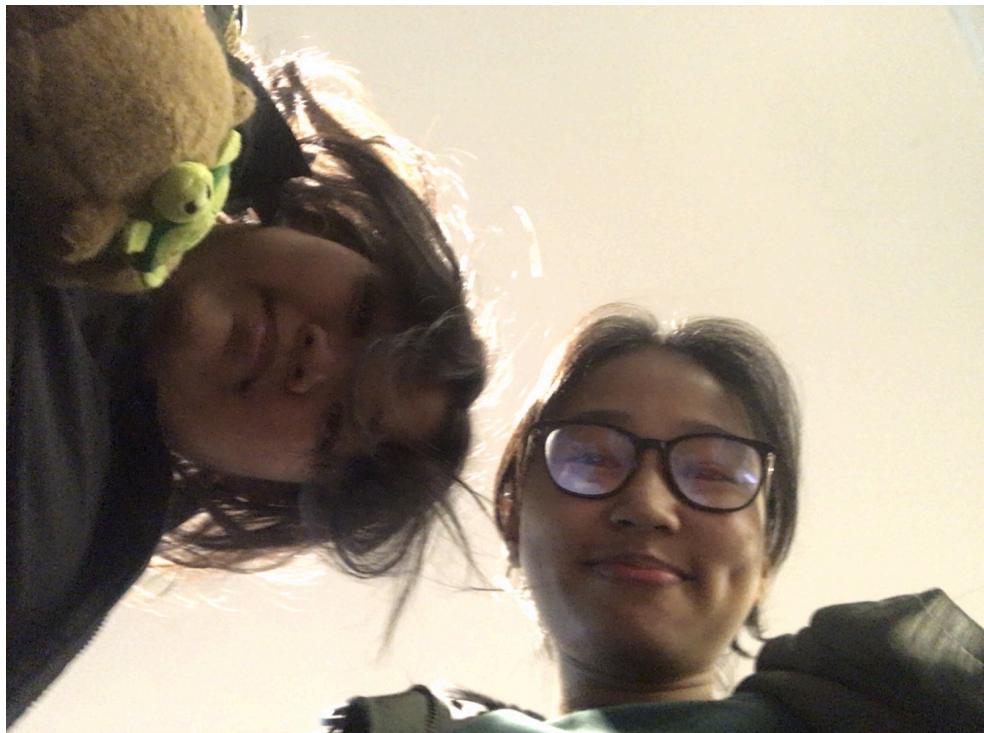


IF2211 Strategi Algoritma

Kompresi Gambar dengan Metode Quadtree

Laporan Tugas Kecil

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma
pada Semester IV Tahun Akademik 2024/2025



Dibuat Oleh

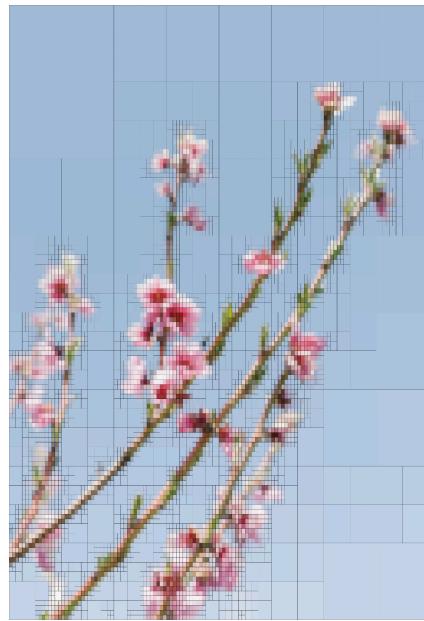
| | |
|----------------------------|----------|
| Bevinda Vivian | 13523120 |
| Samantha Laqueenna Ginting | 13523138 |

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

DAFTAR ISI

| | |
|---|----|
| DAFTAR ISI | 2 |
| BAB 1 DESKRIPSI MASALAH | 3 |
| BAB 2 LANDASAN TEORI | 9 |
| A. Divide and Conquer | 9 |
| B. Quadtree | 9 |
| BAB 3 APLIKASI STRATEGI DIVIDE AND CONQUER | 10 |
| BAB 4 IMPLEMENTASI DAN PENGUJIAN | 12 |
| A. Implementasi Algoritma | 12 |
| a. Op.cpp | 12 |
| b. Quadtree.cpp | 16 |
| c. Main.cpp | 20 |
| B. Source Program | 22 |
| a. Op.cpp | 22 |
| b. Quadtree.cpp | 32 |
| c. Main.cpp | 37 |
| C. Pengujian | 44 |
| a. Menguji dengan metode pengukuran error Variance | 44 |
| b. Menguji dengan metode pengukuran error Mean Absolute Deviation (MAD) | 45 |
| c. Menguji metode pengukuran error Max Pixel Difference | 46 |
| d. Menguji metode pengukuran error Entropy | 47 |
| e. Menguji metode pengukuran error SSIM | 48 |
| f. Menguji masukkan target kompresi | 50 |
| BAB 5 KESIMPULAN DAN SARAN | 52 |
| LAMPIRAN | 53 |
| A. Github | 53 |
| B. Tabel Pemeriksaan | 53 |
| REFERENSI | 54 |

BAB 1 DESKRIPSI MASALAH



Gambar 1. Quadtree dalam Kompresi Gambar

(Sumber:

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

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

tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan.

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



Gambar 2. Proses Pembentukan Quadtree dalam Kompresi Gambar

(Sumber: https://miro.medium.com/v2/resize:fit:640/format:webp/1*LHD7PsbmbgNBFrYkxyG5dA.gif)

Cek sumber untuk melihat animasi GIF

Ilustrasi kasus :

Ide pada tugas kecil 2 ini cukup sederhana, seperti pada pembahasan sebelumnya mengenai Quadtree. Berikut adalah prosedur pada program kompresi gambar yang akan dibuat dalam Tugas Kecil 2 (*Divide and Conquer*):

1. Inisialisasi dan Persiapan Data

Masukkan gambar yang akan dikompresi akan diolah dalam format matriks piksel dengan nilai intensitas berdasarkan sistem warna RGB. Berikut adalah parameter-parameter yang dapat ditentukan oleh pengguna saat ingin melakukan kompresi gambar:

- a) **Metode perhitungan variansi:** pilih metode perhitungan variansi berdasarkan opsi yang tersedia pada *Tabel 1*.
- b) **Threshold variansi:** nilai ambang batas untuk menentukan apakah blok akan dibagi lagi.
- c) **Minimum block size:** ukuran minimum blok piksel yang diperbolehkan untuk diproses lebih lanjut.

2. Perhitungan Error

Untuk setiap blok gambar yang sedang diproses, hitung nilai variansi menggunakan metode yang dipilih sesuai *Tabel 1*.

3. Pembagian Blok

Bandingkan nilai variansi blok dengan threshold:

- ❖ Jika variansi **di atas threshold** (*cek kasus khusus untuk metode bonus*), ukuran blok lebih besar dari **minimum block size**, dan ukuran blok setelah dibagi menjadi empat **tidak kurang dari minimum block size**, blok tersebut dibagi menjadi empat sub-blok, dan proses dilanjutkan untuk setiap sub-blok.
- ❖ Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan untuk blok tersebut.

4. Normalisasi Warna

Untuk blok yang tidak lagi dibagi, lakukanlah normalisasi warna blok sesuai dengan rata-rata nilai RGB blok.

5. Rekursi dan Penghentian

Proses pembagian blok dilakukan secara rekursif untuk setiap sub-blok hingga semua blok memenuhi salah satu dari dua kondisi berikut:

- ❖ Error blok berada di bawah threshold.
- ❖ Ukuran blok setelah dibagi menjadi empat kurang dari *minimum block size*.

6. Penyimpanan dan Output

Rekonstruksi gambar dilakukan berdasarkan struktur QuadTree yang telah dihasilkan selama proses kompresi. Gambar hasil rekonstruksi akan disimpan sebagai file terkompresi. Selain itu, persentase kompresi akan dihitung dan disertakan dengan rumus sesuai dengan yang terlampir pada dokumen ini. Persentase kompresi ini memberikan gambaran mengenai efisiensi metode kompresi yang digunakan.

Parameter:

1. Error Measurement Methods (Metode Pengukuran Error)

Metode yang digunakan untuk menentukan seberapa besar perbedaan dalam satu blok gambar. Jika error dalam blok melebihi ambas batas (*threshold*), maka blok akan dibagi menjadi empat bagian yang lebih kecil.

Tabel 1. Metode Pengukuran Error

| Metode | Formula |
|--------------------------------------|--|
| <u>Variance</u> | $\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$ |
| | $\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$ |
| | 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 |
| <u>Mean Absolute Deviation (MAD)</u> | N = Banyaknya piksel dalam satu blok |

| | |
|--------------------------------------|---|
| <u>Mean Absolute Deviation (MAD)</u> | $MAD_c = \frac{1}{N} \sum_{i=1}^N P_{i,c} - \mu_c $ $MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$ |
| | 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 |

| | |
|--|---|
| Max Pixel Difference | $D_{RGB} = \frac{R + G + B}{3}$ |
| | $D_c = \text{Selisih antara piksel dengan nilai max dan min tiap kanal warna } c \text{ (R, G, B) dalam satu blok}$ $P_{i,c} = \text{Nilai piksel pada posisi } i \text{ untuk channel warna } c$ |
| <u>Entropy</u> | $H_c = - \sum_{i=1}^N P_c(i) \log (P_c(i))_c$ |
| | $H_{RGB} = \frac{H_R + H_G + H_B}{3}$ |
| | $H_c = \text{Nilai entropi tiap kanal warna } c \text{ (R, G, B) dalam satu blok}$ $P_c(i) = \text{Probabilitas piksel dengan nilai } i \text{ dalam satu blok untuk tiap kanal warna } c \text{ (R, G, B)}$ |
| [Bonus] <u>Structural Similarity Index (SSIM)</u> <u>(Referensi tambahan)</u> | $SSIM_c(x, y) = \frac{(2\mu_{x,c} \mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_1^2 + \sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$ |
| | $SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$ |
| | Nilai SSIM yang dibandingkan adalah antara blok gambar sebelum dan sesudah dikompresi. Silakan lakukan eksplorasi untuk memahami serta memperoleh nilai konstanta pada formula SSIM, asumsikan gambar yang akan diuji adalah 24-bit RGB dengan 8-bit per kanal. |

2. Threshold (Ambang Batas)

Threshold adalah nilai batas yang menentukan apakah sebuah blok dianggap cukup seragam untuk disimpan atau harus dibagi lebih lanjut.

3. Minimum Block Size (Ukuran Blok Minimum)

Minimum block size (luas piksel) adalah 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.

4. Compression Percentage (Persentase Kompresi) [BONUS]

Persentasi kompresi menunjukkan seberapa besar ukuran gambar berkurang dibandingkan dengan dengan ukuran aslinya setelah dikompresi menggunakan metode quadtree.

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

BAB 2 LANDASAN TEORI

A. Divide and Conquer

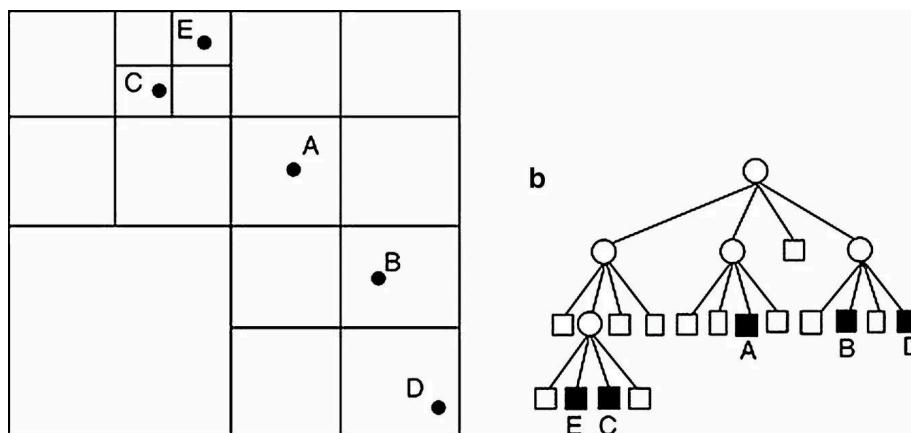
Algoritma *Divide and Conquer* membagi persoalan menjadi lebih kecil dan menyelesaikan masing-masing upa-persoalan. Penyelesaian upa-persoalan digabung sehingga membentuk solusi persoalan semula. Tiap-tiap upa-persoalan memiliki karakteristik yang sama dengan persoalan semula, sehingga dengan metode *Divide and Conquer* umumnya diungkapkan dalam skema rekursif.

Persoalan klasik yang dapat diselesaikan dengan algoritma ini adalah persoalan mencari nilai minimum dan maksimum, persoalan pengurutan (*merge sort* dan *quick sort*), pencarian sepasang titik terdekat, *Convex Hull*, dan *Skyline Problem*.

B. Quadtree

Quadtree adalah sebuah struktur pohon hirarkis di mana setiap simpul internal memiliki tepat empat anak yang terurut. Masing-masing simpul merepresentasikan data dari simpul parent-nya. Tiap simpul di Quad Tree memiliki dua atribut—*val* dan *isLeaf*. *val* bernilai *true* jika simpul tersebut sesuai dengan kisi 1, atau *false* jika sesuai dengan kisi 0. Atribut *isLeaf* bernilai *true* jika simpul tersebut adalah daun (tidak memiliki anak), jika tidak, maka bernilai *false*.

Struktur Quadtree banyak digunakan dalam partisi ruang dua dimensi dengan metode rekursif untuk membagi sebuah daerah menjadi empat kuadran. Oleh karena itu, Quadtree dapat digunakan untuk meauan pengkompresian sebuah gambar dengan melakukan pembagian empat buah daerah gambar dan menggunakan algoritma *Divide and Conquer*.



Gambar 3. Struktur Data Quadtree dalam Kompresi Gambar

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

BAB 3 APLIKASI STRATEGI DIVIDE AND CONQUER

Implementasi algoritma *divide and conquer* pada program kompresi gambar dengan metode quadtree terdiri atas tiga bagian utama:

1. Divide

Pada tahap ini, gambar dibagi secara rekursif menjadi empat bagian yang sama besar. Pembagian dilakukan dengan langkah-langkah berikut ini:

- a. Keseluruhan gambar yang ingin dikompresi dianggap sebagai satu blok
- b. Lalu untuk setiap blok yang ada, periksa apakah *error* dalam konteks ini adalah ketidakhomogenan warna melebihi *threshold* yang ditentukan
- c. Jika *error* melebihi *threshold* dan ukuran blok masih lebih besar dari ukuran minimum, maka bagi blok menjadi empat sub-blok yang berukuran sama.
- d. Lalu lakukan pembagian ini secara rekursif untuk setiap sub-blok.

2. Conquer

Pada tahap ini, dilakukan penyelesaian pada blok-blok terkecil yang tidak perlu dibagi lagi, langkah-langkah penyelesaian tersebut:

- a. Untuk setiap blok yang tidak dapat dibagi lagi yaitu *error* di bawah *threshold* atau ukuran sudah memenuhi kriteria minimum maka hitung warna rata-rata (*avgColor*) dari seluruh piksel di dalam blok.
- b. Lalu informasi terkait *avgColor*, posisi blok, dan juga ukuran blok disimpan untuk tahapan selanjutnya.

3. Combine

Pada tahap ini, dilakukan penggabungan untuk merekonstruksi gambar dari struktur quadtree, langkah-langkah yang dijalankan adalah:

- a. Rekonstruksi gambar dimulai dari node yang paling awal yaitu *root*.
- b. Jika node merupakan leaf (tidak punya anak), maka seluruh area yang direpresentasikan node akan diisi dengan *avgColor*
- c. Jika node memiliki anak (*children*), maka proses rekonstruksi akan terus dilanjutkan secara rekursif untuk setiap anak tersebut.
- d. Jika sudah dilakukan rekonstruksi sampai rekursif selesai, program menghasilkan gambar terkompresi di mana area dengan warna seragam akan direpresentasikan dengan blok besar, dan untuk area yang lebih detail akan direpresentasikan dengan blok yang lebih kecil.

Pada algoritma divide and conquer dalam struktur data quadtree, kasus terburuknya adalah saat minimum block size bernilai 1. Hal ini berarti semua blok gambar dibagi hingga unit terkecilnya, yang adalah 1 piksel.

$$T(n) = 1 + 4 + 4^2 + \dots + 4^{\log_2 n} = \frac{4^{\log_2 n + 1} - 1}{4 - 1} = \frac{4n^2 - 1}{3} = O(n^2)$$

Kasus terbaiknya adalah jika threshold sangat besar untuk metode pengukuran Variance, MAD, MPD, dan Entropy, atau sangat kecil untuk metode SSIM, dimana quadtree tidak dibagi dan hanya tercipta 1 node.

$$T(n) = O(1)$$

Namun kompleksitas waktu secara keseluruhan sebesar $O(n^2 \times \log_2 n)$. Untuk perhitungan error Variance, Mean Absolute Deviation (MAD), Max Pixel Difference (MPD), dan SSIM memiliki kompleksitas per blok sebesar $O(h \times w)$, sehingga tidak mempengaruhi kompleksitas algoritma secara keseluruhan. Namun, perhitungan error dengan metode Entropy dapat meningkatkan kompleksitas menjadi $O(n^4)$ dalam kasus terburuk karena kemungkinan banyaknya nilai piksel unik yang perlu diproses. Meskipun demikian, dalam praktiknya, kompleksitas sering kali tetap mendekati $O(n^2 \times \log_2 n)$ karena distribusi nilai piksel yang tidak sepenuhnya acak.

Berikut juga merupakan penjelasan implementasi bonus yang kami kerjakan:

1. Target Persentase Kompresi

Presentasi kompresi menunjukkan seberapa besar ukuran gambar berkurang dibandingkan dengan dengan ukuran aslinya setelah dikompresi menggunakan metode quadtree. Persentase kompresi dihitung sebagai berikut,

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

Implementasi persentase kompresi yang kami lakukan pada awalnya program akan menerima input target kompresi dalam bentuk *floating point* (0.0 - 1.0) dari masukan user. Jika input 0 maka mode ini tidak akan dijalankan, jika > 0 maka program akan mencari *threshold* yang sesuai dengan menggunakan metode *binary search*. Parameter minimum block size selalu bernilai 1 untuk mode ini. Penyesuaian threshold dilakukan dengan mendeklarasikan threshold maksimum dan minimum:

- [0, 128*128] untuk metode perhitungan galat berdasarkan pengukuran Variance
- [0, 255] untuk metode perhitungan galat berdasarkan MAD dan MPD
- [0, 8] untuk metode perhitungan galat berdasarkan Entropy
- [0, 1] untuk metode perhitungan galat berdasarkan SSIM.

Proses untuk mencapai target persentase kompresi dimulai dengan iterasi dari setengah *threshold* untuk membangun quadtree dan menghitung persentase kompresi yang dihasilkan. Jika kompresinya terlalu kecil, maka *threshold* akan dikurangi, sebaliknya jika terlalu besar, maka *threshold* akan dinaikkan. Proses ini terus berlanjut hingga kompresi mendekati target dengan toleransi 1%.

2. Structural Similarity Index (SSIM)

SSIM merupakan sebuah metode pengukuran error yang membandingkan dua gambar, biasanya antara gambar asli dengan gambar hasil kompresi. Berikut merupakan rumus SSIM untuk suatu gambar yang dihitung per kanal warnanya ($c \in \{R, G, B\}$):

$$SSIM_c(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

$$C_1 = (0,01 \times 255)^2$$

$$C_2 = (0,03 \times 255)^2$$

C_1 dan C_2 merupakan nilai konstan yang ditambahkan untuk mencegah instabilitas saat penyebut mendekati angka nol.

Kemudian nilai SSIM untuk tiap channel diambil rata-ratanya:

$$SSIM_{RGB} = \frac{SSIM_R + SSIM_G + SSIM_B}{3}$$

Namun, terdapat implementasi SSIM yang disederhanakan jika hanya mempunyai satu blok dan blok tersebut ingin dibandingkan dengan warna rata-rata:

$$SSIM = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \times \frac{C_2}{\sigma_x^2 + C_2}$$

Jika diimplementasikan pada program, nilai SSIM blok gambar dihitung pada tiap channelnya. Gabungan atau rata-rata dari nilai SSIM dari tiap channel antara 0 dan 1. Lalu nilai tersebut dibandingkan dengan parameter threshold:

- Jika SSIM lebih kecil dari threshold, blok belum cukup seragam sehingga proses divide dilakukan lagi
- Jika SSIM sudah sama dengan atau lebih besar dari threshold, proses divide diberhentikan.

3. Visualisasi GIF

Implementasi visualisasi GIF dari program bertujuan untuk melihat frame per frame proses *divide and conquer* yang dilakukan oleh program. Frame akan menunjukkan pembagian quadtree di setiap *depth* atau kedalaman. Untuk setiap *depth*, gambar direkonstruksi dengan menerapkan pembatasan *depth* maksimum pada fungsi `reconstructImageForGIF`. Blok yang ada pada *depth* lebih tinggi dari batas akan direpresentasikan oleh warna rata-rata node *rootnya*. Kami membuat GIF dengan menggabungkan rangkaian frame ini menggunakan library gif.h yaitu library GIF yang dibuat oleh Charlie Tangora.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Algoritma

a. Op.cpp

ALGORITMA

```
function hitungAverageColor(pixels)
{
    if pixels is empty then
        return Color(0, 0, 0)

    sumR, sumG, sumB ← 0
    for each pixel in pixels do
        sumR ← sumR + pixel.r
        sumG ← sumG + pixel.g
        sumB ← sumB + pixel.b

    count ← size of pixels
    return Color(sumR / count, sumG / count, sumB / count)
}

function hitungVariance(pixels, avgColor)
{
    if pixels is empty then
        return 0

    sumSqrDiffR, sumSqrDiffG, sumSqrDiffB ← 0
    for each pixel in pixels do
        diffR ← pixel.r - avgColor.r
        diffG ← pixel.g - avgColor.g
        diffB ← pixel.b - avgColor.b
        sumSqrDiffR ← sumSqrDiffR + diffR2
        sumSqrDiffG ← sumSqrDiffG + diffG2
        sumSqrDiffB ← sumSqrDiffB + diffB2

    count ← size of pixels
    return (sumSqrDiffR + sumSqrDiffG + sumSqrDiffB) / (3*count)
}

function hitungMAD(pixels, avgColor)
{
    if pixels is empty then
        return 0

    sumAbsDiffR, sumAbsDiffG, sumAbsDiffB ← 0
    for each pixel in pixels do
        sumAbsDiffR ← sumAbsDiffR + |pixel.r - avgColor.r|
```

```

        sumAbsDiffG ← sumAbsDiffG + |pixel.g -
avgColor.g|
        sumAbsDiffB ← sumAbsDiffB + |pixel.b -
avgColor.b|

        count ← size of pixels
        return (sumAbsDiffR + sumAbsDiffG + sumAbsDiffB) /
(3*count)
    }

function hitungMaxDifference(pixels)
{
    if pixels is empty then
        return 0

    minR, minG, minB ← 255
    maxR, maxG, maxB ← 0

    for each pixel in pixels do
        minR ← min(minR, pixel.r)
        maxR ← max(maxR, pixel.r)
        minG ← min(minG, pixel.g)
        maxG ← max(maxG, pixel.g)
        minB ← min(minB, pixel.b)
        maxB ← max(maxB, pixel.b)

    return ((maxR - minR) + (maxG - minG) + (maxB -
minB)) / 3
}

function hitungEntropy(pixels)
{
    if pixels is empty then
        return 0

    histogramR, histogramG, histogramB ← empty maps
    for each pixel in pixels do
        increment count of pixel.r in histogramR
        increment count of pixel.g in histogramG
        increment count of pixel.b in histogramB

    entropyR, entropyG, entropyB ← 0
    count ← size of pixels

    for each value in histogramR do
        prob ← value.count / count
        entropyR ← entropyR - (prob × log2(prob))

    for each value in histogramG do
        prob ← value.count / count
        entropyG ← entropyG - (prob × log2(prob))

```

```

        for each value in histogramB do
            prob ← value.count / count
            entropyB ← entropyB - (prob × log2(prob))

        return (entropyR + entropyG + entropyB) / 3
    }

function hitungSSIM(pixels, avgColor)
{
    if pixels is empty then
        return 0

    L ← 255
    C1 ← (0.03 × L)2

    sumSqrDiffR, sumSqrDiffG, sumSqrDiffB ← 0

    for each pixel in pixels do
        diffR ← pixel.r - avgColor.r
        diffG ← pixel.g - avgColor.g
        diffB ← pixel.b - avgColor.b
        sumSqrDiffR ← sumSqrDiffR + diffR2
        sumSqrDiffG ← sumSqrDiffG + diffG2
        sumSqrDiffB ← sumSqrDiffB + diffB2

        count ← size of pixels
        varR ← sumSqrDiffR / count
        varG ← sumSqrDiffG / count
        varB ← sumSqrDiffB / count

        ssimR ← C1 / (varR + C1)
        ssimG ← C1 / (varG + C1)
        ssimB ← C1 / (varB + C1)

    return (ssimR + ssimG + ssimB) / 3
}

function estimateThresholdForTargetCompression(image,
errorMethod, minBlockSize, targetCompression,
originalSize)
{
    low ← 0
    high ← if errorMethod = 1 then 1282
            else if errorMethod = 2 or 3 then 255
            else if errorMethod = 4 then 8
            else 1.0

    tolerance ← 0.01
    bestThreshold ← if errorMethod = 5 then high / 2 else
low
}

```

```

        for i ← 1 to 20 do
            mid ← (low + high) / 2
            qt ← new QuadTree
            qt.buildfrImage(image, errorMethod, mid,
minBlockSize)
            reconstructed ← qt.reconstructImage(image.width,
image.height)
            write reconstructed image to temp.jpg
            compressedSize ← size of temp.jpg
            delete temp.jpg

            compressionRatio ← 1 - (compressedSize /
originalSize)

            if |compressionRatio - targetCompression| ≤
tolerance then
                bestThreshold ← mid
                break

            if errorMethod = 5 then
                if compressionRatio < targetCompression then
                    low ← mid
                else
                    high ← mid
            else
                if compressionRatio < targetCompression then
                    high ← mid
                else
                    low ← mid

            bestThreshold ← mid

        return bestThreshold
    }
}

```

File ini berisi kumpulan fungsi yang digunakan untuk menunjang proses kompresi gambar berbasis metode Quadtree dengan algoritma Divide and Conquer. Fungsi-fungsi ini digunakan untuk menghitung nilai rata-rata warna suatu blok, mengukur tingkat keseragaman warna dalam blok (melalui metode error), merekonstruksi gambar, serta memperkirakan nilai threshold optimal untuk mencapai target kompresi tertentu.

Lima fungsi utama dalam file ini berkaitan dengan perhitungan metrik error. Fungsi `hitungAverageColor` digunakan untuk menghitung warna rata-rata dari sekumpulan piksel dalam blok, yang kemudian dijadikan referensi dalam penghitungan error. Fungsi `hitungVariance`, `hitungMAD`, dan `hitungMaxDifference` masing-masing menghitung nilai variance, mean absolute deviation (MAD), dan perbedaan maksimum antar warna terhadap rata-rata. Fungsi `hitungEntropy`

menghitung entropi dari distribusi warna blok, sedangkan `hitungSSIM` menggunakan pendekatan Structural Similarity Index yang disederhanakan.

Fungsi `estimateThresholdForTargetCompression` dipakai untuk menyesuaikan nilai threshold dengan target persentase kompresi yang diinginkan. Fungsi ini bekerja menggunakan binary search, dimana setiap iterasi akan menghasilkan quadtree baru, kemudian gambar direkonstruksi, dan file hasilnya disimpan sementara untuk mengetahui ukuran file yang dikompresi. Rasio kompresi kemudian dibandingkan dengan target, dan threshold disesuaikan sampai toleransi 0.01% tercapai.

Fungsi penunjang seperti `createQuadtreeGIF` digunakan untuk menghasilkan animasi GIF dari hasil pembentukan pohon quadtree. Gambar dihasilkan ulang pada setiap level kedalaman dengan memanggil metode `reconstructImageForGIF`, kemudian setiap frame GIF dibentuk dari gambar yang direkonstruksi. Fungsi `readImage` dan `writeImage` digunakan untuk membaca dan menulis gambar dari dan ke file, serta `getFileSize` dan `getFileExtension` yang membantu dalam proses estimasi ukuran file hasil kompresi.

b. Quadtree.cpp

ALGORITMA

```
procedure buildfrImage(image, errorMethod,
errorThreshold, minBlockSize)
{
    panjang ← lebar gambar (image[0].size)
    lebar ← tinggi gambar (image.size)
    root ← new QuadTreeNode(0, 0, panjang, lebar)
    totalN ← 1
    realWidth ← panjang
    realHeight ← lebar

    buildNode(root, image, errorMethod, errorThreshold,
minBlockSize, 0)
}

procedure buildNode(node, image, errorMethod,
errorThreshold, minBlockSize, currentDepth)
{
    if node = null then return
    maxDepth ← max (maxDepth, currentDepth)

    nodeX ← node.X
    nodeY ← node.Y
    nodePanjang ← node.panjang
    nodeLebar ← node.lebar

    blockPixels ← semua pixel dalam area (nodeX, nodeY,
nodePanjang, nodeLebar)
    avgColor ← hitungAverageColor(blockPixels)
```

```

node.avgColor ← avgColor

switch errorMethod:
    case 1: error ← hitungVariance(blockPixels,
avgColor)
        case 2: error ← hitungMAD(blockPixels, avgColor)
        case 3: error ← hitungMaxDifference(blockPixels)
        case 4: error ← hitungEntropy(blockPixels)
        case 5: error ← hitungSSIM(blockPixels, avgColor)
    default: error ← hitungVariance(blockPixels,
avgColor)

node.error ← error

if nodePanjang ≤ minBlockSize or nodeLebar ≤
minBlockSize then
    node.isLeaf ← true
    return

if (errorMethod = 5 and error ≥ errorThreshold) or
(errorMethod ≠ 5 and error ≤ errorThreshold) then
    node.isLeaf ← true
    return

node.split()
totalN ← totalN + 4

buildNode(node.topLeft, image, errorMethod,
errorThreshold, minBlockSize, currentDepth+1)
    buildNode(node.topRight, image, errorMethod,
errorThreshold, minBlockSize, currentDepth+1)
    buildNode(node.bottomLeft, image, errorMethod,
errorThreshold, minBlockSize, currentDepth+1)
    buildNode(node.bottomRight, image, errorMethod,
errorThreshold, minBlockSize, currentDepth+1)
}

procedure fillImage(image, node)
{
    if node = null then return

    if node.isLeaf then
        startX ← getX();
        startY ← getY();
        endX ← std::min(startX + getpanjang(),
(int)image[0].size());
        int endY = std::min(startY + getlebar(),
(int)image.size());

        for (int y = startY; y < endY; y++)
            for (int x = startX; x < endX; x++)

```

```

                image[y][x] ← getAvgColor();
        else
            fillImage(image, node.topLeft)
            fillImage(image, node.topRight)
            fillImage(image, node.bottomLeft)
            fillImage(image, node.bottomRight)
    }

int hitungCompressedSize()
{
    sizePerNode ← (2 * sizeof(int)) + (2 * sizeof(int))
+ (3 * sizeof(char)) + sizeof(bool);
    return totalN * sizePerNode;
}

procedure reconstructImageLimitedDepth(maxDepth)
{
    image(realHeight, std::vector<Color>(realWidth));
    reconstructLimitedHelper(root, image, 0, 0,
realWidth, realHeight, 0, maxDepth);
    return image;
}

procedure reconstructImageForGIF(depth)
{
    result(realHeight, std::vector<Color>(realWidth));
    bgColor ← getAvgColor();
    for (int y = 0; y < realHeight; y++)
        for (int x = 0; x < realWidth; x++)
            result[y][x] = bgColor;

    fillImageLimited(result, root, depth, 0);

    return result;
}

procedure fillImageLimited(image, node, maxDepth,
currentDepth)
{
    if (!node) return;

    if (node->isLeafNode() || currentDepth >= maxDepth)
    {
        startX ← getX();
        startY ← getY();
        endX ← min(startX + getpanjang(),
(int)image[0].size());
        int endY = std::min(startY + node->getlebar(),
(int)image.size());

        for (int y = startY; y < endY; y++)

```

```

        for (int x = startX; x < endX; x++)
            Image[y][x] ← getAvgColor();
    } else {
        fillImageLimited(image, getTopLeft(), maxDepth,
        currentDepth + 1);
        fillImageLimited(image, getTopRight(), maxDepth,
        currentDepth + 1);
        fillImageLimited(image, getBottomLeft(),
        maxDepth, currentDepth + 1);
        fillImageLimited(image, getBottomRight(),
        maxDepth, currentDepth + 1);
    }
}

```

File ini berisi algoritma utama untuk membangun dan merekonstruksi pohon Quadtree dalam proses kompresi gambar berbasis metode Divide and Conquer. Algoritma dimulai dari `buildfrImage`, yang menginisialisasi root node dari quadtree dan memanggil fungsi rekursif `buildNode` untuk membagi gambar ke dalam blok-blok lebih kecil. Proses pembagian dilakukan berdasarkan nilai error suatu blok dibandingkan dengan threshold tertentu. Jika ukuran blok sudah lebih kecil dari batas minimum (`minBlockSize`), atau nilai error telah memenuhi kriteria penghentian, maka node ditandai sebagai daun (`isLeaf`) dan tidak akan dibagi lagi. Jika tidak, node akan terus membagi menjadi empat kuadran (top-left, top-right, bottom-left, bottom-right).

Fungsi `fillImage` akan mengisi ulang piksel-piksel gambar berdasarkan warna rata-rata tiap node (blok) yang merupakan daun. Selain itu, `fillImageLimited` juga berguna dalam visualisasi proses pembentukan pohon. Sementara `hitungCompressedSize` berfungsi untuk mengestimasi ukuran data hasil kompresi, dengan asumsi bahwa tiap simpul (node) menyimpan informasi posisi, ukuran, rata-rata warna, dan status daun. Dengan mengalikan jumlah simpul dengan ukuran per simpul, fungsi ini memberikan gambaran kasar terhadap efisiensi penyimpanan struktur QuadTree jika disimpan sebagai data terstruktur.

c. Main.cpp

| |
|--|
| ALGORITMA |
| <pre> int main() { repeat print("Masukkan alamat absolut gambar yang ingin dikompresi:") inputFile ← bacaInput() until validateInputFile(inputFile) print("Pilih metode perhitungan error:") print("1 - Variance") </pre> |

```

print("2 - Mean Absolute Deviation")
print("3 - Max Pixel Difference")
print("4 - Entropy")
print("5 - Structural Similarity Index (SSIM) ")

repeat
    print("Pilihan kamu (1-5): ")
    errorMethod ← bacaInput()
until errorMethod=1 or errorMethod=2 or errorMethod=3 or
errorMethod=4 or errorMethod=5}

repeat
    print("Masukkan ambang batas (threshold): ")
    threshold ← bacaInput()
    validThreshold ← true

    if threshold < 0 then
        print("Threshold tidak bisa negatif")
        validThreshold ← false
    else if errorMethod = 1 and threshold > 128×128
then
    print("Threshold untuk Variance harus ≤
128²")
        validThreshold ← false
    else if (errorMethod=2 or errorMethod=3) and
threshold > 255 then
        print("Threshold untuk MAD atau Max
Difference harus ≤ 255")
        validThreshold ← false
    else if errorMethod=4 and threshold > 8 then
        print("Threshold Entropy harus ≤ 8")
        validThreshold ← false
    else if errorMethod=5 and threshold > 1 then
        print("Threshold SSIM harus ≤ 1")
        validThreshold ← false
until validThreshold

repeat
    print("Masukkan ukuran blok minimum: ")
    minBlockSize ← bacaInput()
until minBlockSize ≥ 1

repeat
    print("Masukkan target kompresi (0.0-1.0, 0 untuk
menonaktifkan): ")
    targetCompression ← bacaInput()
until 0.0 ≤ targetCompression ≤ 1.0

if targetCompression = 0.0 then
    isTarget ← false
else

```

```

        isTarget ← true
        minBlockSize ← 1

        repeat
            print("Masukkan alamat output gambar hasil
kompresi:")
            outputFile ← bacaInput()
            until validateOutputPath(outputFile)

            print("Masukkan alamat absolut GIF:")
            gifFile ← bacaInput()

            startTime ← waktu sekarang

            image ← bacaGambar(inputFile, width, height)
            if !readImage(inputFile, image, width, height) then
                print("Gagal baca gambar")
                return

            originalSize ← getFileSize(inputFile)

            if isTarget then
                threshold ←
estimateThresholdForTargetCompression(
                image, errorMethod, minBlockSize,
targetCompression, originalSize)

                quadtree ← objek QuadTree baru
                quadtree.buildfrImage(image, errorMethod, threshold,
minBlockSize)

                reconstructedImage ← quadtree.reconstructImage(width,
height)

                if !writeImage(outputFile, reconstructedImage) then
                    print("Gagal tulis file")
                    return

                if gifFile ≠ "" then
                    print("Memroses GIF...")
                    createQuadtreeGIF(gifFile, image, quadtree,
errorMethod, threshold, minBlockSize)

                endTime ← waktu sekarang
                duration ← endTime - startTime (dalam milidetik)

                compressedSize ← getFileSize(outputFile)
                compressionPercentage ← (1 - (compressedSize /
originalSize)) × 100

                // Tampilkan hasil

```

```

Cetak garis tabel
Cetak "HASIL KOMPRESI QUADTREE" di tengah
Cetak garis tabel

Cetak baris "Waktu eksekusi" ← duration dalam ms
Cetak baris "Ukuran gambar awal" ← originalSize
Cetak baris "Ukuran gambar akhir" ← compressedSize
Cetak baris "Persentase kompresi" ←
compressionPercentage %
Cetak baris "Kedalaman pohon" ← quadtree.maxDepth
Cetak baris "Banyak simpul" ← quadtree.totalN
Cetak garis

Cetak baris "Gambar tersimpan di" ← outputFile
Cetak garis

if gifFile ≠ "" then
    Cetak baris "GIF tersimpan di" ← gifFile
    Cetak garis

return
}

```

Pada fungsi main, pengguna diminta memasukkan path gambar, memilih metode error (Variance, MAD, Max Difference, Entropy, atau SSIM), serta menentukan nilai threshold, ukuran blok minimum, dan target persentase kompresi (opsional). Jika target kompresi diaktifkan, threshold dicari otomatis menggunakan binary search.

Gambar kemudian dibaca dan Quadtree dibangun menggunakan fungsi buildfriImage. Gambar hasil kompresi direkonstruksi dan disimpan dalam bentuk jpg, png, dan/atau gif. Program juga mencetak waktu eksekusi, ukuran file sebelum dan sesudah, persentase kompresi, kedalaman pohon, dan jumlah simpul yang terbuat.

B. Source Program

a. Op.cpp

```

#define STB_IMAGE_IMPLEMENTATION
#include "header/stb_image.h"
#include "header/quadtree.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "header/stb_image_write.h"
#define GIF_STATIC
#define GIF_IMPL
#include "header/gif.h"
#include "header/op.h"
#include <cmath>
#include <algorithm>

```

```

#include <map>
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <string>

Color hitungAverageColor(const std::vector<Color>& pixels) {
    if (pixels.empty()) return Color(0, 0, 0);

    unsigned long long sumR = 0, sumG = 0, sumB = 0;

    for (const auto& pixel : pixels) {
        sumR += pixel.r;
        sumG += pixel.g;
        sumB += pixel.b;
    }

    int count = pixels.size();
    return Color(sumR / count, sumG / count, sumB / count);
}

double hitungVariance(const std::vector<Color>& pixels, const Color&
avgColor) {
    if (pixels.empty()) return 0.0;

    double sumSqrDiffR = 0.0, sumSqrDiffG = 0.0, sumSqrDiffB = 0.0;

    for (const auto& pixel : pixels) {
        double diffR = pixel.r - avgColor.r;
        double diffG = pixel.g - avgColor.g;
        double diffB = pixel.b - avgColor.b;

        sumSqrDiffR += diffR * diffR;
        sumSqrDiffG += diffG * diffG;
        sumSqrDiffB += diffB * diffB;
    }

    int count = pixels.size();
    double varianceR = sumSqrDiffR / count;
    double varianceG = sumSqrDiffG / count;
    double varianceB = sumSqrDiffB / count;
}

```

```

        return (varianceR + varianceG + varianceB) / 3.0;
    }

double hitungMAD(const std::vector<Color>& pixels, const Color&
avgColor) {
    if (pixels.empty()) return 0.0;

    double sumAbsDiffR = 0.0, sumAbsDiffG = 0.0, sumAbsDiffB = 0.0;

    for (const auto& pixel : pixels) {
        sumAbsDiffR += std::abs(pixel.r - avgColor.r);
        sumAbsDiffG += std::abs(pixel.g - avgColor.g);
        sumAbsDiffB += std::abs(pixel.b - avgColor.b);
    }

    int count = pixels.size();
    double madR = sumAbsDiffR / count;
    double madG = sumAbsDiffG / count;
    double madB = sumAbsDiffB / count;

    return (madR + madG + madB) / 3.0;
}

double hitungMaxDifference(const std::vector<Color>& pixels) {
    if (pixels.empty()) return 0.0;

    unsigned char minR = 255, minG = 255, minB = 255;
    unsigned char maxR = 0, maxG = 0, maxB = 0;

    for (const auto& pixel : pixels) {
        minR = std::min(minR, pixel.r);
        minG = std::min(minG, pixel.g);
        minB = std::min(minB, pixel.b);

        maxR = std::max(maxR, pixel.r);
        maxG = std::max(maxG, pixel.g);
        maxB = std::max(maxB, pixel.b);
    }

    double diffR = maxR - minR;

```

```

        double diffG = maxG - minG;
        double diffB = maxB - minB;

        return (diffR + diffG + diffB) / 3.0;
    }

double hitungEntropy(const std::vector<Color>& pixels) {
    if (pixels.empty()) return 0.0;

    std::map<unsigned char, int> histR, histG, histB;

    for (const auto& pixel : pixels) {
        histR[pixel.r]++;
        histG[pixel.g]++;
        histB[pixel.b]++;
    }

    double entropyR = 0.0, entropyG = 0.0, entropyB = 0.0;
    int count = pixels.size();

    for (const auto& pair : histR) {
        double probability = static_cast<double>(pair.second) / count;
        entropyR -= probability * std::log2(probability);
    }

    for (const auto& pair : histG) {
        double probability = static_cast<double>(pair.second) / count;
        entropyG -= probability * std::log2(probability);
    }

    for (const auto& pair : histB) {
        double probability = static_cast<double>(pair.second) / count;
        entropyB -= probability * std::log2(probability);
    }

    return (entropyR + entropyG + entropyB) / 3.0;
}

double hitungSSIM(const std::vector<Color>& pixels, const Color&
avgColor) {
    if (pixels.empty()) return 0.0;
}

```

```

        double L = 255;
        double C1 = (0.03 * L) * (0.03 * L);

        double sumSqrDiffR = 0.0, sumSqrDiffG = 0.0, sumSqrDiffB = 0.0;

        for (const auto& pixel : pixels) {
            double diffR = pixel.r - avgColor.r;
            double diffG = pixel.g - avgColor.g;
            double diffB = pixel.b - avgColor.b;

            sumSqrDiffR += diffR * diffR;
            sumSqrDiffG += diffG * diffG;
            sumSqrDiffB += diffB * diffB;
        }

        int count = pixels.size();
        double varianceR = sumSqrDiffR / count;
        double varianceG = sumSqrDiffG / count;
        double varianceB = sumSqrDiffB / count;

        // SSIM per channel
        double ssimR = C1 / (varianceR + C1);
        double ssimG = C1 / (varianceG + C1);
        double ssimB = C1 / (varianceB + C1);

        return (ssimR + ssimG + ssimB) / 3.0;
    }

    size_t getFileSize(const std::string& filepath) {
        std::ifstream file(filepath, std::ios::binary | std::ios::ate);
        if (!file.is_open()) {
            return 0;
        }
        return file.tellg();
    }

    std::string getFileExtension(const std::string& filename) {
        size_t dotPos = filename.find_last_of('.');
        if (dotPos == std::string::npos || dotPos == filename.length() - 1)
    {

```

```

        return "";
    }

    std::string ext = filename.substr(dotPos + 1);
    for (char& c : ext) {
        c = std::tolower(c);
    }

    return ext;
}

bool readImage(const std::string& filename,
std::vector<std::vector<Color>>& image, int& width, int& height) {
    int channels;
    unsigned char* data = stbi_load(filename.c_str(), &width, &height,
&channels, 3);

    if (!data) {
        std::cerr << "Gagal memuat gambar:" << filename << std::endl;
        std::cerr << "Debug STB: " << stbi_failure_reason() <<
std::endl;
        return false;
    }
    image.resize(height, std::vector<Color>(width));

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int index = (y * width + x) * 3;
            image[y][x] = Color(data[index], data[index + 1], data[index
+ 2]);
        }
    }
    stbi_image_free(data);
    return true;
}

bool writeImage(const std::string& filename, const
std::vector<std::vector<Color>>& image) {
    if (image.empty() || image[0].empty()) {
        std::cerr << "Gambaranya kosong :" << std::endl;
        return false;
    }
}

```

```

    }

    std::cout << "Memroses gambar..." << std::endl;

    int height = image.size();
    int width = image[0].size();
    std::string extension = getFileExtension(filename);

    unsigned char* data = new unsigned char[width * height * 3];

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int index = (y * width + x) * 3;
            data[index] = image[y][x].r;
            data[index + 1] = image[y][x].g;
            data[index + 2] = image[y][x].b;
        }
    }

    bool success = false;

    if (extension == "png") {
        success = stbi_write_png(filename.c_str(), width, height, 3,
data, width * 3);
    }
    else if (extension == "jpg" || extension == "jpeg") {
        success = stbi_write_jpg(filename.c_str(), width, height, 3,
data, 90); // 90 kualitas (0-100)
    }
    else if (extension == "bmp") {
        success = stbi_write_bmp(filename.c_str(), width, height, 3,
data);
    }
    else if (extension == "tga") {
        success = stbi_write_tga(filename.c_str(), width, height, 3,
data);
    }
    else {
        std::cerr << "Format tidak didukung :" << extension <<
std::endl;
        std::cerr << "Format sudah salah satu dari png, jpg, jpeg, bmp,
tga belum? :)" << std::endl;
        delete[] data;
    }
}

```

```

        return false;
    }

    delete[] data;

    if (!success) {
        std::cerr << "Gagal write gambar: " << filename << std::endl;
        return false;
    }
    return true;
}

void createQuadtreeGIF(
    const std::string& outputPath,
    const std::vector<std::vector<Color>>& originalImage,
    QuadTree& quadtree,
    int errorMethod,
    double threshold,
    int minBlockSize
) {
    int maxDepth = quadtree.getMaxDepth();
    int width = originalImage[0].size();
    int height = originalImage.size();

    GifWriter gifWriter = {};
    GifBegin(&gifWriter, outputPath.c_str(), width, height, 100); // 100ms per frame

    for (int depth = 0; depth <= maxDepth; depth++) {
        std::vector<std::vector<Color>> frameImage =
        quadtree.reconstructImageForGIF(depth);

        std::vector<uint8_t> rgbaPixels(width * height * 4);

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int idx = (y * width + x) * 4;

                if (y < frameImage.size() && x < frameImage[y].size()) {
                    Color c = frameImage[y][x];
                    rgbaPixels[idx] = c.r;
                }
            }
        }
        GifWriteFrame(&gifWriter, &rgbaPixels[0], width, height);
    }
    GifEnd(&gifWriter);
}

```

```

        rgbaPixels[idx + 1] = c.g;
        rgbaPixels[idx + 2] = c.b;
        rgbaPixels[idx + 3] = 255;
    } else {
        rgbaPixels[idx] = 0;
        rgbaPixels[idx + 1] = 0;
        rgbaPixels[idx + 2] = 0;
        rgbaPixels[idx + 3] = 255;
    }
}
}

GifWriteFrame(&gifWriter, rgbaPixels.data(), width, height,
100);
}
}

double estimateThresholdForTargetCompression(
    const std::vector<std::vector<Color>>& image,
    int errorMethod,
    int minBlockSize,
    double targetCompression,
    int originalSize
) {
    double low = 0.0;
    double high = (errorMethod == 1) ? 128 * 128 : 1.0; // max threshold tergantung metode
    if (errorMethod == 2 || errorMethod == 3) high = 255;
    if (errorMethod == 4) high = 8.0;

    double tolerance = 0.01; // 1% toleransi
    double bestThreshold = (errorMethod == 5) ? high / 2 : low;

    for (int i = 0; i < 20; i++) {
        // std::cout << "Mencari threshold... " << (i+1) << std::endl;
        double mid = (low + high) / 2.0;

        QuadTree qt;
        qt.buildfrImage(image, errorMethod, mid, minBlockSize);
        std::vector<std::vector<Color>> reconstructed =
            qt.reconstructImage(image[0].size(), image.size());
    }
}
}

```

```

        std::string tempOut = "temp.jpg";
        writeImage(tempOut, reconstructed);
        int compressedSize = getFileSize(tempOut);
        std::remove(tempOut.c_str());

        double compressionRatio = 1.0 -
static_cast<double>(compressedSize) / originalSize;

        // std::cout << "Threshold: " << mid << std::endl;

        if (std::abs(compressionRatio - targetCompression) <= tolerance)
{
            bestThreshold = mid;
            break;
}

        if (errorMethod==5) {
            if (compressionRatio < targetCompression) {
                low = mid;
            } else {
                high = mid;
            }
        } else {
            if (compressionRatio < targetCompression) {
                high = mid;
            } else {
                low = mid;
            }
        }
        bestThreshold = mid;
}
// std::cout << "Final threshold" << bestThreshold << std::endl;
return bestThreshold;
}

```

b. Quadtree.cpp

```

#include "header/quadtree.h"
#include "header/op.h"
#include <algorithm>

```

```

#include <cmath>

QuadTreeNode::QuadTreeNode(int x, int y, int panjang, int lebar)
: x(x), y(y), panjang(panjang), lebar(lebar), isLeaf(true),
  topLeft(nullptr), topRight(nullptr), bottomLeft(nullptr),
  bottomRight(nullptr) {}

QuadTreeNode::~QuadTreeNode() {
    if (topLeft) delete topLeft;
    if (topRight) delete topRight;
    if (bottomLeft) delete bottomLeft;
    if (bottomRight) delete bottomRight;
}

void QuadTreeNode::split() {
    int halfPanjang = panjang / 2;
    int halfLebar = lebar / 2;

    topLeft = new QuadTreeNode(x, y, halfPanjang, halfLebar);
    topRight = new QuadTreeNode(x + halfPanjang, y, panjang -
    halfPanjang, halfLebar);
    bottomLeft = new QuadTreeNode(x, y + halfLebar, halfPanjang, lebar -
    halfLebar);
    bottomRight = new QuadTreeNode(x + halfPanjang, y + halfLebar,
    panjang - halfPanjang, lebar - halfLebar);

    isLeaf = false;
}

bool QuadTreeNode::hasChildren() const {
    return !isLeaf && topLeft != nullptr;
}

QuadTree::QuadTree() : root(nullptr), totalN(0), maxDepth(0) {}

QuadTree::~QuadTree() {
    if (root) delete root;
}

void QuadTree::buildfrImage(const std::vector<std::vector<Color>>&
image, int errorMethod, double errorThreshold, int minBlockSize) {

```

```

        if (image.empty() || image[0].empty()) return;
        int panjang = image[0].size();
        int lebar = image.size();

        realWidth = panjang;
        realHeight = lebar;

        root = new QuadTreeNode(0, 0, panjang, lebar);
        totalN = 1;
        // this->maxDepth = 0;

        buildNode(root, image, errorMethod, errorThreshold, minBlockSize,
0);
    }

void QuadTree::buildNode(QuadTreeNode* node, const
std::vector<std::vector<Color>>& image, int errorMethod, double
errorThreshold, int minBlockSize, int currentDepth) {
    if (!node) return;

    this->maxDepth = std::max(this->maxDepth, currentDepth);

    int nodeX = node->getX();
    int nodeY = node->getY();
    int nodePanjang = node->getpanjang();
    int nodeLebar = node->getlebar();

    std::vector<Color> blockPixels;
    for (int y = nodeY; y < nodeY + nodeLebar; ++y) {
        for (int x = nodeX; x < nodeX + nodePanjang; ++x) {
            if (y < image.size() && x < image[0].size())
                blockPixels.push_back(image[y][x]);
        }
    }

    Color avgColor = hitungAverageColor(blockPixels);
    node->setAvgColor(avgColor);

    double error = 0.0;
    switch (errorMethod) {
        case 1: error = hitungVariance(blockPixels, avgColor); break;

```

```

        case 2: error = hitungMAD(blockPixels, avgColor); break;
        case 3: error = hitungMaxDifference(blockPixels); break;
        case 4: error = hitungEntropy(blockPixels); break;
        case 5: error = hitungSSIM(blockPixels, avgColor); break;
        default: error = hitungVariance(blockPixels, avgColor); break;
    }

    node->setError(error);

    // Ini pengecekan minBlockSize-nya
    if (nodePanjang <= minBlockSize || nodeLebar <= minBlockSize) {
        node->setLeaf(true);
        return;
    }

    if ((errorMethod == 5 && error >= errorThreshold) ||
        (errorMethod != 5 && error <= errorThreshold)) {
        node->setLeaf(true);
        return;
    }

    node->split();
    totalN += 4;

    buildNode(node->getTopLeft(), image, errorMethod, errorThreshold,
minBlockSize, currentDepth + 1);
    buildNode(node->getTopRight(), image, errorMethod, errorThreshold,
minBlockSize, currentDepth + 1);
    buildNode(node->getBottomLeft(), image, errorMethod, errorThreshold,
minBlockSize, currentDepth + 1);
    buildNode(node->getBottomRight(), image, errorMethod,
errorThreshold, minBlockSize, currentDepth + 1);
}

std::vector<std::vector<Color>> QuadTree::reconstructImage(int panjang,
int lebar) {
    //buat gambar sesuai p l
    std::vector<std::vector<Color>> result(lebar,
std::vector<Color>(panjang));

    //node: root, maka isi warna hasil
    if (root) {

```

```

        fillImage(result, root);
    }

    return result;
}

void QuadTree::fillImage(std::vector<std::vector<Color>>& image,
QuadTreeNode* node) {
    if (!node) return;

    if (node->isLeafNode()) {
        //node: leaf, isi warna rata2
        int startX = node->getX();
        int startY = node->getY();
        int endX = std::min(startX + node->getpanjang(),
(int)image[0].size());
        int endY = std::min(startY + node->getlebar(),
(int)image.size());

        for (int y = startY; y < endY; y++) {
            for (int x = startX; x < endX; x++) {
                image[y][x] = node->getAvgColor();
            }
        }
    } else {
        //rekursif tiap anak
        fillImage(image, node->getTopLeft());
        fillImage(image, node->getTopRight());
        fillImage(image, node->getBottomLeft());
        fillImage(image, node->getBottomRight());
    }
}

int QuadTree::hitungCompressedSize() {
    // Compressed Size: total node dikali dengan size per node
    // size per node: posisi (2 int) + ukuran (2 int) + warna (3 byte) +
flag (1 byte)
    int sizePerNode = (2 * sizeof(int)) + (2 * sizeof(int)) + (3 *
sizeof(char)) + sizeof(bool);
    return totalN * sizePerNode;
}

```

```

std::vector<std::vector<Color>> QuadTree::reconstructImageForGIF(int
depth) {
    std::vector<std::vector<Color>> result(realHeight,
std::vector<Color>(realWidth));
    Color bgColor = root->getAvgColor();
    for (int y = 0; y < realHeight; y++) {
        for (int x = 0; x < realWidth; x++) {
            result[y][x] = bgColor;
        }
    }
    fillImageLimited(result, root, depth, 0);

    return result;
}

void QuadTree::fillImageLimited(std::vector<std::vector<Color>>& image,
QuadTreeNode* node, int maxDepth, int currentDepth) {
    if (!node) return;
    if (node->isLeafNode() || currentDepth >= maxDepth) {
        int startX = node->getX();
        int startY = node->getY();
        int endX = std::min(startX + node->getpanjang(),
(int)image[0].size());
        int endY = std::min(startY + node->getlebar(),
(int)image.size());

        for (int y = startY; y < endY; y++) {
            for (int x = startX; x < endX; x++) {
                image[y][x] = node->getAvgColor();
            }
        }
    } else {
        fillImageLimited(image, node->getTopLeft(), maxDepth,
currentDepth + 1);
        fillImageLimited(image, node->getTopRight(), maxDepth,
currentDepth + 1);
        fillImageLimited(image, node->getBottomLeft(), maxDepth,
currentDepth + 1);
        fillImageLimited(image, node->getBottomRight(), maxDepth,
currentDepth + 1);
    }
}

```

```

    }
}

void QuadTreeNode::setError(double err) {
    error = err;
}
double QuadTreeNode::getError() const {
    return error;
}

```

c. Main.cpp

```

#include "header/quadtreenode.h"
#include "header/op.h"
#include <iostream>
#include <string>
#include <chrono>
#include <iomanip>
#include <algorithm>
#include <fstream>
#include <cstdlib>

#define RESET      "\033[0m"
#define RED        "\033[31m"
#define GREEN      "\033[32m"
#define YELLOW     "\033[33m"
#define BLUE       "\033[34m"
#define MAGENTA    "\033[35m"
#define CYAN       "\033[36m"
#define BOLD        "\033[1m"

bool validateInputFile(const std::string& path) {
    std::ifstream file(path);
    if (!file.is_open()) {
        return false;
    }
    file.close();
    return true;
}

```

```

bool validateOutputPath(const std::string& path) {
    if (path.empty()) {
        return false;
    }
    std::ofstream file(path);
    if (!file.is_open()) {
        return false;
    }
    file.close();
    return true;
}

int main() {
    std::string inputFile;
    int errorMethod;
    double threshold;
    int minBlockSize;
    double targetCompression;
    std::string outputFile;
    std::string gifFile;

    // 1. [INPUT] alamat absolut gambar yang akan dikompresi
    do {
        std::cout << YELLOW << "Masukkan alamat absolut gambar yang
ingin dikompresi:\n> " << RESET;
        std::getline(std::cin, inputFile);
        if (!validateInputFile(inputFile)) {
            std::cerr << RED << "Alamat tidak valid atau file tidak
ditemukan :\n" << RESET;
        }
    } while (!validateInputFile(inputFile));

    std::cout << "Pilih metode perhitungan error yang diinginkan:" <<
std::endl;
    std::cout << "1 - Variance" << std::endl;
    std::cout << "2 - Mean Absolute Deviation" << 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;
    do {
        std::cout << "Pilihan kamu (1-5): ";

```

```

    std::cin >> errorMethod;
    if (errorMethod < 1 || errorMethod > 5) {
        std::cerr << "Pilihan tidak valid, seharusnya antara 1-5 :(\n";
    }
} while (errorMethod < 1 || errorMethod > 5);

bool validThreshold = false;
do {
    std::cout << "Masukkan ambang batas (threshold): ";
    std::cin >> threshold;

    validThreshold = true;

    if (threshold < 0) {
        std::cerr << "Threshold tidak bisa negatif :(" << std::endl;
        validThreshold = false;
    }
    if (errorMethod == 1 && threshold > (128*128)) {
        std::cerr << "Threshold metode Variance harus dari 0-128*128
:(" << std::endl;
        validThreshold = false;
    }
    if (errorMethod == 2 && threshold > 255) {
        std::cerr << "Threshold metode Mean Absolute Deviation harus
dari 0-255 :(" << std::endl;
        validThreshold = false;
    }
    if (errorMethod == 3 && threshold > 255) {
        std::cerr << "Threshold metode Max Pixel Difference harus
dari 0-255 :(" << std::endl;
        validThreshold = false;
    }
    if (errorMethod == 4 && threshold > 8) {
        std::cerr << "Threshold metode Entropy gabisa di atas 8 :(
<< std::endl;
        validThreshold = false;
    }
    if (errorMethod == 5 && threshold > 1) {
        std::cerr << "Threshold metode SSIM harus 0-1 yah :(" <<
std::endl;
        validThreshold = false;
    }
}

```

```

        }

    } while (!validThreshold);

    do {
        std::cout << "Masukkan ukuran blok minimum: ";
        std::cin >> minBlockSize;

        if (minBlockSize < 1) {
            std::cerr << "Ukuran blok minimal harus 1 :" << std::endl;
        }
    } while (minBlockSize < 1);

    // [CHECK] belum implement gimmick aja dulu
    do {
        std::cout << "Masukkan target kompresi (0.0-1.0, 0 untuk
menonaktifkan mode ini): ";
        std::cin >> targetCompression;

        if (targetCompression < 0 || targetCompression > 1.0) {
            std::cerr << "Target persentase seharusnya di antara 0.0 -
1.0" << std::endl;
        }
    } while (targetCompression < 0 || targetCompression > 1.0);

    bool isTarget;
    if (targetCompression == 0) {
        isTarget = false;
    } else {
        isTarget = true;
        minBlockSize = 1;
    }

    do {
        std::cin.ignore();
        std::cout << "Masukkan alamat absolut gambar hasil kompresi: ";
        std::getline(std::cin, outputFile);

        if (!validateOutputPath(outputFile)) {
            std::cerr << "Gagal write file :" << std::endl;
        }
    }
}

```

```

} while (!validateOutputPath(outputFile));

std::cout << "Masukkan alamat absolut GIF: ";
std::getline(std::cin, gifFile);

auto startTime = std::chrono::high_resolution_clock::now();
std::vector<std::vector<Color>> image;
int width, height;
if (!readImage(inputFile, image, width, height)) {
    std::cerr << "Gagal read gambar :" << std::endl;
    return 1;
}

size_t originalSize = getFileSize(inputFile);

if (isTarget) {
    threshold = estimateThresholdForTargetCompression(image,
errorMethod, minBlockSize, targetCompression, originalSize);
}

QuadTree quadtree;
quadtree.buildfrImage(image, errorMethod, threshold, minBlockSize);

std::vector<std::vector<Color>> reconstructedImage =
quadtree.reconstructImage(width, height);

if (!writeImage(outputFile, reconstructedImage)) {
    std::cerr << "Gagal write output :" << std::endl;
    return 1;
}

if (!gifFile.empty()) {
    std::cout << "Memroses GIF..." << std::endl;
    createQuadtreeGIF(gifFile, image, quadtree, errorMethod,
threshold, minBlockSize);
}

auto endTime = std::chrono::high_resolution_clock::now();
auto duration =
std::chrono::duration_cast<std::chrono::milliseconds>(endTime -

```

```

startTime) .count();

    // [CHECK]
    size_t compressedSize = getFileSize(outputFile);
    double compressionPercentage = (1.0 -
static_cast<double>(compressedSize) / originalSize) * 100.0;

    //output
    const int lebarTabel = 52;

    auto printLine = [](char ch = '-') {
        std::cout << CYAN << "+" << std::string(lebarTabel - 2, ch) <<
"+" << RESET << std::endl;
    };

    auto printCentered = [&](const std::string& msg) {
        int padding = (lebarTabel - 2 - msg.length()) / 2;
        std::cout << CYAN << "|" << std::string(padding, ' ')
            << BOLD << msg << RESET
            << std::string(lebarTabel - 2 - padding - msg.length(),
' ')
            << CYAN << "|" << RESET << std::endl;
    };

    auto printRow = [&](const std::string& label, const std::string&
value, const std::string& color = RESET) {
        int labelWidth = 24;
        int valueWidth = lebarTabel - 2 - labelWidth;
        std::cout << CYAN << "|" << RESET
            << std::left << std::setw(labelWidth) << label << ":" "
            << color << std::right << std::setw(valueWidth - 2) <<
value << RESET
            << " " << CYAN << "|" << RESET << std::endl;
    };

    printLine();
    printCentered("HASIL KOMPRESI QUADTREE");
    printLine();

    printRow("Waktu eksekusi", std::to_string(duration) + " ms", GREEN);
    printRow("Ukuran gambar awal", std::to_string(originalSize) + "

```

```

bytes", YELLOW);
    printRow("Ukuran gambar akhir", std::to_string(compressedSize) + " bytes", YELLOW);
    std::ostringstream compressionStream;
    compressionStream << std::fixed << std::setprecision(2) << compressionPercentage << " %";
    printRow("Persentase kompresi", compressionStream.str(), MAGENTA);
    printRow("Kedalaman pohon", std::to_string(quadtree.getMaxDepth()), BLUE);
    printRow("Banyak simpul", std::to_string(quadtree.getTotalNodes()), BLUE);
    printLine();

    printRow("Gambar tersimpan di", outputFile);
    printLine();

    if (!gifFile.empty()) {
        printRow("GIF tersimpan di", gifFile);
        printLine();
    }
    return 0;
}

```

C. Pengujian

a. Menguji dengan metode pengukuran error Variance



Gambar 4. snoopy.png

```

Masukkan alamat absolut gambar yang ingin dikompresi:
> C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\snoopy.png
Pilih metode perhitungan error yang diinginkan:
1 - Variance
2 - Mean Absolute Deviation
3 - Max Pixel Difference
4 - Entropy
5 - Structural Similarity Index (SSIM)
Pilihan kamu (1-5): 1
Masukkan ambang batas (threshold): 600
Masukkan ukuran blok minimum: 10
Masukkan target kompresi (0.0-1.0, 0 untuk menonaktifkan mode ini): 0
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\
test\snoopy_VARIANCE_600_10.png
Masukkan alamat absolut GIF: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\snoopy_VARIAN
CE_600_10.gif
Memroses gambar...
Memroses GIF...
+-----+
| HASIL KOMPRESI QUADTREE |
+-----+
| Waktu eksekusi      : 299 ms |
| Ukuran gambar awal   : 115929 bytes |
| Ukuran gambar akhir  : 9465 bytes |
| Persentase kompresi  : 91.84 % |
| Kedalaman pohon      : 6 |
| Banyak simpul        : 1373 |
+-----+
| Gambar tersimpan di E_600_10.png |
+-----+
| GIF tersimpan di     : C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\snoopy_VARIANC
E_600_10.gif |
+-----+

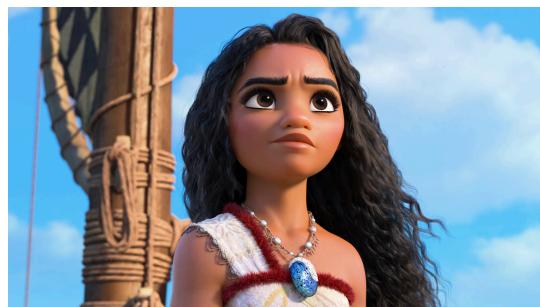
```

Gambar 5. Masukan dan hasil dari pengujian dengan metode error MAD pada snoopy.png



Gambar 6. Hasil pengujian pada snoopy.png

b. Menguji dengan metode pengukuran error Mean Absolute Deviation (MAD)



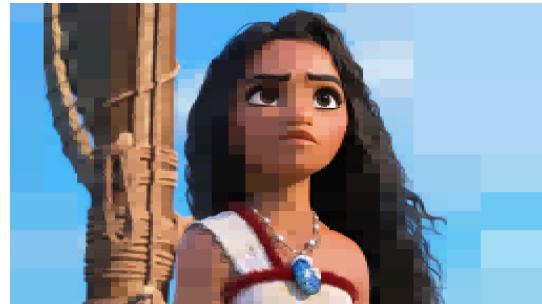
Gambar 7. moana.png

```

Masukkan alamat absolut gambar yang ingin dikompresi:
> C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\moana.png
Pilih metode perhitungan error yang diinginkan:
1 - Variance
2 - Mean Absolute Deviation
3 - Max Pixel Difference
4 - Entropy
5 - Structural Similarity Index (SSIM)
Pilihan kamu (1-5): 2
Masukkan ambang batas (threshold): 10
Masukkan ukuran blok minimum: 5
Masukkan target kompresi (0.0-1.0, 0 untuk menonaktifkan mode ini): 0
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\
test\moana_MAD_10_5.png
Masukkan alamat absolut GIF: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\moana_MAD_10_
5.gif
Memroses gambar...
Memroses GIF...
+-----+
|          HASIL KOMPRESI QUADTREE           |
+-----+
| Waktu eksekusi      :      4513 ms |
| Ukuran gambar awal   :  2361216 bytes |
| Ukuran gambar akhir  : 128008 bytes |
| Persentase kompresi  :     94.58 % |
| Kedalaman pohon      :          8 |
| Banyak simpul        :    22337 |
+-----+
| Gambar tersimpan di : C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\moana_MAD_10_5
.png |
+-----+
| GIF tersimpan di    : C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\moana_MAD_10_5
.gif |
+-----+

```

Gambar 8. Masukan dan hasil dari pengujian dengan metode error MAD pada moana.png



Gambar 9. Hasil pengujian pada moana.png

c. Menguji metode pengukuran error Max Pixel Difference



Gambar 10. joki.jpg

```

Masukkan alamat absolut gambar yang ingin dikompresi:
> C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\joki.jpg
Pilih metode perhitungan error yang diinginkan:
1 - Variance
2 - Mean Absolute Deviation
3 - Max Pixel Difference
4 - Entropy
5 - Structural Similarity Index (SSIM)
Pilihan kamu (1-5): 3
Masukkan ambang batas (threshold): 254
Masukkan ukuran blok minimum: 20
Masukkan target kompresi (0.0-1.0, 0 untuk menonaktifkan mode ini): 0
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\
test\joki_MPД_254_20.jpg
Masukkan alamat absolut GIF: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\joki_MPД_254_20.gif
Memroses gambar...
Memroses GIF...
+-----+
|          HASIL KOMPRESI QUADTREE          |
+-----+
| Waktu eksekusi      :      12678 ms |
| Ukuran gambar awal   :  1772564 bytes |
| Ukuran gambar akhir   :  176718 bytes |
| Persentase kompresi    :     90.03 % |
| Kedalaman pohon       :          7 |
| Banyak simpul        :        245 |
+-----+
| Gambar tersimpan di   : C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\joki_MPД_254_20.jpg |
+-----+
| GIF tersimpan di     : C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\joki_MPД_254_20.gif |
+-----+

```

Gambar 11. Masukan dan hasil dari pengujian dengan metode error MPD pada joki.jpg



Gambar 12. Hasil pengujian pada joki.jpg

d. Menguji metode pengukuran error Entropy



Gambar 13. benelli.jpg

```
Masukkan alamat absolut gambar yang ingin dikompresi:  
> C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\benelli.jpg  
Pilih metode perhitungan error yang diinginkan:  
1 - Variance  
2 - Mean Absolute Deviation  
3 - Max Pixel Difference  
4 - Entropy  
5 - Structural Similarity Index (SSIM)  
Pilihan kamu (1-5): 4  
Masukkan ambang batas (threshold): 6  
Masukkan ukuran blok minimum: 3  
Masukkan target kompresi (0.0-1.0, 0 untuk menonaktifkan mode ini): 0  
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\benelli_ENTROPY_6_3.jpg  
Masukkan alamat absolut GIF: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\benelli_ENTROPY_6_3.gif  
Memroses gambar...  
Memroses GIF...  
+-----+  
| HASIL KOMPRESI QUADTREE |  
+-----+  
| Waktu eksekusi : 15749 ms |  
| Ukuran gambar awal : 375560 bytes |  
| Ukuran gambar akhir : 365874 bytes |  
| Persentase kompresi : 2.58 % |  
| Kedalaman pohon : 8 |  
| Banyak simpul : 15253 |  
+-----+  
| Gambar tersimpan di : C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\benelli_ENTROPY_6_3.jpg |  
+-----+  
| GIF tersimpan di : C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\benelli_ENTROPY_6_3.gif |  
+-----+
```

Gambar 14. Masukan dan hasil dari pengujian dengan metode error Entropy pada benelli.jpg



Gambar 15. Hasil pengujian pada benelli.jpg

e. Menguji metode pengukuran error SSIM



Gambar 16. cursed-katanya.jpg

```

Masukkan alamat absolut gambar yang ingin dikompresi:
> C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\cursed-katanya.jpg
Pilih metode perhitungan error yang diinginkan:
1 - Variance
2 - Mean Absolute Deviation
3 - Max Pixel Difference
4 - Entropy
5 - Structural Similarity Index (SSIM)
Pilihan kamu (1-5): 5
Masukkan ambang batas (threshold): 0.7
Masukkan ukuran blok minimum: 10
Masukkan target kompresi (0.0-1.0, 0 untuk menonaktifkan mode ini): 0
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\
test\cursed-katanya_SSIM_07_10.jpg
Masukkan alamat absolut GIF: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\cursed-katanya_SSIM_07_10.gif
Memroses gambar...
Memroses GIF...
+-----+
| HASIL KOMPRESI QUADTREE |
+-----+
| Waktu eksekusi      : 3797 ms |
| Ukuran gambar awal   : 340413 bytes |
| Ukuran gambar akhir  : 180008 bytes |
| Persentase kompresi   : 47.12 % |
| Kedalaman pohon       : 7 |
| Banyak simpul        : 16861 |
+-----+
| Gambar tersimpan di   : C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\cursed-katanya_SSIM_07_10.jpg |
+-----+
| GIF tersimpan di     : C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\cursed-katanya_SSIM_07_10.gif |
+-----+

```

Gambar 17. Masukan dan hasil dari pengujian dengan metode error SSIM pada cursed-katanya.jpg



Gambar 18. Hasil pengujian pada cursed-katanya.jpg

f. Menguji masukkan target kompresi



Gambar 19. articmonkeysbw.jpg

```
Masukkan alamat absolut gambar yang ingin dikompresi:  
> C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\articmonkeysbw.jpg  
Pilih metode perhitungan error yang diinginkan:  
1 - Variance  
2 - Mean Absolute Deviation  
3 - Max Pixel Difference  
4 - Entropy  
5 - Structural Similarity Index (SSIM)  
Pilihan kamu (1-5): 5  
Masukkan ambang batas (threshold): 0.7  
Masukkan ukuran blok minimum: 10  
Masukkan target kompresi (0.0-1.0, 0 untuk menonaktifkan mode ini): 0.75  
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\  
test\articmonkeysbw_SSIM_07_10_TC.jpg  
Masukkan alamat absolut GIF: C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\articmonkeys  
bw_SSIM_07_10_TC.gif  
  
Memroses gambar...  
Memroses GIF...  
+-----+  
| HASIL KOMPRESI QUADTREE |  
+-----+  
| Waktu eksekusi : 154521 ms |  
| Ukuran gambar awal : 5452536 bytes |  
| Ukuran gambar akhir : 1854275 bytes |  
| Persentase kompresi : 65.99 % |  
| Kedalaman pohon : 11 |  
| Banyak simpul : 4919477 |  
+-----+  
| Gambar tersimpan di : C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\articmonkeysb  
w_SSIM_07_10_TC.jpg |  
+-----+  
| GIF tersimpan di : C:\Users\Lenovo\Documents\Tucil2_13523120_13523138\test\articmonkeysb  
w_SSIM_07_10_TC.gif |  
+-----+
```

Gambar 20. Masukan dan hasil dari pengujian target kompresi pada articmonkeysbw.jpg



Gambar 21. Hasil pengujian pada articmonkeysbw.jpg

BAB 5 KESIMPULAN DAN SARAN

A. Kesimpulan

Dengan algoritma pendekatan algoritma Divide and Conquer dan struktur data quadtree, penulis telah berhasil membuat sebuah program yang dapat mengkompresi gambar dengan pemberian parameter-parameter tertentu. Melalui penerapan berbagai metode pengukuran error (Variance, MAD, Max Difference, Entropy, dan SSIM), pengguna program diberikan fleksibilitas dalam menentukan tingkat ketelitian segmentasi gambar. Selain itu, pengguna dapat mengkompresi gambar dengan target kompresi tertentu sesuai dengan ruang penyimpanannya. Hasil akhir berupa gambar terkompresi dan visualisasi proses quadtree dalam bentuk GIF.

B. Saran

Saran yang penulis dapat berikan untuk pengembang lain yang ingin mengembangkan program yang serupa adalah:

- Optimalisasi pada file-file gambar yang ukurannya lebih besar, dan
- Terkhusus untuk metode pengukuran error SSIM, gunakan formula yang tidak disederhanakan, dan bandingkan nilai SSIM blok gambar dengan hasil kompresi.

LAMPIRAN

A. Github

https://github.com/bevindav/Tucil2_13523120_13523138

B. Tabel Pemeriksaan

| No | Poin | Ya | Tidak |
|----|---|----|-------|
| 1 | Program berhasil dikompilasi tanpa kesalahan | ✓ | |
| 2 | Program berhasil dijalankan | ✓ | |
| 3 | Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan | ✓ | |
| 4 | Mengimplementasi seluruh metode perhitungan error wajib | ✓ | |
| 5 | [Bonus] Implementasi persentase kompresi sebagai parameter tambahan | ✓ | |
| 6 | [Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error | ✓ | |
| 7 | [Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar | ✓ | |
| 8 | Program dan laporan dibuat (kelompok) sendiri | ✓ | |

REFERENSI

1. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)
2. <https://algo.monster/liteproblems/427>
3. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Makalah/Makalah-Stima-2023-\(69\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Makalah/Makalah-Stima-2023-(69).pdf)
4. <https://github.com/charlietangora/gif-h>