Laporan Tugas Kecil IF2211 Strategi Algoritma

Kompresi Gambar Dengan Metode Quadtree

Disusun Oleh:

Qodri Azkarayan	(13523010)
Andri Nurdianto	(13523145)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

Daftar Isi

Bab 1

1.1 Deskripsi Masalah

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

2,1 Algoritma

2.1.1 Algoritma fungsi quadtree()

A. Pemeriksaan Nilai Error
   - Algoritma pertama-tama menghitung error dari blok data yang diberikan. Tergantung pada masukan pengguna, algoritma memilih cara untuk menghitung error (variance, MAD, MPD, atau entropi)
   - Threshold digunakan untuk membandingkan nilai error. Jika error-nya lebih kecil dari threshold atau ukuran blok terlalu kecil, maka blok dianggap cukup baik dan tidak perlu dibagi lebih lanjut (sudah "normalisasi").

B. Pembagian Blok (Divide)
   - Jika error melebihi threshold dan ukuran blok setelah dibagi 4 masih lebih besar dari ukuran minimum blok, algoritma membagi blok menjadi empat bagian yang lebih kecil. Pembagian dilakukan berdasarkan posisi tengah (midX dan midY) dari blok, yaitu membagi lebar dan tinggi menjadi dua bagian yang sama.
   - Keempat bagian ini disebut q1, q2, q3, dan q4:
     - q1 adalah bagian kiri atas.
     - q2 adalah bagian kanan atas.
     - q3 adalah bagian kiri bawah.
     - q4 adalah bagian kanan bawah.

C. Rekursi (Conquer)
   Setelah membagi blok menjadi empat bagian, algoritma memanggil dirinya sendiri (rekursif) untuk memproses masing-masing bagian ini. Proses ini terus berulang untuk setiap bagian yang lebih kecil sampai memenuhi kondisi pembatas (misalnya error di bawah threshold atau ukuran blok terlalu kecil).

D. Menyusun Kembali
   Setelah pemrosesan selesai untuk setiap bagian, algoritma akan menyusun kembali hasil-hasil tersebut ke dalam blok asli. Ini dilakukan dengan menggantikan nilai-nilai dalam blok asli dengan hasil dari keempat kuadran yang telah diproses sebelumnya.

2.2 Pseudocode

```
FUNCTION Quadtree(blok, width, height, errorMeasurement,
threshold, minSize, simpul, depth, maxDepth)
    simpul++
    IF depth > maxDepth THEN
        maxDepth = depth
```

```
    // Hitung error berdasarkan metode yang dipilih
    IF errorMeasurement == 1 THEN
        errorValue = HitungVariance(blok)
    ELSE IF errorMeasurement == 2 THEN
        errorValue = HitungMAD(blok)
    ELSE IF errorMeasurement == 3 THEN
        errorValue = HitungMPD(blok)
    ELSE IF errorMeasurement == 4 THEN
        errorValue = HitungEntropy(blok)

    // Jika error kecil atau ukuran terlalu kecil, normalisasi
blok dan berhenti
    IF errorValue <= threshold OR (width * height / 4) <
minSize THEN
        NormalisasiWarna(blok)
        RETURN

    // Bagi gambar menjadi 4 kuadran
    midX = width / 2
    midY = height / 2
    rightWidth  = IF genap THEN midX else midX + 1
    bottomHeight = IF genap THEN midY else midY + 1

    Kuadran q1, q2, q3, q4

    UNTUK setiap pixel (x, y) di blok:
        index = y * width + x
        IF y < midY:
            IF x < midX: TAMBAH blok[index] ke q1
            ELSE:        TAMBAH blok[index] ke q2
        ELSE:
            IF x < midX: TAMBAH blok[index] ke q3
            ELSE:        TAMBAH blok[index] ke q4

    // Rekursif ke masing-masing kuadran
    Quadtree(q1, midX, midY, errorMeasurement, threshold,
minSize, simpul, depth + 1, maxDepth)
    Quadtree(q2, rightWidth, midY, errorMeasurement,
threshold, minSize, simpul, depth + 1, maxDepth)
    Quadtree(q3, midX, bottomHeight, errorMeasurement,
threshold, minSize, simpul, depth + 1, maxDepth)
    Quadtree(q4, rightWidth, bottomHeight, errorMeasurement,
threshold, minSize, simpul, depth + 1, maxDepth)
```

```
    // Gabungkan kembali hasil dari 4 kuadran ke blok utama
    UNTUK setiap pixel (x, y) di blok:
        index = y * width + x
        IF y < midY:
            IF x < midX: blok[index] = q1[y * midX + x]
            ELSE:        blok[index] = q2[y * rightWidth + (x
- midX)]
        ELSE:
            IF x < midX: blok[index] = q3[(y - midY) * midX +
x]
            ELSE:        blok[index] = q4[(y - midY) *
rightWidth + (x - midX)]

END FUNCTION
```

2.3 Source Program

main.cpp

```
#include <iostream>
#include <chrono>
#include "compress.cpp"
#include "image.cpp"
#include "menu.cpp"
#include "pixel.hpp"

extern std::vector<Pixel> blok;
extern int width, height;

int main() {
    std::cout << "Masukan alamat gambar yang ingin
dikompresi: " << std::endl;
    inputFile();
    checkInput();
    imageToRGB(filePath);
    uintmax_t inputSize = getFileSize(filePath);
    // Pilih Error Measurement
    inputMeasurement();

    // Pilih ambang batas
    inputThreshold();

    // Pilih ukuran blok minimum
    inputBlockMinimum();

    auto start = std::chrono::high_resolution_clock::now();
    int simpul=0;
    int maxDepth=0;
```

```
    quadtree(blok, width, height, choiceMeasurement,
choiceThreshold, choiceBlockMinimum, simpul, 0, maxDepth);
    auto end = std::chrono::high_resolution_clock::now();

    std::cout << "Masukan alamat gambar hasil kompresi: " <<
std::endl;

    std::cin.ignore();
    std::string outputPath;
    std::getline(std::cin, outputPath);
    RGBToImage(blok, outputPath, height, width);
    uintmax_t outputSize = getFileSize(outputPath);

    auto duration =
std::chrono::duration_cast<std::chrono::milliseconds>(end -
start);
    std::cout << "Waktu eksekusi: " << duration.count() <<
"ms" << std::endl;
    std::cout << "Ukuran file awal: " << inputSize/1024 <<
"kb" << std::endl;
    std::cout << "Ukuran file akhir: " << outputSize/1024 <<
"kb" << std::endl;
    std::cout << "Besar kompresi: " << (inputSize -
outputSize) * 100 / inputSize << "%" << std::endl;
    std::cout << "Jumlah simpul: " << simpul << std::endl;
    std::cout << "Kedalaman pohon: " << maxDepth <<
std::endl;
    return 0;
}
```

pixel.hpp

```
#ifndef PIXEL_HPP
#define PIXEL_HPP

struct Pixel {
    int r, g, b;
    Pixel(int R, int G, int B);
    Pixel& operator=(const Pixel& other);
};

#endif
```

pixel.cpp

```
#include "pixel.hpp"

Pixel::Pixel(int R, int G, int B) : r(R), g(G), b(B) {}

Pixel& Pixel::operator=(const Pixel& other) {
```

```
    if (this == &other) {
        return *this;
    }
    r = other.r;
    g = other.g;
    b = other.b;
    return *this;
}
```

menu.cpp

```
#include <iostream>

int choiceMeasurement;
double choiceThreshold;
double choiceBlockMinimum;

void inputMeasurement() {
    try {
        std::cout << "Pilih metode perhitungan error" <<
std::endl;
        std::cout << "1. Variance" << std::endl;
        std::cout << "2. Mean Absolute Deviation (MAD)" <<
std::endl;
        std::cout << "3. Max Pixel Difference" << std::endl;
        std::cout << "4. Entropy" << std::endl;
        std::cout << "Masukan nomor: ";
        std::cin >> choiceMeasurement;

        if (std::cin.fail()) {
            throw "Input harus berupa angka";
        }

        if (choiceMeasurement == 1) {
            std::cout << "Anda memilih Variance" << std::endl
<< std::endl;
        } else if (choiceMeasurement == 2) {
            std::cout << "Anda memilih Mean Absolute
Deviation" << std::endl << std::endl;
        } else if (choiceMeasurement == 3) {
            std::cout << "Anda memilih Max Pixel Difference"
<< std::endl << std::endl;
        } else if (choiceMeasurement == 4) {
            std::cout << "Anda memilih Entropy" << std::endl
<< std::endl << std::endl;
        } else {
            std::cout << "\nAngka tersebut tidak ada dalam
pilihan, silahkan pilih lagi." << std::endl;
            inputMeasurement();
        }
```

```
    } catch (...) {
        std::cout << "Input harus berupa angka" << std::endl;
        std::cin.clear();
        std::cin.ignore(1000, '\n');
        inputMeasurement();
    }
}

void inputThreshold() {
    std::cout << "Tentukan ambang batas" << std::endl;

    double minThreshold = 0;
    double maxThreshold = 0;

    if (choiceMeasurement == 1) {
        std::cout << "Ambang batas berada pada rentang 0 -
1000" << std::endl ;
        maxThreshold = 1000;
    } else if (choiceMeasurement == 2) {
        std::cout << "Ambang batas berada pada rentang 0 -
25" << std::endl;
        maxThreshold = 25;
    } else if (choiceMeasurement == 3) {
        std::cout << "Ambang batas berada pada rentang 0 -
50" << std::endl;
        maxThreshold = 50;
    } else if (choiceMeasurement == 4) {
        std::cout << "Ambang batas berada pada rentang 0 -
10" << std::endl;
        maxThreshold = 10;
    }

    bool valid = false;

    do {
        try {
            std::cout << "Masukan ambang batas: ";
            std::cin >> choiceThreshold;

            if (std::cin.fail()) {
                throw "Input harus berupa angka";
            }

            if (choiceThreshold >= minThreshold &&
choiceThreshold <= maxThreshold) {
                valid = true;
            } else {
                std::cout << "Ambang batas tidak valid.
Silakan coba lagi." << std::endl;
            }
```

```
        } catch (...) {
            std::cout << "Input harus berpa angka" <<
std::endl;
            std::cin.clear();
            std::cin.ignore(1000, '\n');
        }
    } while (!valid);

    std::cout << std::endl;
}

void inputBlockMinimum() {
    std::cout << "Masukan ukuran blok minimum: ";
    std::cin >> choiceBlockMinimum;
    std::cout << std::endl;
}
```

inputhandling.cpp

```
#include <iostream>
#include <string>
#include <filesystem>

std::string filePath;
bool isImageFormat(const std::string directory);

void inputFile() {
    std::getline(std::cin, filePath);
}

void checkInput() {
    while (true) {
        if (std::filesystem::exists(filePath) &&
isImageFormat(filePath)) {
            std::cout << "File ditemukan" << std::endl <<
std::endl;
            break;
        } else if (std::filesystem::exists(filePath) &&
!isImageFormat(filePath)) {
            std::cout << "File bukan merupakan format gambar
(.jpg, .png, .jpeg)" << std::endl;
        } else {
            std::cout << "File tidak ditemukan" << std::endl
<< std::endl;
        }

        std::cout << "Masukan alamat gambar yang ingin
dikompresi: " << std::endl;
        std::getline(std::cin, filePath);
    }
```

```
}

bool isImageFormat(const std::string directory) {
    std::filesystem::path filePath(directory);
    std::string format = filePath.extension().string();

    return format == ".jpg" || format == ".png" || format ==
".jpeg";
}
```

image.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <tuple>
#include "inputhandling.cpp"
#include "pixel.hpp"

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.hpp"

#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.hpp"

std::vector<Pixel> blok;
int width;
int height;

uintmax_t getFileSize(const std::string& filePath) {
    return std::filesystem::file_size(filePath);
}

void imageToRGB(std::string& filePath) {
    int channels;

    unsigned char* image = stbi_load(filePath.c_str(),
&width, &height, &channels, 3);

    if (!image) {
        std::cerr << "Gagal membuka gambar \n";
        return;
    }

    blok.clear();
    blok.reserve(width * height);

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int idx = (y * width + x) * 3;
```

```
            int r = image[idx];
            int g = image[idx + 1];
            int b = image[idx + 2];

            blok.push_back(Pixel(r, g, b));
        }
    }

    stbi_image_free(image);
}

void RGBToImage(std::vector<Pixel>& blok, std::string&
outputPath, int height, int width) {
    std::vector<unsigned char> imageData(width * height * 3);
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int idx = (y * width + x) * 3;
            const Pixel& pixel = blok[y * width + x];
            imageData[idx] = static_cast<unsigned
char>(pixel.r);
            imageData[idx + 1] = static_cast<unsigned
char>(pixel.g);
            imageData[idx + 2] = static_cast<unsigned
char>(pixel.b);
        }
    }
    if (stbi_write_jpg(outputPath.c_str(), width, height, 3,
imageData.data(), 90)) {
        std::cout << "Gambar berhasil disimpan ke: " <<
outputPath << std::endl;
    } else {
        std::cerr << "Gagal menyimpan gambar." << std::endl;
    }
}
```

calculation.hpp

```
#ifndef CALCULATION_HPP
#define CALCULATION_HPP

#include <vector>
#include "pixel.hpp"

double rataRata(const std::vector<Pixel>& blok, char
channel);
Pixel averageColor(const std::vector<Pixel>& blok);
void normalize(std::vector<Pixel>& blok);

#endif
```

calculation.cpp

```
#include <vector>
#include "calculation.hpp"
#include "pixel.hpp"

double rataRata(const std::vector<Pixel>& blok, char channel)
{
    double sum = 0;
    for (int i = 0; i < blok.size(); i++) {
        const Pixel& pixel = blok[i];
        if (channel == 'r') sum += pixel.r;
        else if (channel == 'g') sum += pixel.g;
        else if (channel == 'b') sum += pixel.b;
    }
    return sum / blok.size();
}

Pixel averageColor(const std::vector<Pixel>& blok){
    return Pixel(rataRata(blok,'r'), rataRata(blok,'g'),
rataRata(blok,'b'));
}

void normalize(std::vector<Pixel>& blok) {
    Pixel avg = averageColor(blok);
    for (int i = 0; i < blok.size(); i++) {
        blok[i] = avg;
    }
}
```

compress.cpp

```
#include <iostream>
#include <vector>
#include "pixel.hpp"
#include "calculation.hpp"
#include "ErrorMeasurement/variance.cpp"
#include "ErrorMeasurement/mad.cpp"
#include "ErrorMeasurement/mpd.cpp"
#include "ErrorMeasurement/entropy.cpp"

double rataRata(const std::vector<Pixel>& blok, char channel)
{
    double sum = 0;
    for (int i = 0; i < blok.size(); i++) {
        const Pixel& pixel = blok[i];
        if (channel == 'r') sum += pixel.r;
        else if (channel == 'g') sum += pixel.g;
        else if (channel == 'b') sum += pixel.b;
    }
    return sum / blok.size();
```

```
}

Pixel averageColor(const std::vector<Pixel>& blok) {
    return Pixel(rataRata(blok, 'r'), rataRata(blok, 'g'),
rataRata(blok, 'b'));
}

void normalize(std::vector<Pixel>& blok) {
    Pixel avg = averageColor(blok);
    for (int i = 0; i < blok.size(); i++) {
        blok[i] = avg;
    }
}

void quadtree(std::vector<Pixel>& blok, int width, int
height, int errorMeasurement, double threshold, double
minSize, int &simpul, int depth, int &maxDepth) {
    simpul++;
    if (depth > maxDepth) {
        maxDepth = depth;
    }

    double errorValue;
    if (errorMeasurement == 1) {
        errorValue = varianceError(blok);
    } else if (errorMeasurement == 2) {
        errorValue = madError(blok);
    } else if (errorMeasurement == 3) {
        errorValue = mpdError(blok);
    } else if (errorMeasurement == 4) {
        errorValue = entropyError(blok);
    }

    if (errorValue <= threshold || width * height / 4 <
minSize) {
        normalize(blok);
        return;
    }

    bool widthEven = (width % 2 == 0);
    bool heightEven = (height % 2 == 0);
    int midX = width / 2;
    int midY = height / 2;
    int rightWidth = widthEven ? midX : midX + 1;
    int bottomHeight = heightEven ? midY : midY + 1;

    std::vector<Pixel> q1, q2, q3, q4;

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
```

```
            int index = y * width + x;
            if (y < midY) {
                if (x < midX)
                    q1.push_back(blok[index]);
                else
                    q2.push_back(blok[index]);
            } else {
                if (x < midX)
                    q3.push_back(blok[index]);
                else
                    q4.push_back(blok[index]);
            }
        }
    }

    quadtree(q1, midX, midY, errorMeasurement, threshold,
minSize, simpul, depth + 1, maxDepth);
    quadtree(q2, rightWidth, midY, errorMeasurement,
threshold, minSize, simpul, depth + 1, maxDepth);
    quadtree(q3, midX, bottomHeight, errorMeasurement,
threshold, minSize, simpul, depth + 1, maxDepth);
    quadtree(q4, rightWidth, bottomHeight, errorMeasurement,
threshold, minSize, simpul, depth + 1, maxDepth);

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int index = y * width + x;
            if (y < midY) {
                if (x < midX)
                    blok[index] = q1[y * midX + x];
                else
                    blok[index] = q2[y * rightWidth + (x -
midX)];
            } else {
                if (x < midX)
                    blok[index] = q3[(y - midY) * midX + x];
                else
                    blok[index] = q4[(y - midY) * rightWidth
+ (x - midX)];
            }
        }
    }
}
```

variance.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include "../pixel.hpp"
```

```
#include "../calculation.hpp"

double varians(const std::vector<Pixel>& blok, char color,
double mean) {
    double sum = 0;
    for (int i = 0; i < blok.size(); i++) {
        const Pixel& pixel = blok[i];
          if (color == 'r'){
            sum += (pixel.r - mean) * (pixel.r - mean);
        }
        else if (color == 'g'){
            sum += (pixel.g - mean) * (pixel.g - mean);
        }
        else if (color == 'b'){
            sum += (pixel.b - mean) * (pixel.b - mean);
        }
    }
    return sum / blok.size();
}

double varianceError(const std::vector<Pixel>& blok) {
    double meanR = rataRata(blok, 'r');
    double meanG = rataRata(blok, 'g');
    double meanB = rataRata(blok, 'b');
    double varR = varians(blok, 'r', meanR);
    double varG = varians(blok, 'g', meanG);
    double varB = varians(blok, 'b', meanB);
    double error = (varR + varG + varB) / 3;
    return error;
}
```

mad.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include "../pixel.hpp"
#include "../calculation.hpp"

double difference(const std::vector<Pixel>& blok, char color,
double mean) {
    double sum = 0;
    for (int i = 0; i < blok.size(); i++) {
        const Pixel& pixel = blok[i];
          if (color == 'r'){
            sum += fabs(pixel.r - mean);
        }
        else if (color == 'g'){
            sum += fabs(pixel.g - mean);
        }
```

```
        else if (color == 'b'){
            sum += fabs(pixel.b - mean);
        }
    }
    return sum / blok.size();
}

double madError(const std::vector<Pixel>& blok) {
    double meanR = rataRata(blok, 'r');
    double meanG = rataRata(blok, 'g');
    double meanB = rataRata(blok, 'b');
    double varR = difference(blok, 'r', meanR);
    double varG = difference(blok, 'g', meanG);
    double varB = difference(blok, 'b', meanB);
    double error = (varR + varG + varB) / 3;
    return error;
}
```

mpd.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include "../pixel.hpp"
#include "../calculation.hpp"

double mpdError(const std::vector<Pixel>& blok) {
    double maksR;
    double maksG;
    double maksB;
    double minR;
    double minG;
    double minB;
    for (int i = 0; i < blok.size(); i++){
        const Pixel& pixel = blok[i];
        if(i==0){
            maksR = pixel.r;
            maksG = pixel.g;
            maksB = pixel.b;
            minR = pixel.r;
            minG = pixel.g;
            minB = pixel.b;
        }else{
            if(pixel.r > maksR){
                maksR = pixel.r;
            }
            if(pixel.r < minR){
                minR = pixel.r;
            }
            if(pixel.g > maksG){
```

```
                maksG = pixel.g;
            }
            if(pixel.g < minG){
                minG = pixel.g;
            }
            if(pixel.b > maksB){
                maksB = pixel.b;
            }
            if(pixel.b < minB){
                minB = pixel.b;
            }
        }
    }
    double error = (maksR + maksG + maksB - minR - minG -
minB)/3;
    //std::cout << "Error dalam blok: " << error <<
std::endl;
    return error;
}
```

entropy.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include "../pixel.hpp"
#include "../calculation.hpp"

int count(const std::vector<Pixel>& blok, char color, int p){
    int hasil = 0;
    for (int i = 0; i < blok.size(); i++) {
        const Pixel& pixel = blok[i];
        if (color == 'r' && pixel.r == p) hasil++;
        else if (color == 'g' && pixel.g == p) hasil++;
        else if (color == 'b' && pixel.b == p) hasil++;
    }
    return hasil;
}

double entropyR(const std::vector<Pixel>& blok){
    double total = 0;
    for (int i = 0; i < 256; i++){
        double jumlah = count(blok, 'r', i);
        if(jumlah != 0){
          double prob = jumlah / blok.size();
          total += prob * log(prob) / log(2);
          }
    }
    return -1 * total;
}
```

```
double entropyG(const std::vector<Pixel>& blok){
    double total = 0;
    for (int i = 0; i < 256; i++){
        double jumlah = count(blok, 'g', i);
        if(jumlah != 0){
          double prob = jumlah / blok.size();
          total += prob * log(prob) / log(2);
          }
    }
    return -1 * total;
}

double entropyB(const std::vector<Pixel>& blok){
    double total = 0;
    for (int i = 0; i < 256; i++){
        double jumlah = count(blok, 'b', i);
        if(jumlah != 0){
          double prob = jumlah / blok.size();
          total += prob * log(prob) / log(2);
          }
    }
    return -1 * total;
}

double entropyError(const std::vector<Pixel>& blok) {
    return (entropyR(blok) + entropyG(blok) +
entropyB(blok))/3;
}
```

Bab 3

3.1 Hasil Eksperimen

3.1.1 Test Case 1

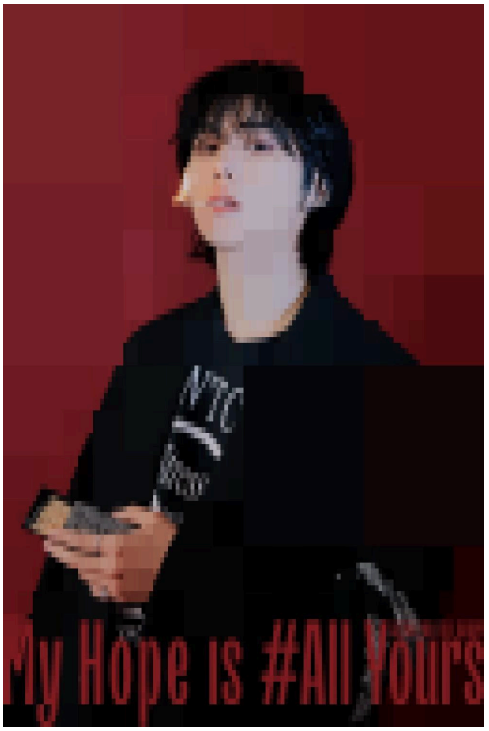
Input terminal

```
Masukan alamat gambar yang ingin dikompresi:
C:\Stima\rocky.jpeg
File ditemukan

Pilih metode perhitungan error
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukan nomor: 1
Anda memilih Variance

Tentukan ambang batas
Ambang batas berada pada rentang 0 - 1000
Masukan ambang batas: 200

Masukan ukuran blok minimum: 25

Masukan alamat gambar hasil kompresi:
C:\Stima\rocky_compressed.jpeg
```

Output terminal

```
Gambar berhasil disimpan ke: C:\Stima\rocky_compressed.jpeg
Waktu eksekusi: 556ms
Ukuran file awal: 132kb
Ukuran file akhir: 110kb
Besar kompresi: 16%
Jumlah simpul: 5157
Kedalaman pohon: 7
```

Gambar awal	Gambar akhir



3.1.2 Test Case 2

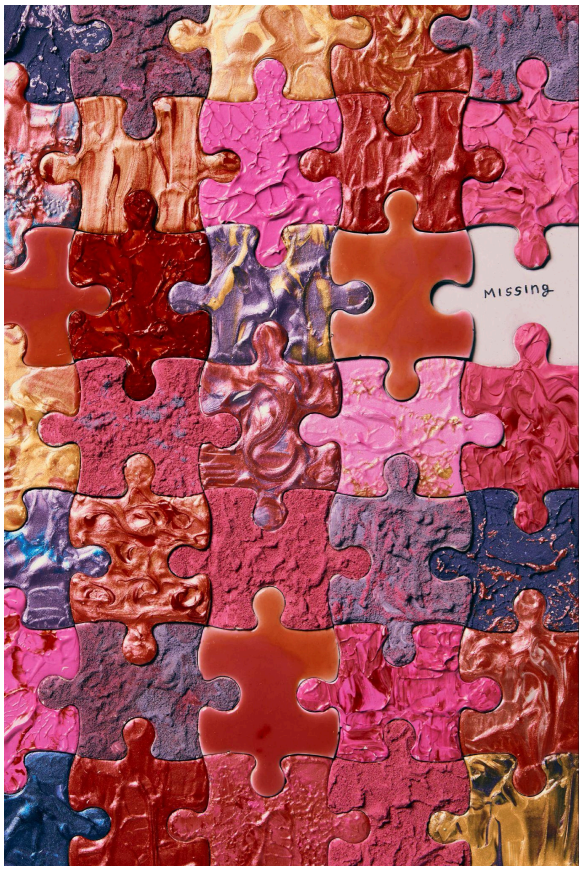
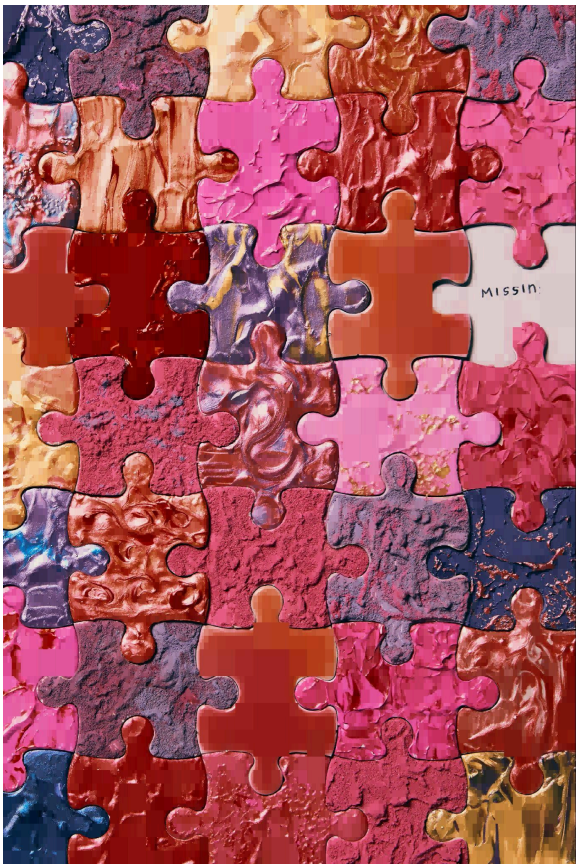
Input terminal

```
Masukan alamat gambar yang ingin dikompresi:
C:\Stima\puzzle.jpeg
File ditemukan

Pilih metode perhitungan error
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukan nomor: 2
Anda memilih Mean Absolute Deviation

Tentukan ambang batas
Ambang batas berada pada rentang 0 - 25
Masukan ambang batas: 15

Masukan ukuran blok minimum: 1

Masukan alamat gambar hasil kompresi:
C:\Stima\puzzle_compressed.jpeg
```

Output terminal

```
Gambar berhasil disimpan ke: C:\Stima\puzzle_compressed.jpeg
Waktu eksekusi: 9110ms
```

Ukuran file awal: 2784kb
Ukuran file akhir: 2681kb
Besar kompresi: 3%
Jumlah simpul: 1609297
Kedalaman pohon: 12

Gambar awal	Gambar akhir
	

3.1.3 Test Case 3

Input terminal

```
Masukan alamat gambar yang ingin dikompresi:
C:\Stima\chuu.jpeg
File ditemukan

Pilih metode perhitungan error
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukan nomor: 3
Anda memilih Max Pixel Difference
```

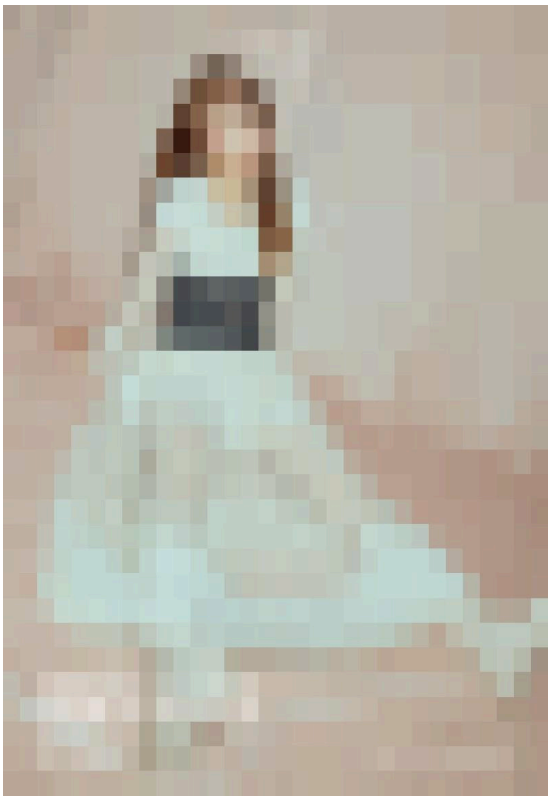
Tentukan ambang batas
Ambang batas berada pada rentang 0 - 50
Masukan ambang batas: 30

Masukan ukuran blok minimum: 400

Masukan alamat gambar hasil kompresi:
C:\Stima\chuu_compressed.jpeg

Output terminal

Gambar berhasil disimpan ke: C:\Stima\chuu_compressed.jpeg
Waktu eksekusi: 239ms
Ukuran file awal: 239kb
Ukuran file akhir: 50kb
Besar kompresi: 78%
Jumlah simpul: 1077
Kedalaman pohon: 5

Gambar awal	Gambar akhir
	

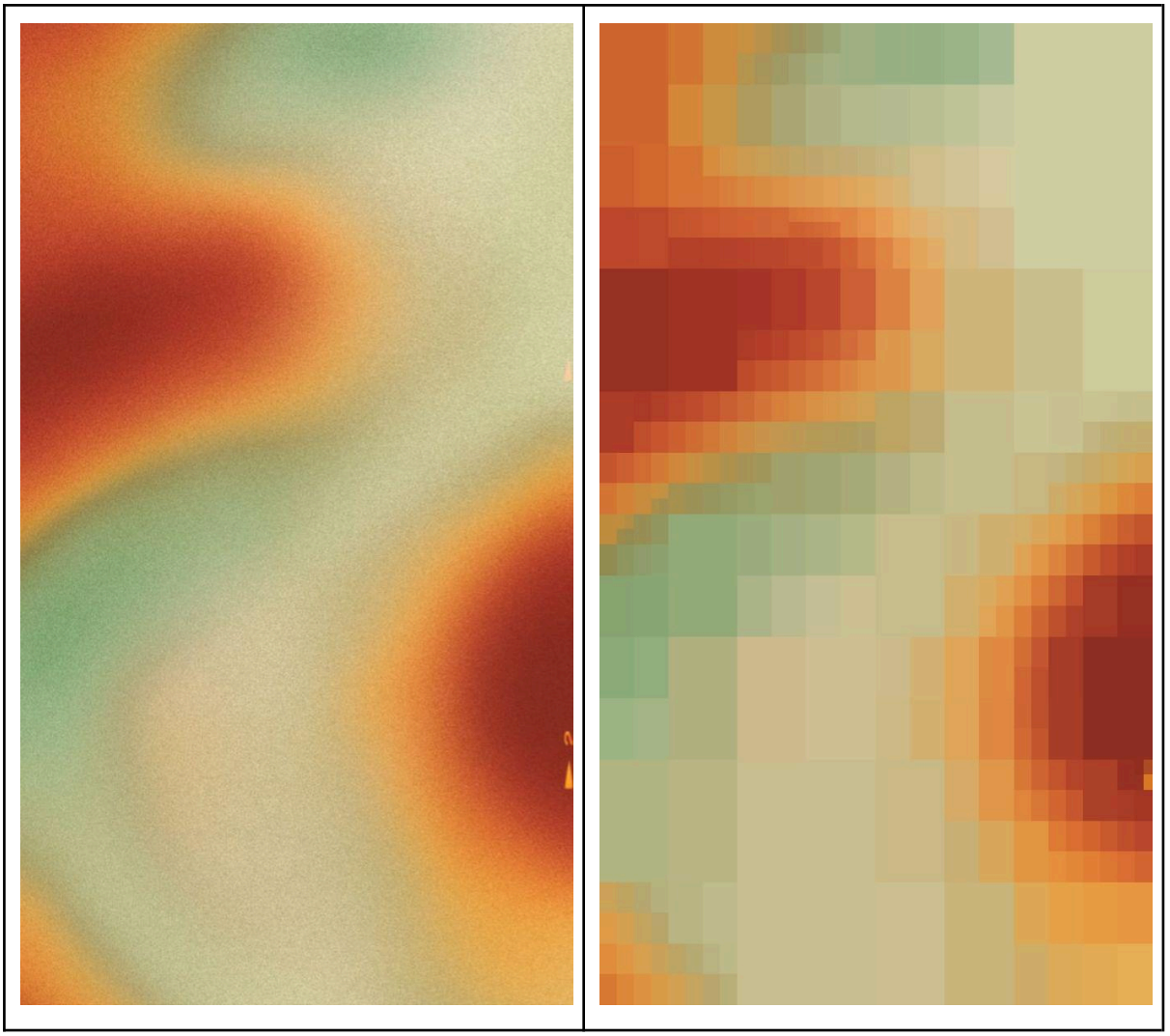
3.1.4 Test Case 4
Input terminal

```
Masukan alamat gambar yang ingin dikompresi:
C:\Stima\gradient.jpeg
File ditemukan
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukan nomor: 4
Anda memilih Entropy


Tentukan ambang batas
Ambang batas berada pada rentang 0 - 5
Masukan ambang batas: 3

Masukan ukuran blok minimum: 100

Masukan alamat gambar hasil kompresi:
C:\Stima\gradient_compressed.jpeg
```

Output terminal

```
Gambar berhasil disimpan ke: C:\Stima\gradient_compressed.jpeg
Waktu eksekusi: 38621ms
Ukuran file awal: 80kb
Ukuran file akhir: 51kb
Besar kompresi: 36%
Jumlah simpul: 5421
Kedalaman pohon: 6
```

Gambar awal	Gambar akhir



3.1.5 Test Case 5

Input terminal

```
C:\Stima\atta.png
File ditemukan

Pilih metode perhitungan error
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukan nomor: 1
Anda memilih Variance

Tentukan ambang batas
Ambang batas berada pada rentang 0 - 1000
Masukan ambang batas: 200

Masukan ukuran blok minimum: 16
```

```
Masukan alamat gambar hasil kompresi:
C:\Stima\atta_compressed.png
```

Output terminal

```
Gambar berhasil disimpan ke: C:\Stima\atta_compressed.png
Waktu eksekusi: 595ms
Ukuran file awal: 76kb
Ukuran file akhir: 63kb
Besar kompresi: 17%
Jumlah simpul: 5749
Kedalaman pohon: 8
```

Gambar awal	Gambar akhir
	

3.1.6 Test Case 6

Input terminal

```
Masukan alamat gambar yang ingin dikompresi:
C:\Stima\anya.jpg
File ditemukan

Pilih metode perhitungan error
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukan nomor: 2
```

```
Anda memilih Mean Absolute Deviation

Tentukan ambang batas
Ambang batas berada pada rentang 0 - 25
Masukan ambang batas: 25

Masukan ukuran blok minimum: 1

Masukan alamat gambar hasil kompresi:
C:\Stima\anya_compressed.jpg
```

Output terminal

```
Gambar berhasil disimpan ke: C:\Stima\anya_compressed.jpg
Waktu eksekusi: 279ms
Ukuran file awal: 81kb
Ukuran file akhir: 31kb
Besar kompresi: 61%
Jumlah simpul: 6917
Kedalaman pohon: 10
```

Gambar awal	Gambar akhir
	

3.1.7 Test Case 7

Input terminal

```
Masukan alamat gambar yang ingin dikompresi:
C:\Stima\sean.bmp
File ditemukan

Pilih metode perhitungan error
```

```
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukan nomor: 4
Anda memilih Entropy


Tentukan ambang batas
Ambang batas berada pada rentang 0 - 10
Masukan ambang batas: 5

Masukan ukuran blok minimum: 81

Masukan alamat gambar hasil kompresi:
C:\Stima\sean_compressed.bmp
```

Output terminal

```
Gambar berhasil disimpan ke: C:\Stima\sean_compressed.bmp
Waktu eksekusi: 11935ms
Ukuran file awal: 65kb
Ukuran file akhir: 56kb
Besar kompresi: 13%
Jumlah simpul: 2033
Kedalaman pohon: 6
```

Gambar awal	Gambar akhir
	

3.2 Analisis

3.2.1 Analisis kompleksitas algoritma

Fungsi quadtree akan mengecek terlebih dahulu apakah error pada blok kurang dari threshold atau tidak, jika error lebih besar quadtree akan membagi menjadi 4 blok yang lebih kecil. Karena error yang dicek satu per satu adalah masing-masing blok yang berjumlah n = lebar x tinggi, kompleksitas yang waktunya adalah O(n).

Rekursi dari setiap quadtree akan membagi 4 blok kembali hingga seterusnya dan menggabungkan kembali 4 blok tersebut sehingga didapatkan kompleksitas T(n) = 4 x T(n/4). Dengan menggabungkan waktu yang dibutuhkan dan rekursi, didapatkan bahwa T(n) = 4 x T(n/4) + n.

Menggunakan theorema master $T(n)= a \times T(n/b) + cn^d$ didapatkan bahwa a = 4, b = 4, c = 1, d = 1. Karena $a = b^d$ didapatkan bahwa T(n) adalah $O(n^d \times \log(n)) = O(n \log(n))$. Sehingga kompleksitas algoritma untuk quadtree adalah O(n log(n)).

3.2.2 Pengaruh Parameter Threshold dan Ukuran Minimum Blok

Semakin kecil nilai threshold yang ditentukan, maka proses pembagian blok akan terjadi lebih sering karena toleransi terhadap variasi warna atau intensitas menjadi semakin ketat. Hal ini menghasilkan kompresi gambar yang lebih detail, namun di sisi lain juga meningkatkan jumlah simpul dalam pohon Quadtree dan memperpanjang waktu eksekusi program. Sementara itu, ukuran blok minimum berfungsi untuk membatasi seberapa kecil pembagian blok dapat dilakukan. Jika ukuran minimum ini ditetapkan terlalu kecil, proses pembagian dapat berlanjut hingga blok-blok sangat kecil, yang menyebabkan waktu proses bertambah lama tanpa memberikan peningkatan kualitas kompresi yang berarti.

3.2.3 Perbandingan Metode Pengukuran Error

Variance sensitif terhadap pixel outlier, sehingga metode ini cocok digunakan pada gambar yang memiliki variasi warna besar atau perubahan warna yang cukup ekstrem.

Metode MAD tidak dapat mendeteksi outliner, sehingga lebih tepat digunakan pada gambar dengan penyimpangan warna yang kecil atau perubahan warna yang lebih halus.

Metode MPD mirip seperti metode MAD. Metode ini efektif untuk gambar yang memiliki banyak tepi atau kontur yang kuat.

Metode Entropy mengukur tingkat ketidakpastian distribusi warna dalam gambar sehingga cocok untuk gambar dengan distribusi warna yang kompleks, namun membutuhkan waktu komputasi yang paling lama dibandingkan metode lainnya.

Lampiran

Pranala Repository

https://github.com/drianto/Tucil2_13523010_13523145

Tabel Ketercapaian

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