Tugas Kecil 2 IF2211 Strategi Algoritma

Semester II tahun 2024/2025

Kompresi Gambar dengan Metode Quadtree

Muhammad Ra’if Alkautsar 13523011

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

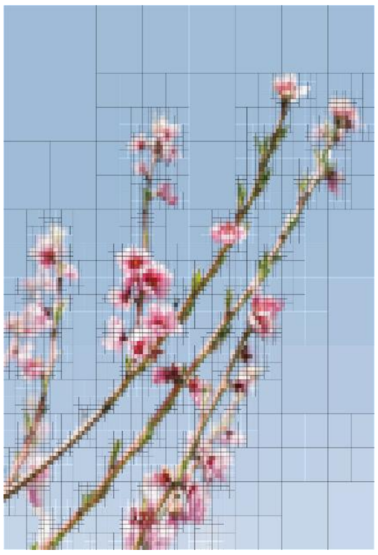
INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

BAB I
DESKRIPSI PERMASALAHAN

A. Deskripsi

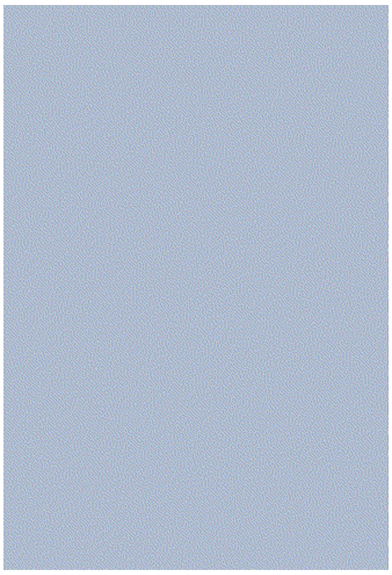


Gambar 1 Quadtree dalam Kompresi Gambar

(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.



Gambar 2. Proses Pembentukan Quadtree dalam Kompresi Gambar

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

BAB II
ALGORITMA DIVIDE AND CONQUER

Algoritma Utama

Algoritma yang digunakan dalam program ini menerapkan prinsip Divide and Conquer, menggunakan struktur pohon Quadtree untuk mengompresi gambar dengan cara membagi area gambar menjadi blok-blok yang semakin kecil, hanya jika blok tersebut memiliki tingkat keseragaman (homogenitas) yang rendah.

Blok-blok yang cukup homogen atau seragam tidak dibagi lebih lanjut dan disimpan sebagai leaf node (simpul daun), mewakili warna rata-rata dari blok tersebut. Ada tiga parameter yang mempengaruhi kompresi, yaitu *threshold*, *minBlockSize*, dan *errorChoice*. Langkah-langkah algoritma utama dari program adalah sebagai berikut:

1. Divide (Pembagian Blok)
   Gambar awal diwakili oleh simpul akar (root) yang mencakup seluruh area gambar. Untuk setiap simpul (blok) pada pohon, akan dihitung nilai error dari blok (contohnya variansi, *mean absolute difference* (MAD), *max pixel difference*, *entropy*, atau SSIM). Jika error lebih dari threshold dan ukuran blok masih lebih besar dari *minimum block size*, maka blok dibagi menjadi empat sub-blok kuadran: kiri atas (top-left), kanan atas (top-right), kiri bawah (bottom-left), dan kanan bawah (bottom right).
2. Conquer (Rekursi)
   Untuk setiap sub-blok, proses divide dipanggil secara rekursif. Ini terus dilakukan hingga nilai error dari blok sudah lebih kecil dari threshold dan ukuran blok mencapai ukuran minimum.
3. Combine atau Build
   Setelah semua simpul dan mewakili sebuah area gambar dan diisi warna rata-rata blok tersebut, seluruh gambar dibangun kembali dari simpul-simpul daun ini untuk menghasilkan gambar terkompresi.

Analisis Kompleksitas Waktu

Untuk sebuah blok berukuran $W \times H$, aksi yang dibutuhkan untuk menghitung error proporsional dengan jumlah pixel dalam blok tersebut, yaitu $\Theta(W \times H)$. Dalam skenario terburuk, algoritma membagi blok menjadi empat kuadran dengan dimensi $W/2 \times H/2$. Oleh karena itu, jika $T(W, H)$ adalah waktu untuk memproses sebuah blok selebar W dan setinggi H, maka rekurensinya menjadi:

$$T(W,H) = \begin{cases} \Theta(W \cdot H) & \text{if } E(W,H) \le \theta \text{ or } W \le m \text{ or } H \le m, \\ 4T\left(\frac{W}{2}, \frac{H}{2}\right) + \Theta(W \cdot H) & \text{if } E(W,H) > \theta \text{ and } W > m \text{ and } H > m. \end{cases}$$

Apabila kita menganalisis kasus rekursifnya, kita bisa mereduksinya menjadi persamaan berikut supaya dapat dianalisis.

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Apabila rekurensi dianalisis menggunakan Teorema Master, maka a = 4, b = 2, dan f(n) = $\Theta(n^2)$. Bentuk tersebut memenuhi kasus kedua dari Teorema Master, yaitu apabila terdapat sebuah bentuk di mana

$$a \ge 1, b > 1, and\ f(n) = \Theta(n^{\log_b a})$$

Maka algoritma memiliki kompleksitas waktu yaitu

$$T(n) = \Theta(n^{\log_b a} \log n)$$

Dari situ, bisa dihitung

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

Jika disubstitusikan n^2 dengan W × H, diperoleh worst-case running time untuk algoritma yaitu

$$\Theta\Big(W \cdot H \cdot \log\big(\min(W,H)\big)\Big)$$

Struktur, Kelas, dan Metode Penghitung Error

Dalam implementasinya, program mengimplementasikan sejumlah struktur dan kelas.

- Struktur *Color* pada program ini berisi empat nilai integer r, g, b, a yang memiliki rentang 0 sampai 255.
- Kelas *QuadTreeNode* mewakili satu simpul (node) dalam pohon Quadtree dan memiliki properti *x, y* (koordinat pojok kiri atas dari blok yang diwakili), *width, height* (ukuran blok), is_leaf (menunjukkan apakah node ini adalah leaf tidak dibagi lagi), color (warna rata-rata blok jika leaf), top_left_child, top_right_child, bottom_left_child dan bottom_right_child (empat anak node jika blok dibagi).

- Kelas QuadTree mewakili satu struktur Quadtree untuk seluruh gambar dan memiliki properti root (akar pohon Quadtree), source_image_matrix (gambar asli), dan result_image_matrix (gambar hasil kompresi).

Lima metode penghitung error untuk mengukur homogenitas suatu blok yaitu:

1. Variansi

$$\mathrm{Var} = \frac{1}{\mathrm{N}} \sum (x_i - \mu)^2$$

2. Mean Absolute Deviation

$$MAD = \frac{1}{N} \sum |x_i - \mu|$$

3. Max Pixel Difference

$$\mathrm{MaxDiff} = \max(x_i) - \min(x_i)$$

4. Entropy

$$H = -\sum p_i \log_2 p_i$$

Pembuatan GIF

Metode penyimpanan GIF pada program ini digunakan untuk memvisualisasikan proses kompresi gambar menggunakan Quadtree secara bertahap. Setiap frame dalam GIF menunjukkan hasil kompresi hingga kedalaman tertentu dari pohon Quadtree. Proses diawali dengan membuat gambar hasil untuk tiap level kedalaman, mulai dari 0 hingga maksimum, lalu menyimpannya sebagai frame animasi.

Gambar-gambar tersebut dikonversi ke format RGBA, lalu disimpan ke file GIF menggunakan pustaka `gif.h`. Fungsi `GifBegin` digunakan untuk memulai file GIF, `GifWriteFrame` menambahkan setiap frame, dan `GifEnd` menutup file. Delay antar frame bisa diatur agar animasi terlihat lebih halus atau lambat. Hasil akhirnya adalah file GIF yang memperlihatkan bagaimana blok-blok gambar dibagi secara bertahap, dari bentuk kasar hingga gambar akhir yang lebih detail. Metode ini cocok untuk memperlihatkan cara kerja kompresi Quadtree secara visual.

Implementasi SSIM

Implementasi SSIM (Structural Similarity Index) dalam program ini digunakan sebagai salah satu metode untuk mengukur homogenitas blok gambar dalam proses kompresi. Alih-alih membandingkan dua gambar berbeda, SSIM di sini disederhanakan untuk membandingkan sebuah blok terhadap versi rata-ratanya sendiri. Artinya, blok dianggap akan dikompresi menjadi satu warna rata-rata, dan program mengevaluasi seberapa besar perbedaan struktural yang hilang akibat penyederhanaan tersebut.

Perhitungannya dilakukan dengan cara mencari rata-rata dan variansi dari setiap kanal warna (R, G, B) dalam blok. Karena blok yang dikompresi dianggap seragam, maka nilai rata-rata blok yang dikompresi akan sama dengan nilai rata-rata blok aslinya, dan variansinya diasumsikan nol. Berdasarkan asumsi ini, rumus SSIM disederhanakan menjadi $SSIM = \frac{C_2}{\sigma^2 + C_2}$, di mana σ^2 adalah variansi kanal warna dan C_2 adalah konstanta kecil untuk menjaga kestabilan. Nilai SSIM dihitung untuk setiap kanal warna, lalu dirata-ratakan, dan nilai akhir yang digunakan sebagai error adalah $1-$SSIM. Semakin kecil error ini, semakin mirip blok dengan versi rata-ratanya, dan semakin besar kemungkinan blok tersebut tidak perlu dibagi lebih lanjut.

Fitur Target Kompresi

Fitur target kompresi dalam program ini memungkinkan pengguna menentukan persentase kompresi yang ingin dicapai, tanpa perlu secara manual menetapkan nilai threshold. Saat fitur ini diaktifkan, program akan mencari nilai threshold yang paling mendekati target kompresi menggunakan metode binary search. Proses ini dilakukan dengan mengukur berapa persen piksel dari gambar asli yang dapat direpresentasikan sebagai leaf node dalam Quadtree, lalu membandingkannya dengan target kompresi yang ditentukan. Jika kompresi terlalu rendah, threshold akan dinaikkan agar lebih banyak blok dianggap homogen dan tidak dibagi; sebaliknya, jika kompresi terlalu tinggi, threshold diturunkan agar pembagian blok lebih ketat. Proses ini diulang sampai perbedaan antara hasil kompresi dan target berada dalam batas toleransi, atau sampai jumlah iterasi maksimum tercapai. Pendekatan ini membuat proses kompresi lebih fleksibel dan mudah digunakan, karena pengguna cukup menentukan seberapa besar kompresi yang diinginkan tanpa memahami detail teknis seperti nilai error atau threshold.

BAB III
KODE SUMBER

A. Main

```cpp
#include <iostream>
#include "quadtree.hpp"
#include "utils.hpp"
#include <chrono>
#include <filesystem>

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"

int main() {
    std::string inputFile, outputFile, gifOutput;
    bool to_gif = true;
    int errorChoice;
    double threshold;
    int minBlockSize;
    double targetCompression;

    std::cout << "[1/7] Enter absolute input image path: ";
    std::getline(std::cin, inputFile);
    std::cout << "[2/7] Enter absolute output image path: ";
    std::getline(std::cin, outputFile);
    std::cout << "Available Error Choices" << std::endl;
    std::cout << "1. Variance" << std::endl;
    std::cout << "2. Mean Absolute Deviance" << std::endl;
    std::cout << "3. Max Pixel Difference" << std::endl;
    std::cout << "4. Entropy" << std::endl;
    std::cout << "5. Structural Similarity Index (SSIM)" << std::endl;
    std::cout << "[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: 
Entropy, 5: SSIM): ";
    std::cin >> errorChoice;
    std::cout << "[4/7] Enter target compression (floating point, 1.0 = no 
compression, 0 to disable): ";
    std::cin >> targetCompression;
    if (targetCompression == 0) {
        std::cout << "[5/7] Enter threshold: ";
        std::cin >> threshold;
        std::cout << "[6/7] Enter minimum block size: ";
        std::cin >> minBlockSize;
    }
    std::cin.ignore();
    std::cout << "[7/7] Enter absolute GIF output path (blank to not save GIF): ";
    std::getline(std::cin, gifOutput);
    if (gifOutput.empty()) {
        to_gif = false;
    }

    auto startTime = std::chrono::high_resolution_clock::now();

    Image input_image = load_image_rgba(inputFile);

    bool smallfile = false;
    double originalFileSizeKB = std::filesystem::exists(inputFile) ? 
std::filesystem::file_size(inputFile) / 1024.0 : 0;

    if (originalFileSizeKB < 100) {
        smallfile = true;
    }

    if (input_image.empty()) {
        std::cerr << "Error: Could not load input image from " << inputFile << "\n";
        return -1;
    }
    std::string srcFormat = get_extension(inputFile);
    if (srcFormat.empty()) {
        std::cerr << "Warning: Could not determine source image format. Defaulting to PNG.\n";
        srcFormat = "png";
    }

    // Membuat sebuah pointer QuadTree dan mengisinya
    QuadTree* qt = new QuadTree(input_image);

    // Jika mode target compression menyala, sesuaikan threshold secara iteratif
    if (targetCompression > 0) {
        qt->adjust_threshold_for_target(errorChoice, targetCompression);
    } else {
        qt->build_quad_tree(errorChoice, threshold, minBlockSize);
    }

    // Build the final image.
    if (to_gif) {
        qt->build_image_gradual();
        qt->save_gif(gifOutput, 50);
    } else {
        qt->build_image_fast();
    }

    qt->save_image_rgba_with_format(outputFile, srcFormat, smallfile);


    auto endTime = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed = endTime - startTime;

    int origHeight = input_image.size();
    int origWidth = input_image[0].size();
    double compressedFileSizeKB = std::filesystem::exists(outputFile) ? std::filesystem::file_size(outputFile) / 1024.0 : 0;
    int originalPixels = origHeight * origWidth;
    int leafCount = qt->get_leaf_count();
    double compressionPercentage = (1.0 - compressedFileSizeKB / originalFileSizeKB) * 100.0;
    int treeDepth = qt->get_tree_depth();
    int totalNodes = qt->get_total_node_count();

    std::cout << "[Post-Compression Information]" << std::endl;
    std::cout << "Execution Time: " << elapsed.count() << " seconds" << std::endl;
    std::cout << "Original File Size: "
              << originalFileSizeKB << " KB" << std::endl;
    std::cout << "Compressed File Size: "
              << compressedFileSizeKB << " KB" << std::endl;
    std::cout << "Compression Percentage: " << compressionPercentage << "%" << std::endl;
    std::cout << "Tree Depth: " << treeDepth << std::endl;
    std::cout << "Total Number of Tree Nodes: " << totalNodes << std::endl;

    delete qt; // Bersih-bersih.
    return 0;
}

```

B. QuadTree

Header

```cpp
// Header
#ifndef QUADTREE_HPP
#define QUADTREE_HPP

#include "quadtreenode.hpp"
#include "utils.hpp"
#include <string>

class QuadTree {
    private:
        QuadTreeNode* root;
        Image source_image_matrix;
        Image result_image_matrix;
    public:
    /* Constructor QuadTree dengan referensi ke input image */
        QuadTree(const Image& input_image);
    /* Destructor QuadTree */
        ~QuadTree();
    /* Membangun QuadTree berdasarkan parameter dan gambar yang ada */
        void build_quad_tree(int error_choice, double threshold, int min_block_size);
    /* Mode cepat pembuatan gambar dari QuadTree (untuk opsi tanpa GIF) */
        void build_image_fast();
    /* Mode gradual pembuatan gambar dari QuadTree (untuk opsi dengan GIF) */
        void build_image_gradual();
    /* Menyimpan gambar RGBA menjadi file */
        void save_image_rgba_with_format(const std::string& filename, const
std::string& srcFormat, bool smallfile);
    /* Menyimpan GIF */
        void save_gif(const std::string& gifOutputPath, int delay);
    /* Memperoleh kedalaman pohon */
        int get_tree_depth() const;
    /* Memperoleh jumlah simpul dalam pohon */
        int get_total_node_count() const;
    /* Metode untuk mode target compression */
        void adjust_threshold_for_target(int errorChoice,
            double targetCompression, double tolerance = 2.0,
            int maxIterations = 30);
    /* Memperoleh jumlah daun dalam pohon*/
        int get_leaf_count() const;
};

#endif

```

Implementation

```cpp
#include "quadtree.hpp"
#include "gif.h"
#include "stb_image.h"
#include "stb_image_write.h"
#include <iostream>
#include <vector>

/* Constructor QuadTree dengan referensi ke input image */
QuadTree::QuadTree(const Image& input_image) {
    root = new QuadTreeNode(0, 0, input_image[0].size(), input_image.size());
    source_image_matrix = std::move(input_image);
    int height = source_image_matrix.size();
    int width = source_image_matrix[0].size();
    result_image_matrix = Image(height, std::vector<Color>(width, Color{0, 0, 0,
255}));
}
```

```cpp

/* Destructor QuadTree */
QuadTree::~QuadTree() {
    delete root;
}

/* Membangun QuadTree berdasarkan parameter dan gambar yang ada */
void QuadTree::build_quad_tree(int error_choice, double threshold, int min_block_size) {
    if (root) {
        root->divide_node(source_image_matrix, error_choice, threshold, min_block_size);
    }
}

/* Mode cepat pembuatan gambar dari QuadTree (untuk opsi tanpa GIF) */
void QuadTree::build_image_fast() {
    if (root) {
        root->draw_node_fast(result_image_matrix);
    }
}

/* Mode gradual pembuatan gambar dari QuadTree (untuk opsi dengan GIF) */
void QuadTree::build_image_gradual() {
    gifFrames.clear();
    int height = result_image_matrix.size();
    int width  = result_image_matrix[0].size();

    int actualMaxDepth = (root) ? root->max_depth() : 0; // Mendapatkan max depth

    Image lastFrame;
    // Loop dari 0 ke kedalaman maksimum.
    for (int d = 0; d <= actualMaxDepth; d++) {
        Image temp(height, std::vector<Color>(width, Color{0, 0, 0, 255}));
        if (root) {
            root->draw_node_at_depth(temp, 0, d);
        }
        gifFrames.push_back(temp);
        lastFrame = temp; // Penyetoran frame paling recent
    }
    result_image_matrix = lastFrame; // Update gambar terakhir
}

/* Menyimpan gambar RGBA menjadi file */
void QuadTree::save_image_rgba_with_format(const std::string& filename, const std::string& srcFormat, bool smallfile) {
    if (result_image_matrix.empty() || result_image_matrix[0].empty()) {
        std::cerr << "Empty image, cannot save.\n";
        return;
    }
    int height = result_image_matrix.size(), width = result_image_matrix[0].size();
    std::vector<unsigned char> buffer(width * height * 4);
    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            int idx = (y * width + x) * 4;
            const Color& c = result_image_matrix[y][x];
            buffer[idx + 0] = static_cast<unsigned char>(c.r);
            buffer[idx + 1] = static_cast<unsigned char>(c.g);
            buffer[idx + 2] = static_cast<unsigned char>(c.b);
            buffer[idx + 3] = static_cast<unsigned char>(c.a);
        }
    }
    if (srcFormat == "png") {
        if (!stbi_write_png(filename.c_str(), width, height, 4, buffer.data(), width * 4))
            std::cerr << "Failed to save PNG image: " << filename << "\n";
    } else if (srcFormat == "jpg" || srcFormat == "jpeg") {
        int quality;
        if (smallfile) {
            quality = 70;
        } else {
            quality = 90;
        }
        if (!stbi_write_jpg(filename.c_str(), width, height, 4, buffer.data(), quality))
            std::cerr << "Failed to save JPEG image: " << filename << "\n";
    } else if (srcFormat == "bmp") {
        if (!stbi_write_bmp(filename.c_str(), width, height, 4, buffer.data()))
            std::cerr << "Failed to save BMP image: " << filename << "\n";
    } else if (srcFormat == "tga") {
        if (!stbi_write_tga(filename.c_str(), width, height, 4, buffer.data()))
            std::cerr << "Failed to save TGA image: " << filename << "\n";
    } else {
        std::cerr << "Unknown source format \"" << srcFormat << "\"; defaulting to PNG.\n";
        if (!stbi_write_png(filename.c_str(), width, height, 4, buffer.data(), width * 4))
            std::cerr << "Failed to save PNG image: " << filename << "\n";
    }
    std::cout << "Saved compressed image to: " << filename << "\n";
}

/* Menyimpan GIF */
void QuadTree::save_gif(const std::string& gifOutputPath, int delay) {
    if (gifFrames.empty()) {
        std::cerr << "No frames recorded for GIF. Make sure gradual build is performed.\n";
        return;
    }
    int height = gifFrames[0].size();
    int width = gifFrames[0][0].size();
    GifWriter writer = {};
    if (!GifBegin(&writer, gifOutputPath.c_str(), width, height, delay)) {
        std::cerr << "Failed to initialize GIF writer.\n";
        return;
    }
    for (const auto &frame : gifFrames) {
        std::vector<unsigned char> buffer = convert_image_to_RGBA(frame);
        GifWriteFrame(&writer, buffer.data(), width, height, delay);
    }
    GifEnd(&writer);
    std::cout << "Saved compression process GIF to: " << gifOutputPath << "\n";
}

/* Memperoleh kedalaman pohon */
int QuadTree::get_tree_depth() const {
    return (root) ? root->get_depth() : 0;
}

/* Memperoleh jumlah simpul dalam pohon */
int QuadTree::get_total_node_count() const {
    return (root) ? root->get_node_count() : 0;
}

/* Metode untuk mode target compression */
void QuadTree::adjust_threshold_for_target(int errorChoice, double targetCompression,
                                           double tolerance, int maxIterations) {
    int minBlockSize = 2;
    double threshold, lowThreshold, highThreshold;
    std::cout << "Error Choice: " << errorChoice << std::endl;
    switch (errorChoice) {
        case 1:
            threshold = 8128;
            lowThreshold = 0;
            highThreshold = 16256;
            break;
        case 2:
            threshold = 63;
            lowThreshold = 0;
            highThreshold = 127;
            break;
        case 3:
            threshold = 127;
            lowThreshold = 0;
            highThreshold = 255;
            break;
        case 4:
            threshold = 4;
            lowThreshold = 0;
            highThreshold = 18;
            break;
        case 5:
            threshold = 0.5;
            lowThreshold = 0;
            highThreshold = 1;
            break;
        default:
            threshold = 50;
            lowThreshold = 0;
            highThreshold = 100;
            break;
    }
    int originalPixels = source_image_matrix.size() * source_image_matrix[0].size();
    double currentCompression = 0.0;
    int iter = 0;

    // Membangun pohon awal
    build_quad_tree(errorChoice, threshold, minBlockSize);
    int leafCount = get_leaf_count();
    currentCompression = (1.0 - static_cast<double>(leafCount) / originalPixels) * 100.0;

    while (std::abs(currentCompression - targetCompression * 100.0) > tolerance && iter < maxIterations) {
        std::cout << "Target Compression: " << targetCompression << std::endl;
        std::cout << "Current Compression: " << currentCompression << std::endl;
        if (currentCompression < targetCompression * 100.0) {
            // Kompresi terlalu sedikit sehingga threshold ditingkatkan
            lowThreshold = threshold;
            threshold = (threshold + highThreshold) / 2.0;
            std::cout << "Not enough compression: increasing threshold." << std::endl;
        } else {
            // Kompresi terlalu banyak sehingga threshold dikurangi
            highThreshold = threshold;
            threshold = (lowThreshold + threshold) / 2.0;
            std::cout << "Too much compression: decreasing threshold." << std::endl;
        }
        std::cout << "Threshold now: " << threshold << std::endl;

        // Hapus pohon lama dan bangun kembali
        delete root;
        root = new QuadTreeNode(0, 0, source_image_matrix[0].size(), source_image_matrix.size());
        build_quad_tree(errorChoice, threshold, minBlockSize);
        leafCount = get_leaf_count();
        currentCompression = (1.0 - static_cast<double>(leafCount) / originalPixels) * 100.0;
        iter++;
        std::cout << "Iteration " << iter << ": threshold = " << threshold
                  << ", compression = " << currentCompression << "%\n";
    }
}

/* Memperoleh jumlah daun */
int QuadTree::get_leaf_count() const {
    return (root) ? root->count_leaves() : 0;
}

```

C. QuadTreeNode

Header

```cpp
// Header
#ifndef QUADTREENODE_HPP
#define QUADTREENODE_HPP

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

extern std::vector<Image> gifFrames;
void capture_frame(const Image& img);

class QuadTreeNode {
    private:
        int x, y;
        int width, height;
        bool is_leaf;
        Color color;
        QuadTreeNode* top_left_child;
        QuadTreeNode* top_right_child;
        QuadTreeNode* bottom_left_child;
        QuadTreeNode* bottom_right_child;
    public:
    /* Konstruktor untuk QuadTreeNode */
        QuadTreeNode(int _x, int _y, int _width, int _height);
    /* Destruktor untuk QuadTreeNode */
        ~QuadTreeNode();
    /* Melakukan pembagian terhadap QuadTreeNode berdasarkan threshold dan min_block_size */
        void divide_node(const Image& image, int error_choice, double threshold, int min_block_size);
    /* Penggambaran node untuk mode cepat */
        void draw_node_fast(Image& result_image);
    /* Penggambaran node untuk mode gradual GIF */
        void draw_node_at_depth(Image& result_image, int currentDepth, int maxDepth);
    /* Menghitung jumlah daun dari suatu QuadTreeNode */
        int count_leaves() const;
    /* Menghitung kedalaman maksimal dari suatu QuadTreeNode */
        int max_depth() const;
    /* Menghitung kedalaman dari suatu QuadTreeNode */
        int get_depth() const;
    /* Menghitung jumlah dari suatu QuadTreeNode */
        int get_node_count() const;
};

#endif

```

Implementation

```cpp
#include "quadtreenode.hpp"
#include "utils.hpp"
#include <iostream>

/* Vektor untuk frames GIF */
std::vector<Image> gifFrames;
/* Fungsi untuk penangkapan frame GIF */
void capture_frame(const Image& img) {
    gifFrames.push_back(img);
}

/* Konstruktor untuk QuadTreeNode */
QuadTreeNode::QuadTreeNode(int _x, int _y, int _width, int _height)
    : x(_x), y(_y), width(_width), height(_height),
      is_leaf(false),
      top_left_child(nullptr), top_right_child(nullptr),
      bottom_left_child(nullptr), bottom_right_child(nullptr)
{
    color = {0, 0, 0, 255};
}

/* Destruktor untuk QuadTreeNode */
QuadTreeNode::~QuadTreeNode() {
    delete top_left_child;
    delete top_right_child;
    delete bottom_left_child;
    delete bottom_right_child;
}

/* Melakukan pembagian terhadap QuadTreeNode berdasarkan threshold dan min_block_size
*/
void QuadTreeNode::divide_node(const Image& image, int error_choice, double threshold,
int min_block_size) {
    double error = compute_block_error(image, x, y, width, height, error_choice);
    if ((error > threshold) && (width > min_block_size) && (height > min_block_size))
{
        int halfW = width / 2;
        int halfH = height / 2;

        top_left_child = new QuadTreeNode(x, y, halfW, halfH);
        top_left_child->divide_node(image, error_choice, threshold, min_block_size);

        top_right_child = new QuadTreeNode(x + halfW, y, width - halfW, halfH);
        top_right_child->divide_node(image, error_choice, threshold, min_block_size);

        bottom_left_child = new QuadTreeNode(x, y + halfH, halfW, height - halfH);
        bottom_left_child->divide_node(image, error_choice, threshold,
min_block_size);

        bottom_right_child = new QuadTreeNode(x + halfW, y + halfH, width - halfW,
height - halfH);
        bottom_right_child->divide_node(image, error_choice, threshold,
min_block_size);
    } else {
        is_leaf = true;
        color = compute_average_color(image, x, y, width, height);
    }
}

/* Penggambaran node untuk mode cepat */
void QuadTreeNode::draw_node_fast(Image& result_image) {
    if (is_leaf) {
        // Mengisi semua pixel dalam area ini dengan warna
        for (int row = y; row < y + height; ++row) {
            for (int col = x; col < x + width; ++col) {
                result_image[row][col] = color;
            }
        }
    } else {
        // Rekursi
        if (top_left_child)    top_left_child->draw_node_fast(result_image);
        if (top_right_child)   top_right_child->draw_node_fast(result_image);
        if (bottom_left_child) bottom_left_child->draw_node_fast(result_image);
        if (bottom_right_child) bottom_right_child->draw_node_fast(result_image);
    }
}

/* Penggambaran node untuk mode gradual GIF */
void QuadTreeNode::draw_node_at_depth(Image& result_image, int currentDepth, int maxDepth) {
    if (currentDepth >= maxDepth || is_leaf) {
        for (int row = y; row < y + height; ++row) {
            for (int col = x; col < x + width; ++col) {
                result_image[row][col] = color;
            }
        }
        return;
    }
    if (top_left_child)    top_left_child->draw_node_at_depth(result_image, currentDepth + 1, maxDepth);
    if (top_right_child)   top_right_child->draw_node_at_depth(result_image, currentDepth + 1, maxDepth);
    if (bottom_left_child) bottom_left_child->draw_node_at_depth(result_image, currentDepth + 1, maxDepth);
    if (bottom_right_child) bottom_right_child->draw_node_at_depth(result_image, currentDepth + 1, maxDepth);
}

/* Menghitung jumlah daun dari suatu QuadTreeNode */
int QuadTreeNode::count_leaves() const {
    if (is_leaf)
        return 1;
    int count = 0;
    if (top_left_child)    count += top_left_child->count_leaves();
    if (top_right_child)   count += top_right_child->count_leaves();
    if (bottom_left_child) count += bottom_left_child->count_leaves();
    if (bottom_right_child)count += bottom_right_child->count_leaves();
    return count;
}

/* Menghitung kedalaman maksimal dari suatu QuadTreeNode */
int QuadTreeNode::max_depth() const {
    if (is_leaf)
        return 0;
    int depthTL = top_left_child ? top_left_child->max_depth() : 0;
    int depthTR = top_right_child ? top_right_child->max_depth() : 0;
    int depthBL = bottom_left_child ? bottom_left_child->max_depth() : 0;
    int depthBR = bottom_right_child ? bottom_right_child->max_depth() : 0;
    return 1 + std::max({depthTL, depthTR, depthBL, depthBR});
}

/* Menghitung kedalaman dari suatu QuadTreeNode */
int QuadTreeNode::get_depth() const {
    if (is_leaf)
        return 1;
    int depthTL = top_left_child ? top_left_child->max_depth() : 0;
    int depthTR = top_right_child ? top_right_child->max_depth() : 0;
    int depthBL = bottom_left_child ? bottom_left_child->max_depth() : 0;
    int depthBR = bottom_right_child ? bottom_right_child->max_depth() : 0;
    return 1 + std::max({depthTL, depthTR, depthBL, depthBR});
}
```

```

/* Menghitung jumlah dari suatu QuadTreeNode */
int QuadTreeNode::get_node_count() const {
    int count = 1; // node ini
    if (top_left_child)    count += top_left_child->get_node_count();
    if (top_right_child)   count += top_right_child->get_node_count();
    if (bottom_left_child) count += bottom_left_child->get_node_count();
    if (bottom_right_child) count += bottom_right_child->get_node_count();
    return count;
}

```

D. Utils

Header

```
#ifndef UTILS_HPP
#define UTILS_HPP

#include <iostream>
#include <vector>
#include <cmath>
#include <map>
#include <algorithm>
#include <string>
#include <cctype>

#include "stb_image.h"
#include "stb_image_write.h"

struct Color {
    int r, g, b;
    int a; // alpha channel
};

typedef std::vector<std::vector<Color>> Image;

/* Menghitung error untuk variansi, MAD, max pixel difference, dan entropi */
double compute_block_error(const Image& image, int startX, int startY, int blockWidth,
int blockHeight, int errorChoice);

/* Menghitung rerata warna dari suatu blok*/
Color compute_average_color(const Image& image, int startX, int startY, int blockWidth,
int blockHeight);

/* Konversi data gambar menjadi array RGBA */
std::vector<unsigned char> convert_image_to_RGBA(const Image& image);

/* Fungsi pembantu untuk load image */
Image load_image_rgba(const std::string& filename);

/* Fungsi pembantu untuk memperoleh ekstensi */
std::string get_extension(const std::string& filename);

#endif

```

Implementation

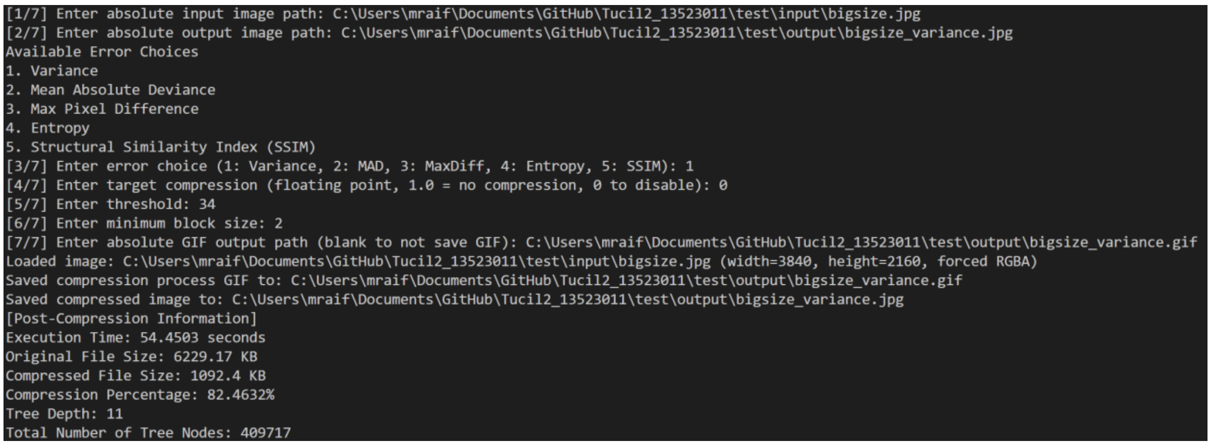
```
#include "utils.hpp"

/* Menghitung error SSIM */
static double compute_ssim_error_for_block(const Image& image, int startX, int startY,
int blockWidth, int blockHeight) {
```

```cpp
    int count = blockWidth * blockHeight;
    double sumR = 0.0, sumG = 0.0, sumB = 0.0;
    for (int y = startY; y < startY + blockHeight; ++y) {
        for (int x = startX; x < startX + blockWidth; ++x) {
            const Color &c = image[y][x];
            sumR += c.r;
            sumG += c.g;
            sumB += c.b;
        }
    }
    double meanR = sumR / count;
    double meanG = sumG / count;
    double meanB = sumB / count;

    double varR = 0.0, varG = 0.0, varB = 0.0;
    for (int y = startY; y < startY + blockHeight; ++y) {
        for (int x = startX; x < startX + blockWidth; ++x) {
            const Color &c = image[y][x];
            varR += (c.r - meanR) * (c.r - meanR);
            varG += (c.g - meanG) * (c.g - meanG);
            varB += (c.b - meanB) * (c.b - meanB);
        }
    }
    varR /= count;
    varG /= count;
    varB /= count;

    // Constants SSIM untuk gambar 8-bit
    const double C1 = (0.01 * 255) * (0.01 * 255); // ≈ 6.5025
    const double C2 = (0.03 * 255) * (0.03 * 255); // ≈ 58.5225

    // Untuk sebuah blok seragam (blok terkompresi), μ_y sama dengan μ_x dan variansinya adalah 0.
    // Oleh karena itu, SSIM tiap channel dapat disederhanakan menjadi:
    // SSIM_c = (2 μ_x^2 + C1)*(C2) / ((2 μ_x^2 + C1)*(σ_x^2 + C2)) = C2 / (σ_x^2 + C2)
    double ssimR = C2 / (varR + C2);
    double ssimG = C2 / (varG + C2);
    double ssimB = C2 / (varB + C2);
    double ssim = (ssimR + ssimG + ssimB) / 3.0;
    // Error didefinisikan sebagai 1 - SSIM.
    return 1.0 - ssim;
}

/* Menghitung error untuk variansi, MAD, max pixel difference, dan entropi */
double compute_block_error(const Image& image, int startX, int startY, int blockWidth, int blockHeight, int errorChoice) {
    int imageHeight = static_cast<int>(image.size());
    if (imageHeight == 0) return 0.0;
    int imageWidth = static_cast<int>(image[0].size());
    if (startX < 0 || startY < 0 || startX + blockWidth > imageWidth || startY + blockHeight > imageHeight) {
        std::cerr << "Error: Block parameters out of image bounds." << std::endl;
        return 0.0;
    }

    const int N = blockWidth * blockHeight;
    if (N == 0) return 0.0;

    if (errorChoice == 5) {
        return compute_ssim_error_for_block(image, startX, startY, blockWidth, blockHeight);
    }

    std::vector<int> pixelsR, pixelsG, pixelsB;
    pixelsR.reserve(N);
    pixelsG.reserve(N);
    pixelsB.reserve(N);

    for (int yy = startY; yy < startY + blockHeight; ++yy) {
        for (int xx = startX; xx < startX + blockWidth; ++xx) {
            const Color& c = image[yy][xx];
            pixelsR.push_back(c.r);
            pixelsG.push_back(c.g);
            pixelsB.push_back(c.b);
        }
    }

    auto computeChannelError = [&](const std::vector<int>& pix) -> double {
        int minVal = 255, maxVal = 0;
        long long sumVal = 0;
        for (int val : pix) {
            if(val < minVal) minVal = val;
            if(val > maxVal) maxVal = val;
            sumVal += val;
        }
        double mean = static_cast<double>(sumVal) / pix.size();
        switch(errorChoice) {
            case 1: { // Variance
                double var = 0.0;
                for (int val : pix) {
                    double diff = val - mean;
                    var += diff * diff;
                }
                return var / pix.size();
            }
            case 2: { // Mean Absolute Deviation (MAD)
                double mad = 0.0;
                for (int val : pix) {
                    mad += std::fabs(val - mean);
                }
                return mad / pix.size();
            }
            case 3: { // Max Pixel Difference
                return static_cast<double>(maxVal - minVal);
            }
            case 4: { // Entropy
                std::map<int,int> freq;
                for (int val : pix) {
                    freq[val]++;
                }
                double entropy = 0.0;
                for (const auto& kv : freq) {
                    double p = static_cast<double>(kv.second) / pix.size();
                    entropy += -p * std::log2(p);
                }
                return entropy;
            }
            default:
                std::cerr << "Unknown errorChoice! Defaulting to variance.\n";
                double var = 0.0;
                for (int val : pix) {
                    double diff = val - mean;
                    var += diff * diff;
                }
                return var / pix.size();
        }
    };

    double errorR = computeChannelError(pixelsR);
    double errorG = computeChannelError(pixelsG);
    double errorB = computeChannelError(pixelsB);
    return (errorR + errorG + errorB) / 3.0;
}

/* Menghitung rerata warna dari suatu blok*/
Color compute_average_color(const Image& image, int startX, int startY, int blockWidth, int blockHeight) {
    long long sumR = 0, sumG = 0, sumB = 0, sumA = 0;
    int count = blockWidth * blockHeight;
    for (int yy = startY; yy < startY + blockHeight; ++yy) {
        for (int xx = startX; xx < startX + blockWidth; ++xx) {
            const Color &c = image[yy][xx];
            sumR += c.r;
            sumG += c.g;
            sumB += c.b;
            sumA += c.a;
        }
    }
    Color avg;
    avg.r = static_cast<int>(sumR / count);
    avg.g = static_cast<int>(sumG / count);
    avg.b = static_cast<int>(sumB / count);
    avg.a = static_cast<int>(sumA / count);
    return avg;
}

/* Konversi data gambar menjadi array RGBA */
std::vector<unsigned char> convert_image_to_RGBA(const Image& image) {
    int height = image.size();
    int width = (height > 0) ? image[0].size() : 0;
    std::vector<unsigned char> buffer(width * height * 4);
    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            int idx = (y * width + x) * 4;
            const Color &c = image[y][x];
            buffer[idx + 0] = static_cast<unsigned char>(c.r);
            buffer[idx + 1] = static_cast<unsigned char>(c.g);
            buffer[idx + 2] = static_cast<unsigned char>(c.b);
            buffer[idx + 3] = static_cast<unsigned char>(c.a);
        }
    }
    return buffer;
}

/* Fungsi pembantu untuk memperoleh ekstensi */
std::string get_extension(const std::string& filename) {
    size_t pos = filename.find_last_of('.');
    if (pos == std::string::npos) return "";
    std::string ext = filename.substr(pos + 1);
    for (auto &ch : ext) {
        ch = std::tolower(ch);
    }
    return ext;
}

/* Fungsi pembantu untuk load image */
Image load_image_rgba(const std::string& filename) {
    int width, height, channels;
    unsigned char* data = stbi_load(filename.c_str(), &width, &height, &channels, STBI_rgb_alpha);
    if (!data) {
        std::cerr << "Failed to load image: " << filename << "\n";
        return {};
    }
    Image img(height, std::vector<Color>(width, Color{0, 0, 0, 255}));
    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            int idx = (y * width + x) * 4;
            Color c;
            c.r = data[idx + 0];
            c.g = data[idx + 1];
            c.b = data[idx + 2];
            c.a = data[idx + 3];
            img[y][x] = c;
        }
    }
    stbi_image_free(data);
    std::cout << "Loaded image: " << filename
              << " (width=" << width << ", height=" << height << ", forced RGBA)\n";
    return img;
}

```

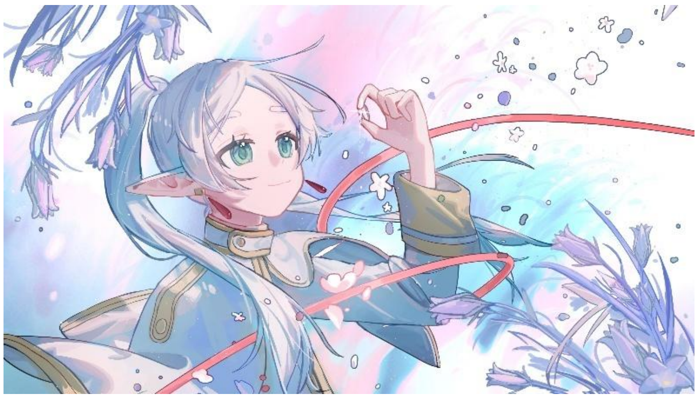
BAB IV
PENGUJIAN

Program dikompilasi dari *root folder* proyek dengan perintah "g++ -o \bin\main \src\main.cpp \src\quadtree.cpp \src\quadtreenode.cpp \src\utils.cpp". Setelah itu, program dijalankan dengan perintah "./bin/main".

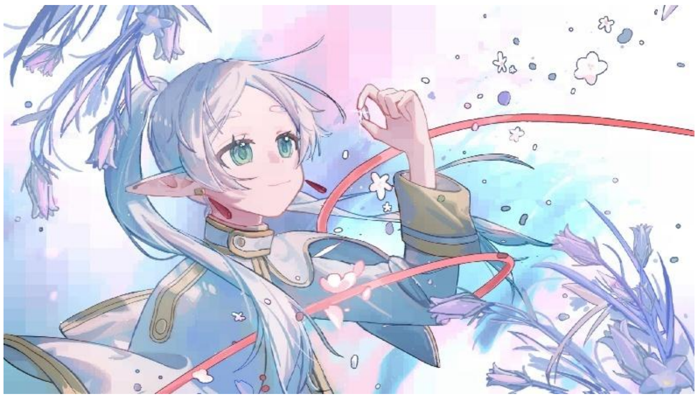
1) Gambar Berukuran Besar – Variansi



```
[1/7] Enter absolute input image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\bigsize.jpg
[2/7] Enter absolute output image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_variance.jpg
Available Error Choices
1. Variance
2. Mean Absolute Deviance
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 1
[4/7] Enter target compression (floating point, 1.0 = no compression, 0 to disable): 0
[5/7] Enter threshold: 34
[6/7] Enter minimum block size: 2
[7/7] Enter absolute GIF output path (blank to not save GIF): C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_variance.gif
Loaded image: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\bigsize.jpg (width=3840, height=2160, forced RGBA)
Saved compression process GIF to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_variance.gif
Saved compressed image to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_variance.jpg
[Post-Compression Information]
Execution Time: 54.4503 seconds
Original File Size: 6229.17 KB
Compressed File Size: 1092.4 KB
Compression Percentage: 82.4632%
Tree Depth: 11
Total Number of Tree Nodes: 409717
```



Gambar Input

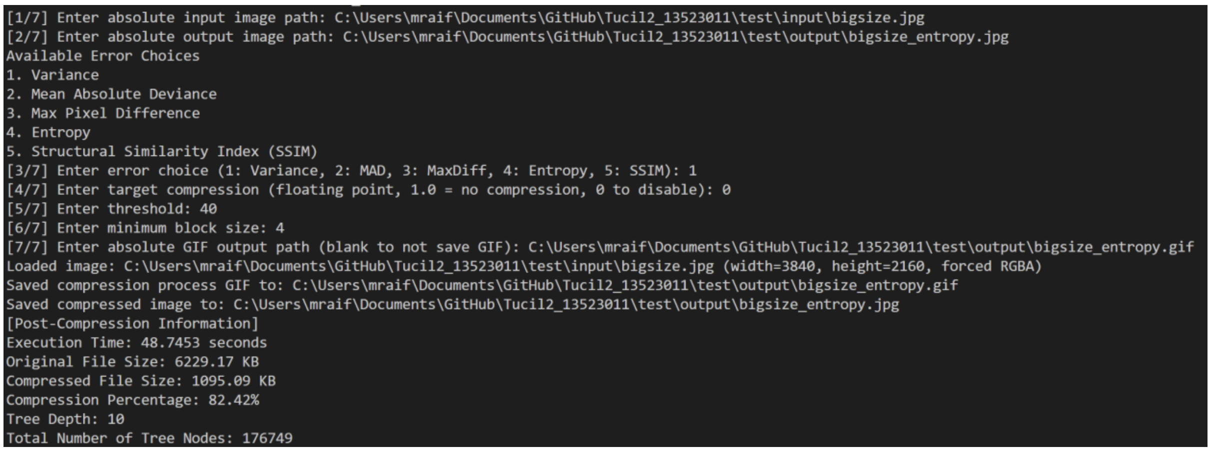


Gambar Output

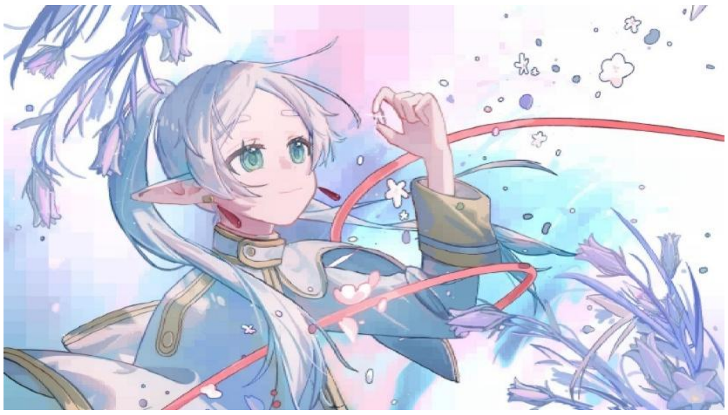
https://drive.google.com/file/d/1c0T2EG-IC35SileBdNDPw7ziAYAXz1Rd/view?usp=drive_link

GIF Output

2) Gambar Berukuran Besar – Entropi



```
[1/7] Enter absolute input image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\bigsize.jpg
[2/7] Enter absolute output image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_entropy.jpg
Available Error Choices
1. Variance
2. Mean Absolute Deviance
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 1
[4/7] Enter target compression (floating point, 1.0 = no compression, 0 to disable): 0
[5/7] Enter threshold: 40
[6/7] Enter minimum block size: 4
[7/7] Enter absolute GIF output path (blank to not save GIF): C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_entropy.gif
Loaded image: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\bigsize.jpg (width=3840, height=2160, forced RGBA)
Saved compression process GIF to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_entropy.gif
Saved compressed image to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_entropy.jpg
[Post-Compression Information]
Execution Time: 48.7453 seconds
Original File Size: 6229.17 KB
Compressed File Size: 1095.09 KB
Compression Percentage: 82.42%
Tree Depth: 10
Total Number of Tree Nodes: 176749
```

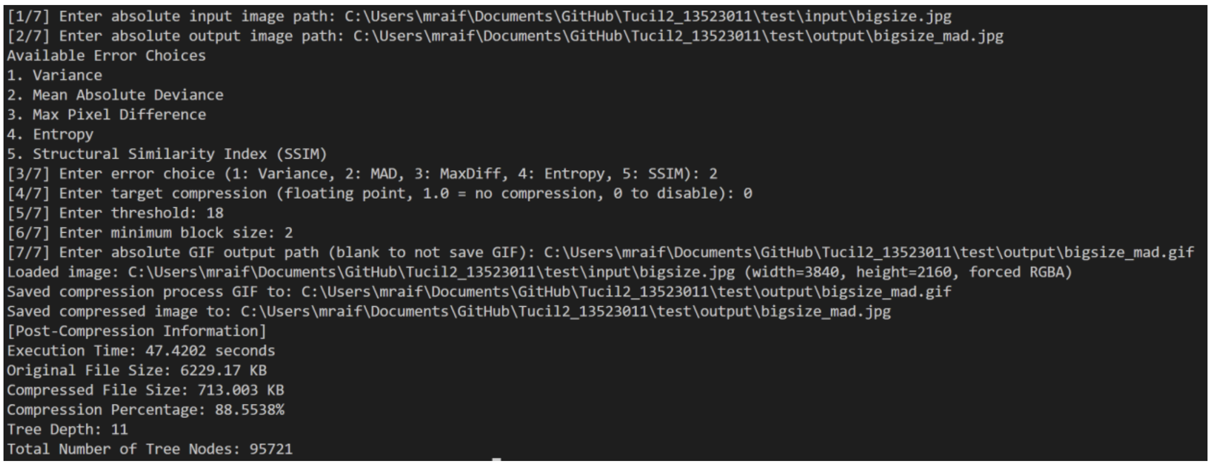


Gambar Output

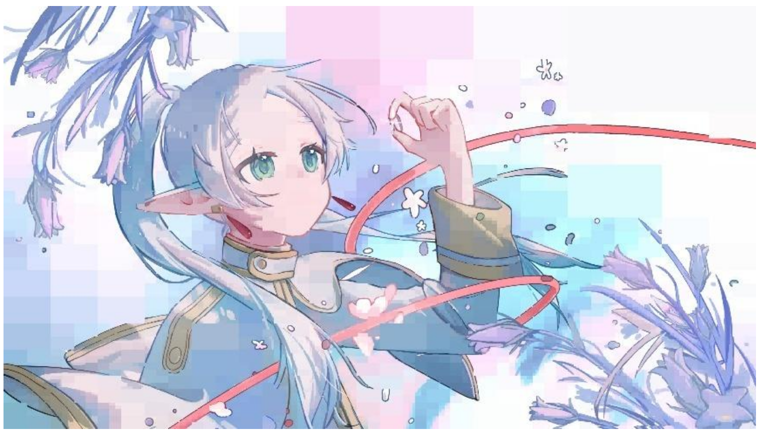
https://drive.google.com/file/d/1kreZPKSQzvhOU9qEn8BzakjRCI1FgOW3/view?usp=drive_link

GIF Output

3) Gambar Berukuran Besar – Mean Absolute Deviation (MAD)



```
[1/7] Enter absolute input image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\bigsize.jpg
[2/7] Enter absolute output image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_mad.jpg
Available Error Choices
1. Variance
2. Mean Absolute Deviance
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 2
[4/7] Enter target compression (floating point, 1.0 = no compression, 0 to disable): 0
[5/7] Enter threshold: 18
[6/7] Enter minimum block size: 2
[7/7] Enter absolute GIF output path (blank to not save GIF): C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_mad.gif
Loaded image: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\bigsize.jpg (width=3840, height=2160, forced RGBA)
Saved compression process GIF to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_mad.gif
Saved compressed image to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_mad.jpg
[Post-Compression Information]
Execution Time: 47.4202 seconds
Original File Size: 6229.17 KB
Compressed File Size: 713.003 KB
Compression Percentage: 88.5538%
Tree Depth: 11
Total Number of Tree Nodes: 95721
```

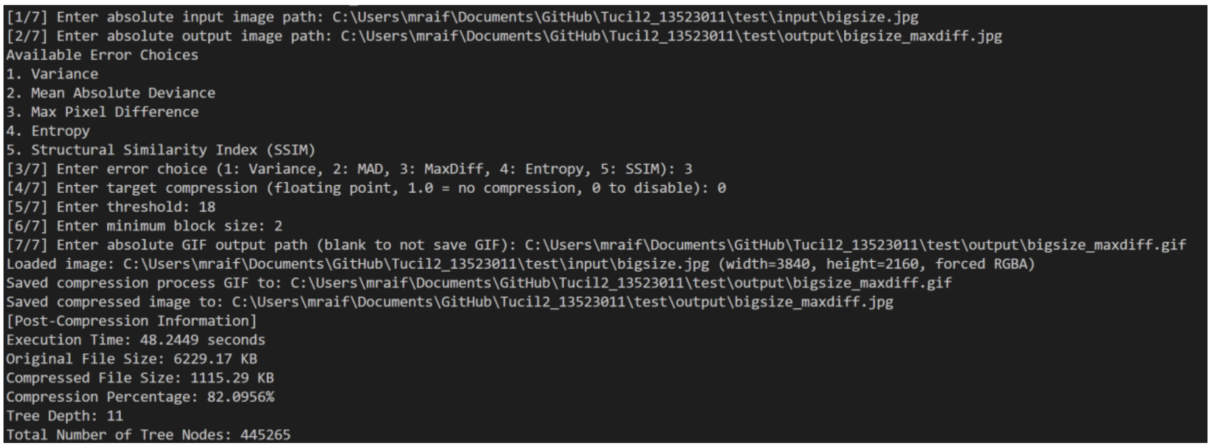


Gambar Output

https://drive.google.com/file/d/1f2uMW52yVxCZp0bjFggPyqO4soXSyUsS/view?usp=drive_link

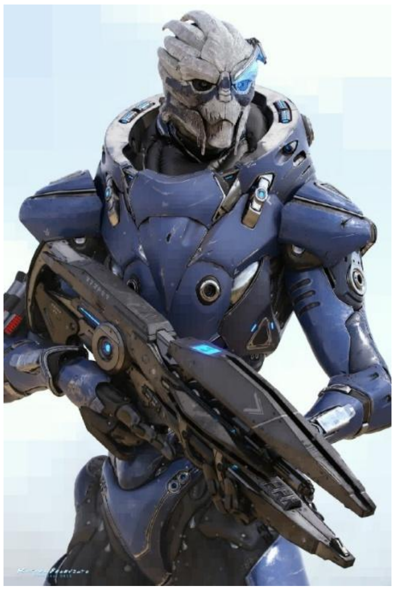
GIF Output

4) Gambar Berukuran Sedang – Max Pixel Difference



```
[1/7] Enter absolute input image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\bigsize.jpg
[2/7] Enter absolute output image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_maxdiff.jpg
Available Error Choices
1. Variance
2. Mean Absolute Deviance
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 3
[4/7] Enter target compression (floating point, 1.0 = no compression, 0 to disable): 0
[5/7] Enter threshold: 18
[6/7] Enter minimum block size: 2
[7/7] Enter absolute GIF output path (blank to not save GIF): C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_maxdiff.gif
Loaded image: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\bigsize.jpg (width=3840, height=2160, forced RGBA)
Saved compression process GIF to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_maxdiff.gif
Saved compressed image to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\bigsize_maxdiff.jpg
[Post-Compression Information]
Execution Time: 48.2449 seconds
Original File Size: 6229.17 KB
Compressed File Size: 1115.29 KB
Compression Percentage: 82.0956%
Tree Depth: 11
Total Number of Tree Nodes: 445265
```

Gambar Input

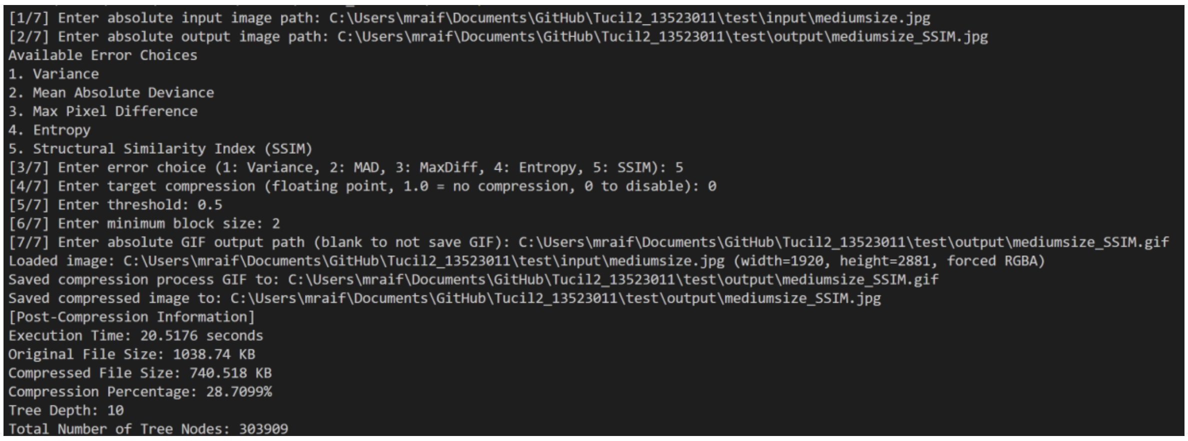


Gambar Output

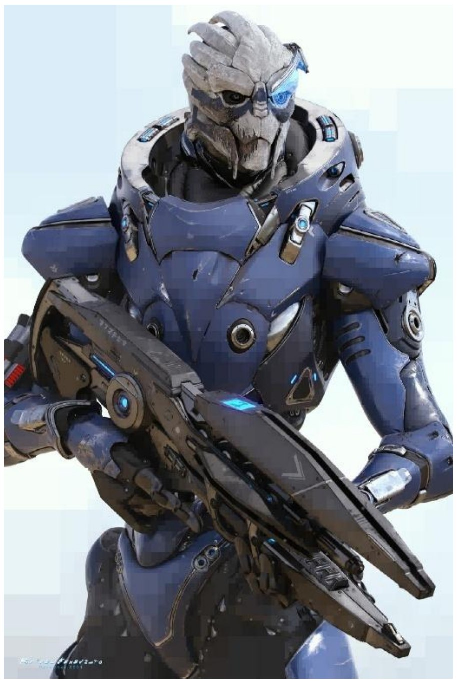
https://drive.google.com/file/d/1IEniQm7nq6hWEL80KKYqN8r9q-Wnjcr2/view?usp=drive_link

GIF Output

5) Gambar Berukuran Sedang – SSIM



```
[1/7] Enter absolute input image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\mediumsize.jpg
[2/7] Enter absolute output image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\mediumsize_SSIM.jpg
Available Error Choices
1. Variance
2. Mean Absolute Deviance
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 5
[4/7] Enter target compression (floating point, 1.0 = no compression, 0 to disable): 0
[5/7] Enter threshold: 0.5
[6/7] Enter minimum block size: 2
[7/7] Enter absolute GIF output path (blank to not save GIF): C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\mediumsize_SSIM.gif
Loaded image: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\mediumsize.jpg (width=1920, height=2881, forced RGBA)
Saved compression process GIF to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\mediumsize_SSIM.gif
Saved compressed image to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\mediumsize_SSIM.jpg
[Post-Compression Information]
Execution Time: 20.5176 seconds
Original File Size: 1038.74 KB
Compressed File Size: 740.518 KB
Compression Percentage: 28.7099%
Tree Depth: 10
Total Number of Tree Nodes: 303909
```

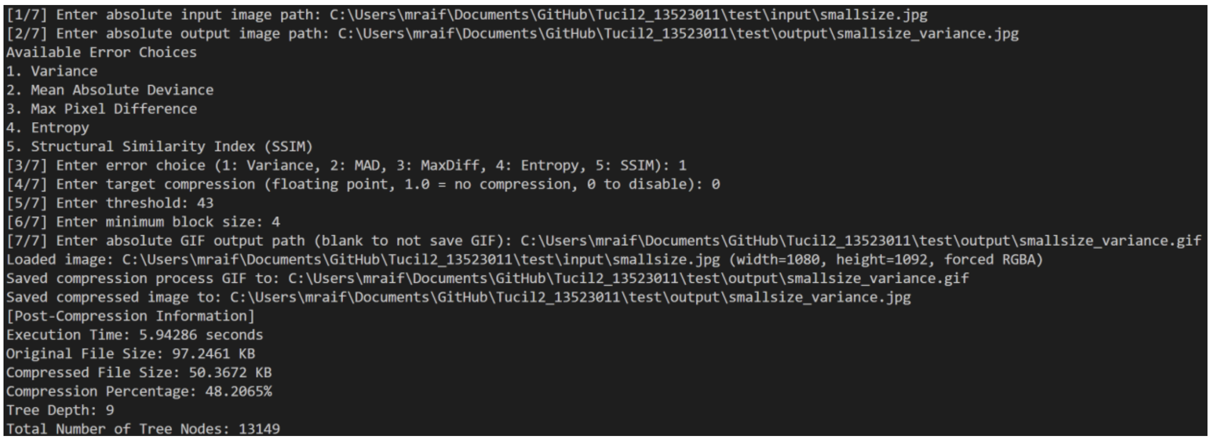


Gambar Output

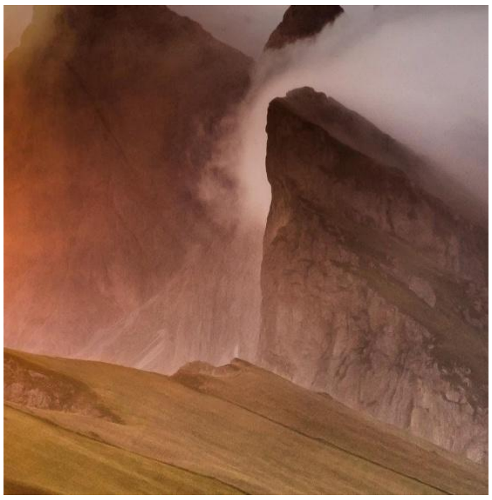
https://drive.google.com/file/d/1L5nxeIwFrlIkB-UjZo0qJEmis_eszYVx/view?usp=drive_link

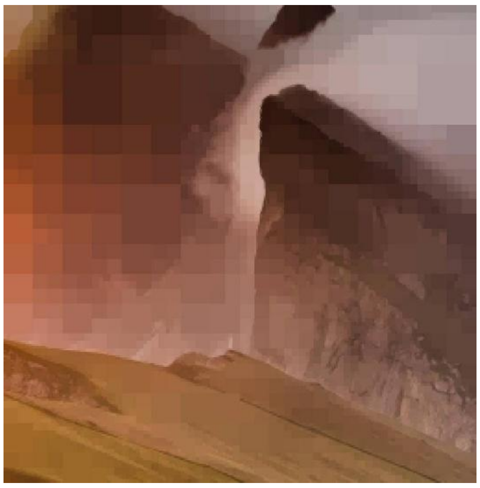
GIF Output

6) Gambar Berukuran Kecil – Variansi



```
[1/7] Enter absolute input image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\smallsize.jpg
[2/7] Enter absolute output image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\smallsize_variance.jpg
Available Error Choices
1. Variance
2. Mean Absolute Deviance
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 1
[4/7] Enter target compression (floating point, 1.0 = no compression, 0 to disable): 0
[5/7] Enter threshold: 43
[6/7] Enter minimum block size: 4
[7/7] Enter absolute GIF output path (blank to not save GIF): C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\smallsize_variance.gif
Loaded image: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\smallsize.jpg (width=1080, height=1092, forced RGBA)
Saved compression process GIF to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\smallsize_variance.gif
Saved compressed image to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\smallsize_variance.jpg
[Post-Compression Information]
Execution Time: 5.94286 seconds
Original File Size: 97.2461 KB
Compressed File Size: 50.3672 KB
Compression Percentage: 48.2065%
Tree Depth: 9
Total Number of Tree Nodes: 13149
```

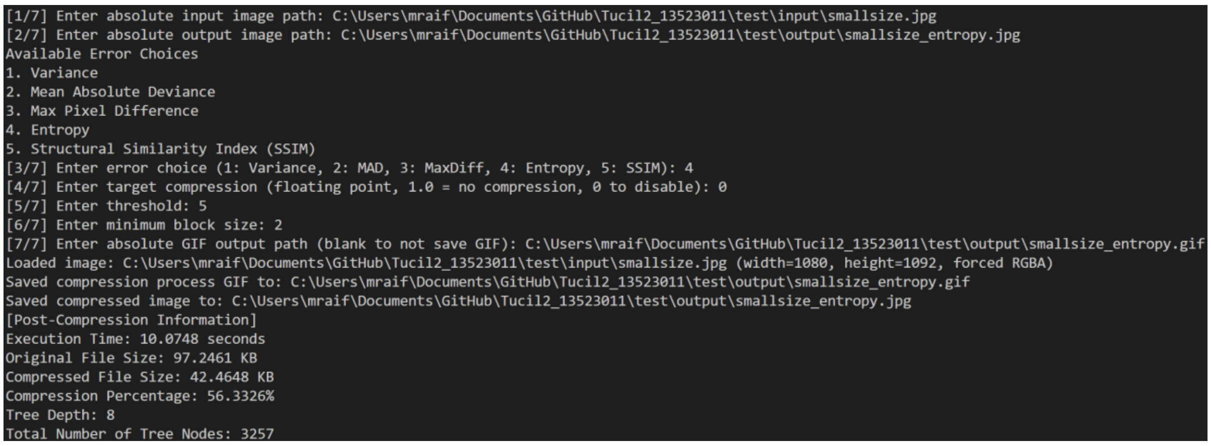

Gambar Input


Gambar Output

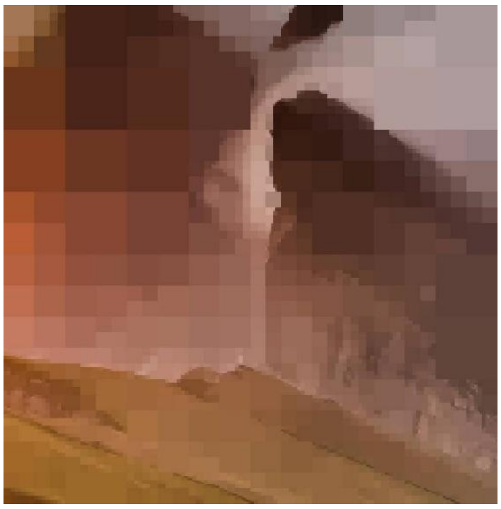
https://drive.google.com/file/d/19W4BsNVS_k-MGqTn-bHWf4ssfGZxO7Gl/view?usp=drive_link

GIF Output

7) Gambar Berukuran Kecil – Entropi



```
[1/7] Enter absolute input image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\smallsize.jpg
[2/7] Enter absolute output image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\smallsize_entropy.jpg
Available Error Choices
1. Variance
2. Mean Absolute Deviance
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 4
[4/7] Enter target compression (floating point, 1.0 = no compression, 0 to disable): 0
[5/7] Enter threshold: 5
[6/7] Enter minimum block size: 2
[7/7] Enter absolute GIF output path (blank to not save GIF): C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\smallsize_entropy.gif
Loaded image: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\smallsize.jpg (width=1080, height=1092, forced RGBA)
Saved compression process GIF to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\smallsize_entropy.gif
Saved compressed image to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\smallsize_entropy.jpg
[Post-Compression Information]
Execution Time: 10.0748 seconds
Original File Size: 97.2461 KB
Compressed File Size: 42.4648 KB
Compression Percentage: 56.3326%
Tree Depth: 8
Total Number of Tree Nodes: 3257
```

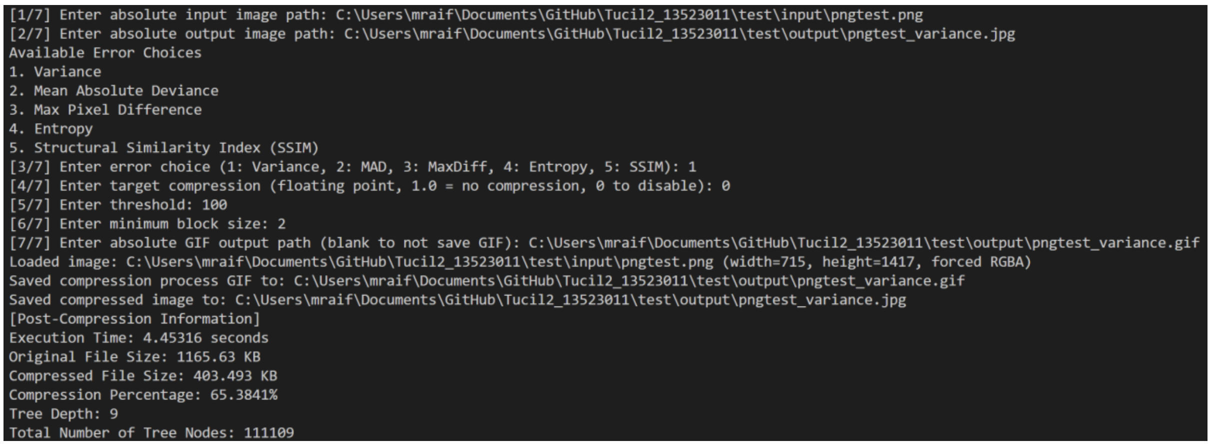


Gambar Output

https://drive.google.com/file/d/10NaHCpjd8NCXELOgCsodkm5TW7grurWQ/view?usp=drive_link

GIF Output

8) Gambar PNG – Variansi

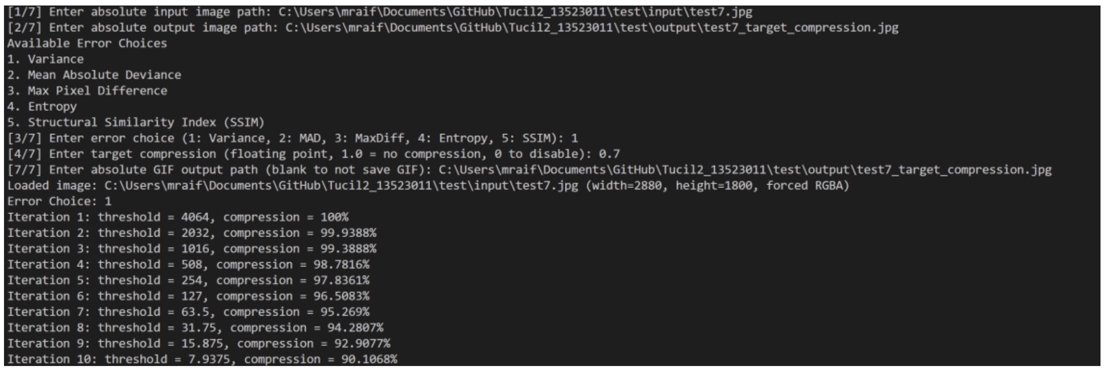


```
[1/7] Enter absolute input image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\pngtest.png
[2/7] Enter absolute output image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\pngtest_variance.jpg
Available Error Choices
1. Variance
2. Mean Absolute Deviance
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 1
[4/7] Enter target compression (floating point, 1.0 = no compression, 0 to disable): 0
[5/7] Enter threshold: 100
[6/7] Enter minimum block size: 2
[7/7] Enter absolute GIF output path (blank to not save GIF): C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\pngtest_variance.gif
Loaded image: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\pngtest.png (width=715, height=1417, forced RGBA)
Saved compression process GIF to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\pngtest_variance.gif
Saved compressed image to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\pngtest_variance.jpg
[Post-Compression Information]
Execution Time: 4.45316 seconds
Original File Size: 1165.63 KB
Compressed File Size: 403.493 KB
Compression Percentage: 65.3841%
Tree Depth: 9
Total Number of Tree Nodes: 111109
```

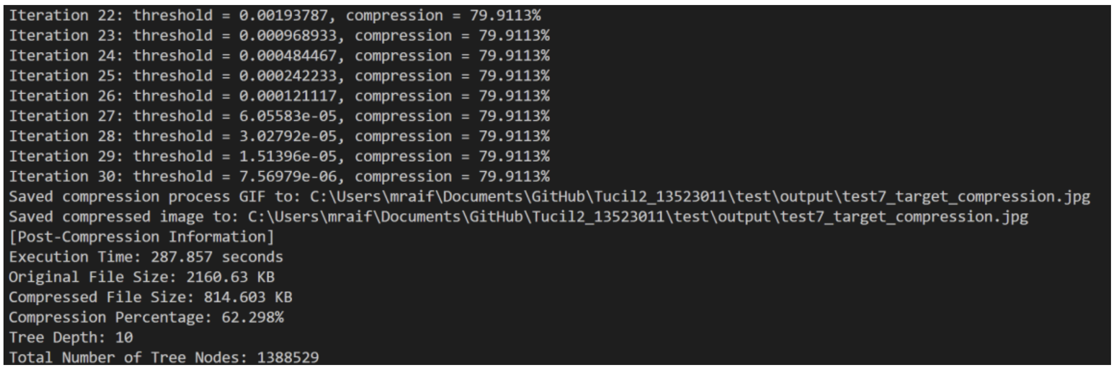
Gambar Input

Gambar Output

9) Gambar Berukuran Sedang – Variansi (Target Compression)

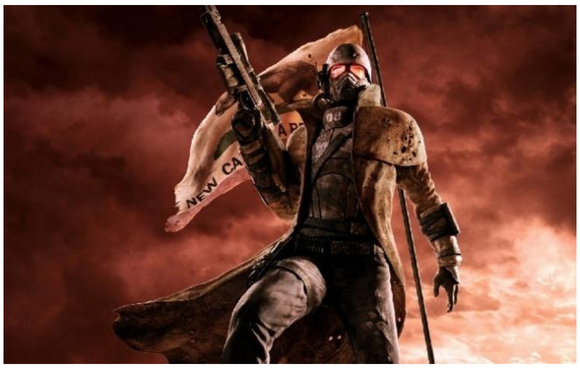


```
[1/7] Enter absolute input image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\test7.jpg
[2/7] Enter absolute output image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\test7_target_compression.jpg
Available Error Choices
1. Variance
2. Mean Absolute Deviance
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 1
[4/7] Enter target compression (floating point, 1.0 = no compression, 0 to disable): 0.7
[7/7] Enter absolute GIF output path (blank to not save GIF): C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\test7_target_compression.jpg
Loaded image: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\test7.jpg (width=2880, height=1800, forced RGBA)
Error Choice: 1
Iteration 1: threshold = 4064, compression = 100%
Iteration 2: threshold = 2032, compression = 99.9388%
Iteration 3: threshold = 1016, compression = 99.3888%
Iteration 4: threshold = 508, compression = 98.7816%
Iteration 5: threshold = 254, compression = 97.8361%
Iteration 6: threshold = 127, compression = 96.5083%
Iteration 7: threshold = 63.5, compression = 95.269%
Iteration 8: threshold = 31.75, compression = 94.2807%
Iteration 9: threshold = 15.875, compression = 92.9077%
Iteration 10: threshold = 7.9375, compression = 90.1068%
```



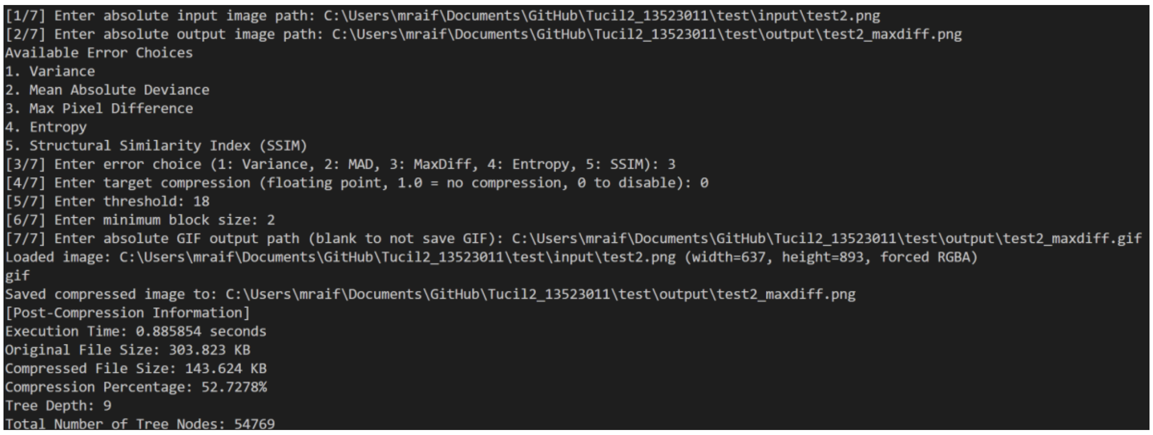
```
Iteration 22: threshold = 0.00193787, compression = 79.9113%
Iteration 23: threshold = 0.000968933, compression = 79.9113%
Iteration 24: threshold = 0.000484467, compression = 79.9113%
Iteration 25: threshold = 0.000242233, compression = 79.9113%
Iteration 26: threshold = 0.000121117, compression = 79.9113%
Iteration 27: threshold = 6.05583e-05, compression = 79.9113%
Iteration 28: threshold = 3.02792e-05, compression = 79.9113%
Iteration 29: threshold = 1.51396e-05, compression = 79.9113%
Iteration 30: threshold = 7.56979e-06, compression = 79.9113%
Saved compression process GIF to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\test7_target_compression.jpg
Saved compressed image to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\test7_target_compression.jpg
[Post-Compression Information]
Execution Time: 287.857 seconds
Original File Size: 2160.63 KB
Compressed File Size: 814.603 KB
Compression Percentage: 62.298%
Tree Depth: 10
Total Number of Tree Nodes: 1388529
```

Gambar Input



Gambar Output

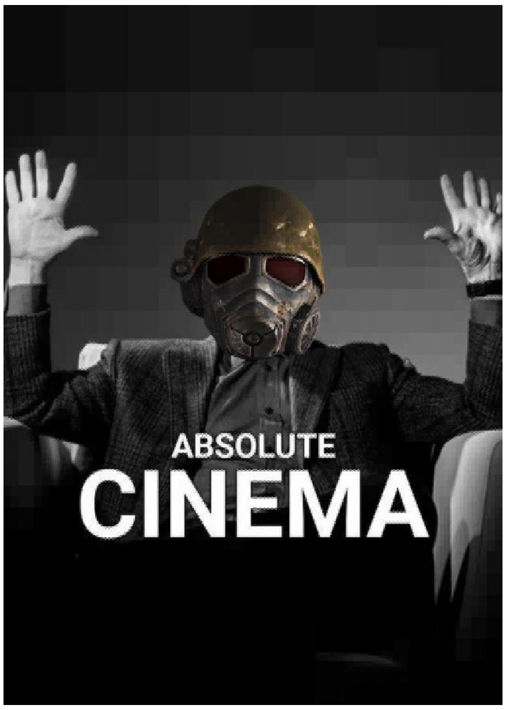
10) Gambar Berukuran Sedang – Max Pixel Difference



```
[1/7] Enter absolute input image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\test2.png
[2/7] Enter absolute output image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\test2_maxdiff.png
Available Error Choices
1. Variance
2. Mean Absolute Deviance
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 3
[4/7] Enter target compression (floating point, 1.0 = no compression, 0 to disable): 0
[5/7] Enter threshold: 18
[6/7] Enter minimum block size: 2
[7/7] Enter absolute GIF output path (blank to not save GIF): C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\test2_maxdiff.gif
Loaded image: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\test2.png (width=637, height=893, forced RGBA)
gif
Saved compressed image to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\test2_maxdiff.png
[Post-Compression Information]
Execution Time: 0.885854 seconds
Original File Size: 303.823 KB
Compressed File Size: 143.624 KB
Compression Percentage: 52.7278%
Tree Depth: 9
Total Number of Tree Nodes: 54769
```

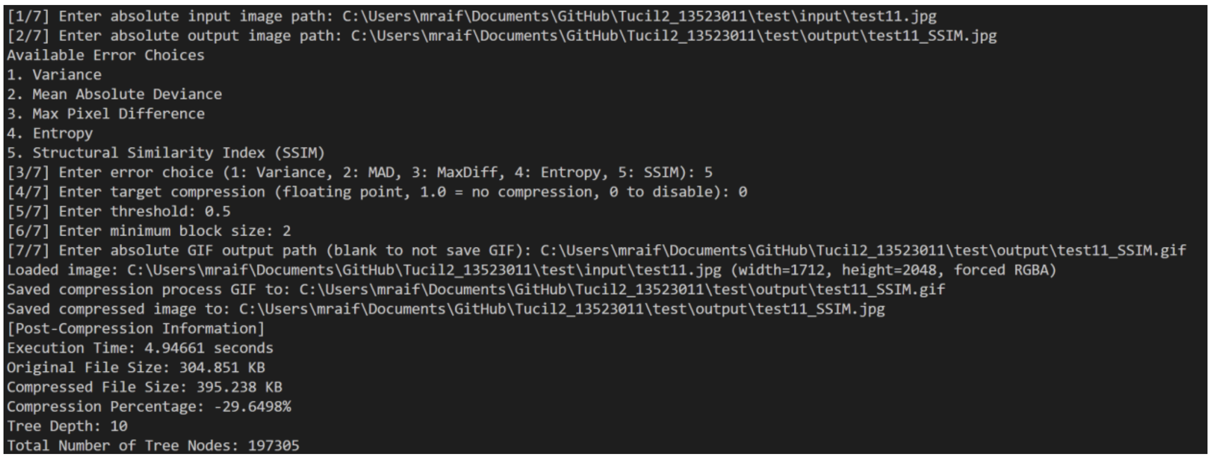
ABSOLUTE
CINEMA

Gambar Input

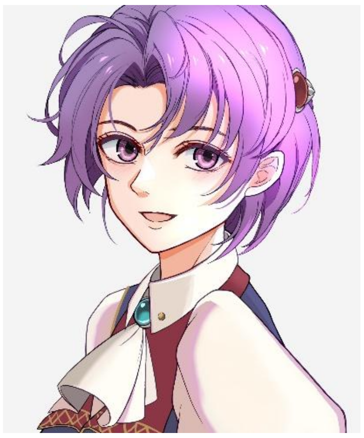


Gambar Output

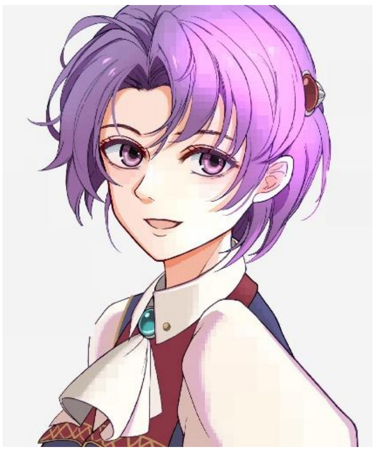
11) Gambar Berukuran Kecil – SSIM



```
[1/7] Enter absolute input image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\test11.jpg
[2/7] Enter absolute output image path: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\test11_SSIM.jpg
Available Error Choices
1. Variance
2. Mean Absolute Deviance
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[3/7] Enter error choice (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 5
[4/7] Enter target compression (floating point, 1.0 = no compression, 0 to disable): 0
[5/7] Enter threshold: 0.5
[6/7] Enter minimum block size: 2
[7/7] Enter absolute GIF output path (blank to not save GIF): C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\test11_SSIM.gif
Loaded image: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\input\test11.jpg (width=1712, height=2048, forced RGBA)
Saved compression process GIF to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\test11_SSIM.gif
Saved compressed image to: C:\Users\mraif\Documents\GitHub\Tucil2_13523011\test\output\test11_SSIM.jpg
[Post-Compression Information]
Execution Time: 4.94661 seconds
Original File Size: 304.851 KB
Compressed File Size: 395.238 KB
Compression Percentage: -29.6498%
Tree Depth: 10
Total Number of Tree Nodes: 197305
```



Gambar Input



Gambar Output

LAMPIRAN

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	✓	

Link Repository: https://github.com/mraifalkautsar/Tucil2_13523011