

Kompresi Gambar Dengan Metode Quadtree



Oleh:

13523036 – Yonatan Edward Njoto

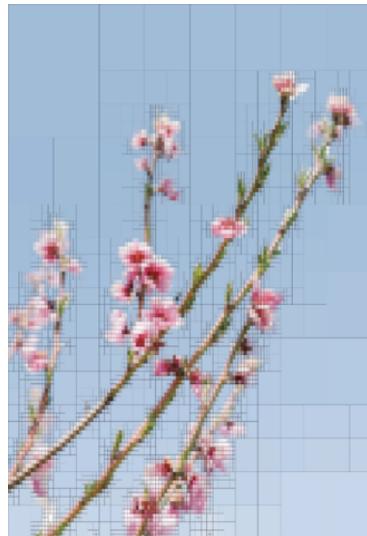
13523075 – Muhammad Dicky Isra

IF2211 – Semester 4
Strategi Algoritma
Tahun Pembelajaran 2024/2025

Daftar Isi

Daftar Isi.....	1
BAB 1. Deskripsi Masalah.....	2
BAB 2. Perancangan.....	3
2.1 Algoritma Divide and Conquer yang Digunakan.....	3
2.2 Penjelasan tentang Spesifikasi Bonus yang Dikerjakan.....	4
1. Target Persentase Kompresi (Compression Target menggunakan Binary Search).....	4
2. SSIM (Structural Similarity Index Measure).....	5
3. GIF (Animasi Proses Rekonstruksi QuadTree).....	6
BAB 3. Implementasi.....	8
3.1 Implementasi Perhitungan Error.....	8
3.2 Implementasi Divide and Conquer di QuadTreeNode.....	15
BAB 4. Pengujian.....	20
4.1 Hasil Pengujian dan analisis output.....	20
Gambar 1 - Grayscale (test/a/a.png).....	20
Gambar 2 - rgba (test/b/b.png).....	25
Gambar 3 - rgb (test/c/c.jpg).....	30
4.2 Analisis Kompleksitas Algoritma Program.....	34
Lampiran.....	36

BAB 1. Deskripsi Masalah



Gambar 1. Quadtree dalam Kompresi Gambar

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

Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, Quadtree membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat bagian, lalu memeriksa apakah setiap bagian memiliki nilai yang seragam berdasarkan analisis **sistem warna RGB**, yaitu dengan membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada piksel-piksel di dalamnya. Jika bagian tersebut tidak seragam, maka bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan.

Dalam implementasi teknis, sebuah Quadtree direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. QuadTree sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

BAB 2. Perancangan

2.1 Algoritma Divide and Conquer yang Digunakan

Algoritma divide and conquer yang digunakan dalam implementasi struktur QuadTree ini bertujuan untuk merepresentasikan gambar dalam bentuk blok-blok piksel yang lebih homogen dengan menggabungkan warnanya, sehingga dapat dimanfaatkan untuk kompresi. Berikut adalah langkah-langkah algoritmanya:

1. Kondisi Berhenti (Base Case):

Pembagian sebuah blok gambar akan dihentikan apabila ukuran blok tersebut telah lebih kecil atau sama dengan nilai minimum yang ditentukan (`minBlockSize`), atau ketika lebar atau tinggi blok sudah mencapai satu piksel. Selain itu, pembagian juga akan dihentikan jika nilai error (tingkat perbedaan warna dalam blok) lebih kecil dari ambang batas (`threshold`) yang telah ditentukan. Dalam kondisi ini, blok dianggap cukup homogen dan tidak perlu dibagi lagi. Nilai rata-rata piksel dari blok tersebut kemudian disimpan dan digunakan sebagai representasi dari seluruh blok.

2. Divide (Pemisahan Wilayah):

Jika kondisi berhenti tidak terpenuhi, maka blok akan dibagi menjadi empat sub-blok, yaitu northwest, northeast, southwest, dan southeast. Proses pembagian ini dilakukan dengan mencari titik tengah dari koordinat blok saat ini. Setelah dibagi, masing-masing sub-blok kemudian diproses secara rekursif dengan cara yang sama: dicek apakah perlu dibagi lagi, dan jika ya, akan dibagi kembali menjadi empat bagian yang lebih kecil. Proses ini terus berlangsung hingga semua blok mencapai kondisi berhenti.

3. Conquer (Pembangunan Subtree):

Setelah proses pembagian (divide) dilakukan, masing-masing sub-blok diproses secara rekursif untuk menentukan apakah mereka juga perlu dibagi. Jika iya, proses ini akan terus berlangsung hingga mencapai kondisi berhenti. Hasil dari tiap proses rekursif tidak digabungkan kembali menjadi satu blok gambar secara langsung, melainkan disusun dalam bentuk struktur pohon. Setiap node akan menyimpan referensi ke keempat anaknya (northwest, northeast, southwest, southeast), yang bersama-sama membentuk subtree dari pohon QuadTree utama. Dengan cara ini, representasi gambar tersimpan dalam bentuk hierarki blok-blok homogen, tanpa perlu digabungkan kembali, namun tetap dapat dimanfaatkan untuk berbagai keperluan seperti kompresi atau visualisasi.

```
// Fungsi utama untuk membangun QuadTree dari sebuah gambar
function BuildQuadTree(image, koordinatArea, threshold, minBlockSize):
    // Langkah 1: Hitung error homogenitas dari area saat ini
    // Jika error < threshold atau ukuran blok terlalu kecil:
    //     -> Anggap blok ini cukup homogen, berhenti membagi
```

```

//      -> Hitung rata-rata warna blok
//      -> Buat node daun (leaf) dengan nilai rata-rata tersebut
//      -> Kembalikan node sebagai hasil rekursi

// Langkah 2: Jika kondisi berhenti tidak terpenuhi:
//      -> Tentukan titik tengah dari area saat ini
//      -> Bagi area menjadi empat subregion:
//          - Northwest
//          - Northeast
//          - Southwest
//          - Southeast

// Langkah 3: Rekursif ke keempat subregion:
//      -> NW = BuildQuadTree(image, area_NW, threshold, minBlockSize)
//      -> NE = BuildQuadTree(image, area_NE, threshold, minBlockSize)
//      -> SW = BuildQuadTree(image, area_SW, threshold, minBlockSize)
//      -> SE = BuildQuadTree(image, area_SE, threshold, minBlockSize)

// Langkah 4: Bangun node internal (non-leaf) dengan referensi ke keempat anaknya
//      -> Hitung rata-rata warna dari keempat anak
//      -> Simpan ke dalam node sebagai representasi blok saat ini

// Langkah 5: Kembalikan node hasil pembentukan subtree

```

2.2 Penjelasan tentang Spesifikasi Bonus yang Dikerjakan

1. Target Persentase Kompresi (Compression Target menggunakan Binary Search)

Fitur ini secara otomatis mencari nilai threshold terbaik agar hasil kompresi memenuhi target persentase kompresi yang diinginkan.

Metode:

- Menggunakan algoritma **binary search** pada rentang threshold dari 0.0 hingga 65025.0 karena menggunakan variansi.
- Pada setiap iterasi, dijalankan langkah berikut:
 1. Bangun struktur QuadTree menggunakan threshold mid.
 2. Rekonstruksi gambar hasil kompresi ke dalam matriks piksel.
 3. Simpan hasil gambar sementara untuk menghitung ukuran file.
 4. Hitung persentase kompresi yang tercapai:

$$\text{compressionAchieved} = 1.0 - (\text{outputSize} / \text{inputSize})$$
 5. Jika persentase belum memenuhi target, naikkan nilai low. Jika sudah, simpan mid sebagai kandidat threshold terbaik dan turunkan nilai high.

Output:

- Threshold terbaik (bestThreshold) yang menghasilkan kompresi sesuai target.

- Log debug yang menunjukkan threshold yang diuji dan persentase kompresi yang dicapai.

```

function FindBestThreshold(image, targetKompresi, pathInput):
    low = 0.0
    high = 65025.0
    bestThreshold = 0.0

    while low <= high:
        mid = (low + high) / 2.0

        // Langkah 1: Buat QuadTree dengan threshold saat ini
        quadTree = BuildQuadTree(image, seluruhArea, mid, minBlockSize)

        // Langkah 2: Rekonstruksi gambar hasil kompresi
        hasilMatrix = quadTree.reconstructToMatrix()

        // Langkah 3: Simpan sementara hasilnya ke file
        SimpanImage(hasilMatrix, tempPath)

        // Langkah 4: Hitung ukuran file input dan output
        sizeInput = GetUkuranFile(pathInput)
        sizeOutput = GetUkuranFile(tempPath)

        // Langkah 5: Hitung kompresi yang tercapai
        kompresi = 1.0 - (sizeOutput / sizeInput)

        // Langkah 6: Lanjutkan binary search
        if kompresi < targetKompresi:
            low = mid + stepKecil
        else:
            bestThreshold = mid
            high = mid - stepKecil

    return bestThreshold

```

2. SSIM (Structural Similarity Index Measure)

Digunakan untuk mengukur kualitas visual hasil kompresi dengan membandingkan struktur citra hasil dengan citra asli.

Rumus SSIM:

$$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$$

Komponen yang dihitung:

- Rata-rata piksel (mu1, mu2)

- Variansi (var1, var2)
- Kovariansi (cov)

Dukungan Multi-Channel:

- Mendukung 1 (grayscale), 3 (RGB), atau 4 (RGBA) channel.
- Pembobotan tiap channel disesuaikan:
 - RGB: 0.2125 (R), 0.7154 (G), 0.0721 (B)
 - RGBA: Tambahan 0.1 untuk Alpha channel

Output:

- Nilai SSIM antara area tertentu pada gambar asli dan hasil kompresi.
- Digunakan sebagai parameter kualitas visual.

```
function HitungSSIM(imageAsli, imageHasil, area, channel):
    // Langkah 1: Hitung rata-rata piksel (mu) dari imageAsli dan imageHasil
    mu1 = HitungRataRata(imageAsli, area, channel)
    mu2 = HitungRataRata(imageHasil, area, channel)

    // Langkah 2: Hitung variansi masing-masing
    var1 = HitungVariansi(imageAsli, area, channel, mu1)
    var2 = HitungVariansi(imageHasil, area, channel, mu2)

    // Langkah 3: Hitung kovariansi antar citra
    cov = HitungKovariansi(imageAsli, imageHasil, area, channel, mu1, mu2)

    // Langkah 4: Terapkan rumus SSIM
    SSIM = ((2 * mu1 * mu2 + C1) * (2 * cov + C2)) / ((mu1^2 + mu2^2 + C1) * (var1 + var2 + C2))

    return SSIM
```

3. GIF (Animasi Proses Rekonstruksi QuadTree)

Fitur ini membuat file animasi GIF untuk menampilkan proses rekonstruksi gambar dari struktur QuadTree berdasarkan kedalaman tertentu.

Langkah-langkah:

1. Inisialisasi GIF writer dan buffer gambar.
2. Untuk setiap tingkat kedalaman dari 1 hingga maxDepth:
 - Gambar direkonstruksi sebagian menggunakan reconstructImageMatrix.
 - Matriks dikonversi ke data RGBA.
 - Frame ditambahkan ke animasi GIF.
3. Proses diulang hingga kedalaman maksimal tercapai.

Spesifikasi Teknis:

- Delay antar frame: 50 ms
- Gambar dikodekan dalam format 32-bit RGBA
- Semua frame dihasilkan langsung dari hasil rekonstruksi QuadTree per depth

Output:

- File .gif yang memperlihatkan proses visual bagaimana QuadTree merepresentasikan gambar dari bentuk kasar hingga penuh.

```
function GenerateGIF(quadTree, maxDepth, width, height, pathGIF):  
    Inisialisasi GIFWriter  
  
    for depth dari 1 hingga maxDepth:  
        // Langkah 1: Rekonstruksi gambar sampai kedalaman saat ini  
        matrix = quadTree.reconstructToMatrix(depth)  
  
        // Langkah 2: Konversi matriks ke RGBA (frameData)  
        frame = ConvertToRGBA(matrix)  
  
        // Langkah 3: Tambahkan frame ke GIF  
        TambahkanFrameGIF(writer, frame)  
  
    Selesai menulis GIF
```

BAB 3. Implementasi

3.1 Implementasi Perhitungan Error

Perhitungan error yang dipertimbangkan dalam program di bawah sudah termasuk pada channel grayscale (1), rgb (3), dan rgba (4). Dapat dilihat juga bahwa setiap metode perhitungan error, baik itu variance, mean absolute deviation, maximum pixel difference, entropy, maupun SSIM, mempertimbangkan jumlah channel yang terdapat pada gambar, dan mengakumulasi hasilnya berdasarkan nilai channel tersebut.

Untuk channel:

- Grayscale (1 channel):
Hanya channel r yang dihitung (atau channel tunggal pada representasi Pixel), dan error langsung didasarkan pada nilai tersebut.
- RGB (3 channel):
Setiap metode error melakukan perhitungan pada ketiga channel (r, g, b) dan merata-ratakan hasilnya, atau dalam kasus SSIM, memberikan bobot berbeda untuk setiap channel:
R: 0.2125, G: 0.7154, B: 0.0721 sesuai dengan persepsi penglihatan manusia terhadap warna.
- RGBA (4 channel):
Sama seperti RGB, tetapi juga memperhitungkan channel alpha (a) untuk representasi transparansi. Dalam kasus SSIM, diberikan bobot tambahan (A: 0.1).

Inti penting lainnya:

- Fungsi computeError menjadi pengendali utama pemilihan metode error dan juga normalisasi berdasarkan jumlah channel.
- SSIM memiliki perlakuan khusus, dibandingkan terhadap citra rata-rata (mean image) dan hasil akhirnya adalah 1.0 - SSIM untuk mengekspresikan error (karena nilai SSIM 1 menunjukkan kemiripan sempurna).
- Seluruh metode dijaga agar tidak menghasilkan nilai di luar rentang 0–255, dengan menggunakan fungsi clamp (min dan max).

```
#include "QuadTreeNode.hpp"
#include <algorithm>
#include <cmath>
```

```

#include <iostream>
#include <stdexcept>
#include <string>
#include <unordered_map>
#include <vector>

const double C1 = (0.01 * 255) * (0.01 * 255);
const double C2 = (0.03 * 255) * (0.03 * 255);
using namespace std;

double QuadTreeNode::computeError(int pickMethod, const Image &image, int fromX, int fromY, int
toX, int toY) const {
    Pixel errorPixel;
    double errorValue = 0.0;

    if (pickMethod == 1) {
        errorPixel = getVariance(image, fromX, fromY, toX, toY);
    } else if (pickMethod == 2) {
        errorPixel = getMeanAbsoluteDeviation(image, fromX, fromY, toX, toY);
    } else if (pickMethod == 3) {
        errorPixel = getMaxPixelDifference(image, fromX, fromY, toX, toY);
    } else if (pickMethod == 4) {
        errorPixel = getEntropy(image, fromX, fromY, toX, toY);
    } else if (pickMethod == 5) {
        // For SSIM, we want error = 1 - SSIM since SSIM of 1 means perfect similarity
        // Create a temporary image with just the mean pixel value for comparison
        Image meanImage = createMeanImage(image, fromX, fromY, toX, toY);
        errorValue = 1.0 - getSSIM(image, meanImage, fromX, fromY, toX, toY);
        return errorValue;
    } else {
        throw invalid_argument("Invalid pick method.");
    }

    int channels = image.channels;
    double sum = 0.0;

    if (channels >= 1)
        sum += static_cast<double>(errorPixel.r);
    if (channels >= 2)
        sum += static_cast<double>(errorPixel.g);
    if (channels >= 3)
        sum += static_cast<double>(errorPixel.b);
    if (channels >= 4)
        sum += static_cast<double>(errorPixel.a);

    return sum / static_cast<double>(channels);
}

```

```

Image QuadTreeNode::createMeanImage(const Image &sourceImage, int fromX, int fromY, int toX, int
toY) const {
    Image meanImage;
    meanImage.width = toX - fromX;
    meanImage.height = toY - fromY;
    meanImage.channels = sourceImage.channels;

    // Initialize the pixel data
    meanImage.pixels.resize(meanImage.height);
    for (int y = 0; y < meanImage.height; y++) {
        meanImage.pixels[y].resize(meanImage.width, meanPixel);
    }

    return meanImage;
}

Pixel QuadTreeNode::getVariance(const Image &image, int fromX, int fromY, int toX, int toY) const
{
    double retR = 0, retG = 0, retB = 0, retA = 0;
    int count = (toX - fromX) * (toY - fromY);

    if (count == 0)
        return Pixel(0, 0, 0, 0);

    for (int y = fromY; y < toY; ++y) {
        for (int x = fromX; x < toX; ++x) {
            Pixel p = image.pixels[y][x];
            double diffR = static_cast<double>(p.r) - static_cast<double>(meanPixel.r);
            double diffG = static_cast<double>(p.g) - static_cast<double>(meanPixel.g);
            double diffB = static_cast<double>(p.b) - static_cast<double>(meanPixel.b);
            double diffA = static_cast<double>(p.a) - static_cast<double>(meanPixel.a);

            retR += (diffR * diffR) / count;
            retG += (diffG * diffG) / count;
            retB += (diffB * diffB) / count;
            retA += (diffA * diffA) / count;
        }
    }

    // Clamp values to valid unsigned char range
    return Pixel(
        min(255.0, max(0.0, retR)),
        min(255.0, max(0.0, retG)),
        min(255.0, max(0.0, retB)),
        min(255.0, max(0.0, retA)));
}

```

```

Pixel QuadTreeNode::getMeanAbsoluteDeviation(const Image &image, int fromX, int fromY, int toX,
int toY) const {
    double retR = 0, retG = 0, retB = 0, retA = 0;
    int count = (toX - fromX) * (toY - fromY);

    if (count == 0)
        return Pixel(0, 0, 0, 0);

    for (int y = fromY; y < toY; ++y) {
        for (int x = fromX; x < toX; ++x) {
            Pixel p = image.pixels[y][x];
            retR += abs(static_cast<double>(p.r) - static_cast<double>(meanPixel.r)) / count;
            retG += abs(static_cast<double>(p.g) - static_cast<double>(meanPixel.g)) / count;
            retB += abs(static_cast<double>(p.b) - static_cast<double>(meanPixel.b)) / count;
            retA += abs(static_cast<double>(p.a) - static_cast<double>(meanPixel.a)) / count;
        }
    }

    // Clamp values to valid unsigned char range
    return Pixel(
        min(255.0, max(0.0, retR)),
        min(255.0, max(0.0, retG)),
        min(255.0, max(0.0, retB)),
        min(255.0, max(0.0, retA)));
}

Pixel QuadTreeNode::getMaxPixelDifference(const Image &image, int fromX, int fromY, int toX,
int toY) const {
    int maxDiffR = 0, maxDiffG = 0, maxDiffB = 0, maxDiffA = 0;

    for (int y = fromY; y < toY; ++y) {
        for (int x = fromX; x < toX; ++x) {
            Pixel p = image.pixels[y][x];
            maxDiffR = max(maxDiffR, abs(static_cast<int>(p.r) - static_cast<int>(meanPixel.r)));
            maxDiffG = max(maxDiffG, abs(static_cast<int>(p.g) - static_cast<int>(meanPixel.g)));
            maxDiffB = max(maxDiffB, abs(static_cast<int>(p.b) - static_cast<int>(meanPixel.b)));
            maxDiffA = max(maxDiffA, abs(static_cast<int>(p.a) - static_cast<int>(meanPixel.a)));
        }
    }

    return Pixel(maxDiffR, maxDiffG, maxDiffB, maxDiffA);
}

double QuadTreeNode::computeEntropy(unordered_map<int, int> &frequencyMap, int totalPixels) const
{
    double entropy = 0.0;
}

```

```

        for (const auto &[value, count] : frequencyMap) {
            double p = static_cast<double>(count) / totalPixels;
            if (p > 0)
                entropy -= p * log2(p);
        }

        return entropy;
    }

Pixel QuadTreeNode::getEntropy(const Image &image, int fromX, int fromY, int toX, int toY) const
{
    Pixel entropyPixel = {0, 0, 0, 0};
    unordered_map<int, int> freqR, freqG, freqB, freqA;
    int totalPixels = (toX - fromX) * (toY - fromY);

    if (totalPixels == 0)
        return entropyPixel;

    for (int y = fromY; y < toY; ++y) {
        for (int x = fromX; x < toX; ++x) {
            Pixel p = image.pixels[y][x];
            freqR[p.r]++;
            freqG[p.g]++;
            freqB[p.b]++;
            freqA[p.a]++;
        }
    }

    double entropyR = computeEntropy(freqR, totalPixels);
    double entropyG = computeEntropy(freqG, totalPixels);
    double entropyB = computeEntropy(freqB, totalPixels);
    double entropyA = computeEntropy(freqA, totalPixels);

    // Convert entropy to 0-255 scale per channel and clamp
    entropyPixel.r = static_cast<unsigned char>(min(255.0, max(0.0, entropyR)));
    entropyPixel.g = static_cast<unsigned char>(min(255.0, max(0.0, entropyG)));
    entropyPixel.b = static_cast<unsigned char>(min(255.0, max(0.0, entropyB)));
    entropyPixel.a = static_cast<unsigned char>(min(255.0, max(0.0, entropyA)));

    return entropyPixel;
}

double QuadTreeNode::SSIMmean(const Image &img, int fx, int fy, int tx, int ty, int channel)
const {
    double sum = 0.0;
    int count = (tx - fx) * (ty - fy);
}

```

```

if (count == 0)
    return 0.0;

for (int y = fy; y < ty; ++y) {
    for (int x = fx; x < tx; ++x) {
        const Pixel &p = img.pixels[y - fy][x - fx];
        if (channel == 0)
            sum += p.r;
        else if (channel == 1)
            sum += p.g;
        else if (channel == 2)
            sum += p.b;
        else if (channel == 3)
            sum += p.a;
    }
}
return sum / count;
}

double QuadTreeNode::SSIMvariance(const Image &img, int fx, int fy, int tx, int ty, int channel,
double mu) const {
    double sumSq = 0.0;
    int count = (tx - fx) * (ty - fy);
    if (count == 0)
        return 0.0;

    for (int y = fy; y < ty; ++y) {
        for (int x = fx; x < tx; ++x) {
            const Pixel &p = img.pixels[y - fy][x - fx];
            double val = 0.0;

            if (channel == 0)
                val = p.r;
            else if (channel == 1)
                val = p.g;
            else if (channel == 2)
                val = p.b;
            else if (channel == 3)
                val = p.a;

            sumSq += (val - mu) * (val - mu);
        }
    }
    return sumSq / count;
}

double QuadTreeNode::SSIMcovariance(const Image &img1, const Image &img2, int fx, int fy, int tx,

```

```

int ty, int channel, double mu1, double mu2) const {
    double sumCov = 0.0;
    int count = (tx - fx) * (ty - fy);
    if (count == 0)
        return 0.0;

    for (int y = fy; y < ty; ++y) {
        for (int x = fx; x < tx; ++x) {
            const Pixel &p1 = img1.pixels[y][x];
            const Pixel &p2 = img2.pixels[y - fy][x - fx];
            double val1 = 0.0, val2 = 0.0;

            if (channel == 0) {
                val1 = p1.r;
                val2 = p2.r;
            } else if (channel == 1) {
                val1 = p1.g;
                val2 = p2.g;
            } else if (channel == 2) {
                val1 = p1.b;
                val2 = p2.b;
            } else if (channel == 3) {
                val1 = p1.a;
                val2 = p2.a;
            }

            sumCov += (val1 - mu1) * (val2 - mu2);
        }
    }
    return sumCov / count;
}

double QuadTreeNode::getSSIM(const Image &original, const Image &compressed, int fromX, int
fromY, int toX, int toY) const {
    int channels = original.channels;

    if (channels <= 0 || channels > 4) {
        throw std::invalid_argument("Unsupported number of channels.");
    }

    vector<double> weights(channels, 1.0);
    if (channels == 3) {
        weights = {0.2125, 0.7154, 0.0721};
    } else if (channels == 4) {
        weights = {0.2125, 0.7154, 0.0721, 0.1};
    }
}

```

```

        double totalWeight = 0.0;
        for (double w : weights)
            totalWeight += w;

        double totalSSIM = 0.0;

        for (int c = 0; c < channels; ++c) {
            double mu1 = SSIMmean(original, fromX, fromY, toX, toY, c);
            double mu2 = SSIMmean(compressed, fromX, fromY, toX, toY, c);

            double var1 = SSIMvariance(original, fromX, fromY, toX, toY, c, mu1);
            double var2 = SSIMvariance(compressed, fromX, fromY, toX, toY, c, mu2);

            double cov = SSIMcovariance(original, compressed, fromX, fromY, toX, toY, c, mu1, mu2);

            double numerator = (2 * mu1 * mu2 + C1) * (2 * cov + C2);
            double denominator = (mu1 * mu1 + mu2 * mu2 + C1) * (var1 + var2 + C2);

            double ssim = (denominator == 0.0) ? 1.0 : numerator / denominator;
            totalSSIM += ssim * weights[c];
        }

        return totalSSIM / totalWeight;
    }
}

```

3.2 Implementasi Divide and Conquer di QuadTreeNode

Dalam konteks Quadtree, Divide and Conquer ini digunakan untuk:

- Membagi gambar menjadi empat kuadran jika masih belum memenuhi kriteria tertentu (seperti threshold error).
- Menaklukkan (selesaikan) setiap kuadran dengan memprosesnya secara rekursif sampai tiap kuadran cukup seragam atau terlalu kecil untuk dibagi lagi.
- Menggabungkan hasil berupa pohon (tree) yang menyimpan rata-rata piksel dari masing-masing node.

Implementasi QuadTreeNode dapat dilihat di bawah:

```

#include "QuadTreeNode.hpp"
#include <algorithm>
#include <cmath>
#include <iostream>

```

```

#include <stdexcept>
#include <string>
#include <unordered_map>
#include <vector>

using namespace std;

QuadTreeNode::QuadTreeNode(const Image &image, int pickMethod, double threshold, int minBlockSize)
: fromX(0), fromY(0), width(image.width), height(image.height),
  divided(false), northwest(nullptr), northeast(nullptr),
  southwest(nullptr), southeast(nullptr) {

    if (image.getPixelCount() == 0) {
        cout << "[QuadTree] Image is empty." << endl;
        return;
    }

    meanPixel = image.getMean(0, 0, width, height);
    buildTree(image, pickMethod, threshold, minBlockSize);
}

QuadTreeNode::QuadTreeNode(const Image &image, int fromX, int fromY, int toX, int toY,
                         int pickMethod, double threshold, int minBlockSize)
: fromX(fromX), fromY(fromY), width(toX - fromX), height(toY - fromY),
  divided(false), northwest(nullptr), northeast(nullptr),
  southwest(nullptr), southeast(nullptr) {

    if (width <= 0 || height <= 0) {
        throw invalid_argument("Invalid dimensions for QuadTreeNode.");
    }

    meanPixel = image.getMean(fromX, fromY, toX, toY);
    buildTree(image, pickMethod, threshold, minBlockSize);
}

bool QuadTreeNode::isCanDivide(int width, int height, int minBlockSize) const {
    double minBlockSizeD = static_cast<double>(minBlockSize);
    double curSize = static_cast<double>(width * height);
    return ((curSize / 4) > minBlockSizeD) && width > 1 && height > 1;
}

void QuadTreeNode::buildTree(const Image &image, int pickMethod, double threshold, int minBlockSize) {
    if (!isCanDivide(width, height, minBlockSize)) {
        return;
    }
}

```

```

        double error = computeError(pickMethod, image, fromX, fromY, fromX + width, fromY + height);
        if (error < threshold) {
            return;
        }

        // If we reach here, we need to divide the node
        divided = true;
        int midX = fromX + width / 2;
        int midY = fromY + height / 2;

        // Create the four quadrants
        northwest = new QuadTreeNode(image, fromX, fromY, midX, midY, pickMethod, threshold,
minBlockSize);
        northeast = new QuadTreeNode(image, midX, fromY, fromX + width, midY, pickMethod, threshold,
minBlockSize);
        southwest = new QuadTreeNode(image, fromX, midY, midX, fromY + height, pickMethod, threshold,
minBlockSize);
        southeast = new QuadTreeNode(image, midX, midY, fromX + width, fromY + height, pickMethod,
threshold, minBlockSize);

        // Update mean pixel based on children
        vector<Pixel> pixelGroup = {
            northwest->meanPixel,
            northeast->meanPixel,
            southwest->meanPixel,
            southeast->meanPixel};
        meanPixel = computeAveragePixel(pixelGroup);
    }

    void QuadTreeNode::deleteChildren() {
        delete northwest;
        northwest = nullptr;
        delete northeast;
        northeast = nullptr;
        delete southwest;
        southwest = nullptr;
        delete southeast;
        southeast = nullptr;
    }

    QuadTreeNode::~QuadTreeNode() {
        deleteChildren();
    }

    Pixel QuadTreeNode::computeAveragePixel(vector<Pixel> &pixels) const {
        int retR = 0, retG = 0, retB = 0, retA = 0;
        int count = pixels.size();

```

```

if (count == 0)
    return Pixel(0, 0, 0, 0);

for (const auto &p : pixels) {
    retR += p.r;
    retG += p.g;
    retB += p.b;
    retA += p.a;
}

return Pixel(retR / count, retG / count, retB / count, retA / count);
}

void QuadTreeNode::showTree(int depth) const {
    for (int i = 0; i < depth; ++i)
        cout << " ";
    cout << "[" << width << "x" << height << (divided ? "] - Subdivided" : "] - Leaf") << " " <<
meanPixel << endl;
    for (int i = 0; i < depth; ++i)
        cout << " ";
    cout << "\\" << fromX << ", " << fromY << ") to (" << fromX + width << ", " << fromY +
height << ")" << endl;
    if (divided) {
        northwest->showTree(depth + 1);
        northeast->showTree(depth + 1);
        southwest->showTree(depth + 1);
        southeast->showTree(depth + 1);
    }
}

void QuadTreeNode::debugTree() const {
    cout << "\n[QuadTree Structure]" << endl;
    showTree(0);
}

void QuadTreeNode::buildMatrix(vector<vector<Pixel>> &imageMatrix) const {
    if (divided) {
        northwest->buildMatrix(imageMatrix);
        northeast->buildMatrix(imageMatrix);
        southwest->buildMatrix(imageMatrix);
        southeast->buildMatrix(imageMatrix);
    } else {
        for (int y = fromY; y < fromY + height; ++y) {
            for (int x = fromX; x < fromX + width; ++x) {
                imageMatrix[y][x] = meanPixel;
            }
        }
    }
}

```

```
        }
    }

pair<int, int> QuadTreeNode::getStat() const {
    if (!divided) {
        return {1, 1}; // Depth = 1, Node count = 1 (Leaf node)
    } else {
        pair<int, int> northwestStat = northwest ? northwest->getStat() : make_pair(0, 0);
        pair<int, int> northeastStat = northeast ? northeast->getStat() : make_pair(0, 0);
        pair<int, int> southwestStat = southwest ? southwest->getStat() : make_pair(0, 0);
        pair<int, int> southeastStat = southeast ? southeast->getStat() : make_pair(0, 0);

        int depth = 1 + max({northwestStat.first, northeastStat.first, southwestStat.first,
southeastStat.first});
        int nodeCount = 1 + northwestStat.second + northeastStat.second + southwestStat.second +
southeastStat.second;

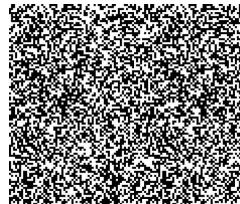
        return {depth, nodeCount};
    }
}
```

BAB 4. Pengujian

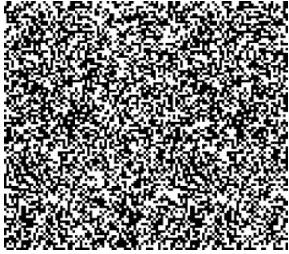
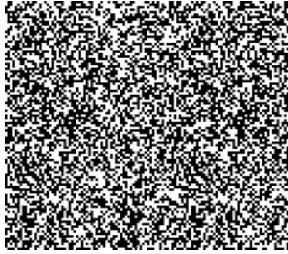
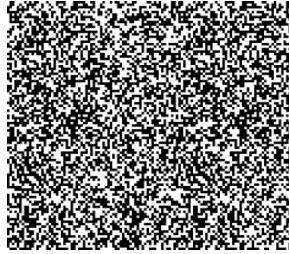
4.1 Hasil Pengujian dan analisis output

Pengujian ini dilakukan $3 \times 6 \times 3 = 54$ untuk mengetes channel berbeda-beda.

Gambar 1 - Grayscale (test/a/a.png)



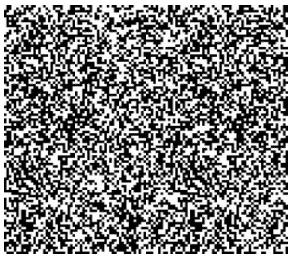
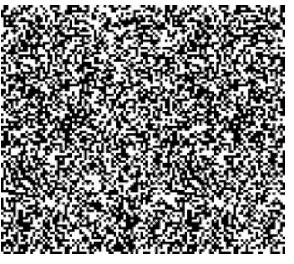
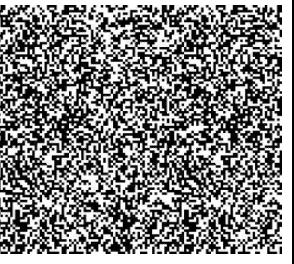
Variance

method percentage threshold min block size	test/a/a.png 0 1 10 1	test/a/a.png 0 1 10 2	test/a/a.png 0 1 20 1
stats	===== stats ===== Waktu Eksekusi: 0.108557 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 171786 bit Persentase kompresi: 15.797% Kedalaman QuadTree: 10 Jumlah Simpul: 142001	===== stats ===== Waktu Eksekusi: 0.0680603 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 100278 bit Persentase kompresi: 50.8475% Kedalaman QuadTree: 10 Jumlah Simpul: 77849	===== stats ===== Waktu Eksekusi: 0.0885009 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 159312 bit Persentase kompresi: 21.9112% Kedalaman QuadTree: 10 Jumlah Simpul: 129189
output			
gif			

Dapat dilihat bahwa persentase kompresi besar saat min block size dinaikkan, hal ini terjadi karena perbedaan warna pada pixel bersebelahan sangat besar (hitam vs putih) jadi agak

kurang memungkinkan untuk bisa disatukan (variansi tinggi), menyebabkan minBlockSize 2 mendapat persentase besar dalam kompresi.

Mean Absolute Deviation

method percentage threshold min block size	test/a/a.png 0 2 10 1	test/a/a.png 0 2 10 2	test/a/a.png 0 2 20 1
stats	===== stats ===== Waktu Eksekusi: 0.0823448 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 145899 bit Persentase kompresi: 28.4858% Kedalaman QuadTree: 10 Jumlah Simpul: 115349	===== stats ===== Waktu Eksekusi: 0.0633714 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 94549 bit Persentase kompresi: 53.6556% Kedalaman QuadTree: 10 Jumlah Simpul: 70197	===== stats ===== Waktu Eksekusi: 0.0789446 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 136586 bit Persentase kompresi: 33.0507% Kedalaman QuadTree: 10 Jumlah Simpul: 104841
output			
gif			

Mirip dengan sebelumnya, perbedaan warna antara pixel tinggi, menyebabkan mean absolute deviation error juga besar, menyebabkan mereka awalnya tidak menyatu kecuali yang dipaksa dengan minBlockSize 2.

Max Pixel Difference

method percentage threshold min block size	test/a/a.png 0 3 10 1	test/a/a.png 0 3 10 2	test/a/a.png 0 3 20 1
---	-----------------------------------	-----------------------------------	-----------------------------------

stats	<pre>===== stats ===== Waktu Eksekusi: 0.0825274 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 157847 bit Persentase kompresi: 22.6293% Kedalaman QuadTree: 10 Jumlah Simpul: 127677</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.0683909 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 98504 bit Persentase kompresi: 51.717% Kedalaman QuadTree: 10 Jumlah Simpul: 75625</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.0764004 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 147063 bit Persentase kompresi: 27.1915% Kedalaman QuadTree: 10 Jumlah Simpul: 116617</pre>
output			
gif			

Sama lagi dengan sebelumnya, perbedaan warna antara pixel tinggi, menyebabkan max pixel deviation maksimum, menyebabkan mereka awalnya tidak menyatu kecuali yang dipaksa dengan minBlockSize 2.

Entropy

method percentage threshold min block size	test/a/a.png 0 4 1 1	test/a/a.png 0 4 1 2	test/a/a.png 0 4 2 1
stats	<pre>===== stats ===== Waktu Eksekusi: 0.524515 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 201597 bit Persentase kompresi: 1.18472% Kedalaman QuadTree: 10 Jumlah Simpul: 180301</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.443813 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 107468 bit Persentase kompresi: 47.3232% Kedalaman QuadTree: 10 Jumlah Simpul: 91141</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.467106 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 97550 bit Persentase kompresi: 52.1847% Kedalaman QuadTree: 9 Jumlah Simpul: 73357</pre>
output			

gif			
-----	--	--	--

Sama lagi dengan sebelumnya, perbedaan warna antara pixel tinggi, menyebabkan entropi error tinggi, sehingga mereka awalnya tidak menyatu kecuali yang dipaksa dengan minBlockSize 2. Pada threshold 2, entropi 2 sudah cukup besar untuk memungkinkan blending antara putih dan hitam untuk menjadi abu-abu.

Structural Similarity Index

method percentage threshold min block size	test/a/a.png 0 5 0.005 1	test/a/a.png 0 5 0.005 2	test/a/a.png 0 5 0.01 1
stats	===== stats ===== Waktu Eksekusi: 0.176972 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 183277 bit Persentase kompresi: 10.1645% Kedalaman QuadTree: 10 Jumlah Simpul: 169857	===== stats ===== Waktu Eksekusi: 0.134136 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 105689 bit Persentase kompresi: 48.1952% Kedalaman QuadTree: 10 Jumlah Simpul: 90941	===== stats ===== Waktu Eksekusi: 0.148182 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 179820 bit Persentase kompresi: 11.859% Kedalaman QuadTree: 10 Jumlah Simpul: 165757
output			
gif			

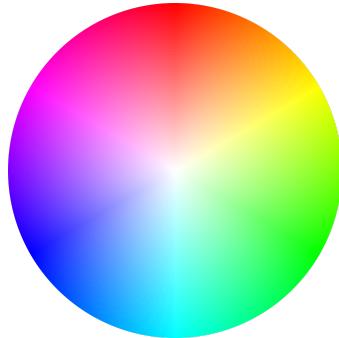
Sama lagi dengan sebelumnya, perbedaan warna antara pixel tinggi, menyebabkan max pixel deviation maksimum, menyebabkan mereka awalnya tidak menyatu kecuali yang dipaksa dengan minBlockSize 2.

Persentase

persentase	test/a/a.png 0.1	test/a/a.png 0.2	test/a/a.png 0.25
stats	<pre>===== stats ===== Waktu Eksekusi: 0.0821446 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 182918 bit Persentase kompresi: 10.3405% Kedalaman QuadTree: 10 Jumlah Simpul: 155353</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.0782112 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 162663 bit Persentase kompresi: 20.2687% Kedalaman QuadTree: 10 Jumlah Simpul: 132421</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.0706664 seconds Ukuran Gambar sebelum: 204014 bit Ukuran Gambar setelah: 152916 bit Persentase kompresi: 25.0463% Kedalaman QuadTree: 10 Jumlah Simpul: 122865</pre>
output			
gif			

Disini kita dapat melihat bahwa kita mendapatkan persentase kompresi untuk 0.2 0.2 dan 0.25 karena adanya persebaran warna yang memungkinkan kita untung menyatukannya, akan tetapi jika dicoba 0.5 maka akan membuat gambar full abu-abu.

Gambar 2 - rgba (test/b/b.png)

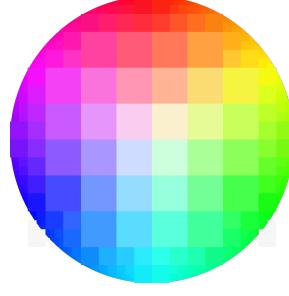
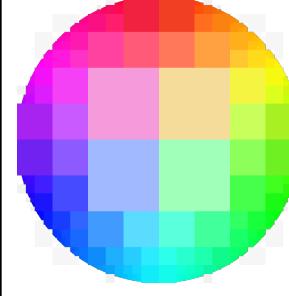
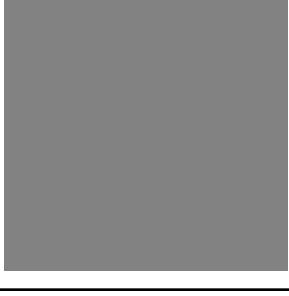


Variance

method percentage threshold min block size	test/b/b.png 0 1 10 1	test/b/b.png 0 1 10 2	test/b/b.png 0 1 20 1
stats	===== stats ===== Waktu Eksekusi: 0.907348 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 215403 bit Persentase kompresi: 78.1209% Kedalaman QuadTree: 11 Jumlah Simpul: 18625	===== stats ===== Waktu Eksekusi: 0.917179 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 215403 bit Persentase kompresi: 78.1209% Kedalaman QuadTree: 11 Jumlah Simpul: 18625	===== stats ===== Waktu Eksekusi: 0.879057 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 209749 bit Persentase kompresi: 78.6952% Kedalaman QuadTree: 11 Jumlah Simpul: 15749
output			
gif			

Disini dapat dilihat bahwa dengan meningkatkan minimum block size tidak mengurangi jumlah simpul ataupun persentase kompresi untuk threshold 10, dengan kata lain block size yang dibentuk sudah ≥ 2

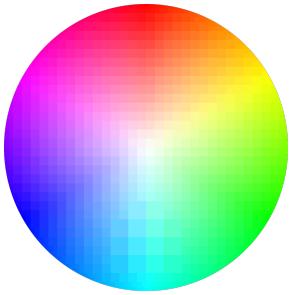
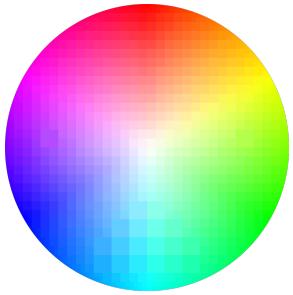
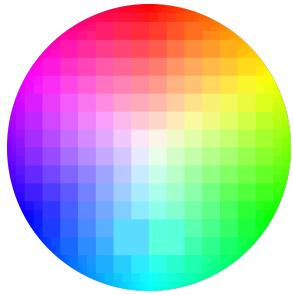
Mean Absolute Deviation

method percentage threshold min block size	test/b/b.png 0 2 10 1	test/b/b.png 0 2 10 2	test/b/b.png 0 2 20 1
stats	===== stats ===== Waktu Eksekusi: 1.21466 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 199785 bit Persentase kompresi: 79.7073% Kedalaman QuadTree: 11 Jumlah Simpul: 8945	===== stats ===== Waktu Eksekusi: 1.21038 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 199785 bit Persentase kompresi: 79.7073% Kedalaman QuadTree: 11 Jumlah Simpul: 8945	===== stats ===== Waktu Eksekusi: 1.20204 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 185850 bit Persentase kompresi: 81.1227% Kedalaman QuadTree: 11 Jumlah Simpul: 4449
output			
gif			

Disini dapat dilihat bahwa dengan meningkatkan minimum block size tidak mengurangi jumlah simpul ataupun persentase kompresi untuk threshold 10, dengan kata lain block size yang dibentuk sudah ≥ 2

Max Pixel Difference

method percentage threshold min block size	test/b/b.png 0 3 10 1	test/b/b.png 0 3 10 2	test/b/b.png 0 3 20 1
---	-----------------------------------	-----------------------------------	-----------------------------------

stats	<pre>===== stats ===== Waktu Eksekusi: 0.891385 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 199744 bit Persentase kompresi: 79.7115% Kedalaman QuadTree: 11 Jumlah Simpul: 8933</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.918054 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 210382 bit Persentase kompresi: 78.6309% Kedalaman QuadTree: 11 Jumlah Simpul: 15937</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.908496 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 208766 bit Persentase kompresi: 78.7951% Kedalaman QuadTree: 11 Jumlah Simpul: 14849</pre>
output			
gif			

Disini dapat dilihat bahwa dengan meningkatkan minimum block size tidak mengurangi jumlah simpul ataupun persentase kompresi untuk threshold 10, dengan kata lain block size yang dibentuk sudah ≥ 2

Entropy

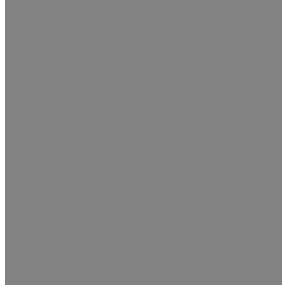
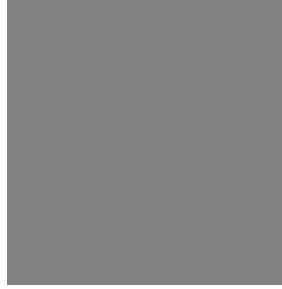
method percentage threshold min block size	test/b/b.png 0 4 1 1	test/b/b.png 0 4 1 2	test/b/b.png 0 4 2 1
stats	<pre>===== stats ===== Waktu Eksekusi: 4.09926 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 270314 bit Persentase kompresi: 72.5435% Kedalaman QuadTree: 11 Jumlah Simpul: 70857</pre>	<pre>===== stats ===== Waktu Eksekusi: 4.16925 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 270314 bit Persentase kompresi: 72.5435% Kedalaman QuadTree: 11 Jumlah Simpul: 70857</pre>	<pre>===== stats ===== Waktu Eksekusi: 5.82047 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 172538 bit Persentase kompresi: 82.4748% Kedalaman QuadTree: 8 Jumlah Simpul: 3393</pre>

output			
gif			

Disini dapat dilihat bahwa dengan meningkatkan minimum block size tidak mengurangi jumlah simpul ataupun persentase kompresi untuk threshold 10, dengan kata lain block size yang dibentuk sudah ≥ 2

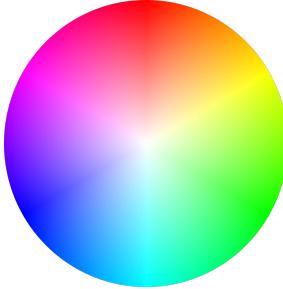
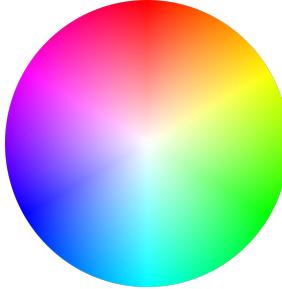
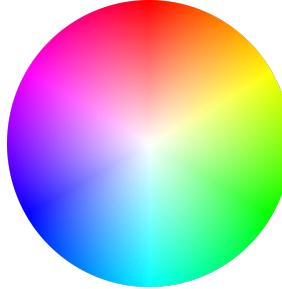
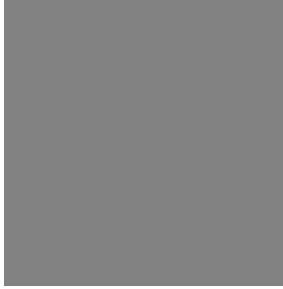
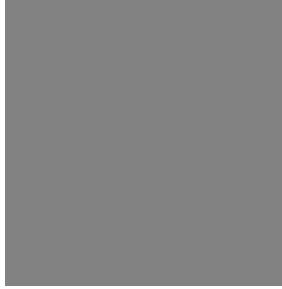
Structural Similarity Index

method percentage threshold min block size	test/b/b.png 0 5 0.005 1	test/b/b.png 0 5 0.005 2	test/b/b.png 0 5 0.01 1
stats	===== stats ===== Waktu Eksekusi: 3.64761 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 823885 bit Persentase kompresi: 16.3157% Kedalaman QuadTree: 11 Jumlah Simpul: 1105441	===== stats ===== Waktu Eksekusi: 3.86834 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 823885 bit Persentase kompresi: 16.3157% Kedalaman QuadTree: 11 Jumlah Simpul: 1105441	===== stats ===== Waktu Eksekusi: 3.69919 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 823956 bit Persentase kompresi: 16.3085% Kedalaman QuadTree: 11 Jumlah Simpul: 1105409
output			

gif			
-----	---	--	---

Disini dapat dilihat bahwa dengan meningkatkan minimum block size tidak mengurangi jumlah simpul ataupun persentase kompresi untuk threshold 10, dengan kata lain block size yang dibentuk sudah ≥ 2

Persentase

persentase	test/b/b.png 0.3	test/b/b.png 0.4	test/b/b.png 0.5
stats	<pre>===== stats ===== Waktu Eksekusi: 46.0565 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 591400 bit Persentase kompresi: 39.9299% Kedalaman QuadTree: 11 Jumlah Simpul: 265489</pre>	<pre>===== stats ===== Waktu Eksekusi: 48.6179 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 448123 bit Persentase kompresi: 54.4829% Kedalaman QuadTree: 11 Jumlah Simpul: 157341</pre>	<pre>===== stats ===== Waktu Eksekusi: 46.4448 seconds Ukuran Gambar sebelum: 984516 bit Ukuran Gambar setelah: 448123 bit Persentase kompresi: 54.4829% Kedalaman QuadTree: 11 Jumlah Simpul: 157341</pre>
output			
gif			

Disini dapat dilihat bahwa 0.4 dan 0.5 membuat kompresi ke 54.4829% (sama), karena variansi dengan presisi $1e^{-8}$ dalam step binary search tidak memisahkan warna yang selisihnya dekat, jadi dianggap menjadi satu blok.

Gambar 3 - rgb (test/c/c.jpg)



Variance

method percentage threshold min block size	test/c/c.jpg 0 1 10 1	test/c/c.jpg 0 1 10 2	test/c/c.jpg 0 1 20 1
stats	===== stats ===== Waktu Eksekusi: 0.0257045 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5509 bit Persentase kompresi: 52.6108% Kedalaman QuadTree: 5 Jumlah Simpul: 341	===== stats ===== Waktu Eksekusi: 0.0255843 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5509 bit Persentase kompresi: 52.6108% Kedalaman QuadTree: 5 Jumlah Simpul: 341	===== stats ===== Waktu Eksekusi: 0.0274488 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5509 bit Persentase kompresi: 52.6108% Kedalaman QuadTree: 5 Jumlah Simpul: 341
output			
gif			

Variansi dari gambar ini rendah, karena warnanya gradasi yang lumayan smooth sehingga semuanya membuat quadtree yang sama, menyebabkan kompresi yang sama. Untuk min block size 1 ke 2 tetap sama dalam treshold 10 karena pada threshold 10 pun sudah mengambil >2 blok.

Mean Absolute Deviation

method percentage threshold min block size	test/c/c.jpg 0 2 10 1	test/c/c.jpg 0 2 10 2	test/c/c.jpg 0 2 20 1
stats	===== stats ===== Waktu Eksekusi: 0.0244728 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5839 bit Persentase kompresi: 49.772% Kedalaman QuadTree: 3 Jumlah Simpul: 21	===== stats ===== Waktu Eksekusi: 0.0250525 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5839 bit Persentase kompresi: 49.772% Kedalaman QuadTree: 3 Jumlah Simpul: 21	===== stats ===== Waktu Eksekusi: 0.020534 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5319 bit Persentase kompresi: 54.2452% Kedalaman QuadTree: 2 Jumlah Simpul: 5
output			
gif			

Disini kita juga dapat melihat hal yang terjadi di variansi karena untuk treshold 10, setiap simpul sudah >2 block size. Untuk threshold 20, warna dari gambar menyatu karena dibawah threshold, membuatnya jadi 5 simpul saja.

Max Pixel Difference

method percentage threshold min block size	test/c/c.jpg 0 3 10 1	test/c/c.jpg 0 3 10 2	test/c/c.jpg 0 3 20 1
stats	===== stats ===== Waktu Eksekusi: 0.029451 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5526 bit Persentase kompresi: 52.4645% Kedalaman QuadTree: 5 Jumlah Simpul: 329	===== stats ===== Waktu Eksekusi: 0.0398759 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5526 bit Persentase kompresi: 52.4645% Kedalaman QuadTree: 5 Jumlah Simpul: 329	===== stats ===== Waktu Eksekusi: 0.0360313 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 6013 bit Persentase kompresi: 48.2753% Kedalaman QuadTree: 4 Jumlah Simpul: 69
output			

gif			
-----	---	--	---

Untuk Max Pixel Difference terjadi hal yang mirip yaitu pada threshold 10, setiap simpul sudah memiliki block size > 2 jadi tidak berpengaruh pada perubahan block size 1 ke 2. Namun untuk threshold 20 warna menyatu lagi (lagi-lagi ini gampang terjadi karena gradasi).

Entropy

method percentage threshold min block size	test/c/c.jpg 0 4 1 1	test/c/c.jpg 0 4 1 2	test/c/c.jpg 0 4 2 1
stats	<pre>===== stats ===== Waktu Eksekusi: 0.207224 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5171 bit Persentase kompresi: 55.5183% Kedalaman QuadTree: 10 Jumlah Simpul: 7937</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.210355 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5171 bit Persentase kompresi: 55.5183% Kedalaman QuadTree: 9 Jumlah Simpul: 7893</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.16779 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5474 bit Persentase kompresi: 52.9118% Kedalaman QuadTree: 6 Jumlah Simpul: 1149</pre>
output			
gif			

Untuk entropi 2 itu sudah besar sekali beda dengan 1, dapat dilihat bahwa minimum block size berpengaruh pada entropy threshold 1, berarti ada simpul yang block sizenya 1 pixel.

Structural Similarity Index

method percentage threshold min block size	test/c/c.jpg 0 5 0.005 1	test/c/c.jpg 0 5 0.005 2	test/c/c.jpg 0 5 0.01 1
---	--------------------------------------	--------------------------------------	-------------------------------------

stats	<pre>===== stats ===== Waktu Eksekusi: 0.18079 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5212 bit Percentase kompresi: 55.1656% Kedalaman QuadTree: 10 Jumlah Simpul: 162033</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.153979 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5191 bit Percentase kompresi: 55.3462% Kedalaman QuadTree: 9 Jumlah Simpul: 87249</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.189669 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5212 bit Percentase kompresi: 55.1656% Kedalaman QuadTree: 10 Jumlah Simpul: 161745</pre>
output			
gif			

Untuk SSIM, dapat dilihat bahwa benar jumlah simpul lebih banyak saat threshold lebih kecil (lebih presisi, bandingkan threshold 0.005 dengan 0.01), dan minimum block size 1 ke 2 menyatakan bahwa threshold 0.005 ada simpul dengan block size = 1 pixel.

Persentase

persentase	test/c/c.jpg 0.2	test/c/c.jpg 0.3	test/c/c.jpg 0.4
stats	<pre>===== stats ===== Waktu Eksekusi: 0.0411949 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5192 bit Percentase kompresi: 55.3376% Kedalaman QuadTree: 10 Jumlah Simpul: 71953</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.0518487 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5192 bit Percentase kompresi: 55.3376% Kedalaman QuadTree: 10 Jumlah Simpul: 71953</pre>	<pre>===== stats ===== Waktu Eksekusi: 0.0517768 seconds Ukuran Gambar sebelum: 11625 bit Ukuran Gambar setelah: 5192 bit Percentase kompresi: 55.3376% Kedalaman QuadTree: 10 Jumlah Simpul: 71953</pre>
output			
gif			

Dengan persentase kita mendapat 3 hasil yang sama, hal ini bisa terjadi karena kita menggunakan metode yang sama lalu dilakukan binary search, yang berarti saat suatu metode tidak bisa membedakan lagi, maka tidak akan terbentuk perbedaan

4.2 Analisis Kompleksitas Algoritma Program

Quadtree merupakan implementasi algoritma divide and conquer yang membagi gambar menjadi empat kuadran secara rekursif hingga memenuhi kriteria tertentu. Algoritma ini bekerja dengan pendekatan top-down, dimulai dari gambar utuh kemudian membaginya hingga mencapai kondisi pemberhentian.

Kompleksitas Waktu

1. **Konstruksi Quadtree:** Kompleksitas waktu untuk membangun Quadtree adalah $O(n \log n)$, dimana n adalah jumlah piksel dalam gambar. Ini karena:
 - Setiap piksel diproses setidaknya sekali.
 - Pada setiap level tree, seluruh gambar diproses ulang untuk menghitung error.
 - Kedalaman maksimum tree adalah $\log_4(n)$ karena setiap node dibagi menjadi 4 node anak.
2. **Perhitungan Error:** Metode perhitungan error yang digunakan memiliki kompleksitas $O(w \times h)$ dimana w dan h adalah lebar dan tinggi blok yang sedang diproses:
 - **Variance (pickMethod 1):** Memerlukan dua pass melalui seluruh piksel di blok.
 - **Mean Absolute Deviation (pickMethod 2):** Memerlukan satu pass untuk menghitung selisih absolut.
 - **Max Pixel Difference (pickMethod 3):** Memerlukan satu pass untuk menemukan perbedaan maksimum.
 - **Entropy (pickMethod 4):** Memerlukan satu pass untuk menghitung frekuensi dan kemudian perhitungan entropy.
 - **SSIM (pickMethod 5):** Paling kompleks, memerlukan beberapa pass untuk menghitung mean, variance, dan covariance.
3. **Rekonstruksi Gambar:** Kompleksitas waktu untuk rekonstruksi gambar (`buildMatrix`) adalah $O(n)$, karena setiap piksel dikunjungi tepat sekali untuk menggambar gambar hasil kompresi.

Kompleksitas Ruang

1. **Struktur Quadtree:** Dalam kasus terburuk (semua node terus dibagi), kompleksitas ruang adalah $O(n)$, karena jumlah total node dalam Quadtree lengkap dengan n piksel dapat mencapai $4n/3$.
2. **Gambar Sementara:** Metode error (seperti SSIM) memerlukan pembuatan gambar sementara, yang memerlukan $O(w \times h)$ ruang tambahan.

Algoritma Quadtree untuk kompresi gambar mendemonstrasikan kekuatan pendekatan divide and conquer dalam pemrosesan gambar. Kompleksitas waktu $O(n \log n)$ menjadikannya efisien untuk gambar dengan ukuran sedang. Metode error yang berbeda menawarkan trade-off antara kecepatan komputasi dan akurasi visual, dengan SSIM memberikan hasil paling mendekati persepsi manusia meskipun memiliki kompleksitas komputasi tertinggi.

Implementasi ini menggunakan strategi pemberhentian ganda (threshold error dan ukuran blok minimum) yang menyeimbangkan kualitas gambar dan efisiensi komputasi, membuatnya cocok untuk aplikasi kompresi gambar yang membutuhkan kualitas visual yang dapat disesuaikan.

Lampiran

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

Github Repository

https://github.com/yonatan-nyo/Tucil2_13523036_13523075

Link Google Docs (untuk melihat GIF ;-; udh dipasang tapi tidak dapat dilihat dengan pdf)

[https://docs.google.com/document/d/1nkakzxu1gYFFFizl3OcCdl8tEZceOE4IZskAfS2TMzl/edit?
usp=sharing](https://docs.google.com/document/d/1nkakzxu1gYFFFizl3OcCdl8tEZceOE4IZskAfS2TMzl/edit?usp=sharing)