

LAPORAN TUGAS KECIL 2

IF2211 Strategi Algoritma

Kompresi Gambar Dengan Metode Quadtree



Dipersiapkan oleh:

M. Rayhan Farrukh - 13523035

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Daftar Isi

Daftar Isi	1
Bab 1: Algoritma Divide and Conquer	2
1.1. Divide and Conquer	2
1.2. Penerapan Algoritma Divide and Conquer	2
Bab 2: Implementasi Program	3
2.1. Interface	4
2.2. Processing	8
2.3. Quadtree	10
2.4. Error	13
2.5. Utility	14
Bab 3: Implementasi Bonus	17
3.1. Target Persentase Kompresi	17
3.2. Output GIF	17
3.3. Structural Similarity Index Measure (SSIM)	19
Bab 4: Pengujian	19
4.1. Kasus Uji	19
4.2. Analisis Hasil Percobaan	21
Lampiran	22
Referensi	23

Bab 1: Algoritma Divide and Conquer

1.1. Divide and Conquer

Algoritma Divide and Conquer adalah algoritma dimana suatu permasalahan dipecah menjadi beberapa permasalahan yang lebih kecil yang lebih mudah diselesaikan, dimana solusi permasalahan kecil tersebut akan digabungkan sehingga membentuk solusi untuk permasalahan awal.

1.2. Penerapan Algoritma Divide and Conquer

Secara umum, algoritma yang diterapkan dalam program ini adalah membagi-bagi pixel-pixel gambar menjadi empat bagian sama besar, kemudian menggunakan Quadtree, terus membagi bagian-bagian ini hingga dicapai suatu kondisi berdasarkan nilai ambang batas serta ukuran minimum bagian gambar, yang kemudian disimpan sebagai data pada setiap node Quadtree. Setelah gambar dinilai tidak bisa dibagi lagi, maka bagian-bagian tersebut akan dikompresi dengan mengubah seluruh warna menjadi satu warna saja, yaitu warna rata-rata dari bagian tersebut. Untuk langkah-langkah lebih detail, adalah sebagai berikut.

- 1) Dimulai dengan bagian pertama, yaitu keseluruhan gambar.
- 2) Bagian ini dievaluasi melalui perhitungan error dengan metode tertentu, dan dibandingkan terhadap suatu nilai ambang batas yang dipilih. Ukuran dari bagian ini juga dibandingkan terhadap ukuran minimum yang dibolehkan.
- 3) Jika bagian ini memenuhi kriteria tertentu berdasarkan perbandingan diatas, maka gambar dinilai sudah dapat disimpan pada node Quadtree (*Conquered*), dan bagian ini diubah menjadi warna monoton, yaitu warna rata-rata dari bagian ini.
- 4) Jika tidak memenuhi kriteria, maka gambar akan dibagi menjadi empat bagian sama besar (*Divided*). Dimana langkah 1 hingga langkah ini (4) akan diulang secara rekursif untuk bagian-bagian yang lebih kecil tersebut.
- 5) Proses ini akan selesai ketika semua bagian telah ter-*conquered* dan dapat digabung menjadi satu gambar yang telah terkompresi, dimana Quadtree dianggap telah dibangun.
- 6) Setelah proses rekursi membangun Quadtree selesai, akan telusuri node-node pada Quadtree dan kemudian membangun gambar yang ingin disimpan dengan meletakkan setiap node daun pada quadtree pada posisi pixel gambar yang bersesuaian (*Combine*).

Pseudocode:

Bagian Divide and Conquer

```
FUNCTION BangunQuadtree(region, min_block_size , threshold, fungsi_error):  
    // Base case
```

```

IF error_bagian <= threshold ATAU ukuran(bagian) <= min_block_size MAKA
    tandai bagian sebagai node DAUN
    RETURN
ELSE
    tandai bagian sebagai node bukan DAUN

    // Divide
    sub_region_TL, sub_region_TR, sub_region_BL, sub_region_BR ←
    gJadiEmpatKuadran(region)

    // Panggil rekursif untuk setiap sub-region
    BangunQuadtree(sub_region_TL, gambar, min_block, threshold, fungsi_error)
    BangunQuadtree(sub_region_TR, gambar, min_block, threshold, fungsi_error)
    BangunQuadtree(sub_region_BL, gambar, min_block, threshold, fungsi_error)
    BangunQuadtree(sub_region_BR, gambar, min_block, threshold, fungsi_error)

```

Bagian Combine

```

ACTION BentukGambar(node, gambar):

IF node = Daun MAKA
    FOR setiap piksel (px, py) dalam node.region:
        target_gambar[px, py] <- node.warna_rata_rata
    RETURN

ELSE
    UNTUK i DARI 0 HINGGA 3:
        JIKA node.child[i] ADA MAKA
            GambarQuadtree(node.child[i], gambar)
        AKHIR JIKA
    AKHIR UNTUK
AKHIR IF

KHIR FUNCTION

```

Bab 2: Implementasi Program

Semua kode yang ditampilkan pada bab ini merupakan implementasi dari file yang bersangkutan dalam bahasa C++. Jika ingin melihat *header file* dari program silahkan merujuk ke [repository github](#). Selain itu, implementasi yang ditampilkan hanya berisi kode yang merupakan logika inti dari fungsi-fungsi yang ditampilkan, sehingga tidak merepresentasikan kode program secara utuh. Untuk implementasi bonus, kode akan ditampilkan pada [Bab 3: Implementasi Bonus](#).

2.1. Interface

File ini berisi *User Interface* (UI) sebagai perantara interaksi pengguna dan program.

```
#include "header/interface.hpp"

void interface(){
    int choice;
    while (true) {
        cls();
        cout << endl << endl;
        clr(welcome_banner, 196);
        wait_for_input();

        while(true){
            clr(" ----- ", 227);
            clr(" | Image Compression Using Quadtree | ", 227);
            clr(" ----- ", 227);
            clr(sleepy_cat, 230);
            cout << "Select an option: ";
            input(choice);
            if (choice == 1){main_program();}
            else if (choice == 2){exit(0);}
            else {
                input_error();
                continue;
            }
        }
    }
}

void main_program(){
```

```

// Input File Path
while (true){
    cls();
    cout << "Enter the absolute path of the image (make sure the file exists)" << endl;
    input(input_path);
    trim(input_path);
    if (input_path.find("/") == string::npos && input_path.find("\\")) == string::npos){
// Default input_path
        input_path = "io/input/" + input_path;
    }
    if (!filesystem::exists(input_path) || !regex_match(input_path, image_regex)){
        cout << endl;
        clr("File doesn't exist or doesn't have a valid file extension!", 196);
        input_error();
        continue;}
    break;
}

// Error Methods
while (true){
    cls();
    cout << "Pick the Error Measurement Methods" << endl;
    if (!input(mode) || (mode < 1 || mode > 5)){
        input_error();
        continue;}
    range = get_threshold_range(mode);
    break;
}

// Min Block Size
while (true){
    cls();
    cout << "Enter minimum block size" << endl << endl;
    if (!input(min_block)){
        input_error();
        continue;
    }
    break;
}

```

```

}

// Output File Path
while (true){
    cls();
    cout << "Enter the absolute path of where to save the image" << endl;
    input(output_path);
    trim(output_path);
    if (output_path.find("/") == string::npos && output_path.find("\") == string::npos){
        output_path = "io/output/" + output_path;
    }
    if (!regex_match(output_path, image_regex)){
        cout << endl;
        clr("Invalid path and/or filename!", 196);
        input_error();
        continue;
    }
    if (input_path == output_path){
        cout << endl;
        clr("Output path cannot be the same as Input path", 196);
        input_error();
        continue;
    }
    break;
}

// GIF
while (true){
    cls();
    cout << "Save compression to GIF? (y/n)" << endl << endl;
    cout << ">> ";
    input(gif);
    if (gif != "y" && gif != "Y" && gif != "n" && gif != "N"){
        cout << endl;
        clr("Enter y or n!", 196);
        input_error();
        continue;
    }
    break;
}

```

```

}

// GIF PATH
if (gif == "y" || gif == "Y"){
    while(true){
        cls();
        cout << "Enable dithering for GIF? (y/n)" << endl;
        input(gif);
        if (gif != "y" && gif != "Y" && gif != "n" && gif != "N"){
            cout << endl;
            clr("Enter y or n!", 196);
            input_error();
            continue;
        }
    }
}

while(true){
    cls();
    cout << "Enter the absolute path of where to save the GIF" << endl;
    input(gif_path);
    trim(gif_path);
    if (gif_path.find("/") == string::npos && gif_path.find("\\") == string::npos){
        gif_path = "io/output/" + gif_path;
    }
    if (!regex_match(gif_path, gif_regex)){
        input_error();
        continue;
    }
    gif_path = gif + gif_path;
    break;
}
}

cls();
compression(input_path, output_path, gif_path, mode, threshold, min_block);
wait_for_input();
}

void wait_for_input(){

```

```

cout << endl << "Press enter to continue...";
cin.sync();
cin.get();
cls();
}

void cls(){
    cout << "\033[2J\033[H";
}

void input_error(){

    cout << endl;
    clr(string(15, '='), 196);
    clr("Invalid input!", 196);
    clr(string(15, '='), 196);
    cout << endl;

    cin.clear();

    wait_for_input();
}

```

2.2. Processing

File ini berisi fungsi-fungsi yang digunakan untuk membaca file, menyimpan file, serta bertanggung jawab memanggil fungsi yang melakukan kompresi gambar. Pembacaan dan penyimpanan file gambar dilakukan dengan library stb_image dan stb_image_write.

```

#define STB_IMAGE_IMPLEMENTATION
#define STB_IMAGE_WRITE_IMPLEMENTATION

#include "header/processing.hpp"

Image read_image_file(string path){
    Image img;
    img.data = stbi_load(path.c_str(), &img.width, &img.height, &img.channels, 3);
    img.type = filesystem::path(path).extension().string();
    img.channels = 3;
    return img;
}

```

```

}

void save_image_file(Image img, string path){
    if (img.type == ".png"){
        stbi_write_png(path.c_str(), img.width, img.height, img.channels, img.data, img.width
* img.channels);
    }
    else if (img.type == ".jpg" || img.type == ".jpeg"){
        stbi_write_jpg(path.c_str(), img.width, img.height, img.channels, img.data, 85);
    }
}

void compression(string in_path, string out_path, string gif_path, int mode, double
threshold, int min_block){
    Image img = read_image_file(in_path);

    if (img.data == nullptr) {
        clr(string(15, '='), 196);
        clr("Failed to read image file!", 196);
        clr(string(15, '='), 196);
        return;
    }

    typedef double (*ErrorFunc)(const Image&, const Region&, const vector<uint8_t>&, int);
    function<double(const Image&, const Region&, const vector<uint8_t>&, int)> error_func;

    switch (mode){
        case 1:
            error_func = (ErrorFunc) variance;
            break;
        case 2:
            error_func = (ErrorFunc) mean_absolute_deviation;
            break;
        case 3:
            error_func = (ErrorFunc) max_pixel_difference;
            break;
        case 4:
            error_func = (ErrorFunc) entropy;
            break;
    }
}

```

```

    case 5:
        error_func = (ErrorFunc) ssim;
        break;
    default:
        error_func = nullptr;
        break;
}

Quadtree tree = Quadtree(0, 0, img.width, img.height);
tree.build(img, min_block, threshold, error_func);
tree.draw_to_depth(img, 0, max_depth);
try {
    create_file(out_path);
    save_image_file(img, out_path);

    if (gif_path != "null"){
        create_gif(gif_path, tree, img, max_depth);
    }
} catch (const exception &e){
    clr("Failed to save file!", 196);
}
}

```

2.3. Quadtree

File yang berisi implementasi dari kelas *Quadtree* dan logika inti dari algoritma *Divide and Conquer*.

```

#include "header/quadtree.hpp"

Quadtree::Quadtree(int x, int y, int width, int height) :
    region({x, y, width, height}), isLeaf(true){
    for (int i = 0; i < 4; i++) children[i] = nullptr;
    this->region.pixel_count = (long long)region.width * region.height;
}

Quadtree::~Quadtree(){
    for (int i = 0; i < 4; i++){
        delete children[i];
    }
}

```

```

}

int Quadtree::get_depth() {
    if (isLeaf) {
        return 0;
    } else {
        int max_child_depth = 0;
        for (int i = 0; i < 4; ++i) {
            max_child_depth = max(max_child_depth, children[i]->get_depth());
        }
        return 1 + max_child_depth;
    }
}

long long Quadtree::get_node_count() {
    long long count = 1; // Current node
    if (!isLeaf) {
        for (int i = 0; i < 4; ++i) {
            count += children[i]->get_node_count();
        }
    }
    return count;
}

vector<uint8_t> Quadtree::calculate_mean_color(const Image& img, const Region& region){
    vector<uint8_t> mean(img.channels, 0);
    vector<long long> pixel_sum(img.channels, 0);

    for (int y = region.y; y < region.y + region.height; ++y) {
        for (int x = region.x; x < region.x + region.width; ++x) {
            for (int c = 0; c < img.channels; ++c) {
                pixel_sum[c] += img.data[(y * img.width + x) * img.channels + c];
            }
        }
    }
    for (int c = 0; c < img.channels; ++c) mean[c] = (uint8_t)(pixel_sum[c] / (double)region.pixel_count);

    return mean;
}

```

```

}

void Quadtree::build(const Image& img, int min_block, double threshold, function<double(const Image&, const Region&, const vector<uint8_t>&, int)>error_func){
    this->mean_color = calculate_mean_color(img, region);
    double error = (error_func(img, region, mean_color, 0) + error_func(img, region, mean_color, 1) + error_func(img, region, mean_color, 2)) / 3.0;

    int half_width = region.width / 2;
    int half_height = region.height / 2;

    if ((img.mode == 5 && error <= threshold) || region.width <= min_block || region.height <= min_block || error <= threshold) {
        isLeaf = true;
        return;
    }

    isLeaf = false;

    children[0] = new Quadtree(region.x, region.y, half_width, half_height);
    children[1] = new Quadtree(region.x + half_width, region.y, half_width, half_height);
    children[2] = new Quadtree(region.x, region.y + half_height, half_width, half_height);
    children[3] = new Quadtree(region.x + half_width, region.y + half_height, half_width, half_height);

    for (int i = 0; i < 4; ++i) {
        children[i]->build(img, min_block, threshold, error_func);
    }
}

void Quadtree::draw_to_depth(Image& img, int curr_depth, int limit_depth){
    if (isLeaf || curr_depth == limit_depth) {
        for (int y = region.y; y < region.y + region.height; ++y) {
            uint8_t* row_ptr = &img.data[(y * img.width + region.x) * img.channels];
            for (int x = 0; x < region.width; ++x) {
                for (int c = 0; c < img.channels; ++c) {
                    row_ptr[x * img.channels + c] = mean_color[c];
                }
            }
        }
    }
}

```

```

    }
    return;
}

if (!isLeaf && curr_depth < limit_depth) {
    for (int i = 0; i < 4; ++i) {
        children[i]->draw_to_depth(img, curr_depth + 1, limit_depth);
    }
}
}

```

2.4. Error

File yang berisi implementasi fungsi-fungsi *error method*.

```

#include "header/error.hpp"

double variance(const Image& img, const Region& region, const vector<uint8_t>& mean, int ch_idx) {
    double var = 0;
    for (int y = region.y; y < region.y + region.height; ++y) {
        for (int x = region.x; x < region.x + region.width; ++x) {
            var += pow(img.data[(y * img.width + x) * img.channels + ch_idx] - mean[ch_idx], 2);
        }
    }
    return var / (double) region.pixel_count;
}

double mean_absolute_deviation(const Image& img, const Region& region, const vector<uint8_t>& mean, int ch_idx){
    double mad = 0;
    for (int y = region.y; y < region.y + region.height; ++y) {
        for (int x = region.x; x < region.x + region.width; ++x) {
            mad += abs(img.data[(y * img.width + x) * img.channels + ch_idx] - mean[ch_idx]);
        }
    }
    return mad / (double) region.pixel_count;
}

```

```

}

double max_pixel_difference(const Image& img, const Region& region, const vector<uint8_t>& mean, int ch_idx) {
    int max_vals = 0;
    int min_vals = 255;
    for (int y = region.y; y < region.y + region.height; ++y) {
        for (int x = region.x; x < region.x + region.width; ++x) {
            int val = img.data[(y * img.width + x) * img.channels + ch_idx];
            max_vals = max(max_vals, val);
            min_vals = min(min_vals, val);
        }
    }
    return max_vals - min_vals;
}

double entropy(const Image& img, const Region& region, const vector<uint8_t>& mean, int ch_idx){
    vector<int> counts(256, 0);
    double entropy;
    for (int y = region.y; y < region.y + region.height; ++y) {
        for (int x = region.x; x < region.x + region.width; ++x) {
            int pix_val = img.data[(y * img.width + x) * img.channels + ch_idx];

            counts[pix_val]++;
        }
    }
    return calculate_entropy(counts, region.pixel_count);;
}

double calculate_entropy(const vector<int>& counts, long long pixel_count){
    double probability;
    double channel_entropy = 0;
    for (int i = 0; i < 256; i++){
        if (counts[i] > 0){
            probability = counts[i] / (double) pixel_count;
            channel_entropy -= probability * log2(probability);
        }
    }
}

```

```
    return channel_entropy;
}
```

2.5. Utility

File ini berisi fungsi-fungsi tambahan yang digunakan untuk memudahkan penulisan program.

```
#include "header/utility.hpp"

void trim(string &str) {
    size_t first = str.find_first_not_of(" \t\n\r");
    if (first == string::npos) {
        str.clear();
        return;
    }
    size_t last = str.find_last_not_of(" \t\n\r");
    str = str.substr(first, last - first + 1);
}

void create_file(string path){
    filesystem::path p(path);
    filesystem::create_directories(p.parent_path());

    ofstream file(path);
    if (!file.is_open()) {
        cerr << "Error creating file: " << path << endl;
    } else {
        file.close();
    }
}

vector<int> get_threshold_range(int mode){
    switch (mode){
        case 1:
            return {0, 2250};
        case 2:
            return {0,50};
        case 3:
            return {0,250};
        case 4:
```

```

        return {0,8};
    case 5:
        return {-1,1};
    default:
        return {0,0};
    }
}

bool validate_threshold(vector<int> range, int threshold){
    return threshold >= range[0] && threshold <= range[1];
}

template<class T>
bool input(T &var) {
    string line;
    if (getline(cin, line)) {
        istringstream iss(line);
        if (iss >> var) {
            return true;
        }
    }
    cin.clear();
    return false;
}

template<class T>
void clr(const T& word, int code, bool enln = true) {
    cout << "\033[38;5;" << code << "m" << word << "\033[0m";
    if (enln) cout << endl;
}

```

Bab 3: Implementasi Bonus

3.1. Target Persentase Kompresi

Bonus ini merupakan fitur dimana pengguna dapat memasukkan target persentase kompresi dari gambar. Persentase kompresi adalah nilai yang menyatakan seberapa besar ukuran gambar berkurang jika dibandingkan dengan ukuran gambar original. Persentase kompresi dihitung menggunakan rumus berikut.

$$\text{Persentase Kompresi} = \left(1 - \frac{\text{Ukuran Gambar Terkompresi}}{\text{Ukuran Gambar Asli}}\right) \times 100\%$$

Dimana semakin besar persentase kompresi maka semakin kecil ukuran gambar hasil kompresi.

Untuk mencapai suatu target kompresi, maka harus menyesuaikan parameter yang digunakan untuk kompresi. Parameter yang disesuaikan adalah parameter *threshold* sedangkan *minimum block size* dijadikan 0 agar tidak mempengaruhi hasil kompresi.

Penyesuaian *threshold* yang digunakan sangat sederhana, yaitu menggunakan rentang nilai *threshold* yang valid (sesuai masing-masing *error method*), lalu mengubah rentang nilai tersebut menjadi skala 100 dengan mengalikan nilai target kompresi yang dimasukkan pengguna.

Implementasi:

```
cout << "Enter Target Compression Percentage" << endl;
cout << "Range: 0 - 1" << endl;
cout << "Enter 0 to disable this feature" << endl << endl;
cout << ">> ";
if(!input(target) || target < 0 || target > 1){
    input_error();
    continue;
}
if (target){
    threshold = (double)(range[1]-range[0]) * target;
    min_block = 0;
}
```

3.2. Output GIF

Bonus ini merupakan fitur yang memberikan pengguna pilihan untuk menyimpan file output GIF. File GIF ini menunjukkan proses pembentukan Quadtree untuk kompresi gambar. Jumlah frame pada GIF adalah jumlah dari *depth* Quadtree. Cara menyimpan frame adalah dengan “menggambar” image dari *depth* 0 hingga masing-masing *depth* sampai *maximum depth* tercapai, secara iteratif.

Untuk menyimpan frame-frame dan menggabungkan menjadi sebuah file GIF, digunakan *library* gif.h oleh [charlietangora](#). Pengguna juga dapat memilih untuk mengaktifkan *dithering* atau tidak. Dithering adalah fungsional yang diberikan oleh gif.h untuk fungsi menggambar GIF agar terlihat lebih halus.

Implementasi:

```
void create_gif(string path, Quadtree& tree, const Image& img, int max_depth){  
    int w = img.width;  
    int h = img.height;  
  
    try {  
        GifWriter writer;  
        uint32_t delay = 50;  
        bool dither = false;  
        if (path.rfind(";;;;;", 0) == 0) {  
            path.erase(0, 5);  
            dither = true;  
        }  
  
        if (!GifBegin(&writer, path.c_str(), w, h, delay, 8, dither)) {  
            __throw_runtime_error;  
        }  
  
        vector<uint8_t> rgb_buffer(static_cast<size_t>(w) * h * 3);  
        vector<uint8_t> rgba_buffer(static_cast<size_t>(w) * h * 4);  
  
        Image frame;  
        frame.width = w;  
        frame.height = h;  
        frame.channels = 3;  
        frame.data = rgb_buffer.data();  
  
        for (int d = 0; d <= max_depth; ++d) {  
            fill(rgb_buffer.begin(), rgb_buffer.end(), 0);  
            tree.draw_to_depth(frame, 0, d);  
  
            for (size_t i = 0, j = 0; i < rgb_buffer.size(); i += 3, j += 4) {  
                rgba_buffer[j + 0] = rgb_buffer[i + 0];  
                rgba_buffer[j + 1] = rgb_buffer[i + 1];  
            }  
        }  
    }  
}
```

```

        rgba_buffer[j + 2] = rgb_buffer[i + 2];
        rgba_buffer[j + 3] = 255;
    }

    if (!GifWriteFrame(&writer, rgba_buffer.data(), w, h, delay, 8, dither)) {
        break;
    }
}
GifEnd(&writer);
cout << endl;
cout << "GIF saved successfully to " << path << endl;
} catch (const exception& e){
    clr("Failed to save GIF!", 196);
}
}
}

```

3.3. Structural Similarity Index Measure (SSIM)

Structural Similarity Index Measure adalah teknik perhitungan *error* yang mengukur kemiripan visual antara dua gambar dengan membandingkan luminans (kecerahan), kontras, dan informasi struktural keduanya. Tujuannya adalah meniru persepsi visual manusia agar kompresi terlihat lebih natural. Nilai SSIM dihitung menggunakan rumus berikut.

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

$$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$$

Dengan:

- x : blok piksel gambar pertama
- y : blok piksel gambar kedua
- μ : nilai rata-rata dari intensitas atau warna piksel pada blok
- C : nilai konstanta stabilisasi SSIM
- c : nilai pada satu channel (RGB)
- w : bobot SSIM dari masing-masing channel RGB (dapat menggunakan luminance weighting agar warna tampak natural di persepsi manusia)

Untuk SSIM yang digunakan pada program ini, gambar yang dibandingkan adalah gambar original dan gambar hasil kompresi, dengan melakukan perbandingan ini per blok, atau per region pada Quadtree. Karena blok yang dibandingkan adalah blok original dan blok kompresi, nilai rata-rata kedua blok tersebut akan sama, karena kompresi adalah pergantian semua nilai piksel pada blok dengan nilai rata-rata blok. Karena nilai rata-rata kedua blok sama, maka ada perubahan pada perhitungan nilai SSIM. Nilai piksel pada blok terkompresi sama dengan nilai rata-rata sehingga variansi dan kovariansi menjadi 0. Sehingga formula SSIM dapat disederhanakan seperti berikut.

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

Implementasi:

```
double ssim(const Image& img, const Region& region, const vector<uint8_t>& mean, int ch_idx){
    const double C1 = pow(0.01 * 255, 2);
    const double C2 = pow(0.03 * 255, 2);

    double ssim_sum = 0;
    vector<double> weights = {0.2989, 0.5870, 0.1140}; // Luminance Weighting
    double var_x = variance(img, region, mean, ch_idx);
    return 1 - (weights[ch_idx] * (C2 / (var_x + C2)));
}
```

Hasil yang dikembalikan adalah $1 - \text{SSIM}$ karena pada algoritma Quadtree, digunakan cek error $\leq \text{threshold}$, namun seharusnya pada SSIM cek ini dibalik menjadi error $\geq \text{SSIM}$. Dengan menggunakan $1 - \text{SSIM}$, kita dapat membiarkan algoritma utama dan hanya memodifikasi fungsi SSIM saja.

Hasil dari implementasi kurang memuaskan karena hanya nilai threshold 0.8 hingga 1 (range seharusnya 0 - 1) saja yang membuat hasil yang bermakna sedangkan nilai selain itu akan menghasilkan gambar yang sama dengan gambar original. Untuk menangani masalah ini, khusus untuk SSIM, nilai threshold disesuaikan sehingga input 0 - 1 selalu menghasilkan nilai 0 atau nilai pada rentang 0.8 - 1.

```
if (threshold) threshold = 0.8 + (0.2 * threshold);
```

Bab 4: Pengujian

4.1. Kasus Uji

Image yang digunakan untuk pengujian



1. Kasus Uji 1

a. Parameter

Method: Variansi

Target Kompresi: 0

Threshold: 500

Min. Block Size: 10

b. Output Terminal

```
Compressed image saved successfully to test/var_0_500_10.png
Execution time: 1448.205680 ms

Parameters:
Error Method: 1
Threshold: 500.000000
Minimum Block Size: 10

Original Size: 1839700 Bytes
Compressed Size: 296282 Bytes
Compression Percentage: 83.895092%
Depth of Tree: 7
Amount of Nodes: 8285

Press enter to continue...
```

c. File Output



2. Kasus Uji 2

a. Parameter

Method: MAD

Target Kompresi: 0

Threshold: 30

Min. Block Size: 5

b. Output Terminal

```
Compressed image saved successfully to io/output/mad_0_30_5.png
Execution time: 314.281707 ms
```

Parameters:

Error Method: 2

Threshold: 30.000000

Minimum Block Size: 5

Original Size: 1839700 Bytes

Compressed Size: 160338 Bytes

Compression Percentage: 91.284557%

Depth of Tree: 8

Amount of Nodes: 3973

c. File Output



3. Kasus Uji 3

a. Parameter

Method: MPD

Target Kompresi: 0

Threshold: 182

Min. Block Size: 0

b. Output Terminal

```
Compressed image saved successfully to test/mpd_0.73_0_0.png
Execution time: 400.976439 ms
```

Parameters:

Error Method: 3

Threshold: 182.500000

Minimum Block Size: 0

Original Size: 1839700 Bytes

Compressed Size: 150609 Bytes

Compression Percentage: 91.813393%

Depth of Tree: 10

Amount of Nodes: 8133

c. File Output



4. Kasus Uji 4

a. Parameter

Method: Entropi
Target Kompresi: 0
Threshold: 6
Min. Block Size: 15

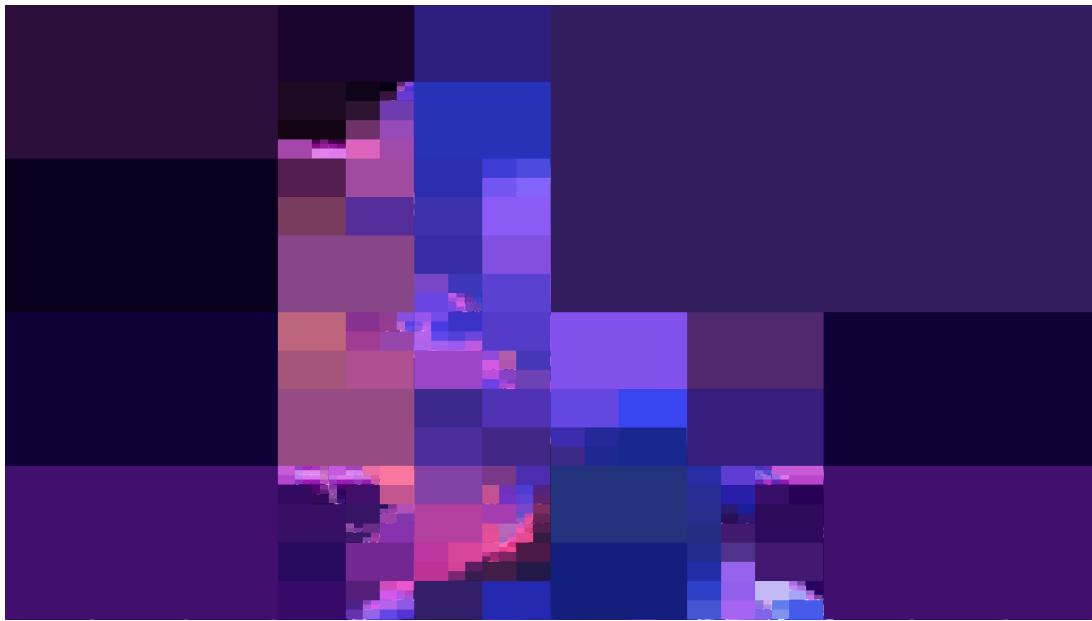
b. Output Terminal

```
Compressed image saved successfully to test/ent_0_6_15.png
Execution time: 253.912799 ms
```

```
Parameters:
Error Method: 4
Threshold: 6.000000
Minimum Block Size: 15

Original Size: 1839700 Bytes
Compressed Size: 88620 Bytes
Compression Percentage: 95.182910%
Depth of Tree: 7
Amount of Nodes: 457
```

c. File Output



5. Kasus Uji 5

a. Parameter

Method: SSIM

Target Kompresi: 0

Threshold: 0.85

Min. Block Size: 5

b. Output Terminal

```
Compressed image saved successfully to test/ssim_0_0-85_5.png
Execution time: 1679.063841 ms

Parameters:
Error Method: 5
Threshold: 0.970000
Minimum Block Size: 5

Original Size: 1839700 Bytes
Compressed Size: 561207 Bytes
Compression Percentage: 69.494646%
Depth of Tree: 8
Amount of Nodes: 32941
```

NOTE: seperti pada [penjelasan SSIM](#), nilai threshold yang diinput disesuaikan agar bisa mendapatkan hasil yang bagus.

c. File Output



6. Kasus Uji 6

a. Parameter

Method: Variansi

Target Kompresi: 0.66

Threshold: -

Min. Block Size: -

b. Output Terminal

```
Compressed image saved successfully to io/output/lolol.png
Execution time: 1127.464923 ms
```

Parameters:

Error Method: 1

Threshold: 1485.000000

Minimum Block Size: 0

Original Size: 1839700 Bytes

Compressed Size: 211500 Bytes

Compression Percentage: 88.503560%

Depth of Tree: 10

Amount of Nodes: 11157

c. File Output



7. Kasus Uji 7 (GIF)

a. Parameter

Method: Variansi

Target Kompresi: 0

Threshold: 0

Min. Block Size: 0

Dithering: No

b. Output Terminal

```
Compressed image saved successfully to test/gif_no_dither.png
Execution time: 3060.072461 ms
```

```
GIF saved successfully to test/no_dither.gif
Execution time: 11214.155375 ms
```

Parameters:

Error Method: 1

Threshold: 0.000000

Minimum Block Size: 0

Original Size: 1839700 Bytes

Compressed Size: 2115707 Bytes

Compression Percentage: -15.002827%

Depth of Tree: 10

Amount of Nodes: 1080305

c. File Output

Image:



GIF: [no_dither.gif](#)

8. Kasus Uji 8 (GIF)

a. Parameter

Method: Variansi

Target Kompresi: 0

Threshold: 0

Min. Block Size: 0

Dithering: Yes

b. Output Terminal

```
Compressed image saved successfully to test/gif_no_dither.png
Execution time: 3060.072461 ms

GIF saved successfully to test/no_dither.gif
Execution time: 11214.155375 ms

Parameters:
Error Method: 1
Threshold: 0.000000
Minimum Block Size: 0

Original Size: 1839700 Bytes
Compressed Size: 2115707 Bytes
Compression Percentage: -15.002827%
Depth of Tree: 10
Amount of Nodes: 1080305
```

c. File Output

Image:



GIF: [dither.gif](#)

9. Analisis Hasil Percobaan

Kompleksitas Waktu

Build:

Fungsi *build* adalah fungsi untuk membangun pohon Quadtree, kompleksitas waktu tergantung pada jumlah operasi pemrosesan piksel dan rekursi yang bergantung kepada nilai *error* dan *threshold* serta *minimum block size*. Operasi pada satu blok (menghitung warna rata-rata) dengan jumlah piksel n membutuhkan sejumlah cn operasi.

Untuk *worst case scenario*, yaitu ketika subdivisi blok mencapai titik terdalam hingga satu blok mewakili satu pixel, jumlah operasi menjadi $4T(n/4)$ operasi karena divide and conquer dilakukan sebanyak 4 kali dengan membagi ukuran data menjadi 4 untuk masing-masing rekursi. Sehingga total operasi pada fungsi build adalah $4T(n/4) + cn$.

Draw:

Fungsi *draw* adalah fungsi untuk merekonstruksi gambar berdasarkan struktur pohon Quadtree yang sudah dibangun. Fungsi ini bekerja secara rekursif mengunjungi setiap node (blok) pada pohon. Total operasi untuk pengunjungan node, pada *worst case*, yaitu ketika jumlah node sama dengan jumlah piksel, sehingga ada cn operasi.

Jika node yang dikunjungi adalah *leaf*, fungsi akan memproses semua piksel pada node tersebut dan meletakkan piksel warna rata-rata pada posisi yang bersesuaian di gambar yang direkonstruksi. Jumlah operasi untuk satu *leaf* sebanding dengan jumlah pikselnya, yaitu cn operasi. Jika node bukan *leaf*, fungsi draw akan melakukan 4 rekursi untuk anak-anaknya.

Karena dua proses tersebut, total operasi draw menjadi $2cn$.

Total:

Berdasarkan dua fungsi tersebut, total operasi yang dibutuhkan adalah $4T(n/4) + 3cn$. Dengan teorema master ($a = 4$, $b = 4$, $d = 1$), didapat $a = b^d$, sehingga kompleksitas algoritma proses kompresi menggunakan quadtree adalah $O(n \log n)$.

Kompleksitas Ruang

Pada program, penyimpanan yang dibutuhkan adalah untuk Quadtree, serta untuk menyimpan data gambar ketika konstruksi. Untuk gambar dengan jumlah piksel n , data gambar membutuhkan $O(n)$, sedangkan untuk Quadtree, pada kasus terburuk, dimana jumlah node sama dengan piksel, juga membutuhkan $O(n)$, sehingga pada kasus terburuk, kompleksitas ruang menjadi $O(2n) = O(n)$.

Lampiran

Repository: https://github.com/grwna/Tucil2_13523035

Tabel Checklist:

No	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	✓	

Referensi

- [1]. R. Munir, "Algoritma Divide and Conquer, Bagian 1" Institut Teknologi Bandung, Bandung, Indonesia, 2024-2025. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)
- [2]. R. Munir, "Algoritma Divide and Conquer, Bagian 2" Institut Teknologi Bandung, Bandung, Indonesia, 2024-2025. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-(2025)-Bagian2.pdf)
- [3]. R. Munir, "Algoritma Divide and Conquer, Bagian 3" Institut Teknologi Bandung, Bandung, Indonesia, 2024-2025. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/09-Algoritma-Divide-and-Conquer-\(2025\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/09-Algoritma-Divide-and-Conquer-(2025)-Bagian3.pdf)
- [4]. R. Munir, "Algoritma Divide and Conquer, Bagian 4" Institut Teknologi Bandung, Bandung, Indonesia, 2024-2025. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/10-Algoritma-Divide-and-Conquer-\(2025\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/10-Algoritma-Divide-and-Conquer-(2025)-Bagian4.pdf)
- [5]. T. W. York, "Quadtrees for Image Processing," Medium, [Online]. Available: <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>