = 0, maxB = 0;
87     double minR = 255, minG = 255, minB = 255;
88
89     for (int i = y; i < y + height; i++) {
90         for (int j = x; j < x + width; j++) {
91             double r = img[i][j][0];
92             double g = img[i][j][1];
93             double b = img[i][j][2];
94
95             if (r > maxR) maxR = r;
96             if (g > maxG) maxG = g;
97             if (b > maxB) maxB = b;
98
99             if (r < minR) minR = r;
100            if (g < minG) minG = g;
101            if (b < minB) minB = b;
102        }
103    }

```

|    |                    |   |
|----|--------------------|---|
|    |                    | <pre> 104     } 105 106     return (maxR - minR + maxG - minG + maxB - minB) / 3.0; 107 } 108 109 110 double entropy(const vector&lt;vector&lt;vector&lt;double&gt;&gt;&gt;&amp; img, int x, int y, int width, int height) { 111     int N = width * height; 112 113     double histR[256] = {0}, histG[256] = {0}, histB[256] = {0}; 114 115     for (int i = y; i &lt; y + height; i++) { 116         for (int j = x; j &lt; x + width; j++) { 117             histR[(int)img[i][j][0]]++; 118             histG[(int)img[i][j][1]]++; 119             histB[(int)img[i][j][2]]++; 120         } 121     } 122 123     double sumR = 0, sumG = 0, sumB = 0; 124 125     for (int i = 0; i &lt; 256; i++) { 126         if (histR[i] &gt; 0) sumR += histR[i] * log2(histR[i] / N); 127         if (histG[i] &gt; 0) sumG += histG[i] * log2(histG[i] / N); 128         if (histB[i] &gt; 0) sumB += histB[i] * log2(histB[i] / N); 129     } 130 131     double entropyR = -sumR; 132     double entropyG = -sumG; 133     double entropyB = -sumB; 134 135     return (entropyR + entropyG + entropyB) / (3 * N); 136 } 137 138 139 double structuralSimilarityIndex(const vector&lt;vector&lt;vector&lt;double&gt;&gt;&gt;&amp; img, int x, int y, int width, int height) { 140     double stdDevR = standardDeviation(img, 0, x, y, width, height); 141     double stdDevG = standardDeviation(img, 1, x, y, width, height); 142     double stdDevB = standardDeviation(img, 2, x, y, width, height); 143 144     double c1 = 6.5025, c2 = 58.5225; //c1 = (0.01 * 255) ^ 2, c2 = (0.03 * 255) ^ 2 145     double ssimR = 0, ssimG = 0, ssimB = 0; 146     ssimR = c2 / (stdDevR * stdDevR + c2); //rumus cepet angjayy 147     ssimG = c2 / (stdDevG * stdDevG + c2); 148     ssimB = c2 / (stdDevB * stdDevB + c2); 149 150     return 0.299 * ssimR + 0.587 * ssimG + 0.114 * ssimB; 151 } </pre> |
| 3. | FileProcessing.cpp | <pre> 1  #ifndef FILEPROCESSING_HPP 2  #define FILEPROCESSING_HPP 3 4  #include &lt;vector&gt; 5  #include &lt;string&gt; 6  #include &lt;stdexcept&gt; 7  #include &lt;iostream&gt; 8  #include &lt;cmath&gt; 9  #include &lt;algorithm&gt; 10 #include &lt;chrono&gt; 11 #include &lt;ctime&gt; 12 #include &lt;iomanip&gt; 13 #include &lt;sstream&gt; 14 15 16 class FileProcessing { 17 public: 18     struct Image { 19         int width; 20         int height; 21         int channels; 22         std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&gt; data; 23     }; 24 25     static Image loadImageAsDouble(const std::string&amp; filename); 26     static void printImage(const Image&amp; image); 27     static void saveImageAsPNG(const std::string&amp; filename, const std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img); 28 }; 29 30 31 #endif </pre>  |



4.

FileProcess  
ing.cpp

```

1  #include "FileProcessing.hpp"
2
3  #define STB_IMAGE_WRITE_IMPLEMENTATION
4  #define STB_IMAGE_IMPLEMENTATION
5  #include "stb_image.h"
6  #include "stb_image_write.h"
7
8  using namespace std;
9
10 FileProcessing::Image FileProcessing::loadImageAsDouble(const string& filename) {
11     int width, height, channels;
12     unsigned char* img = stbi_load(filename.c_str(), &width, &height, &channels, 0);
13     if (!img) {
14         throw std::runtime_error("Failed to load image: " + filename);
15     }
16
17     FileProcessing::Image image;
18     image.width = width;
19     image.height = height;
20     image.channels = channels;
21     image.data.resize(height, std::vector<std::vector<double>>(width, std::vector<double>(channels)));
22
23     for (int y = 0; y < height; ++y) {
24         for (int x = 0; x < width; ++x) {
25             for (int c = 0; c < 3; ++c) {
26                 int index = (y * width + x) * channels + c;
27                 image.data[y][x][c] = static_cast<double>(img[index]);
28             }
29         }
30     }
31
32     stbi_image_free(img);
33     return image;
34 }
35
36 void FileProcessing::printImage(const FileProcessing::Image& image) {
37     for (int y = 0; y < image.height; ++y) {
38         for (int x = 0; x < image.width; ++x) {
39             for (int c = 0; c < 3; ++c) {
40                 cout << endl;
41             }
42             cout << endl;
43         }
44     }
45 }
46
47 void FileProcessing::saveImageAsPNG(const std::string& pathName, const std::vector<std::vector<std::vector<double>>>& img) {
48     int height = img.size();
49     int width = img[0].size();
50     int channels = img[0][0].size();
51
52     vector<unsigned char> output(width * height * channels);
53
54     for (int y = 0; y < height; ++y) {
55         for (int x = 0; x < width; ++x) {
56             for (int c = 0; c < channels; ++c) {
57                 double val = img[y][x][c];
58                 val = clamp(val, 0.0, 255.0);
59                 output[(y * width + x) * channels + c] = static_cast<unsigned char>(val);
60             }
61         }
62     }
63
64     cout << "Saving image to " << pathName << endl;
65     stbi_write_jpg(pathName.c_str(), width, height, channels, output.data(), 100);
66
67 }
68

```

|    |                  |  |
|----|------------------|--|
| 5. | QuadTree.<br>hpp | <pre> 1  #ifndef QUADTREE_HPP 2  #define QUADTREE_HPP 3 4  #include &lt;vector&gt; 5  #include &lt;algorithm&gt; 6  #include &lt;functional&gt; 7  #include &lt;queue&gt; 8 9  class QuadTree { 10 public: 11     struct Node { 12         int x; 13         int y; 14         int width; 15         int height; 16         Node* topLeft; 17         Node* topRight; 18         Node* bottomLeft; 19         Node* bottomRight; 20     }; 21 22     QuadTree(); 23     ~QuadTree(); 24 25     static Node* buildTree(const std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img, int x, int y, int width, int height); 26     static void divideTree(Node* node, const std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img, int method, double threshold, int minBlockSize); 27     static void nodeToMatrix(Node* node, std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img); 28     static void printTree(Node* node); 29     static int countNode(Node* node); 30     static int getDepth(Node* node); 31     static void deleteTree(Node* node); 32     static void bfsAveragePerLevelImage(Node* node, const std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; img, std::function&lt;void(const std::vect 33     static void avgTheRGB_fromOriginal(const Node* node, std::vector&lt;std::vector&lt;std::vector&lt;double&gt;&gt;&gt;&amp; targetImg, const std::vector&lt;std::vector&lt; 34 35 }; 36 #endif </pre>  |
| 6. | QuadTree.c<br>pp | <pre> 1  #include "QuadTree.hpp" 2  #include "EMM.hpp" 3  #include &lt;iostream&gt; 4  #include &lt;vector&gt; 5 6  using namespace std; 7 8  QuadTree::QuadTree() {} 9  QuadTree::~QuadTree() {} 10 11 QuadTree::Node* QuadTree::buildTree(const vector&lt;vector&lt;vector&lt;double&gt;&gt;&gt;&amp; img, int x, int y, int width, int height) { 12     QuadTree::Node* node = new QuadTree::Node(); 13     node-&gt;x = x; 14     node-&gt;y = y; 15     node-&gt;width = width; 16     node-&gt;height = height; 17     node-&gt;topLeft = nullptr; 18     node-&gt;topRight = nullptr; 19     node-&gt;bottomLeft = nullptr; 20     node-&gt;bottomRight = nullptr; 21 22     return node; 23 } 24 25 void QuadTree::divideTree(QuadTree::Node* node, const vector&lt;vector&lt;vector&lt;double&gt;&gt;&gt;&amp; img, int method, double threshold, int minBlockSize) { 26     if (node-&gt;width * node-&gt;height &lt;= minBlockSize) return; 27 28     double error; 29     switch (method) { 30         case 1: error = variance(img, node-&gt;x, node-&gt;y, node-&gt;width, node-&gt;height); break; 31         case 2: error = meanAbsoluteDeviation(img, node-&gt;x, node-&gt;y, node-&gt;width, node-&gt;height); break; 32         case 3: error = maxPixelDifference(img, node-&gt;x, node-&gt;y, node-&gt;width, node-&gt;height); break; 33         case 4: error = entropy(img, node-&gt;x, node-&gt;y, node-&gt;width, node-&gt;height); break; 34         case 5: error = structuralSimilarityIndex(img, node-&gt;x, node-&gt;y, node-&gt;width, node-&gt;height); break; 35         default: return; 36     } </pre> |

```

38     if (error <= threshold && method < 5) {return;}
39     if (error >= threshold && method == 5) {
40         return;
41     }
42
43     int w1 = node->width / 2;
44     int w2 = node->width - w1;
45     int h1 = node->height / 2;
46     int h2 = node->height - h1;
47
48     node->topLeft = buildTree(img, node->x, node->y, w1, h1);
49     node->topRight = buildTree(img, node->x + w1, node->y, w2, h1);
50     node->bottomLeft = buildTree(img, node->x, node->y + h1, w1, h2);
51     node->bottomRight = buildTree(img, node->x + w1, node->y + h1, w2, h2);
52
53     divideTree(node->topLeft, img, method, threshold, minBlockSize);
54     divideTree(node->topRight, img, method, threshold, minBlockSize);
55     divideTree(node->bottomLeft, img, method, threshold, minBlockSize);
56     divideTree(node->bottomRight, img, method, threshold, minBlockSize);
57 }
58
59
60 void QuadTree::nodeToMatrix(QuadTree::Node* node, vector<vector<vector<double>>>& img) {
61     if (node == nullptr) return;
62
63     if (node->topLeft == nullptr) {
64         double avgR = 0;
65         double avgG = 0;
66         double avgB = 0;
67         for (int i = node->y; i < node->y + node->height; i++) {
68             for (int j = node->x; j < node->x + node->width; j++) {
69                 avgR += img[i][j][0];
70                 avgG += img[i][j][1];
71                 avgB += img[i][j][2];
72             }
73         }
74         avgR /= node->width * node->height;
75         avgG /= node->width * node->height;
76         avgB /= node->width * node->height;
77         for (int i = node->y; i < node->y + node->height; i++) {
78             for (int j = node->x; j < node->x + node->width; j++) {
79                 img[i][j][0] = avgR;
80                 img[i][j][1] = avgG;
81                 img[i][j][2] = avgB;
82             }
83         }
84     } else {
85         nodeToMatrix(node->topLeft, img);
86         nodeToMatrix(node->topRight, img);
87         nodeToMatrix(node->bottomLeft, img);
88         nodeToMatrix(node->bottomRight, img);
89     }
90 }
91
92 void QuadTree::printTree(QuadTree::Node* node) {
93     if (node == nullptr) return;
94
95     cout << "Node: (" << node->x << ", " << node->y << ") " << node->width << "x" << node->height << endl;
96     printTree(node->topLeft);
97     printTree(node->topRight);
98     printTree(node->bottomLeft);
99     printTree(node->bottomRight);
100 }
101
102 int QuadTree::countNode(QuadTree::Node* node) {
103     if (node == nullptr) return 0;
104
105     return 1 + countNode(node->topLeft) + countNode(node->topRight) + countNode(node->bottomLeft) + countNode(node->bottomRight);

```

```

106 }
107
108 int QuadTree::getDepth(QuadTree::Node* node) {
109     if (node == nullptr) return 0;
110     if (!node->topLeft) return 1;
111     return 1 + max(getDepth(node->topLeft), getDepth(node->topRight), getDepth(node->bottomLeft), getDepth(node->bottomRight));
112 }
113
114 void QuadTree::deleteTree(QuadTree::Node* node) {
115     if (node == nullptr) return;
116
117     deleteTree(node->topLeft);
118     deleteTree(node->topRight);
119     deleteTree(node->bottomLeft);
120     deleteTree(node->bottomRight);
121     delete node;
122 }
123
124 void QuadTree::avgTheRGB_fromOriginal(const Node* node, vector<vector<vector<double>>>& targetImg, const vector<vector<vector<double>>>& sourceImg)
125 {
126     double avgR = 0, avgG = 0, avgB = 0;
127     for (int i = node->y; i < node->y + node->height; i++) {
128         for (int j = node->x; j < node->x + node->width; j++) {
129             avgR += sourceImg[i][j][0];
130             avgG += sourceImg[i][j][1];
131             avgB += sourceImg[i][j][2];
132         }
133     }
134     int area = node->width * node->height;
135     avgR /= area;
136     avgG /= area;
137     avgB /= area;
138
139     for (int i = node->y; i < node->y + node->height; i++) {
140         for (int j = node->x; j < node->x + node->width; j++) {
141             targetImg[i][j][0] = avgR;
142             targetImg[i][j][1] = avgG;
143             targetImg[i][j][2] = avgB;
144         }
145     }
146 }
147
148 void QuadTree::bfsAveragePerLevelImage(Node* root, const vector<vector<vector<double>>>& originalImg, function<void(const vector<vector<vector<double>>>& targetImg, const vector<vector<vector<double>>>& sourceImg)>
149 if (!root) return;
150
151 std::queue<std::pair<Node*, int>> q;
152 q.push({root, 0});
153 int currentDepth = 0;
154
155 std::vector<Node*> currentLevelNodes;
156 auto currentImg = originalImg;
157
158 while (!q.empty()) {
159     Node* node = q.front().first;
160     int depth = q.front().second;
161     q.pop();
162
163     if (depth != currentDepth) {
164         auto imgCopy = currentImg;
165         for (Node* n : currentLevelNodes) {
166             avgTheRGB_fromOriginal(n, imgCopy, originalImg);
167         }
168
169         handleLevelImage(imgCopy, currentDepth);
170         currentImg = imgCopy;
171         currentLevelNodes.clear();
172         currentDepth = depth;
173     }
174
175     if (node->topLeft) q.push({node->topLeft, depth + 1});
176     if (node->topRight) q.push({node->topRight, depth + 1});
177     if (node->bottomLeft) q.push({node->bottomLeft, depth + 1});
178     if (node->bottomRight) q.push({node->bottomRight, depth + 1});
179 }

```

|    |          |   |
|----|----------|---|
|    |          | <pre> 174 175         currentLevelNodes.push_back(node); 176         if (node-&gt;topLeft) { 177             q.push({node-&gt;topLeft, depth + 1}); 178             q.push({node-&gt;topRight, depth + 1}); 179             q.push({node-&gt;bottomLeft, depth + 1}); 180             q.push({node-&gt;bottomRight, depth + 1}); 181         } 182     } 183 184     if (!currentLevelNodes.empty()) { 185         auto imgCopy = currentImg; 186         for (Node* n : currentLevelNodes) { 187             avgTheRGB_fromOriginal(n, imgCopy, originalImg); 188         } 189         handleLevelImage(imgCopy, currentDepth); 190     } 191 } </pre>  |
| 7. | main.cpp | <pre> 1  #include &lt;iostream&gt; 2  #include &lt;string&gt; 3  #include &lt;limits&gt; 4  #include &lt;cmath&gt; 5  #include &lt;filesystem&gt; 6  #include &lt;chrono&gt; 7  #include &lt;algorithm&gt; 8 9  #include "FileProcessing.hpp" 10 #include "EMM.hpp" 11 #include "QuadTree.hpp" 12 13 using namespace std; 14 namespace fs = std::filesystem; 15 16 int main() { 17     string inputPath, outputPath, gifPath, fileName, fileType; 18     fs::path inputPathObj, gifPathObj; 19     uintmax_t originalFileSize; 20     int errorMethod; 21     double threshold; 22     int minBlockSize; 23     double targetCompression; 24     bool targetCompressionEnabled = false; 25     bool targetCompressionValid = false; 26 27     // === INPUTS === 28 29     while (true) { 30         cout &lt;&lt; "Alamat absolut gambar yang akan dikompresi: "; 31         getline(cin, inputPath); 32 33         inputPathObj = fs::path(inputPath); 34 35         if (!inputPathObj.is_absolute()) { 36             cout &lt;&lt; "Path harus absolut. Masukkan path lengkap.\n"; 37             continue; </pre> |

```

38     }
39
40     if (!fs::exists(inputPathObj)) {
41         cout << "File tidak ditemukan. Silakan masukkan ulang.\n";
42         continue;
43     }
44
45     string ext = inputPathObj.extension().string();
46     fileType = ext;
47     for (char& c : ext) c = std::tolower(c);
48     cout << "File type: " << fileType << endl;
49     if (ext != ".png" && ext != ".jpg" && ext != ".jpeg") {
50         cout << "File bukan PNG atau JPG. Masukkan file yang valid.\n";
51         continue;
52     }
53
54     originalFileSize = fs::file_size(inputPath);
55     break;
56 }
57
58
59 fs::path pathObj(inputPath);
60 fileName = pathObj.filename().string();
61
62 FileProcessing::Image image = FileProcessing::loadImageAsDouble(inputPath);
63
64 cout << "Pilih metode perhitungan error:\n";
65 cout << "1. Variance\n";
66 cout << "2. Mean Absolute Deviation (MAD)\n";
67 cout << "3. Max Pixel Difference\n";
68 cout << "4. Entropy\n";
69 cout << "5. Structural Similarity Index (SSIM)\n";
70 cout << "Masukkan nomor metode: ";
71

```

```

72 while (!(cin >> errorMethod) || errorMethod < 1 || errorMethod > 5) {
73     cout << "Metode tidak valid. Masukkan hanya bisa dalam rentang 1 - 5: ";
74     cin.clear();
75     cin.ignore(numeric_limits<streamsize>::max(), '\n');
76 }
77
78 cout << "Ambang batas error (threshold): ";
79 while (!(cin >> threshold) || threshold <= 0) {
80     cout << "Threshold harus lebih dari 0: ";
81     cin.clear();
82     cin.ignore(numeric_limits<streamsize>::max(), '\n');
83 }
84
85 cout << "Ukuran blok minimum (>= 1): ";
86 while (!(cin >> minBlockSize) || minBlockSize < 1) {
87     cout << "Ukuran blok harus >= 1: ";
88     cin.clear();
89     cin.ignore(numeric_limits<streamsize>::max(), '\n');
90 }
91
92 cout << "Target persentase kompresi 0 - 100%: ";
93 while (!(cin >> targetCompression) || targetCompression < 0.0 || targetCompression > 1.0) {
94     cout << "Persentase kompresi harus di antara 0 hingga 1: ";
95     cin.clear();
96     cin.ignore(numeric_limits<streamsize>::max(), '\n');
97 }
98 if (targetCompression > 0.0) {
99     targetCompressionEnabled = true;
100     cout << "Target persentase kompresi diaktifkan: " << targetCompression << "%" << endl;
101 } else {
102     cout << "Target persentase kompresi dinonaktifkan." << endl;
103 }
104
105 cin.ignore(numeric_limits<streamsize>::max(), '\n');
106
107 while (true) {
108     std::cout << "Alamat absolut gambar hasil kompresi: ";
109     std::getline(std::cin, outputPath);
110
111     fs::path outputPathObj(outputPath);
112     fs::path parentDir = outputPathObj.parent_path();
113     std::string ext = outputPathObj.extension().string();
114
115     for (char& c : ext) c = std::tolower(c);
116
117     if (!outputPathObj.is_absolute()) {
118         std::cout << "Path harus absolut. Masukkan path lengkap.\n";
119         continue;
120     }
121
122     if (ext != ".jpg" && ext != ".jpeg" && ext != ".png") {
123         std::cout << "Ekstensi file tidak valid! Gunakan .jpg, .jpeg, atau .png.\n";
124         continue;
125     }
126
127     if (parentDir.empty() || fs::exists(parentDir)) {
128         break;
129     } else {
130         std::cout << "Direktori tidak ditemukan: \"" << parentDir.string() << "\"\n";
131         std::cout << "Silakan masukkan path yang valid.\n";
132     }
133 }
134
135 while (true) {
136     std::cout << "Alamat output GIF proses kompresi: ";
137     std::getline(std::cin, gifPath);
138
139     gifPath.erase(std::remove(gifPath.begin(), gifPath.end(), ' '), gifPath.end());
140
141     fs::path outputPathObj(gifPath);
142     fs::path parentDir = outputPathObj.parent_path();

```

```

143     std::string ext = outputPathObj.extension().string();
144
145     for (char& c : ext) c = std::tolower(c);
146
147     if (!outputPathObj.is_absolute()) {
148         std::cout << "Path harus absolut. Masukkan path lengkap.\n";
149         continue;
150     }
151
152     if (ext != ".gif") {
153         std::cout << "Ekstensi file tidak valid! Gunakan .gif.\n";
154         continue;
155     }
156
157     if (parentDir.empty() || fs::exists(parentDir)) {
158         break;
159     } else {
160         std::cout << "Direktori tidak ditemukan: \"" << parentDir.string() << "\"\n";
161         std::cout << "Silakan masukkan path yang valid.\n";
162     }
163 }
164
165 // === QUADTREE PROCESSING ===
166
167 auto start = chrono::high_resolution_clock::now();
168
169 auto compressedImage = image.data;
170
171 QuadTree quadTree;
172 QuadTree::Node* root;
173 double maxThreshold;
174 if (errorMethod == 1) {
175     maxThreshold = 16256.25;
176 } else if (errorMethod == 2) {
177     maxThreshold = 127.5;
178 } else if (errorMethod == 3) {
179     maxThreshold = 255;
180 } else if (errorMethod == 4) {
181     maxThreshold = 8.0;
182 } else if (errorMethod == 5) {
183     maxThreshold = 1.0;
184 }
185
186 double low = 0.0, high = maxThreshold, tolerance = 0.01;
187 int maxIterations = 50, iteration = 0;
188
189 while (!targetCompressionValid && iteration < maxIterations) {
190     compressedImage = image.data;
191     if (targetCompressionEnabled) {
192         threshold = (low + high) / 2.0;
193     }
194
195     quadTree = QuadTree();
196     root = quadTree.buildTree(image.data, 0, 0, image.width, image.height);
197     quadTree.divideTree(root, image.data, errorMethod, threshold, minBlockSize);
198     quadTree.nodeToMatrix(root, compressedImage);
199
200     if (!targetCompressionEnabled) {
201         break;
202     } else {
203         fs::create_directories("targetCompressionTemp");
204
205         outputPath = "targetCompressionTemp/" + fileName;
206         FileProcessing::saveImageAsPNG(outputPath, compressedImage);
207         uintmax_t compressedFileSize = fs::file_size(outputPath);
208         fs::remove(outputPath);
209
210         double compressionNew = 1.0 - static_cast<double>(compressedFileSize) / originalFileSize;

```



```

210         double compressionNew = 1.0 - static_cast<double>(compressedFileSize) / originalFileSize;
211
212         if (abs((targetCompression) - compressionNew) <= tolerance) {
213             targetCompression = compressionNew;
214         } else {
215             if (compressionNew < targetCompression) {
216                 if (errorMethod < 5) {
217                     low = threshold;
218                 }
219                 else {
220                     high = threshold;
221                 }
222             } else {
223                 if (errorMethod < 5) {
224                     high = threshold;
225                 }
226                 else {
227                     low = threshold;
228                 }
229             }
230         }
231
232         cout << "Compressed file size: " << compressedFileSize << " bytes" << endl;
233         cout << "Target compression: " << targetCompression << "%" << endl;
234         cout << iteration << " Threshold: " << threshold << ", Compression ratio: " << compressionNew << endl;
235     }
236     iteration++;
237 }
238
239 cout << "\nSaving image to " << outputPath << endl;
240 FileProcessing::saveImageAsPNG(outputPath, compressedImage);
241 auto end = chrono::high_resolution_clock::now();
242
243 // === GIF PROCESSING ===
244 auto gifStart = chrono::high_resolution_clock::now();
245
246 fs::create_directories("gifTemp");
247 int counter = 0;
248 QuadTree::bfsAveragePerLevelImage(root, image.data, [&counter, fileType](const std::vector<std::vector<std::vector<double>>>& img, int level) {
249     if (fileType == ".png") {
250         FileProcessing::saveImageAsPNG("gifTemp/level_" + std::to_string(level + 100) + ".png", img);
251     }
252     else if (fileType == ".jpg") {
253         FileProcessing::saveImageAsPNG("gifTemp/level_" + std::to_string(level + 100) + ".jpg", img);
254     }
255     else if (fileType == ".jpeg") {
256         FileProcessing::saveImageAsPNG("gifTemp/level_" + std::to_string(level + 100) + ".jpeg", img);
257     }
258     else {
259         cout << "Ekstensi file tidak valid!" << endl;
260     }
261     counter += 1;
262 });
263
264 cout << "Saving GIF to " << gifPath << endl;
265
266 string command = "magick -delay 50 -loop 0 ";
267
268 for (int i = 100; i < 100 + counter; i++) {
269     command += "gifTemp/level_" + std::to_string(i) + fileType + " ";
270 }
271
272 command += gifPath;
273
274 int result = system(command.c_str());

```

```

278     if (result != 0) {
279         cout << "Failed to create GIF. Check if ImageMagick is properly installed and available in PATH." << endl;
280     } else {
281         cout << "GIF successfully created." << endl;
282     }
283
284     auto gifEnd = chrono::high_resolution_clock::now();
285
286     for(int i = 100; i < 100 + counter; i++){
287         if(fileType == ".png"){
288             string filename = "gifTemp/level_" + std::to_string(i) + ".png";
289             fs::remove(filename);
290         }
291         else if(fileType == ".jpg"){
292             string filename = "gifTemp/level_" + std::to_string(i) + ".jpg";
293             fs::remove(filename);
294         }
295         else if(fileType == ".jpeg"){
296             string filename = "gifTemp/level_" + std::to_string(i) + ".jpeg";
297             fs::remove(filename);
298         }
299     }
300
301     uintmax_t compressedFileSize = fs::file_size(outputPath);
302
303     int depth = quadTree.getDepth(root);
304     int nodeCount = quadTree.countNode(root);
305
306     quadTree.deleteTree(root);
307
308     chrono::duration<double> duration = end - start;
309     chrono::duration<double> gifDuration = gifEnd - gifStart;
310
311     // === OUTPUTS ===
312     double compressionRatio = 100 * (1.0 - (double)compressedFileSize / originalFileSize);
313
314     cout << "\n=== HASIL EKSEKUSI ===" << endl;
315     cout << "Waktu eksekusi image: " << duration.count() << " detik" << endl;
316     cout << "Waktu eksekusi GIF: " << gifDuration.count() << " detik" << endl;
317     cout << "Ukuran gambar sebelum: " << originalFileSize << " bytes" << endl;
318     cout << "Ukuran gambar setelah: " << compressedFileSize << " bytes" << endl;
319     cout << "Persentase kompresi: " << compressionRatio << "%" << endl;
320     cout << "Kedalaman pohon: " << depth - 1 << endl;
321     cout << "Banyak simpul pada pohon: " << nodeCount << endl;
322
323     return 0;
324 }

```


### BAB 3

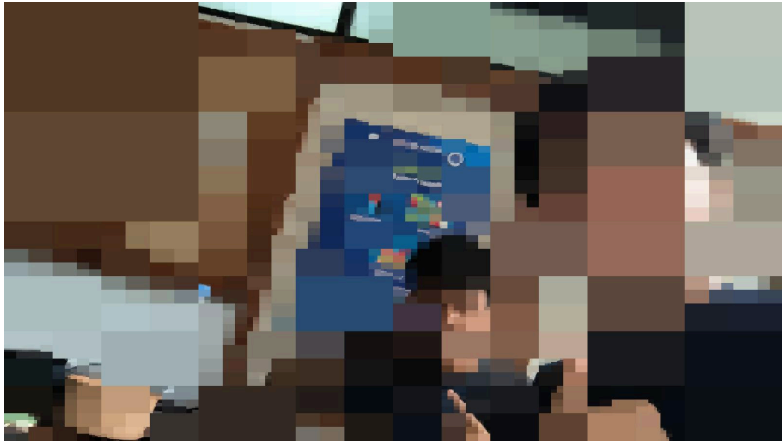

## Input dan Output






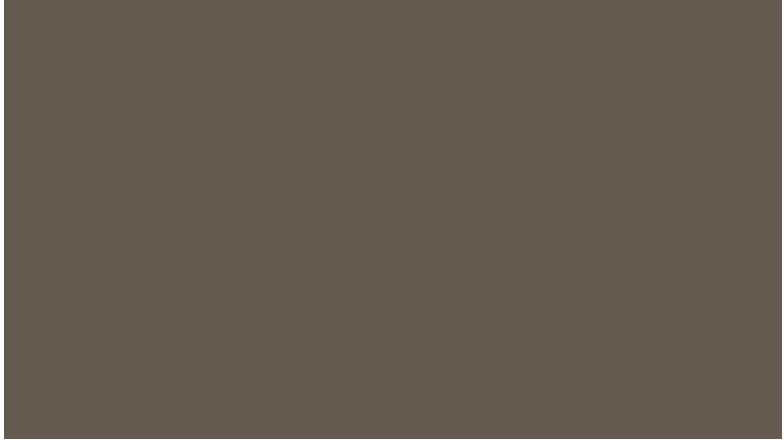
**Gambar 2.** Foto Sampel Pengujian

Gambar di atas adalah gambar awal dan seluruh pengujian menggunakan minBlockSize 1.

| No. | Method   | Threshold | Result   |
|-----|----------|-----------|--|
| 1.  | variance | 100       |  |

|    |     |    |   |
|----|-----|----|---|
|    |     |    | <pre> === HASIL EKSEKUSI === Waktu eksekusi: 5.24657 detik Ukuran gambar sebelum: 111862 Ukuran gambar setelah: 635683 Persentase kompresi: -468.274% Kedalaman pohon: 12 Banyak simpul pada pohon: 91165 </pre>  |
| 2. | MAD | 30 |  <pre> === HASIL EKSEKUSI === Waktu eksekusi: 4.30943 detik Ukuran gambar sebelum: 111862 Ukuran gambar setelah: 255862 Persentase kompresi: -128.73% Kedalaman pohon: 12 Banyak simpul pada pohon: 3329 </pre> |
| 3. | MPD | 50 |   |

|    |         |     |   |
|----|---------|-----|---|
|    |         |     | <pre> === HASIL EKSEKUSI === Waktu eksekusi: 4.41223 detik Ukuran gambar sebelum: 111862 Ukuran gambar setelah: 634860 Persentase kompresi: -467.539% Kedalaman pohon: 12 Banyak simpul pada pohon: 42329 </pre>  |
| 4. | entropy | 4   |  <pre> === HASIL EKSEKUSI === Waktu eksekusi: 3.8616 detik Ukuran gambar sebelum: 111862 Ukuran gambar setelah: 714339 Persentase kompresi: -538.59% Kedalaman pohon: 10 Banyak simpul pada pohon: 48249 </pre> |
| 5. | SSIM    | 0.8 |  <pre> Waktu eksekusi: 5.40431 detik Ukuran gambar sebelum: 111862 Ukuran gambar setelah: 661688 Persentase kompresi: -491.522% Kedalaman pohon: 12 Banyak simpul pada pohon: 368433 </pre>                   |

|    |                          |     |  |
|----|--------------------------|-----|--|
| 6. | SSIM + target compressed | 10% |  <pre> === HASIL EKSEKUSI === Waktu eksekusi: 37.7886 detik Ukuran gambar sebelum: 111862 Ukuran gambar setelah: 102090 Persentase kompresi: 8.73576% Kedalaman pohon: 11 Banyak simpul pada pohon: 265 </pre> |
| 7. | SSIM (GIF)               | 0.8 |  <p> <a href="https://drive.google.com/drive/folders/1zVLd58Vfto6VPwPd058H_AFO9Qbec-KY?usp=drive_link">https://drive.google.com/drive/folders/1zVLd58Vfto6VPwPd058H_AFO9Qbec-KY?usp=drive_link</a> </p>       |

## BAB 4

### Hasil Analisis

#### 1. Nilai Maksimum Error

Karena pada bonus target compression menggunakan binary search sehingga perlu mengetahui range dari setiap error. Maka dari itu, berikut cara mendapatkan nilai maksimum error untuk setiap metode:

a. Variance

Pada  $(P_{i,c} - \mu_c)^2$  nilai terbesar yang mungkin dihasilkan adalah  $127.5^2$  dimana nilai tersebut didapat karena 255 merupakan nilai terbesar untuk RGB dan 0 nilai terkecilnya sehingga ketika dimasukkan ke rumus keseluruhan nilai maksimumnya adalah **16256.25**.

b. Mean Absolute Deviation (MAD)

Sama dengan variance, namun karena tidak dipangkat nilai yang didapatkan adalah **127.5**.

c. Max Pixel Difference

Pada metode ini, nilai error didapatkan dengan mengurangi nilai maksimum dengan minimum dari RGB sehingga nilai maksimum error untuk metode ini adalah **255**.

d. Entropy

Pada metode entropy, perhitungan tidak dijumlahkan berdasarkan besar gambar seperti variance dan MAD, perhitungan dilakukan dengan bilangan konstan yaitu 255. Maka dari itu kita dapat mengetahui batas maksimum dari entropy dengan cara mengasumsikan bahwa kemunculan warna sangatlah minimum (yaitu setiap warna hanya memiliki peluang  $\frac{1}{256}$  untuk kemunculannya) sehingga seperti berikut  $-256(\frac{1}{256}) \times \log_2(\frac{1}{256})$  sehingga didapatkan nilai **8**.

e. Structural Similarity Index (SSIM)

Pada metode ini dapat dilihat langsung pada rumus akhir di mana nilai SSIM dinormalisasikan karena greyscaling yaitu  $0.299 \cdot SSIM_r + 0.587 \cdot SSIM_g + 0.114 \cdot SSIM_b$  sehingga nilai maksimumnya adalah **1**.

#### 2. Kompleksitas Waktu

Fungsi divideTree bekerja secara rekursif untuk membagi sebuah blok gambar menjadi empat sub-blok lebih kecil menggunakan pendekatan *divide and conquer*. Pada setiap langkah rekursi, fungsi akan mengecek apakah sebuah blok perlu dibagi lebih lanjut berdasarkan nilai *error measurement*. Proses rekursif ini akan berhenti ketika blok tidak memenuhi syarat pembagian (misalnya ukuran terlalu kecil atau error terlalu rendah).

Pola pemanggilan secara rekursif yang memecah satu blok menjadi empat bagian setiap kali dijalankan, maka dapat **dimodelkan waktu eksekusi dalam bentuk relasi rekurens**. Selain

melakukan pemanggilan rekursif, fungsi juga menghitung nilai error dari blok saat ini dengan kompleksitas linier terhadap jumlah piksel dalam blok tersebut.

$$T(N) = 4T\left(\frac{N}{4}\right) + cN$$

$T(N)$  : waktu untuk memproses blok berukuran  $N$  piksel (dengan  $N = h \times w$ )

$h, w$  : ukuran gambar height dan width

$cN$  : waktu untuk memproses perhitungan error pada blok tersebut

Maka berdasarkan teorema master dapat dibentuk:

$$O(N^1 \log_4 N) = O(N \log_4 N) = O((h * w) \log_4 (h * w))$$



## BAB 5

### BONUS

#### 1. Target Persentase Kompresi

Tujuan dari bonus ini adalah untuk mengotimalkan hasil agar terkompresi sesuai keinginan. Karena diketahui bahwa semakin besar threshold, semakin terkompresi gambar (sebaliknya untuk SSIM). Maka terdapat berbagai pendekatan untuk menyelesaikan masalah ini. Pada bonus ini kami menggunakan pendekatan binary search. Karena menggunakan binary search membutuhkan rentang, maka setelah kami perhitungkan berikut merupakan jarak untuk setiap ambang batas:

1. Variance = {0 - 16256.25}
2. MAD = {0 - 127.5}
3. MPD = {0 - 255}
4. Entropy = {0 - 8}
5. SSIM = {0 - 1}

Dengan mengetahui jarak ambang batas, maka algoritma dapat kami buat seperti berikut:

1. Pengujian ambang batas pertama menggunakan median dari jarak ambang batas.
2. Setelah gambar berhasil dibuat, cek ukuran byte dari file tersebut dan bandingkan dengan target kompresi.
3. Jika masih kurang terkompresi, naikan nilai threshold menggunakan cara binary search dan jika terlalu terkompresi lakukan sebaliknya.
4. Setelah gambar yang didapatkan terbukti terkompresi dengan toleransi 1% atau 50 kali iterasi, maka program akan berhenti.

#### 2. Structural Similarity Index (SSIM)

SSIM merupakan salah satu metode pengukuran error pada tucil ini. Berbeda dengan metode lainnya, SSIM tidak mengandalkan warna dari gambar tetapi dari bentuk gambar. Dasarnya SSIM menggunakan 3 parameter yaitu: luminance, contrast, dan struktur. Berdasarkan parameter tersebut didapatkan rumus berikut.

$$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$$
$$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$$

Gambar 3. Rumus Gabungan SSIM

(Sumber: <https://ieeexplore.ieee.org/document/1284395> )

Namun pengaplikasian SSIM pada tugas ini akan dibandingkan dengan gambar terkompresi di mana terkompresi di sini bermakna gambar dengan terpenuhi nilai rata-rata dari suatu block pada gambar awal. Sehingga misalkan gambar  $x$  adalah gambar awal dan gambar  $y$  adalah gambar yang terkompresi, didapatkan bahwa rata-rata gambar  $x$  = rata - rata gambar  $y$  dan standar deviasi dari gambar  $y$  = 0. Sehingga rumus di atas dapat disimplifikasi menjadi.

$$SSIM_c(x, y) = \frac{c_2}{(\sigma_{x,c}^2 + c_2)}$$

Menggunakan rumus tersebut, SSIM dapat diaplikasikan seperti metode error lainnya. Namun terdapat perbedaan terkait sifat SSIM, untuk metode lain, semakin besar ambang batas, maka gambar akan semakin terkompresi, namun sifat tersebut terbalik untuk SSIM sehingga pengaplikasian pada kode perlu sedikit perubahan agar sesuai.

### 3. GIF

Tujuan utama dari pembuatan GIF ini adalah untuk memvisualisasikan alur kerja algoritma kompresi, terutama bagaimana pohon Quadtree terbentuk dan berkembang saat membagi gambar. Perubahan pada setiap level pembagian gambar ditampilkan sehingga proses dapat lebih mudah memahami bagaimana algoritma mengambil keputusan untuk membagi atau mempertahankan suatu block pixel.

Konsep yang digunakan dalam pembuatan GIF ini adalah dengan memanfaatkan struktur pohon yang telah dibentuk oleh fungsi `QuadTree::divideTree` dan mengakses tiap simpul (node) secara Breadth-First Search (BFS). Setiap kali proses BFS berpindah ke level (kedalaman) berikutnya, image yang telah diproses di level sebelumnya akan digunakan kembali, lalu ditimpa oleh masing-masing simpul pada level saat ini. Penimpaan dilakukan dengan mengambil rata-rata warna dari gambar asli (original image) yang terdefinisi sesuai area pada atribut simpul tersebut. Dengan demikian, jika suatu simpul tidak memiliki anak, maka warna area pada image tersebut tidak akan berubah dan tetap menggunakan warna hasil pemrosesan dari level sebelumnya.

Setiap hasil pemrosesan pada tiap level akan di-*generate* dan disimpan dalam folder gifTemp. Seluruh gambar yang telah dihasilkan tersebut kemudian digabungkan menjadi sebuah GIF menggunakan sistem, dengan penambahan jeda (delay) antar frame.

## BAB 6

### Lampiran

| Poin  | Ya                                  | Tidak                    |
|---|-------------------------------------|--------------------------|
| 1. Program berhasil dikompilasi tanpa kesalahan   | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 2. Program berhasil dijalankan  | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 3. Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan                    | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 4. Mengimplementasi seluruh metode perhitungan error wajib  | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 5. <b>[Bonus]</b> Implementasi persentase kompresi sebagai parameter tambahan                     | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 6. <b>[Bonus]</b> Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 7. <b>[Bonus]</b> Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 8. Program dan laporan dibuat (kelompok) sendiri  | <input checked="" type="checkbox"/> | <input type="checkbox"/> |

Link Github: [https://github.com/BobSwagg13/Tucil2\\_13523086\\_13523104](https://github.com/BobSwagg13/Tucil2_13523086_13523104)