

LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA

PENERAPAN *DIVIDE AND CONQUER* DALAM
KOMPRESI GAMBAR DENGAN METODE QUADTREE



DISUSUN OLEH:

Asybel Bintang Pardomuan Sianipar 15223011

Ignacio Kevin Alberiann 15223090

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

DAFTAR ISI.....	2
1. DESKRIPSI MASALAH.....	3
2. TEORI SINGKAT.....	6
2.1. Divide and Conquer.....	6
2.1.1. Divide.....	7
2.1.2. Conquer.....	7
2.1.3. Combine/Merge.....	7
2.2. Quadtree.....	8
2.3. Metode Pengukuran Error.....	9
2.3.1 Varians.....	9
2.3.2. Mean Absolute Deviation (MAD).....	10
2.3.3. Max Pixel Difference.....	10
2.3.4. Entropi.....	11
2.3.5. Structural Similarity Index Measure (SSIM).....	11
3. ALGORITMA PROGRAM.....	14
3.1. DIVIDE.....	14
3.2. CONQUER.....	16
3.3. COMBINE / MERGE.....	17
4. IMPLEMENTASI PROGRAM.....	21
5. IMPLEMENTASI BONUS.....	38
5.1. Structural Similarity Index Measure (SSIM).....	38
5.2. GIF (Graphics Interchange Format).....	39
6. PENGUJIAN DAN ANALISIS.....	40
6.1. Pengujian Program.....	40
6.2. Analisis Algoritma Quad Tree.....	46
7. KESIMPULAN DAN SARAN.....	48
7.1. Kesimpulan.....	48
7.2. Saran.....	48
LAMPIRAN.....	50
1. Tautan Repository Github.....	50
2. Tabel Ketercapaian.....	50
DAFTAR PUSTAKA.....	51

1. DESKRIPSI MASALAH

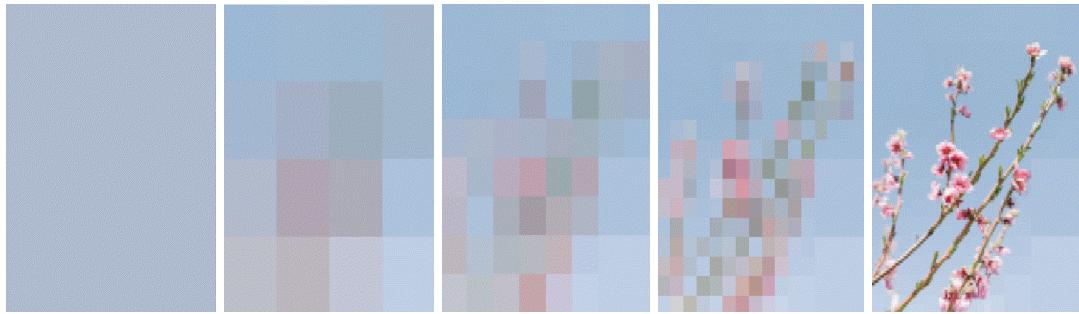


Gambar 1.1. Quadtree dalam kompresi gambar

(Sumber: <https://medium.com/@tannerwykoff/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 1.2. Proses Pembentukan Quadtree dalam Kompresi Gambar (.gif)

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

Pada Tugas Kecil 2 ini, mahasiswa diminta untuk membuat program sederhana dalam bahasa **C/C#/C++/Java** (CLI) yang mengimplementasikan **algoritma divide and conquer** untuk melakukan kompresi gambar berbasis quadtree yang mengimplementasikan **seluruh parameter** sebagai *user input*.

Pengguna menentukan parameter yang disebutkan berikut:

- a. **Metode pengukuran error:** metode yang digunakan untuk menentukan seberapa besar perbedaan dalam satu blok gambar. Jika error dalam blok melebihi ambang batas (*threshold*), maka blok akan dibagi menjadi empat bagian yang lebih kecil. Metode-metode tersebut adalah: **Variance**, **Mean Absolute Deviation (MAD)**, **Max Pixel Difference**, **Entropy**, dan **SSIM (metode bonus)**.
- b. **Threshold (ambang batas):** nilai batas yang menentukan apakah sebuah blok dianggap cukup seragam untuk disimpan atau harus dibagi lebih lanjut.
- c. **Minimum block size (ukuran blok minimum):** ukuran terkecil dari sebuah blok yang diizinkan dalam proses kompresi. Jika ukuran blok yang akan dibagi menjadi empat sub-blok berada di bawah ukuran minimum yang telah dikonfigurasi, maka blok tersebut tidak akan dibagi lebih lanjut, meskipun error masih di atas *threshold*.

Alur program untuk Tugas Kecil 2 ini adalah:

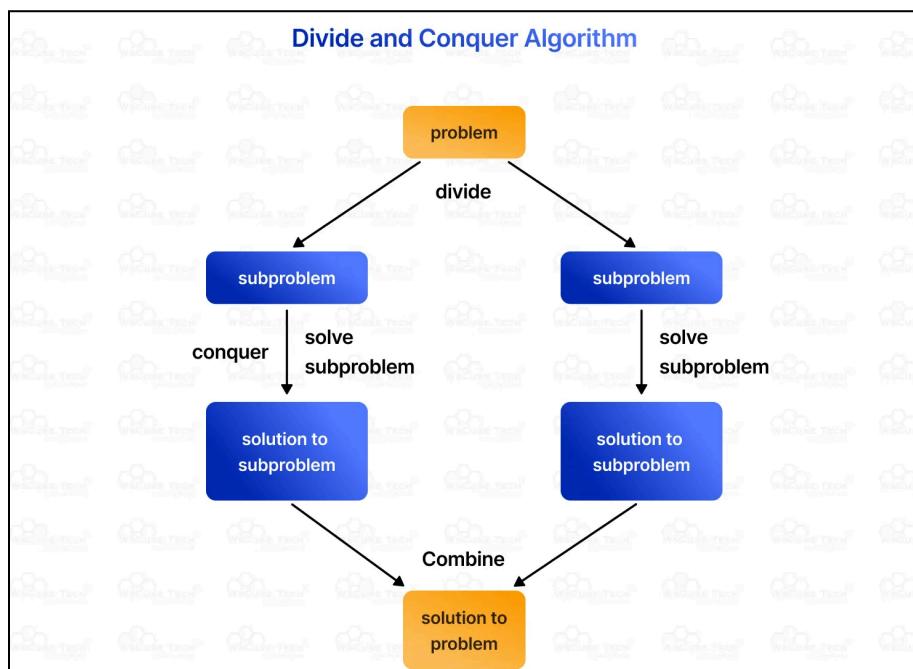
1. [INPUT] **alamat absolut** gambar yang akan dikompresi.
2. [INPUT] metode perhitungan error (gunakan penomoran sebagai input).
3. [INPUT] ambang batas (pastikan range nilai sesuai dengan metode yang dipilih).
4. [INPUT] ukuran blok minimum.

5. [INPUT] **Alamat absolut** gambar hasil kompresi.
6. [INPUT] **Alamat absolut GIF**.
7. [OUTPUT] Waktu eksekusi.
8. [OUTPUT] Ukuran gambar sebelum.
9. [OUTPUT] Ukuran gambar setelah.
10. [OUTPUT] Persentase kompresi.
11. [OUTPUT] Kedalaman pohon.
12. [OUTPUT] Banyak simpul pada pohon.
13. [OUTPUT] Gambar hasil kompresi pada alamat yang sudah ditentukan.
14. [OUTPUT] GIF proses kompresi pada alamat yang sudah ditentukan (bonus).

2. TEORI SINGKAT

2.1. Divide and Conquer

Dalam konteks ilmu informatika, *divide and conquer* adalah sebuah algoritma yang digunakan untuk menyelesaikan masalah besar dengan memecahnya menjadi bagian-bagian yang lebih kecil dan mudah diselesaikan. Sederhananya, algoritma ini **membagi (divide)** masalah menjadi bagian-bagian kecil (yang dinamakan pula sebagai upa-masalah), **menyelesaikan/menguasai/menaklukkan (conquer)** setiap bagian masalah tersebut menjadi solusi, lalu **menggabungkannya (combine/merge)** untuk mendapatkan solusi utama akhir.



Gambar 2.1.1. Proses Algoritma Divide and Conquer
(Sumber: <https://www.wscubetech.com/resources/dsa/divide-and-conquer-algorithm>)

Setiap upa-masalah memiliki karakteristik yang sama dengan karakteristik masalah semula, hanya saja berukuran lebih kecil. Oleh karena tiap upa-masalah dapat dibagi kembali menjadi upa-masalah lainnya yang lebih kecil, maka algoritma ini lebih *natural* direpresentasikan dengan skema rekursif.

Kompleksitas waktu $T(n)$ algoritma *divide and conquer* didapatkan melalui fungsi *piecewise* berikut:

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \dots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

Dimana:

$T(n)$	= Kompleksitas waktu penyelesaian permasalahan yang berukuran n
$g(n)$	= Kompleksitas waktu untuk menyelesaikan permasalahan jika n sudah cukup kecil
$f(n)$	= Kompleksitas waktu untuk menggabungkan solusi dari setiap upa-masalah
$T(n_1) + T(n_2) + \dots + T(n_r)$	= Kompleksitas waktu untuk memproses setiap upa-masalah

2.1.1. Divide

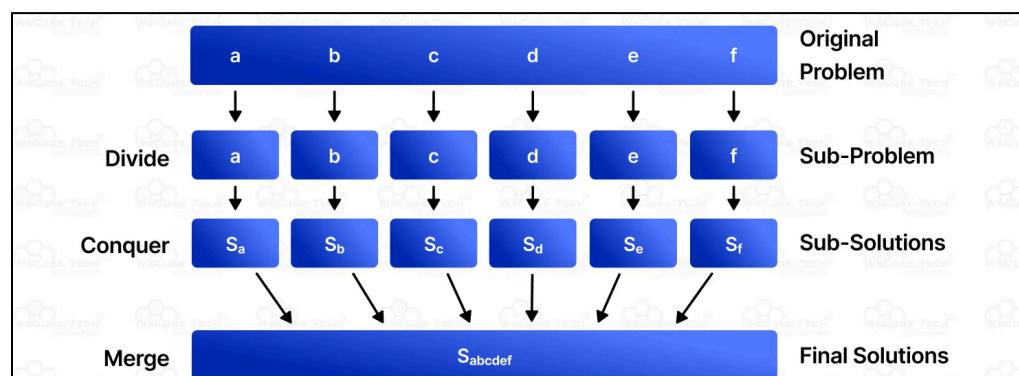
Langkah pertama adalah membagi/memecah (*divide*) masalah yang besar menjadi upa-masalah atau sub-masalah yang lebih kecil dan mudah dikelola. Proses “pembagian” ini berlanjut hingga upa-masalah tersebut dilihat sebagai masalah yang cukup sederhana untuk dapat dipecahkan secara langsung, seringkali ketika mencapai kasus dasar, seperti elemen tunggal atau kumpulan data yang lebih kecil.

2.1.2. Conquer

Langkah kedua adalah menyelesaikan setiap upa-masalah yang lebih kecil. Langkah ini biasanya melibatkan penyelesaian setiap upa-masalah secara individual. Bergantung pada algoritmanya, langkah ini dapat melibatkan rekursi, di mana setiap upa-masalah dipecahkan menggunakan pendekatan *divide and conquer* yang sama, atau melalui komputasi langsung sederhana.

2.1.3. Combine/Merge

Pada langkah terakhir, setelah semua upa-masalah diselesaikan menjadi upa-solusi, seluruh upa-solusi tersebut digabungkan untuk menjadi solusi dari masalah semula yang lebih besar.

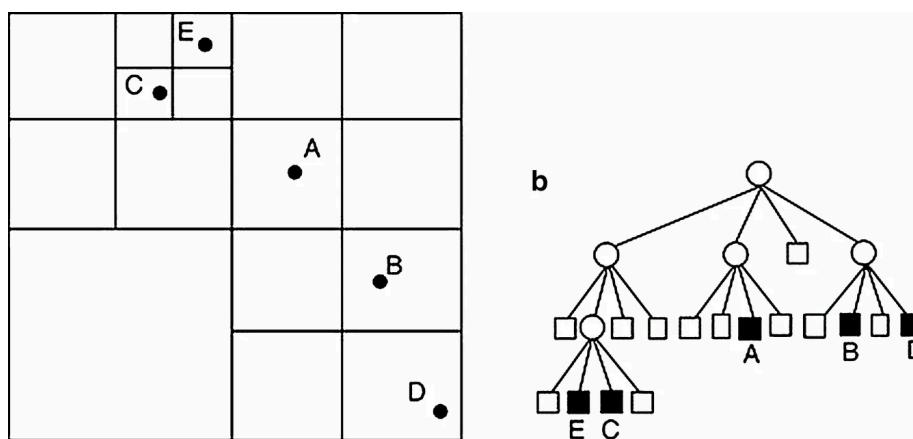


Gambar 2.1.2. Proses Penyelesaian Masalah dengan Algoritma Divide and Conquer
(Sumber: <https://www.wscubetech.com/resources/dsa/divide-and-conquer-algorithm>)

2.2. Quadtree

Quadtree adalah sebuah struktur data berbentuk pohon/tree yang umum digunakan untuk membagi ruang dua dimensi menjadi ruang yang lebih kecil. *Quadtree* mengambil kata “*Quad*” karena setiap simpul anaknya berjumlah empat, sesuai dengan empat kuadran ruang.

Quadtree dimulai dari simpul akar yang mewakili seluruh ruang. Setiap tingkat pohon membagi ruang menjadi empat kuadran yang berukuran sama. Simpul daun mewakili ruang yang lebih kecil tersebut.



Gambar 2.2.1. Struktur Data Quadtree dalam Kompresi Gambar

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

Dalam konteks kompresi gambar, *quadtree* adalah struktur data yang membagi gambar secara rekursif menjadi empat bagian berdasarkan keseragaman warna. Setiap bagian menyimpan rata-rata warna RGB dan nilai error yang menunjukkan seberapa representatif warna tersebut untuk sub-bagiannya. Dengan menggunakan ambang batas error sebagai acuan, *quadtree* dapat menentukan apakah suatu bagian perlu dibagi lagi atau tidak. Pendekatan ini memungkinkan kompresi gambar yang efisien tanpa mengorbankan kualitas, terutama untuk gambar dengan area warna yang luas dan seragam.

Selain untuk kompresi, *quadtree* juga bermanfaat dalam mempercepat proses deteksi tepi pada gambar. Karena fokus pembagian berada pada area yang lebih detail, struktur ini memungkinkan algoritma hanya memproses bagian-bagian penting dari gambar, yaitu node daun dan induknya. Hal ini membuat proses deteksi tepi menjadi lebih cepat dan efisien dibandingkan memproses seluruh gambar secara menyeluruh.

2.3. Metode Pengukuran Error

Dalam proses kompresi gambar berbasis Quadtree, metode pengukuran error pada tugas ini digunakan untuk mengevaluasi tingkat homogenitas warna suatu blok piksel gambar. Tujuan utama dari metode ini adalah menentukan apakah suatu blok gambar cukup seragam atau berbeda sehingga blok perlu dibagi kembali menjadi blok-blok yang lebih kecil. Proses ini membuat kompresi menjadi lebih efisien dengan tetap menjaga kualitas gambar.

Setiap blok gambar akan dihitung besar nilai error-nya berdasarkan salah satu metode pengukuran error yang ditentukan. Apabila nilai error pada suatu blok melebihi ambang batas (*threshold*) yang ditetapkan, maka blok tersebut dianggap tidak seragam (memiliki variasi yang terlalu besar jika direpresentasikan secara homogen). Blok yang tidak dianggap seragam akan dibagi menjadi empat bagian kecil sesuai dengan prinsip dari Quadtree itu sendiri.

Sebaliknya, jika nilai error pada suatu blok tidak melebihi ambang batas (*threshold*), maka blok dianggap cukup seragam dan dapat direpresentasikan sebagai satu blok tanpa perlu dibagi lagi. Metode ini memungkinkan penghematan dalam jumlah simpul (node) Quadtree yang ada, serta mengurangi ukuran data hasil kompresi, yang merupakan salah satu tujuan dari proses kompresi gambar.

Metode pengukuran error yang digunakan pada tugas ini berdasarkan spesifikasi yang ada (*Variance*, *Mean Absolute Deviation*, *Max Pixel Difference*, dan *Entropy*), dengan penambahan metode *Structural Similarity Index Measure* (SSIM) sebagai bonus.

2.3.1 Varians

Untuk mengukur nilai varians per kanal warna dari sampel gambar, digunakan rumus berikut:

$$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$$

Dimana:

σ_c^2 = Variansi tiap kanal warna c (R, G, B) dalam satu blok

$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c

μ_c = Nilai rata-rata tiap piksel dalam satu blok

N = Banyaknya piksel dalam satu blok

Nilai varians dihitung untuk seluruh kanal warna (R, G, B), lalu digabung dengan bobot yang sama sesuai rumus berikut:

$$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$$

2.3.2. Mean Absolute Deviation (MAD)

Untuk mengukur nilai MAD per kanal warna dari sampel gambar, digunakan rumus berikut:

$$MAD_c = \frac{1}{N} \sum_{i=1}^N |P_{i,c} - \mu_c|$$

Dimana:

MAD_c = Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok

$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c

μ_c = Nilai rata-rata tiap piksel dalam satu blok

N = Banyaknya piksel dalam satu blok

Nilai MAD dihitung untuk seluruh kanal warna (R, G, B), lalu digabung dengan bobot yang sama sesuai rumus berikut:

$$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$$

2.3.3. Max Pixel Difference

Untuk mengukur nilai Max Pixel Difference per kanal warna dari sampel gambar, digunakan rumus berikut:

$$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$$

Dimana:

D_c = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok

$P_{i,c}$ = Nilai piksel pada posisi i untuk channel warna c

Nilai Max Pixel Difference dihitung untuk seluruh kanal warna (R, G, B), lalu digabung dengan bobot yang sama sesuai rumus berikut:

$$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$$

2.3.4. Entropi

Untuk mengukur nilai entropi per kanal warna dari sampel gambar, digunakan rumus berikut:

$$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$$

Dimana:

H_c	= Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok	
$P_c(i)$	= Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)	

Nilai entropi dihitung untuk seluruh kanal warna (R, G, B), lalu digabung dengan bobot yang sama sesuai rumus berikut:

$$H_{RGB} = \frac{H_R + H_G + H_B}{3}$$

2.3.5. Structural Similarity Index Measure (SSIM)

SSIM adalah metode untuk mengukur seberapa mirip dua gambar, terutama dari sisi persepsi manusia. Tidak hanya sekadar menghitung perbedaan warna piksel demi piksel (seperti PSNR atau MSE), tapi memperhatikan struktur gambar, kecerahan, dan kontras, yang merupakan aspek-aspek yang lebih dekat dengan bagaimana manusia melihat gambar. SSIM mengukur tiga hal, yaitu: **Luminance** (kecerahan gambar), **Contrast** (perbedaan antara terang dan gelap), dan **Structure** (pola atau bentuk visual). SSIM akan memberikan hasil dalam rentang nilai 0.0 - 1.0 di mana 0.0 artinya sangat berbeda dan 1.0 artinya kedua gambar yang dikomparasi identik sempurna.

Nilai SSIM yang dibandingkan adalah antara blok gambar sebelum dan sesudah dikompresi. Untuk mengukur nilai entropi per kanal warna dari sampel gambar, digunakan rumus berikut:

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

Dimana:

$\mu_{x,c}$	= Rata-rata nilai piksel di blok asli tiap kanal warna c pada satu blok
$\mu_{y,c}$	= Rata-rata nilai piksel di blok kompresi tiap kanal warna c pada satu blok
C_1	= Konstanta stabilitas 1

- C_2 = Konstanta stabilitas 2
 $\sigma_{xy,c}$ = Kovarians antara blok asli dengan blok yang sedang dikompresi
 $\sigma_{x,c}^2$ = Variansi nilai piksel di blok asli tiap kanal warna c pada satu blok
 $\sigma_{y,c}^2$ = Variansi nilai piksel di blok kompresi tiap kanal warna c pada satu blok

Nilai entropi dihitung untuk seluruh kanal warna (R, G, B), lalu digabung dengan bobot yang sama sesuai rumus berikut:

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

2.4. Range Threshold Tiap Metode Pengukuran Error

1. Varians

- Min Threshold: 0, terjadi ketika semua piksel dalam blok memiliki nilai yang sama (tidak ada variasi, misalnya, semua piksel bernilai 255).
- Max Threshold: $(255)^2 = 65.025$, terjadi ketika piksel dalam blok memiliki variasi maksimum, misalnya, setengah piksel bernilai 0 dan setengah lainnya bernilai 255 (untuk gambar 8-bit).

Dalam program, nilai 5000 dipilih sebagai titik tengah dalam rentang praktis penggunaan varians (0 hingga 10.000) untuk menjaga keseimbangan antara pembagian blok dan efisiensi kompresi. Nilai ini memungkinkan blok dengan variasi sedang hingga tinggi (misalnya, tekstur atau tepi) untuk dibagi lebih lanjut, sementara blok yang hampir seragam (varians < 5000) dihindarkan pembagiannya

2. MAD (Mean Absolute Deviation)

- Min Threshold: 0, terjadi ketika semua piksel dalam blok memiliki nilai yang sama (tidak ada deviasi).
- Max Threshold: 127.5, terjadi ketika piksel dalam blok memiliki deviasi maksimum, misalnya, setengah piksel bernilai 0 dan setengah lainnya bernilai 255 (rata-rata = 127.5, deviasi absolut maksimum = 127.5).

Nilai 50 dipilih sebagai batas maksimum dalam program, yang berarti hanya blok dengan deviasi sangat tinggi (mendekati maksimum teoretis) yang akan dibagi lebih lanjut.

3. MPD (Mean Pixel Difference)

- Min Threshold: 0, terjadi ketika semua piksel dalam blok memiliki nilai yang sama (tidak ada perbedaan antar piksel tetangga).
- Max Threshold: 255, terjadi ketika piksel tetangga memiliki perbedaan maksimum, misalnya, piksel bergantian antara 0 dan 255 dalam pola catur.

Nilai 100 dipilih sebagai batas maksimum dalam program, yang berarti hanya blok dengan perbedaan piksel tetangga yang sangat besar (mendekati maksimum teoretis) yang akan dibagi lebih lanjut. Ini mirip dengan pendekatan MAD, di mana fokusnya adalah efisiensi kompresi dengan meminimalkan pembagian, kecuali pada blok dengan kontras lokal ekstrem (misalnya, pola catur).

4. Entropi

- Min Threshold: 0, terjadi ketika semua piksel dalam blok memiliki intensitas yang sama (hanya satu nilai dalam histogram, $p_i = 1$, $\log_2(1) = 0$).
- Max Threshold: $\log_2(K)$, di mana K adalah jumlah intensitas yang mungkin (untuk gambar 8-bit, K = 256, maka maksimum = $\log_2(256) = 8$), terjadi ketika semua intensitas terdistribusi secara seragam ($p_i = 1/256$ untuk setiap i).

Nilai 7 dipilih sebagai *max threshold* dalam program karena berada di dekat batas atas rentang teoritis, yang berarti hanya blok dengan keragaman intensitas sangat tinggi (mendekati distribusi seragam) yang akan dibagi lebih lanjut.

5. SSIM (Structural Similarity Index)

- Min Threshold: -1, terjadi ketika dua blok sangat berbeda (misalnya, satu blok semua piksel 0, yang lain semua piksel 255, dengan konstanta C_1 dan C_2 mendekati 0).
- Max Threshold: 1, terjadi ketika dua blok identik (misalnya, blok asli dan blok representasi memiliki piksel yang sama persis).

Nilai 0.5 untuk $1 - \text{SSIM}$ (atau $\text{SSIM} = 0.5$) dipilih karena SSIM di bawah 0.5 menunjukkan perbedaan struktural yang cukup signifikan secara visual antara blok asli dan blok yang direpresentasikan (dengan warna rata-rata). Ambang batas ini memastikan bahwa blok dengan kesamaan rendah ($\text{SSIM} < 0.5$) akan dibagi lebih lanjut untuk mempertahankan kualitas visual, sementara blok dengan kesamaan lebih tinggi ($\text{SSIM} \geq 0.5$) dihentikan pembagiannya, menyeimbangkan kualitas dan efisiensi kompresi.

3. ALGORITMA PROGRAM

Pada bab ini dijelaskan secara rinci alur kerja algoritma **Quadtree** dengan pendekatan **Divide and Conquer** dalam program kompresi gambar. Penjelasan dibagi menjadi tiga tahap utama: **Divide**, **Conquer**, dan **Combine/Merge**.

3.1. DIVIDE

Tahap Divide adalah langkah awal dalam pendekatan Divide and Conquer, ketika gambar input dibagi menjadi sub-blok yang lebih kecil secara rekursif menggunakan struktur Quadtree. Tujuannya adalah untuk mengidentifikasi area homogen dalam gambar yang dapat direpresentasikan dengan satu warna sehingga mengurangi kompleksitas data dan memungkinkan kompresi yang efisien.

1. Inisialisasi Root Node

Program memulai dengan membuat satu *Node* (simpul) yang mewakili keseluruhan gambar sebagai *root* dari Quadtree. Node ini menyimpan informasi berikut:

- **Koordinat kiri-atas:** ($x=0, y=0$), yang menunjukkan posisi awal blok pada gambar.
- **Ukuran blok:** Lebar (*width*) dan tinggi (*height*) sesuai dengan dimensi gambar input (misalnya, 512×512 piksel).
- **Data piksel:** Representasi piksel dalam bentuk matriks tiga dimensi ($\text{byte}[:, :, :]$), di mana setiap elemen menyimpan nilai RGB (Red, Green, Blue) untuk piksel pada posisi tertentu.

Root node ini menjadi titik awal untuk proses pembagian secara rekursif.

2. Pengukuran Error

Untuk menentukan apakah sebuah blok perlu dibagi lebih lanjut, program menghitung nilai *error* yang mengukur tingkat homogenitas blok. Pengguna dapat memilih salah satu dari lima metode perhitungan error berikut:

- Variance
- Mean Absolute Deviation (MAD)
- Max Pixel Difference
- Entropi
- Structural Similarity Index Measure (SSIM)

Nilai error ini dibandingkan dengan *threshold* yang ditentukan pengguna (atau dihitung secara otomatis jika target kompresi diberikan).

3. Kriteria Pembagian

Sebuah blok akan dibagi lebih lanjut menjadi empat sub-blok jika memenuhi semua kondisi berikut:

1. **Error \geq Threshold:** Nilai error yang dihitung melebihi *threshold*, menunjukkan bahwa blok belum cukup homogen untuk direpresentasikan dengan satu warna.
2. **Ukuran Sub-Blok Masih Di Atas Minimum:** Setelah pembagian, ukuran sub-blok (area blok-induk / 4) harus lebih besar dari *Minimum Block Size*. Secara matematis, kondisi ini diperiksa sebagai:

Kondisi ini memastikan bahwa Quadtree tidak menghasilkan blok yang terlalu kecil, yang dapat meningkatkan jumlah *node* dan mengurangi efisiensi kompresi, terutama untuk format PNG.

4. Penentuan Sub-Blok

Jika kriteria pembagian terpenuhi, blok dipecah menjadi empat kuadran (sub-blok) dengan koordinat dan ukuran sebagai berikut::

1. Top-Left: $(x, y, \lfloor W/2 \rfloor, \lfloor H/2 \rfloor)$
2. Top-Right: $(x+\lfloor W/2 \rfloor, y, W-\lfloor W/2 \rfloor, \lfloor H/2 \rfloor)$
3. Bottom-Left: $(x, y+\lfloor H/2 \rfloor, \lfloor W/2 \rfloor, H-\lfloor H/2 \rfloor)$
4. Bottom-Right: $(x+\lfloor W/2 \rfloor, y+\lfloor H/2 \rfloor, W-\lfloor W/2 \rfloor, H-\lfloor H/2 \rfloor)$

Di mana:

- W adalah lebar blok saat ini (*width*).
- H adalah tinggi blok saat ini (*height*).
- $\lfloor \cdot \rfloor$ adalah operasi *floor* untuk membulatkan ke bawah ke bilangan bulat terdekat.

Pembagian ini memastikan bahwa keempat sub-blok mencakup seluruh area blok asli tanpa tumpang tindih.

5. Rekursi Pembagian

Setiap sub-blok yang dihasilkan kemudian diproses ulang secara rekursif, dimulai dari langkah pengukuran error. Proses ini berulang untuk setiap sub-blok hingga salah satu kondisi berikut terpenuhi:

- Error $< Threshold$, menandakan blok cukup homogen.
- Ukuran sub-blok setelah pembagian akan menjadi kurang dari atau sama dengan *Minimum Block Size*.

Proses rekursi ini adalah inti dari tahap *Divide*, yang memecah gambar menjadi bagian-bagian kecil yang lebih mudah dikelola.

3.2. CONQUER

Tahap *Conquer* berfokus pada penyelesaian upa-masalah yang telah dibagi pada tahap *Divide*. Dalam konteks Quadtree, ini berarti menentukan kapan pembagian harus dihentikan (*base case*) dan memproses setiap *leaf node* untuk menghasilkan representasi yang lebih sederhana (warna rata-rata).

1. Base Case

Pembagian rekursif dihentikan dan sebuah blok ditandai sebagai *leaf node* jika salah satu dari kondisi berikut terpenuhi:

1. **Blok Homogen:** Nilai error yang dihitung kurang dari *threshold* ($\text{Error} < \text{Threshold}$), menunjukkan bahwa blok memiliki variasi warna yang cukup kecil sehingga dapat direpresentasikan dengan satu warna.
2. **Ukuran Minimum Tercapai:** Ukuran sub-blok setelah pembagian akan menjadi kurang dari atau sama dengan *Minimum Block Size*, seperti yang dijelaskan pada kriteria pembagian.

Ketika salah satu kondisi ini terpenuhi, blok tersebut tidak dibagi lagi dan menjadi *leaf node* dalam struktur Quadtree.

2. Perhitungan Average Color

Untuk setiap *leaf node*, program menghitung *Average Color* (rata-rata nilai R, G, B) dari semua piksel dalam blok tersebut. Langkah-langkahnya adalah sebagai berikut:

1. Iterasi melalui semua piksel dalam blok.
2. Untuk setiap kanal warna (R, G, B), jumlahkan nilai piksel dan hitung rata-ratanya:

$$R_{avg} = \frac{\sum Ri}{\text{jumlah piksel}}, G_{avg} = \frac{\sum Gi}{\text{jumlah piksel}}, B_{avg} = \frac{\sum Bi}{\text{jumlah piksel}}$$

3. Simpan nilai rata-rata ini sebagai representasi warna untuk seluruh blok.
4. Warna rata-rata ini akan digunakan pada tahap *Combine* untuk mengisi blok pada gambar keluaran, sehingga mengurangi kompleksitas data dengan merepresentasikan seluruh blok homogen dengan satu warna.

3. Pengumpulan Statistik

Selama proses rekursi, program mencatat statistik penting untuk analisis performa dan efisiensi Quadtree:

1. **Tree Depth:** Kedalaman maksimum pohon Quadtree, yang dihitung sebagai jumlah level dari *root* ke *leaf node* terdalam. Kedalaman ini mencerminkan tingkat detail pembagian gambar.
 - o Misalnya, untuk gambar 512×512 dengan *Minimum Block Size* = 64, kedalaman maksimum adalah 3 ($512 \rightarrow 256 \rightarrow 128 \rightarrow 64$).
2. **Node Count:** Jumlah total *node* (baik *leaf node* maupun *internal node*) dalam Quadtree. Jumlah ini mencerminkan kompleksitas struktur Quadtree.
 - o Contoh: Untuk *Minimum Block Size* = 64 pada gambar 512×512 , total *node* ≈ 21 ($1 + 4 + 16$).

Statistik ini dicatat selama pembangunan Quadtree dan akan ditampilkan pada ringkasan akhir.

3.3. COMBINE / MERGE

Tahap *Combine* atau *Merge* adalah langkah terakhir dalam pendekatan *Divide and Conquer*, di mana hasil dari sub-masalah (representasi warna pada *leaf node*) digabungkan untuk membentuk gambar terkompresi. Selain itu, program juga menghasilkan output tambahan seperti animasi GIF (opsional) dan statistik performa.

1. Rekonstruksi Gambar

Setelah Quadtree selesai dibangun, program melakukan *traversal* (penelusuran) pada struktur Quadtree untuk merekonstruksi gambar:

1. **Traversal ke Leaf Node:** Program mengunjungi setiap *leaf node* dalam Quadtree secara rekursif.
2. **Pengisian Warna:** Untuk setiap *leaf node*, program mengisi semua piksel dalam blok yang sesuai pada gambar keluaran dengan *Average Color* yang telah dihitung pada tahap *Conquer*.
 - o Misalnya, jika sebuah *leaf node* mewakili blok 64×64 piksel dengan *Average Color* (R=100, G=150, B=200), maka semua 64×64 piksel dalam blok tersebut diisi dengan warna tersebut.
3. **Pemutakhiran Matriks Piksel:** Matriks piksel (byte[,]) yang mewakili gambar keluaran diperbarui dengan nilai warna baru.

Proses ini menghasilkan gambar terkompresi di mana area homogen direpresentasikan dengan warna rata-rata, sehingga mengurangi detail yang tidak signifikan.

2. Representasi Blok Homogen

Karena setiap *leaf node* (blok homogen) diwakili oleh satu warna rata-rata, area seragam dalam gambar asli menjadi sangat efisien dalam hal penyimpanan:

- Dalam gambar asli, setiap piksel menyimpan nilai R, G, B (3 byte per piksel). Untuk blok 64×64 , ini berarti $64 \times 64 \times 3 = 12.288$ byte.
- Setelah kompresi, blok tersebut hanya direpresentasikan oleh satu warna (3 byte), ditambah overhead struktur Quadtree.
- Efisiensi ini sangat signifikan untuk format seperti JPEG, yang menggunakan kompresi lossy, dan PNG, yang efisien untuk area seragam.

Namun, untuk format PNG, efisiensi dapat berkurang jika terdapat banyak *leaf node* kecil dengan batas warna yang tajam, karena PNG menggunakan algoritma DEFLATE yang kurang efisien untuk transisi warna yang tajam. Untuk mengatasi ini, program menyesuaikan *Minimum Block Size* (misalnya, minimum 64 untuk PNG) guna mengurangi jumlah *node* dan menciptakan blok yang lebih besar.

Karena setiap blok homogen diwakili satu warna, area seragam dalam gambar asli menjadi efisien secara penyimpanan.

3. Ekspor Hasil

Setelah rekonstruksi selesai, gambar terkompresi dieksport ke lokasi yang ditentukan pengguna:

1. Format Output:

- Jika gambar input adalah JPEG, output disimpan sebagai JPEG dengan kualitas yang dioptimalkan (disesuaikan melalui pencarian biner untuk mencapai target kompresi).
- Jika gambar input adalah PNG, output disimpan sebagai PNG menggunakan *ImageMagick* dengan pengaturan optimal (*Quality = 9*, *MagickFormat.Png8*, dan *Strip()* untuk menghapus metadata).

2. Animasi GIF (Opsiional):

- Jika pengguna mengaktifkan fitur GIF, program merekam setiap tahap pengisian blok sebagai *frame* selama proses rekonstruksi.
- Frame-frame ini kemudian disusun menjadi animasi GIF yang menunjukkan transformasi dari gambar asli ke gambar terkompresi, memberikan visualisasi proses Quadtree.

4. Statistik Program

Di akhir proses, program menampilkan ringkasan hasil untuk memberikan gambaran performa dan efisiensi kompresi:

1. Execution Time:

- Waktu eksekusi untuk setiap tahap utama, diukur dalam milidetik:
 - *Input Processing*: Waktu untuk membaca dan mengonversi gambar input ke matriks piksel.
 - *Finding Optimal Threshold*: Waktu untuk mencari *threshold* dan kualitas JPEG (jika berlaku).
 - *Building Quadtree*: Waktu untuk membangun struktur Quadtree.
 - *Image Reconstruction*: Waktu untuk merekonstruksi gambar.
 - *Save to GIF*: Waktu untuk membuat animasi GIF (jika diaktifkan).

2. Original vs Compressed Size:

- Ukuran file gambar asli dan gambar terkompresi, dalam kilobyte (KB).

3. Compression Percentage:

- Persentase kompresi yang dicapai, dihitung sebagai:

$$\text{Compression Percentage} = \left(1 - \frac{\text{Compressed Size}}{\text{Original Size}}\right) \times 100$$

4. Tree Depth:

Kedalaman maksimum Quadtree, mencerminkan tingkat detail pembagian.

5. Node Count:

Jumlah total *node* dalam Quadtree, mencerminkan kompleksitas struktur.

6. Output Paths:

- Lokasi absolut gambar terkompresi (wajib).
- Lokasi absolut animasi GIF (opsional, jika diaktifkan).

Contoh output statistik:

```
----- Program Statistics -----
- Execution Time
  - Input Processing: 38.74 ms
  - Finding Optimal Threshold: 5073.15 ms
  - Building Quadtree: 83.05 ms
  - Image Reconstruction: 155.39 ms
  - Save to GIF: 3372.60 ms

- Previous Image Size: 38.124 KB
- Compressed Image Size: 7.625 KB
- Compression Percentage: 80.00%
- Quadtree Total Depth: 3
- Node Count: 21
- Output Image Path: C:\Users\balse\Documents\VScode\Tucil2_Stima
master\Tucil2_Stima\test\test7_compressed.png
- Output GIF Path: C:\Users\balse\Documents\VScode\Tucil2_Stima
master\Tucil2_Stima\test\test7_compressed_transform.gif
```

Gambar 3.1 Statistik Program
(Sumber: arsip penulis)

4. IMPLEMENTASI PROGRAM

Program diimplementasikan dalam bahasa pemrograman C# versi .NET 9.0. Paradigma pemrograman yang digunakan adalah paradigma berorientasi objek. Seluruh program kompresi gambar berada dalam proyek bernama **QuadtreeCompression**. Program ini menggunakan pustaka eksternal untuk mendukung pemrosesan gambar dan pembuatan file GIF. Pustaka yang digunakan adalah **System.Drawing.Common** untuk manipulasi gambar dasar dan **Magick.NET-Q16-AnyCPU** untuk pemrosesan gambar tingkat lanjut serta pembuatan GIF.

```
Tucil2_STIMA/
└── QuadtreeCompression/
    ├── Compressor.cs
    ├── InputHandler.cs
    ├── Metrics.cs
    ├── Node.cs
    ├── OutputHandler.cs
    ├── Program.cs
    ├── Quadtree.cs
    └── QuadtreeCompression.csproj

    └── test/
        └── QuadtreeCompression.sln
```

Program.cs

```
1  using System;
2
3  namespace QuadtreeCompression
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              // Initialize handlers
10             InputHandler inputHandler = new InputHandler();
11             OutputHandler outputHandler = new OutputHandler();
12             Compressor compressor = new Compressor(outputHandler);
13
14             // Gather inputs
15             string inputImagePath = inputHandler.GetImagePath();
16             string errorMethod = inputHandler.GetErrorMethod();
17             double threshold = inputHandler.GetThreshold(errorMethod);
18             int minBlockSize = inputHandler.GetMinBlockSize();
19             string outputImagePath = inputHandler.GetOutputImagePath(inputImagePath);
20             string outputGifPath = inputHandler.GetOutputGifPath(inputImagePath);
21
22             // Ask if the user wants to create the GIF
23             Console.WriteLine("Create quadtree-depth GIF? (y/N): ");
24             string? key = Console.ReadLine()?.Trim().ToLowerInvariant();
25             bool createGif = key == "y" || key == "yes";
26
27             // Run the compression
28             compressor.Compress(
29                 inputImagePath,
30                 errorMethod,
31                 threshold,
32                 minBlockSize,
33                 outputImagePath,
34                 outputGifPath,
35                 createGif);
36
37             //Display statistics
38             outputHandler.SetOutputPaths(outputImagePath, createGif ? outputGifPath : null);
39             outputHandler.DisplayStatistics();
40         }
41     }
42 }
```

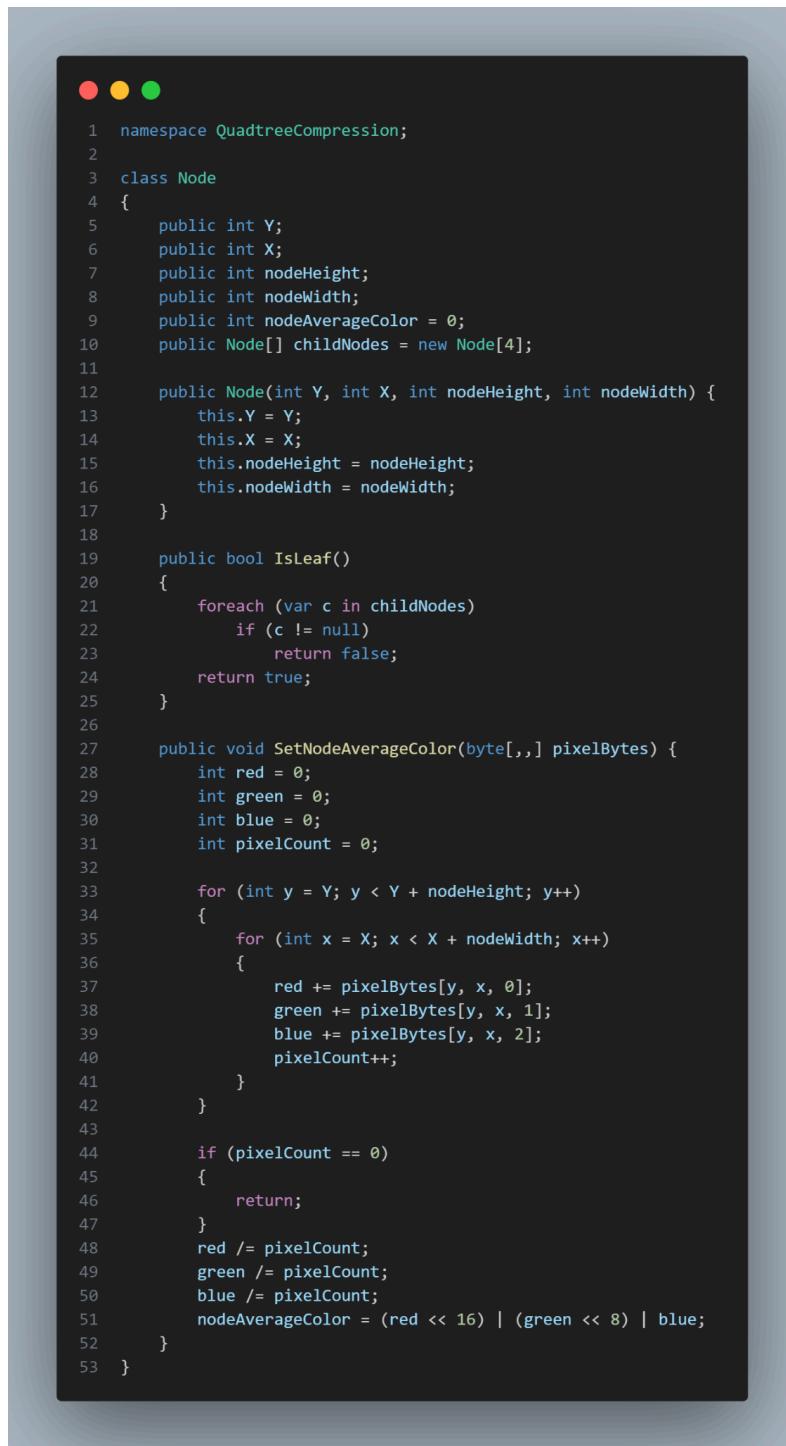
Gambar 4.1. Kelas Program (main entry)
(Sumber: arsip penulis)

Quadtree.cs

```
1  namespace QuadtreeCompression
2  {
3      internal class Compressor
4      {
5          private readonly OutputHandler outputHandler;
6
7          public Compressor(OutputHandler outputHandler)
8          {
9              this.outputHandler = outputHandler;
10         }
11
12         public void Compress(
13             string inputImagePath,
14             string errorMethod,
15             double threshold,
16             int minBlockSize,
17             string outputImagePath,
18             string outputGifPath,
19             bool createGif)
20         {
21             // Time "Input Processing"
22             outputHandler.StartTiming("Input Processing");
23             byte[,] pixelMatrix = InputHandler.ImageToPixelBytes(inputImagePath);
24             if (pixelMatrix.Length == 0)
25             {
26                 outputHandler.StopTiming("Input Processing");
27                 throw new Exception("Failed to load image.");
28             }
29             int height = pixelMatrix.GetLength(0);
30             int width = pixelMatrix.GetLength(1);
31             outputHandler.StopTiming("Input Processing");
32
33             // Get original image size
34             long originalSize = new FileInfo(inputImagePath).Length;
35
36             // Time "Building Quadtree"
37             outputHandler.StartTiming("Building Quadtree");
38             Quadtree quadtree = new Quadtree(pixelMatrix, errorMethod, threshold, minBlockSize);
39             outputHandler.StopTiming("Building Quadtree");
40
41             outputHandler.SetQuadtreeStats(quadtree.GetMaxDepth(), quadtree.GetNodeCount());
42
43             // Time "Image Reconstruction"
44             outputHandler.StartTiming("Image Reconstruction");
45             quadtree.ReconstructImage(pixelMatrix);
46             outputHandler.SaveImage(pixelMatrix, height, width, inputImagePath, outputImagePath);
47             outputHandler.StopTiming("Image Reconstruction");
48             Console.WriteLine("Image is saved successfully.");
49
50             // Time "Save to GIF" if requested
51             if (createGif)
52             {
53                 outputHandler.StartTiming("Save to GIF");
54                 outputHandler.CreateTransformationGif(quadtree, pixelMatrix, height, width, outputGifPath);
55                 outputHandler.StopTiming("Save to GIF");
56                 Console.WriteLine("GIF is saved successfully.");
57             }
58
59             // Set image sizes for statistics
60             long compressedSize = new FileInfo(outputImagePath).Length;
61             outputHandler.SetImageSizes(originalSize, compressedSize);
62         }
63     }
64 }
```

Gambar 4.2. Kelas Quadtree
(Sumber: arsip penulis)

Node.cs



```
1  namespace QuadtreeCompression;
2
3  class Node
4  {
5      public int Y;
6      public int X;
7      public int nodeHeight;
8      public int nodeWidth;
9      public int nodeAverageColor = 0;
10     public Node[] childNodes = new Node[4];
11
12     public Node(int Y, int X, int nodeHeight, int nodeWidth) {
13         this.Y = Y;
14         this.X = X;
15         this.nodeHeight = nodeHeight;
16         this.nodeWidth = nodeWidth;
17     }
18
19     public bool IsLeaf()
20     {
21         foreach (var c in childNodes)
22             if (c != null)
23                 return false;
24         return true;
25     }
26
27     public void SetNodeAverageColor(byte[,] pixelBytes) {
28         int red = 0;
29         int green = 0;
30         int blue = 0;
31         int pixelCount = 0;
32
33         for (int y = Y; y < Y + nodeHeight; y++)
34         {
35             for (int x = X; x < X + nodeWidth; x++)
36             {
37                 red += pixelBytes[y, x, 0];
38                 green += pixelBytes[y, x, 1];
39                 blue += pixelBytes[y, x, 2];
40                 pixelCount++;
41             }
42         }
43
44         if (pixelCount == 0)
45         {
46             return;
47         }
48         red /= pixelCount;
49         green /= pixelCount;
50         blue /= pixelCount;
51         nodeAverageColor = (red << 16) | (green << 8) | blue;
52     }
53 }
```

Gambar 4.3. Kelas Node
(Sumber: arsip penulis)

Metrics.cs

```
1  using System.Runtime.InteropServices;
2
3  namespace QuadtreeCompression;
4
5  // ERROR MEASUREMENT METHOD
6  class Variance
7  {
8      public static double CalculateError(byte[, ,] pixelMatrix, int y, int x, int height, int width)
9      {
10         double sumR = 0, sumG = 0, sumB = 0;
11         int numPixels = height * width;
12
13         for (int i = y; i < y + height; i++)
14         {
15             for (int j = x; j < x + width; j++)
16             {
17                 sumR += pixelMatrix[i, j, 0];
18                 sumG += pixelMatrix[i, j, 1];
19                 sumB += pixelMatrix[i, j, 2];
20             }
21         }
22
23         int avgR = (int)(sumR / numPixels);
24         int avgG = (int)(sumG / numPixels);
25         int avgB = (int)(sumB / numPixels);
26
27         double varR = 0, varG = 0, varB = 0;
28         for (int i = y; i < y + height; i++)
29         {
30             for (int j = x; j < x + width; j++)
31             {
32                 varR += Math.Pow(pixelMatrix[i, j, 0] - avgR, 2);
33                 varG += Math.Pow(pixelMatrix[i, j, 1] - avgG, 2);
34                 varB += Math.Pow(pixelMatrix[i, j, 2] - avgB, 2);
35             }
36         }
37         varR /= numPixels;
38         varG /= numPixels;
39         varB /= numPixels;
40         return varR + varG + varB;
41     }
42 }
```

Gambar 4.4. Kelas Variance
(Sumber: arsip penulis)

```
● ○ ●
1 class MeanAbsoluteDeviation
2 {
3     public static double CalculateError(byte[,] pixelMatrix, int y, int x, int height, int width)
4     {
5         double sumR = 0, sumG = 0, sumB = 0;
6         int numPixels = height * width;
7
8         for (int i = y; i < y + height; i++)
9         {
10             for (int j = x; j < x + width; j++)
11             {
12                 sumR += pixelMatrix[i, j, 0];
13                 sumG += pixelMatrix[i, j, 1];
14                 sumB += pixelMatrix[i, j, 2];
15             }
16         }
17
18         int avgR = (int)(sumR / numPixels);
19         int avgG = (int)(sumG / numPixels);
20         int avgB = (int)(sumB / numPixels);
21
22         double madR = 0, madG = 0, madB = 0;
23         for (int i = y; i < y + height; i++)
24         {
25             for (int j = x; j < x + width; j++)
26             {
27                 madR += Math.Abs(pixelMatrix[i, j, 0] - avgR);
28                 madG += Math.Abs(pixelMatrix[i, j, 1] - avgG);
29                 madB += Math.Abs(pixelMatrix[i, j, 2] - avgB);
30             }
31         }
32         madR /= numPixels;
33         madG /= numPixels;
34         madB /= numPixels;
35         return madR + madG + madB;
36     }
37 }
```

Gambar 4.5. Kelas MeanAbsoluteDifference
(Sumber: arsip penulis)

```
1  class MaxPixelDifference
2  {
3      public static double CalculateError(byte[,] pixelMatrix, int y, int x, int height, int width)
4      {
5          double minR = 0, minG = 0, minB = 0;
6          double maxR = 0, maxG = 0, maxB = 0;
7          int numPixels = height * width;
8
9          for (int i = y; i < y + height; i++)
10         {
11             for (int j = x; j < x + width; j++)
12             {
13                 minR = Math.Min(minR, pixelMatrix[i, j, 0]);
14                 minG = Math.Min(minG, pixelMatrix[i, j, 1]);
15                 minB = Math.Min(minB, pixelMatrix[i, j, 2]);
16                 maxR = Math.Max(maxR, pixelMatrix[i, j, 0]);
17                 maxG = Math.Max(maxG, pixelMatrix[i, j, 1]);
18                 maxB = Math.Max(maxB, pixelMatrix[i, j, 2]);
19             }
20
21         }
22
23         int differenceR = (int)(maxR - minR);
24         int differenceG = (int)(maxG - minG);
25         int differenceB = (int)(maxB - minB);
26
27         differenceR /= numPixels;
28         differenceG /= numPixels;
29         differenceB /= numPixels;
30
31         return differenceR + differenceG + differenceB;
32     }
33 }
```

Gambar 4.6. Kelas MaxPixelDifference
(Sumber: arsip penulis)

```
● ○ ●
1  class Entropy
2  {
3      public static double CalculateError(byte[,] pixelMatrix, int y, int x, int height, int width)
4      {
5
6          int numPixels = height * width;
7          int[] histogramR = new int[256];
8          int[] histogramG = new int[256];
9          int[] histogramB = new int[256];
10
11
12         for (int i = y; i < y + height; i++)
13         {
14             for (int j = x; j < x + width; j++)
15             {
16                 histogramR[pixelMatrix[i, j, 0]]++;
17                 histogramG[pixelMatrix[i, j, 1]]++;
18                 histogramB[pixelMatrix[i, j, 2]]++;
19             }
20         }
21
22         double entropyR = 0, entropyG = 0, entropyB = 0;
23
24         for (int i = 0; i < 256; i++)
25         {
26             double probabilityR = (double)histogramR[i] / numPixels;
27             double probabilityG = (double)histogramG[i] / numPixels;
28             double probabilityB = (double)histogramB[i] / numPixels;
29
30             if (probabilityR > 0) entropyR -= probabilityR * Math.Log(probabilityR, 2);
31             if (probabilityG > 0) entropyG -= probabilityG * Math.Log(probabilityG, 2);
32             if (probabilityB > 0) entropyB -= probabilityB * Math.Log(probabilityB, 2);
33         }
34         entropyR /= numPixels;
35         entropyG /= numPixels;
36         entropyB /= numPixels;
37
38         return entropyR + entropyG + entropyB;
39     }
40
41 }
42
```

Gambar 4.7. Kelas Entropy
(Sumber: arsip penulis)

```

● ● ●

1  class SSIM // Structural Similarity Index
2  {
3      public static double CalculateError(byte[,] pixelMatrix, int y, int x, int height, int width)
4      {
5          // Menghitung C1-C2 berdasarkan rumus dari SSIM (Wang et al., 2004) dengan nilai piksel 255 untuk 8-bit gambar
6          double L = 255; // 8-bit
7          double K1 = 0.01;
8          double K2 = 0.03;
9          double C1 = K1*L*K1*L; // C1 = (K1*L)^2 = 6.5025
10         double C2 = K2*L*K2*L; // C2 = (K2*L)^2 = 58.5225
11
12         byte[,] originalMatrix = InputHandler.GetOriginalMatrix(pixelMatrix);
13
14         double muX_R = 0, muX_G = 0, muX_B = 0;
15         double numPixels = height * width;
16
17         //hitung mean dari gambar blok asli
18         for (int i = y; i < y + height; i++)
19         {
20             for (int j = x; j < x + width; j++)
21             {
22                 muX_R += originalMatrix[i, j, 0];
23                 muX_G += originalMatrix[i, j, 1];
24                 muX_B += originalMatrix[i, j, 2];
25             }
26         }
27
28         muX_R /= numPixels;
29         muX_G /= numPixels;
30         muX_B /= numPixels;
31
32         // Tidak ada Blok Hasil Rekonstruksi, Simulasi Blok Rekonstruksi ketika sedang dalam proses kompresi
33         double muY_R = muX_R;
34         double muY_G = muX_G;
35         double muY_B = muX_B;
36
37         //VARIANCE AND COVARIANCE
38         double varX_R = 0, varX_G = 0, varX_B = 0;
39         double varY_R = 0, varY_G = 0, varY_B = 0;
40         double covXY_R = 0, covXY_G = 0, covXY_B = 0;
41
42         for (int i = y; i < y + height; i++)
43         {
44             for (int j = x; j < x + width; j++)
45             {
46                 double oxR = pixelMatrix[i, j, 0] - muX_R;
47                 double oxG = pixelMatrix[i, j, 1] - muX_G;
48                 double oxB = pixelMatrix[i, j, 2] - muX_B;
49
50                 // blok rekonstruksi konstan, deviasi = 0, maka varY = 0, dan covXY = 0
51                 varX_R += oxR * oxR;
52                 varX_G += oxG * oxG;
53                 varX_B += oxB * oxB;
54             }
55         }
56
57         varX_R /= (numPixels - 1);
58         varX_G /= (numPixels - 1);
59         varX_B /= (numPixels - 1);
60
61         varY_R = 0;
62         varY_G = 0;
63         varY_B = 0;
64         covXY_R = 0;
65         covXY_G = 0;
66         covXY_B = 0;
67     }
}

```

```

68
69     // Hitung SSIM per Kanal warna
70     double numerator_R = ((2 * muX_R * muY_R + C1) * (2 * covXY_R + C2));
71     double denominator_R = ((muX_R * muX_R + muY_R * muY_R + C1) * (varX_R + varY_R + C2));
72     double ssim_R = numerator_R / denominator_R;
73
74     double numerator_G = ((2 * muX_G * muY_G + C1) * (2 * covXY_G + C2));
75     double denominator_G = ((muX_G * muX_G + muY_G * muY_G + C1) * (varX_G + varY_G + C2));
76     double ssim_G = numerator_G / denominator_G;
77
78     double numerator_B = ((2 * muX_B * muY_B + C1) * (2 * covXY_B + C2));
79     double denominator_B = ((muX_B * muX_B + muY_B * muY_B) * (varX_B + varY_B + C2));
80     double ssim_B = numerator_B / denominator_B;
81
82     // SSIM VALUE
83     ssim_R /= numPixels;
84     ssim_G /= numPixels;
85     ssim_B /= numPixels;
86
87     return (1.0 - ssim_R) + (1.0 - ssim_G) + (1.0 - ssim_B);
88 }
89 }
```

Gambar 4.8. Kelas SSIM

(Sumber: arsip penulis)

InputHandler.cs

```
1  using System.Drawing;
2  using System.Drawing.Drawing2D;
3  using System.Drawing.Imaging;
4
5  namespace QuadtreeCompression;
6
7  class InputHandler
8  {
9      // image data extraction
10     public static byte[,] ImageToPixelBytes(string imgPath)
11     {
12         try
13         {
14             byte[] fileBytes = File.ReadAllBytes(imgPath);
15
16             using (MemoryStream ms = new MemoryStream(fileBytes))
17             {
18                 // decode raw bytes (binary representation) from MemoryStream into RGB pixel values.
19                 using (Bitmap bmp = new Bitmap(ms))
20                 {
21                     // LockBits to get pixel data
22                     Rectangle rect = new Rectangle(0, 0, bmp.Width, bmp.Height);
23                     BitmapData bmpData = bmp.LockBits(rect, ImageLockMode.ReadOnly, bmp.PixelFormat);
24
25                     int bytesPerPixel = Image.GetPixelFormatSize(bmpData.PixelFormat) / 8;    // 3 bytes: R, G, B
26                     int totalBytes = bmpData.Stride * bmp.Height;           // stride = pixel bytes + padding bytes
27                     byte[] pixelBytes = new byte[totalBytes];           // instantiating pixelBytes with size = totalBytes
28
29                     // copying raw bytes from Scan0 to pixelBytes
30                     System.Runtime.InteropServices.Marshal.Copy(bmpData.Scan0, pixelBytes, 0, totalBytes);
31
32                     byte[,] pixelMatrix = GetPixelMatrix(pixelBytes, bmp.Height, bmp.Width, bmpData.Stride, bytesPerPixel);
33
34                     bmp.UnlockBits(bmpData);
35                     return pixelMatrix;
36                 }
37             }
38         }
39         catch (Exception ex)
40         {
41             Console.WriteLine($"Failed to load image {imgPath}. Error: {ex.Message}");
42             return new byte[0,0];
43         }
44     }
45 }
```

```

● ● ●

1 // user-input handler
2 public string GetImagePath()
3 {
4     while (true)
5     {
6         Console.Write("Absolute input image path: ");
7         string? path = Console.ReadLine()?.Trim();
8         if(!string.IsNullOrEmpty(path) && File.Exists(path))
9         {
10             return path;
11         }
12         Console.WriteLine("Invalid path or file does not exist. Please try again.");
13     }
14 }
15
16 public string GetErrorMethod()
17 {
18     while (true)
19     {
20         Console.Write("Select error calculation method (1/2/3/4/5): ");
21         string? input = Console.ReadLine()?.Trim();
22         switch (input)
23         {
24             case "1": return "Variance";
25             case "2": return "MAD";
26             case "3": return "MPD";
27             case "4": return "Entropy";
28             case "5": return "SSIM";
29         }
30         Console.WriteLine("Invalid choice. Please enter a number between 1-5.");
31     }
32 }
33
34 public double GetThreshold(string method)
35 {
36     double min, max;
37     switch (method)
38     {
39         case "Variance":
40             min = 0;
41             max = 5000;
42             break;
43         case "MAD":
44             min = 0;
45             max = 50;
46             break;
47         case "MPD":
48             min = 0;
49             max = 100;
50             break;
51         case "Entropy":
52             min = 0;
53             max = 7;
54             break;
55         case "SSIM":
56             min = 0;
57             max = 0.5;
58             break;
59         default:
60             throw new ArgumentException("Invalid error method");
61     }
62
63     while (true)
64     {
65         Console.Write($"Enter threshold value ({min} to {max}): ");
66         if (double.TryParse(Console.ReadLine(), out double threshold) && threshold >= min && threshold <= max)
67         {
68             return threshold;
69         }
70         Console.WriteLine($"Invalid threshold. Please enter a value between {min} and {max}.");
71     }
72 }

```

```

74     public int GetMinBlockSize()
75     {
76         while (true)
77         {
78             Console.Write("Enter minimum block size (block area. e.g. 2, 4, 10 etc): ");
79             if (int.TryParse(Console.ReadLine(), out int size) && size > 0)
80             {
81                 return size;
82             }
83             Console.WriteLine("Invalid block size. Please enter a positive integer.");
84         }
85     }
86
87     public double GetCompressionPercentage()
88     {
89         while (true)
90         {
91             Console.Write("Enter target compression percentage (0.00 to 1.0): ");
92             if (double.TryParse(Console.ReadLine(), out double percent) && percent <= 1)
93             {
94                 return percent;
95             }
96             Console.WriteLine("Invalid percentage. Please enter a value between 0.00 and 1.00");
97         }
98     }
99
100    public string GetOutputImagePath(string inputImagePath)
101    {
102        while (true) {
103            Console.Write("Absolute output image path: ");
104            string? outputPath = Console.ReadLine()?.Trim();
105            if(!string.IsNullOrEmpty(outputPath))
106            {
107                return outputPath;
108            }
109            Console.WriteLine("Invalid path. Please try again.");
110        }
111    }
112
113    public string GetOutputGifPath(string inputImagePath)
114    {
115        while (true) {
116            Console.Write("Absolute GIF path: ");
117            string? outputPath = Console.ReadLine()?.Trim();
118            if(!string.IsNullOrEmpty(outputPath))
119            {
120                return outputPath;
121            }
122            Console.WriteLine("Invalid path. Please try again.");
123        }
124    }
125 }

```

Gambar 4.9. Kelas InputHandler
(Sumber: arsip penulis)

OutputHandler.cs

```

● ● ●
1  using System.Diagnostics;
2  using System.Drawing;
3  using System.Drawing.Imaging;
4  using ImageMagick;
5
6
7  namespace QuadtreeCompression;
8
9  class OutputHandler
10 {
11     public class ProgramStatistics
12     {
13         public Dictionary<string, TimeSpan> ExecutionTimes { get; } = new Dictionary<string, TimeSpan>();
14         public long PreviousImageSizeBytes { get; set; }
15         public long CompressedImageSizeBytes { get; set; }
16         public double CompressionPercentage => PreviousImageSizeBytes > 0 ? (1.0 - (double)CompressedImageSizeBytes / PreviousImageSizeBytes) * 100 : 0;
17         public int QuadtreeTotalDepth { get; set; }
18         public int NodeCount { get; set; }
19         public string? OutputImagePath { get; set; }
20         public string? OutputGifPath { get; set; }
21
22         public override string ToString()
23         {
24             string result = "\n----- Program Statistics -----";
25             result += " Execution Time\n";
26             foreach (var kvp in ExecutionTimes)
27             {
28                 result += $" - {kvp.Key}: {kvp.Value.TotalMilliseconds:F2} ms\n";
29             }
30             result += "\n";
31
32             result += $"- Previous Image Size: {(PreviousImageSizeBytes / 1024.0:F3)} KB\n";
33             result += $"- Compressed Image Size: {(CompressedImageSizeBytes / 1024.0:F3)} KB\n";
34             result += $"- Compression Percentage: {CompressionPercentage:F2}%\n";
35             result += $"- Quadtree Total Depth: {QuadtreeTotalDepth}\n";
36             result += $"- Node Count: {NodeCount}\n";
37             result += $"- Output Image Path: {(OutputImagePath)}\n"; // Always display image path
38             if (!String.IsNullOrEmpty(OutputGifPath)) // Display GIF path only if provided
39             {
40                 result += $"- Output GIF Path: {(OutputGifPath)}\n";
41             }
42             return result;
43         }
44     }
45
46     private readonly ProgramStatistics stats = new ProgramStatistics();
47     private readonly Dictionary<string, Stopwatch> timers = new Dictionary<string, Stopwatch>();
48
49     public void StartTiming(string processName)
50     {
51         if (!timers.ContainsKey(processName))
52         {
53             timers[processName] = new Stopwatch();
54         }
55         timers[processName].Restart();
56     }
57
58     public void StopTiming(string processName)
59     {
60         if (timers.ContainsKey(processName) && timers[processName].IsRunning)
61         {
62             timers[processName].Stop();
63             stats.ExecutionTimes[processName] = timers[processName].Elapsed;
64         }
65     }
66
67     public void SetImageSizes(long previousSize, long compressedSize)
68     {
69         stats.PreviousImageSizeBytes = previousSize;
70         stats.CompressedImageSizeBytes = compressedSize;
71     }
72
73     public void SetQuadtreeStats(int depth, int nodeCount)
74     {
75         stats.QuadtreeTotalDepth = depth;
76         stats.NodeCount = nodeCount;
77     }
78     public void SetOutputPaths(string outputImagePath, string? outputGifPath = null)
79     {
80         stats.OutputImagePath = outputImagePath;
81         stats.OutputGifPath = outputGifPath; // Will be null if GIF is not created
82     }
83     public void DisplayStatistics()
84     {
85         Console.WriteLine(stats.ToString());
86     }

```

```
● ● ●
```

```
1 // Image output
2 public void SaveImage(
3     byte[,,] pixelMatrix,
4     int height,
5     int width,
6     string inputImagePath,
7     string outputPath,
8     long jpegQuality = 75L) // Default quality, overridden for JPEG outputs
9 {
10    if (pixelMatrix == null)
11        throw new ArgumentNullException(nameof(pixelMatrix));
12    if (height <= 0 || width <= 0)
13        throw new ArgumentException("Height and width must be positive");
14    if (string.IsNullOrWhiteSpace(outputImagePath))
15        throw new ArgumentException("Output path is required", nameof(outputImagePath));
16
17    using (Bitmap reconstructedBitmap = new Bitmap(width, height))
18    {
19        for (int y = 0; y < height; y++)
20        {
21            for (int x = 0; x < width; x++)
22            {
23                reconstructedBitmap.SetPixel(x, y, Color.FromArgb(
24                    pixelMatrix[y, x, 0],
25                    pixelMatrix[y, x, 1],
26                    pixelMatrix[y, x, 2]
27                ));
28            }
29        }
30
31        string inputExtension = Path.GetExtension(inputImagePath).ToLowerInvariant();
32        if (inputExtension == ".png")
33        {
34            reconstructedBitmap.Save(outputImagePath, ImageFormat.Png);
35        }
36        else // JPEG
37        {
38            var jpgEncoder = GetEncoder(ImageFormat.Jpeg);
39            var encoderParams = new EncoderParameters(1);
40            encoderParams.Param[0] = new EncoderParameter(Encoder.Quality, jpegQuality);
41            outputPath = Path.ChangeExtension(outputImagePath, ".jpg");
42            reconstructedBitmap.Save(outputImagePath, jpgEncoder, encoderParams);
43        }
44    }
45 }
46
47 private ImageCodecInfo GetEncoder(ImageFormat format)
48 {
49     foreach (var codec in ImageCodecInfo.GetImageEncoders())
50     {
51         if (codec.FormatID == format.Guid)
52             return codec;
53     }
54     throw new InvalidOperationException($"No encoder found for format {format}.");
55 }
56
```

```

1 // GIF output
2 // Use Magick.NET to create the animated GIF
3 public void CreateTransformationGif(Quadtree quadtree, byte[,] pixelMatrix, int height, int width, string outputGifPath)
4 {
5     int maxDepth = GetMaxDepth(quadtree);
6
7     List<Bitmap> frames = new List<Bitmap>();
8
9     for (int depth = 0; depth <= maxDepth; depth++)
10    {
11        byte[,] framePixelMatrix = new byte[height, width, 3];
12        ReconstructImageAtDepth(quadtree.GetRootNode(), framePixelMatrix, depth);
13
14        Bitmap frame = new Bitmap(width, height);
15        for (int y = 0; y < height; y++)
16        {
17            for (int x = 0; x < width; x++)
18            {
19                frame.SetPixel(x, y, Color.FromArgb(
20                    framePixelMatrix[y, x, 0],
21                    framePixelMatrix[y, x, 1],
22                    framePixelMatrix[y, x, 2]));
23            }
24        }
25        frames.Add(frame);
26    }
27    using (var collection = new MagickImageCollection())
28    {
29        foreach (var frame in frames)
30        {
31            using (MemoryStream ms = new MemoryStream())
32            {
33                frame.Save(ms, ImageFormat.Png); // Save frame as PNG in memory
34                ms.Position = 0;
35                var magickImage = new MagickImage(ms);
36                magickImage.AnimationDelay = 100; // 100 centiseconds = 1 second per frame
37                collection.Add(magickImage);
38            }
39            frame.Dispose(); // Dispose of the Bitmap after using it
40        }
41
42        collection.Optimize();
43
44        collection.Write(outputGifPath);
45    }
46 }
47
48 private int GetMaxDepth(Quadtree quadtree)
49 {
50     return GetNodeDepth(quadtree.GetRootNode(), 0);
51 }
52
53 private int GetNodeDepth(Node node, int currentDepth)
54 {
55     if (node.IsLeaf())
56     {
57         return currentDepth;
58     }
59
60     int maxChildDepth = currentDepth;
61     if (!node.IsLeaf())
62     {
63         foreach (var child in node.childNodes)
64         {
65             if (child != null)
66             {
67                 int childDepth = GetNodeDepth(child, currentDepth + 1);
68                 maxChildDepth = Math.Max(maxChildDepth, childDepth);
69             }
70         }
71     }
72     return maxChildDepth;
73 }

```

```

75 // Helper method to reconstruct the image up to a specific depth (should've been joined to image reconstruction in QuadTree.cs)
76 private void ReconstructImageAtDepth(Node node, byte[,] pixelMatrix, int maxDepth, int currentDepth = 0)
77 {
78     if (currentDepth >= maxDepth || node.IsLeaf())
79     {
80         // Fill the region with the node's average color
81         int r = (node.nodeAverageColor >> 16) & 0xFF;
82         int g = (node.nodeAverageColor >> 8) & 0xFF;
83         int b = node.nodeAverageColor & 0xFF;
84         int maxY = Math.Min(node.Y + node.nodeHeight, pixelMatrix.GetLength(0));
85         int maxX = Math.Min(node.X + node.nodeWidth, pixelMatrix.GetLength(1));
86         for (int y = node.Y; y < maxY; y++)
87         {
88             for (int x = node.X; x < maxX; x++)
89             {
90                 pixelMatrix[y, x, 0] = (byte)r;
91                 pixelMatrix[y, x, 1] = (byte)g;
92                 pixelMatrix[y, x, 2] = (byte)b;
93             }
94         }
95         return;
96     }
97     if (!node.IsLeaf())
98     {
99         foreach (var child in node.childNodes)
100         {
101             if (child != null)
102             {
103                 ReconstructImageAtDepth(child, pixelMatrix, maxDepth, currentDepth + 1);
104             }
105         }
106     }
107 }
108 }
109 }
```

Gambar 4.10. Kelas OutputHandler

(Sumber: arsip penulis)

5. IMPLEMENTASI BONUS

5.1. Structural Similarity Index Measure (SSIM)

Fitur bonus pembuatan Structural Similarity Index Measure (SSIM) diimplementasikan sebagai suatu metode pengukuran error yang dapat dipilih oleh pengguna. Berbeda dengan metode-metode pengukuran error sederhana seperti *Variance* atau *Mean Absolute Deviation (MAD)* yang hanya memperhitungkan nilai blok piksel secara statistik, SSIM bekerja dengan membandingkan kemiripan struktural antara dua blok gambar, yaitu blok asli/semula dengan blok hasil representasi rata-rata warna selama proses kompresi, bukan blok akhir hasil kompresi.

Perhitungan SSIM dilakukan dengan rata-rata intensitas, varians, dan kovarians antara dua blok warna piksel dalam setiap kanal c (R, G, B). Fitur ini memberikan alternatif pengukuran yang lebih adaptif dan sesuai dengan persepsi manusia terhadap kualitas gambar sehingga hasil kompresi gambar yang dihasilkan tetap terlihat jelas dan *natural* meskipun ukuran file telah di-*compress* atau dikurangi.

Langkah-langkah implementasi SSIM adalah sebagai berikut:

1. **Inisialisasi dan Validasi Blok Gambar:** Program menerima koordinat awal dan ukuran blok yang akan dianalisis. Validasi dilakukan untuk memastikan blok tidak melampaui batas dimensi gambar.
2. **Pemanggilan Matriks Gambar Asli/Semula/Awal:** Program memanggil fungsi GetOriginalMatrix() untuk mengambil salinan gambar asli sebagai referensi pembanding terhadap blok piksel hasil representasi kompresi.
3. **Perhitungan Rata-Rata Intensitas Piksel:** Untuk tiap kanal warna (R, G, B), program menghitung rata-rata intensitas piksel (μ) pada blok gambar. Nilai ini juga digunakan sebagai rata-rata dari blok piksel hasil representasi, karena blok kompresi berisi warna rata-rata secara seragam.
4. **Perhitungan Varians dan Kovarians:** Program menghitung varians untuk setiap kanal dari blok asli. Oleh karena blok piksel hasil representasi menggunakan warna seragam, varians dan kovariansnya bernilai nol.
5. **Perhitungan Nilai SSIM:** Program menghitung rumus SSIM berdasarkan Wang et al. (2004). Nilai SSIM dihitung untuk masing-masing kanal warna c (R, G, B). Perhitungan ini menggunakan nilai rata-rata intensitas piksel, varians, kovarians, dan konstanta stabilitas C1 dan C2 yang telah ditetapkan.
6. **Perhitungan Nilai Rata-Rata RGB Antar-Kanal dan Transformasi ke Nilai Error:** Nilai SSIM akhir didapatkan dengan menghitung rata-rata SSIM dari ketiga kanal warna c (R, G, B). Nilai ini kemudian dikonversi menjadi nilai error dengan rumus $1 - \text{SSIM}$, agar dapat dibandingkan dengan *threshold* dalam algoritma Quadtree.

5.2. GIF (*Graphics Interchange Format*)

Fitur bonus pembuatan GIF (Graphics Interchange Format) diimplementasikan untuk memvisualisasikan proses pembentukan Quadtree secara bertahap. GIF yang dihasilkan menampilkan langkah-langkah pembagian gambar menjadi blok-blok kecil berdasarkan kedalaman pohon Quadtree. Setiap frame dalam GIF mewakili gambar pada kedalaman tertentu, di mana blok-blok yang telah dibagi hingga kedalaman tersebut diisi dengan warna rata-rata, sedangkan blok yang belum dibagi tetap menampilkan warna asli dari gambar input. Fitur ini membantu pengguna memahami bagaimana Quadtree membagi gambar secara rekursif hingga mencapai kondisi pemberhentian.

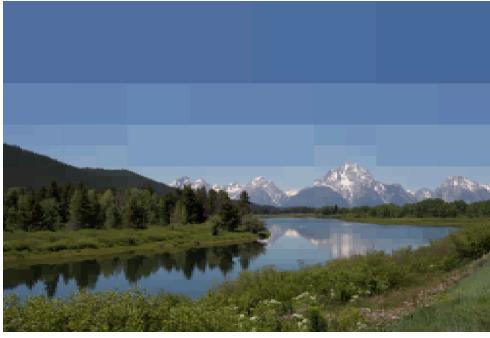
Proses pembuatan GIF melibatkan beberapa langkah utama. Pertama, program menentukan kedalaman maksimum pohon Quadtree yang telah dibentuk. Kedalaman maksimum ini menentukan jumlah frame yang akan dihasilkan dalam GIF. Kedua, untuk setiap kedalaman dari 0 hingga kedalaman maksimum, program menghasilkan sebuah frame gambar dengan cara menelusuri pohon Quadtree hingga kedalaman tersebut. Ketiga, semua frame yang dihasilkan dikompilasi menjadi sebuah file GIF menggunakan library Magick.NET-Q16-AnyCPU..

Langkah-langkah implementasi pembuatan GIF adalah sebagai berikut:

1. **Inisialisasi Koleksi Frame:** Program membuat sebuah koleksi untuk menyimpan frame-frame yang akan digunakan dalam GIF. Library Magick.NET-Q16-AnyCPU menyediakan kelas MagickImageCollection untuk keperluan ini.
2. **Penelusuran Pohon Quadtree per Kedalaman:** Untuk setiap kedalaman d dari 0 hingga kedalaman maksimum, program menelusuri pohon Quadtree dan menghasilkan gambar sementara. Simpul-simpul pada kedalaman kurang dari atau sama dengan d diisi dengan warna rata-rata, sedangkan simpul pada kedalaman lebih besar tetap menggunakan warna asli dari gambar input. Gambar sementara ini kemudian dikonversi menjadi frame menggunakan MagickImage.
3. **Pengaturan Delay dan Kompilasi GIF:** Setiap frame diberi delay sebesar 500 ms agar animasi dapat dilihat dengan jelas. Setelah semua frame ditambahkan ke koleksi, program menyimpan koleksi tersebut sebagai file GIF di folder test dengan nama [nama_file]_quadtree.gif.

6. PENGUJIAN DAN ANALISIS

6.1. Pengujian Program

Input	Output
 <pre>Absolute input image path: C:\Users\balse\Documents\VScode\Tucil2 Select error calculation method (1/2/3/4/5): 1 Enter threshold value (0 to 5000): 200 Enter minimum block size (block area. e.g. 2, 4, 10 etc): 8 Absolute output image path: C:\Users\balse\Documents\VScode\Tucil2 Absolute GIF path: C:\Users\balse\Documents\VScode\Tucil2_Stima m Create quadtree-depth GIF? (y/N): y</pre>	 <pre>----- Program Statistics ----- - Execution Time - Input Processing: 52.14 ms - Building Quadtree: 301.29 ms - Save to GIF: 3747.77 ms - Previous Image Size: 131.239 KB - Compressed Image Size: 54.836 KB - Compression Percentage: 58.22% - Quadtree Total Depth: 8 - Node Count: 32401 - Output Image Path: C:\Users\balse\Documents\VScode\Tucil2 - Output GIF Path: C:\Users\balse\Documents\VScode\Tucil2_Stima m</pre>
 <pre>Absolute input image path: C:\Users\balse\Documents\VScode\Tucil2 Select error calculation method (1/2/3/4/5): 1 Enter threshold value (0 to 5000): 2000 Enter minimum block size (block area. e.g. 2, 4, 10 etc): 8 Absolute output image path: C:\Users\balse\Documents\VScode\Tucil2 Absolute GIF path: C:\Users\balse\Documents\VScode\Tucil2_Stima m</pre>	 <pre>----- Program Statistics ----- - Execution Time - Input Processing: 47.54 ms - Building Quadtree: 178.79 ms - Image Reconstruction: 225.65 ms - Save to GIF: 3927.65 ms - Previous Image Size: 131.239 KB - Compressed Image Size: 28.185 KB - Compression Percentage: 78.52% - Quadtree Total Depth: 8 - Node Count: 3445 - Output Image Path: C:\Users\balse\Documents\VScode\Tucil2 - Output GIF Path: C:\Users\balse\Documents\VScode\Tucil2_Stima m</pre>

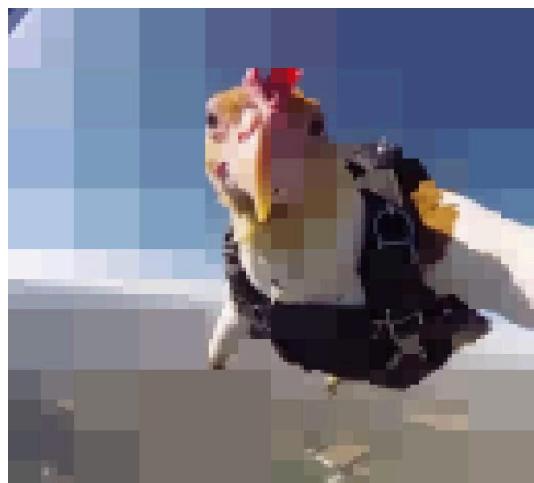


```
Absolute input image path: C:\Users\balse\Documents\VScode\Tucil2
Select error calculation method (1/2/3/4/5): 2
Enter threshold value (0 to 50): 20
Enter minimum block size (block area. e.g. 2, 4, 10 etc): 10
Absolute output image path: C:\Users\balse\Documents\VScode\Tucil2_10x10
Absolute GIF path: C:\Users\balse\Documents\VScode\Tucil2_Stima.m
```



----- Program Statistics -----
- Execution Time
- Input Processing: 43.09 ms
- Building Quadtree: 97.14 ms
- Image Reconstruction: 188.78 ms
- Save to GIF: 2755.81 ms

- Previous Image Size: 39.438 KB
- Compressed Image Size: 36.810 KB
- Compression Percentage: 6.66%
- Quadtree Total Depth: 7
- Node Count: 6109
- Output Image Path: C:\Users\balse\Documents\Tucil2_10x10.jpg
- Output GIF Path: C:\Users\balse\Documents\Tucil2_Stima.gif



```
Absolute input image path: C:\Users\balse\Documents\Vscode\Tuci1  
Select error calculation method (1/2/3/4/5): 2  
Enter threshold value (0 to 50): 40  
Enter minimum block size (block area. e.g. 2, 4, 10 etc): 14  
Absolute output image path: C:\Users\balse\Documents\Vscode\Tuci1  
Absolute GIF path: C:\Users\balse\Documents\Vscode\Tuci12_Stima
```

```
----- Program Statistics -----  
- Execution Time  
  - Input Processing: 37.33 ms  
  - Building Quadtree: 76.33 ms  
  - Image Reconstruction: 192.10 ms  
  - Save to GIF: 2749.75 ms  
  
- Previous Image Size: 39.438 KB  
- Compressed Image Size: 29.938 KB  
- Compression Percentage: 24.09%  
- Quadtree Total Depth: 7  
- Node Count: 2909  
- Output Image Path: C:\Users\balse\Documents\Vscode\Tuci12_Img  
- Output GIF Path: C:\Users\balse\Documents\Vscode\Tuci12_GIF
```



```
Absolute input image path: C:\Users\balse\Documents\Vscode\Tuci1  
Select error calculation method (1/2/3/4/5): 3  
Enter threshold value (0 to 100): 20  
Enter minimum block size (block area. e.g. 2, 4, 10 etc): 2  
Absolute output image path: C:\Users\balse\Documents\Vscode\Tuci1  
Absolute GIF path: C:\Users\balse\Documents\Vscode\Tuci12_Stima
```

```
----- Program Statistics -----  
- Execution Time  
  - Input Processing: 72.03 ms  
  - Building Quadtree: 214.24 ms  
  - Image Reconstruction: 328.93 ms  
  - Save to GIF: 6736.67 ms  
  
- Previous Image Size: 114.909 KB  
- Compressed Image Size: 118.169 KB  
- Compression Percentage: -2.84%  
- Quadtree Total Depth: 9  
- Node Count: 136281  
- Output Image Path: C:\Users\balse\Documents\Vscode\Tuci12_Img  
- Output GIF Path: C:\Users\balse\Documents\Vscode\Tuci12_GIF
```



```
Absolute input image path: C:\Users\balse\Documents\Vscode\Tucil2_Step1.jpg  
Select error calculation method (1/2/3/4/5): 3  
Enter threshold value (0 to 100): 20  
Enter minimum block size (block area. e.g. 2, 4, 10 etc): 10  
Absolute output image path: C:\Users\balse\Documents\Vscode\Tucil2_Step1_Quantized.jpg  
Absolute GIF path: C:\Users\balse\Documents\Vscode\Tucil2_Step1_GIF.gif  
Creating quantized image... (1/1)
```

```
----- Program Statistics -----  
- Execution Time  
- Input Processing: 58.82 ms  
- Building Quadtree: 177.85 ms  
- Image Reconstruction: 341.14 ms  
- Save to GIF: 5525.66 ms  
  
- Previous Image Size: 114.909 KB  
- Compressed Image Size: 112.852 KB  
- Compression Percentage: 1.79%  
- Quadtree Total Depth: 8  
- Node Count: 54045  
- Output Image Path: C:\Users\balse\Documents\Vscode\Tucil2_Step1_Quantized.jpg  
- Output GIF Path: C:\Users\balse\Documents\Vscode\Tucil2_Step1_GIF.gif
```



```
Absolute input image path: C:\Users\balse\Documents\Vscode\Tucil2_Step1_GIF.gif  
Select error calculation method (1/2/3/4/5): 4  
Enter threshold value (0 to 7): 2  
Enter minimum block size (block area. e.g. 2, 4, 10 etc): 2  
Absolute output image path: C:\Users\balse\Documents\Vscode\Tucil2_Step1_GIF.gif  
Absolute GIF path: C:\Users\balse\Documents\Vscode\Tucil2_Step1_GIF.gif  
Creating quantized image... (1/1)
```

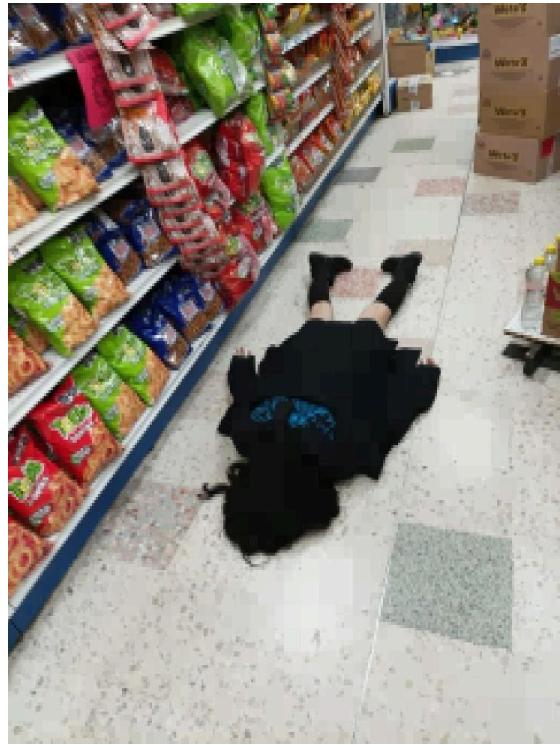
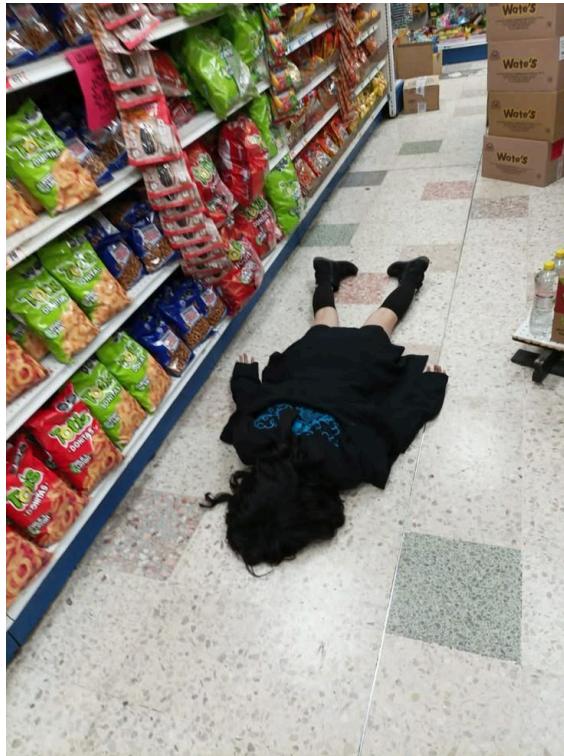


```
----- Program Statistics -----  
- Execution Time  
- Input Processing: 45.98 ms  
- Building Quadtree: 843.44 ms  
- Image Reconstruction: 259.76 ms  
- Save to GIF: 4805.34 ms  
  
- Previous Image Size: 80.429 KB  
- Compressed Image Size: 70.283 KB  
- Compression Percentage: 12.61%  
- Quadtree Total Depth: 9  
- Node Count: 308133  
- Output Image Path: C:\Users\balse\Documents\Vscode\Tucil2_Step1_GIF.gif  
- Output GIF Path: C:\Users\balse\Documents\Vscode\Tucil2_Step1_GIF.gif
```



```
Absolute input image path: C:\Users\balse\Documents\Vscode\Tuci  
Select error calculation method (1/2/3/4/5): 4  
Enter threshold value (0 to 7): 5  
Enter minimum block size (block area. e.g. 2, 4, 10 etc): 20  
Absolute output image path: C:\Users\balse\Documents\Vscode\Tuci  
Absolute GIF path: C:\Users\balse\Documents\Vscode\tuci12_stima
```

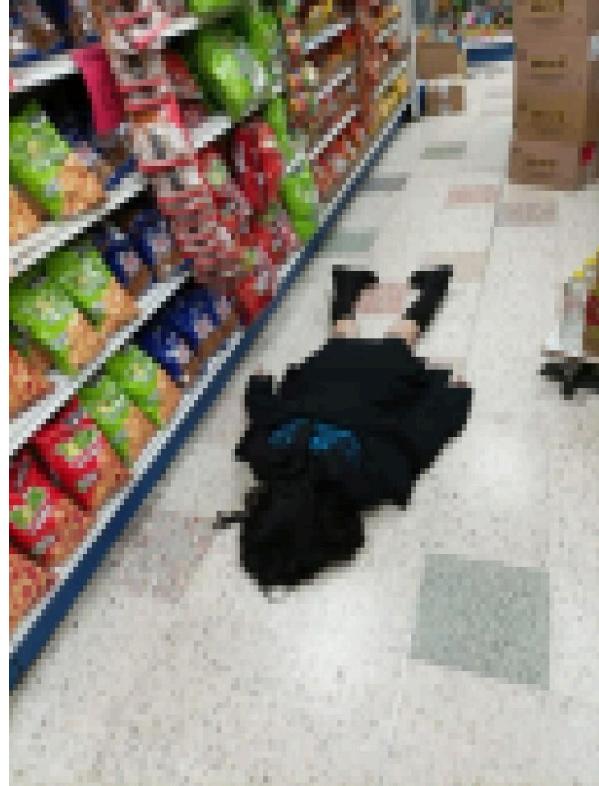
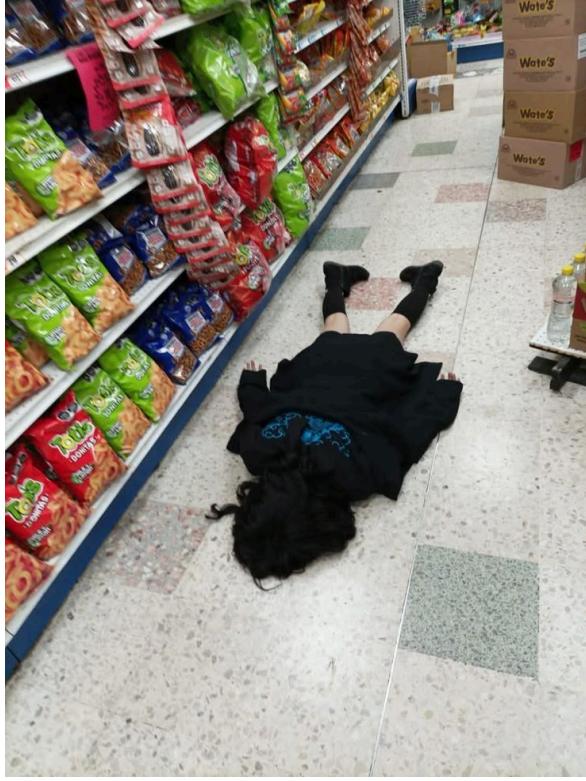
```
----- Program Statistics -----  
- Execution Time  
- Input Processing: 51.45 ms  
- Building Quadtree: 181.66 ms  
- Image Reconstruction: 261.97 ms  
- Save to GIF: 3930.84 ms  
  
- Previous Image Size: 80.429 KB  
- Compressed Image Size: 48.259 KB  
- Compression Percentage: 40.00%  
- Quadtree Total Depth: 7  
- Node Count: 20253  
- Output Image Path: C:\Users\balse\Documents\Vscode\tuci12_stima  
- Output GIF Path: C:\Users\balse\Documents\Vscode\tuci12_stima.gif
```



```
Absolute input image path: C:\Users\balse\Documents\vscode\Tucil2_Stim
Select error calculation method (1/2/3/4/5): 5
Enter threshold value (0 to 0.5): 0.4
Enter minimum block size (block area. e.g. 2, 4, 10 etc): 8
Absolute output image path: C:\Users\balse\Documents\vscode\Tucil2_Stim
Absolute GIF path: C:\Users\balse\Documents\VScode\Tucil2_Stim
```

----- Program Statistics -----

- Execution Time
 - Input Processing: 70.35 ms
 - Building Quadtree: 16402.28 ms
 - Image Reconstruction: 483.92 ms
 - Save to GIF: 4918.38 ms
- Previous Image Size: 124.630 KB
- Compressed Image Size: 107.447 KB
- Compression Percentage: 13.79%
- Quadtree Total Depth: 8
- Node Count: 56317
- Output Image Path: C:\Users\balse\Documents\Tucil2_Stim
- Output GIF Path: C:\Users\balse\Documents\VScode\Tucil2_Stim



```
Absolute input image path: C:\Users\balse\Documents\vscode\Tucil2_Stim
Select error calculation method (1/2/3/4/5): 5
Enter threshold value (0 to 0.5): 0.05
Enter minimum block size (block area. e.g. 2, 4, 10 etc): 30
Absolute output image path: C:\Users\balse\Documents\vscode\Tucil2_Stim
Absolute GIF path: C:\Users\balse\Documents\VScode\Tucil2_Stim
```

----- Program Statistics -----

- Execution Time
 - Input Processing: 50.48 ms
 - Building Quadtree: 4997.75 ms
 - Image Reconstruction: 274.59 ms
 - Save to GIF: 3723.04 ms
- Previous Image Size: 124.630 KB
- Compressed Image Size: 96.348 KB
- Compression Percentage: 22.69%
- Quadtree Total Depth: 7
- Node Count: 21505
- Output Image Path: C:\Users\balse\Documents\Tucil2_Stim
- Output GIF Path: C:\Users\balse\Documents\VScode\Tucil2_Stim

6.2. Analisis Algoritma Quad Tree

Proses pembentukan Quadtree terdiri dari beberapa langkah utama. Pertama, program menghitung nilai rata-rata warna pada setiap blok gambar untuk digunakan dalam perhitungan galat dan sebagai representasi warna simpul. Selanjutnya, galat blok dihitung berdasarkan metode perhitungan *error* yang dipilih oleh pengguna. Terakhir, program memeriksa kondisi pemberhentian: jika galat melebihi ambang batas dan ukuran blok masih lebih besar dari ukuran minimum, maka blok akan dibagi menjadi empat kuadran secara rekursif. Jika tidak, proses rekursi berhenti.

Langkah perhitungan rata-rata warna memiliki kompleksitas waktu $O(n)$ dengan n adalah jumlah piksel dalam blok gambar. Proses ini bersifat linier karena hanya memerlukan iterasi sebanyak jumlah piksel untuk menjumlahkan nilai warna merah (R), hijau (G), dan biru (B), lalu membaginya dengan luas blok.

Setelah mendapatkan nilai rata-rata warna, program menghitung galat blok menggunakan *metric* yang dipilih. Meskipun setiap metode (seperti varians, MAD, MPD, entropi, atau SSIM) memiliki rumus yang berbeda, semua metode tersebut memiliki kompleksitas waktu $O(n)$ karena memerlukan iterasi linier terhadap piksel dalam blok.

Pengecekan kondisi pemberhentian dilakukan dengan membandingkan galat dengan ambang batas serta memeriksa ukuran blok. Proses ini memiliki kompleksitas waktu $O(1)$ karena tidak bergantung pada jumlah piksel. Jika kondisi pembagian terpenuhi, proses yang sama akan diulang sebanyak empat kali dengan ukuran blok menjadi $\frac{n}{4}$.

Jika keseluruhan proses pembentukan Quadtree dilambangkan dengan fungsi $T(n)$, maka dapat dituliskan sebagai berikut:

$$T(n) = O(n) + O(n) + O(1) + 4T\left(\frac{n}{4}\right) = 4T\left(\frac{n}{4}\right) + O(n)$$

Fungsi ini bersifat rekursif sehingga dapat didefinisikan dalam bentuk *piecewise* sebagai berikut.

$$T(n) = \begin{cases} O(n), & n < C_1 \vee \text{error} \leq C_2 \\ 4T\left(\frac{n}{4}\right) + O(n), & n \geq C_1 \wedge \text{error} > C_2 \end{cases}$$

Di mana:

- $T(n)$: Kompleksitas waktu pembentukan Quadtree.

- $O(n)$: Kompleksitas waktu untuk setiap iterasi rekursi.
- error: Nilai galat blok.
- C_1 : Konstanta ukuran blok minimum.
- C_2 : Konstanta ambang batas galat.

Fungsi rekursi ini sesuai dengan bentuk teorema Master. Jadi, kompleksitas waktu pembentukan Quadtree adalah $O(n \log(n))$, di mana n adalah jumlah piksel dalam gambar.

7. KESIMPULAN DAN SARAN

7.1. Kesimpulan

Dari hasil algoritma program, implementasi algoritma, dan pengujian program kompresi gambar berbasis struktur Quadtree ini, dapat disimpulkan bahwa:

1. Struktur Quadtree mampu merepresentasikan gambar secara efisien dengan algoritma *divide and conquer*, yang dapat membagi blok gambar berdasarkan nilai error lokal.
2. Metode pengukuran error yang dapat digunakan, yaitu: *Variance*, *Mean Absolute Deviation (MAD)*, *Max Pixel Difference*, *Entropy*, dan *SSIM*, dapat memberikan fleksibilitas pengguna program untuk mengontrol kualitas dan tingkat kompresi gambar.
3. Kompleksitas waktu pembentukan Quadtree untuk kompresi gambar adalah $O(n \log n)$, di mana n adalah jumlah piksel yang ada pada gambar, mengikuti karakteristik fungsi rekursif dari proses pembagian blok.
4. *Implementasi Structural Similarity Index Measure (SSIM)* dapat menghasilkan kompresi gambar yang mendekati persepsi manusia terhadap kualitas gambar.
5. Program berhasil menghasilkan file kompresi yang lebih kecil dengan kualitas gambar yang masih terjaga dan tidak berubah banyak.

7.2. Saran

Pada Tugas Kecil 2 ini, terdapat beberapa saran yang perlu dikembangkan lebih lanjut untuk memperbaiki dan mengembangkan program:

1. Pilihan metode pengukuran error dapat diperbanyak untuk memperluas opsi pengguna dalam melakukan proses kompresi gambar. Beberapa opsi yang dapat diimplementasikan adalah: *Mean Squared Error (MSE)*, *Peak Signal-to-Noise Ratio (PSNR)*, dan *Mean Absolute Error (MAE)*.
2. Meskipun algoritma *divide and conquer* yang menggunakan struktur Quadtree telah berhasil dalam mengompresi gambar, masih terdapat banyak algoritma yang dapat digunakan secara lebih efisien dan efektif, serta tidak terlalu berpegang pada fungsi rekursif yang membutuhkan manajemen memori

pemrograman yang besar dan lambat. Kompresi gambar dapat dilakukan melalui baik sebuah algoritma baru atau penggabungan algoritma demi mencapai hasil yang lebih baik.

LAMPIRAN

1. Tautan Repository Github

https://github.com/KalengBalsem/Tucil2_15223011_15223090

2. Tabel Ketercapaian

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	✓	

DAFTAR PUSTAKA

- DeepBlock. (t.t.). *Quadtree*. Diambil dari <https://www.deepblock.net/blog/quadtree>
- Fei, L., Yang, J., Wong, W.-K., Zhao, S., Toomey, A., & Deng, J. (2025). Palm-vein images reconstruction against adversarial attacks. *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1–5. doi:10.1109/ICASSP.2025.1284395. Diambil dari <https://ieeexplore.ieee.org/document/1284395>
- Institut Teknologi Bandung. (2025). *Spesifikasi tugas kecil 2 strategi algoritma 2024/2025* [Dokumen Google]. Diambil dari https://docs.google.com/document/d/1N_ENR9VQI9XdOTtVinCBd7vVqb2tbecIMInyARImVcs/edit?tab=t.0
- Lemstra, D. (t.t.). *Magick.NET* [Repositori GitHub]. Diambil dari <https://github.com/dlemstra/Magick.NET>
- Microsoft. (t.t.). *System.Drawing.Common* [Paket NuGet]. Diambil dari <https://www.nuget.org/packages/system.drawing.common/>
- Tanner, W. (t.t.). *Quadtrees for image processing*. Diambil dari <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>
- TutorialsPoint. (t.t.). *Divide and conquer algorithm*. Diambil dari https://www.tutorialspoint.com/data_structures_algorithms/divide_and_conquer.htm
- Wang, Y., & Zhang, Y. (2019). A quadtree-based image compression method using vector quantization and discrete wavelet transform. *Journal of Software Engineering and Applications*, 12(4), 97-108. doi:10.4236/jsea.2019.124007. Diambil dari <https://www.scirp.org/journal/paperinformation?paperid=90911>
- WsCube Tech. (t.t.). *Divide and conquer algorithm*. Diambil dari <https://www.wscubetech.com/resources/dsa/divide-and-conquer-algorithm>