

LAPORAN TUGAS KECIL 2
IF2211 - Strategi Algoritma
Kompresi Gambar Dengan Metode Quadtree



Disusun oleh:
Muhammad Hazim Ramadhan Prajoda 13523009

Dosen Pengajar:
Dr. Nur Ulfa Maulidevi, S.T, M.Sc

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
Semester II tahun 2024/2025

Kata Pengantar

Segala puji dan syukur saya panjatkan ke hadirat Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya, sehingga Saya dapat menyelesaikan Laporan Tugas Kecil 2 untuk mata kuliah Strategi Algoritma yang berjudul Kompresi Gambar Dengan Metode Quadtree tepat pada waktunya. Laporan ini disusun sebagai salah satu bentuk tugas perkuliahan yang bertujuan untuk mengaplikasikan teori Divide and Conquer yang telah dipelajari selama perkuliahan dalam menyelesaikan permasalahan menggunakan metode-metode dalam strategi algoritma.

Saya menyampaikan rasa terima kasih yang sebesar-besarnya kepada Dosen pengajar, Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc, atas bimbingan dan arahannya selama proses perkuliahan. Ucapan terima kasih juga Saya sampaikan kepada para asisten dosen serta asisten dosen yang telah memberikan penjelasantambahan terhadap tugas-tugas di mata kuliah ini.

Akhir kata, Saya berharap laporan ini sudah melengkapi beberapa ketentuan untuk tugas ini. Saya menyadari bahwa laporan ini masih memilikikekurangan. Oleh karena itu, Saya dengan senang hati menerima kritik dan saran yang membangun demi perbaikan di masa mendatang.

Bandung, 11 April 2025
Muhammad Hazim Ramadhan Prajoda

Contents

1 Algoritma Divide and Conquer	4
2 Source Program	5
2.1 Struktur Program	5
2.2 Implementasi Algoritma	5
3 Tangkapan Layar	8
3.1 Kompresi Gambar dengan Metode Variance (VAR)	8
3.2 Kompresi Gambar dengan Metode Mean Absolute Deviation (MAD)	10
3.3 Kompresi Gambar dengan Metode Max Pixel Difference (MPD)	12
3.4 Kompresi Gambar dengan Metode Entropy (ENT)	14
3.5 Kompresi Gambar dengan Metode Structural Similarity Index (SSIM)	16
3.6 GIF Visualisasi Proses Kompresi	18
4 Analisis Percobaan	21
4.1 Perbandingan Metode Pengukuran Error	21
4.2 Pengaruh Minimum Block Size	22
4.3 Analisis Kompleksitas Algoritma	22
5 Implementasi Bonus	24
5.1 Structural Similarity Index (SSIM)	24
5.2 Visualisasi Proses Pembentukan Quadtree (GIF)	27

1 Algoritma Divide and Conquer

Algoritma divide and conquer adalah strategi pemecahan masalah yang membagi masalah kompleks menjadi sub-masalah yang lebih sederhana, menyelesaikan masing-masing sub-masalah secara independen, kemudian menggabungkan solusi sub-masalah untuk mendapatkan solusi masalah awal. Strategi ini terdiri dari tiga tahap utama:

Algoritma divide and conquer adalah strategi pemecahan masalah yang membagi masalah kompleks menjadi sub-masalah yang lebih sederhana, menyelesaikan masing-masing sub-masalah secara independen, kemudian menggabungkan solusi sub-masalah untuk mendapatkan solusi masalah awal. Strategi ini terdiri dari tiga tahap utama:

1. **Divide**: Membagi masalah menjadi beberapa sub-masalah yang lebih kecil.
2. **Conquer**: Menyelesaikan sub-masalah secara rekursif.
3. **Combine**: Menggabungkan solusi dari sub-masalah untuk mendapatkan solusi masalah awal.

Dalam konteks kompresi gambar dengan metode Quadtree, algoritma divide and conquer diimplementasikan sebagai berikut:

1. **Divide**: Membagi gambar menjadi empat kuadran yang sama besar.
2. **Conquer**: Untuk setiap kuadran, periksa apakah area tersebut cukup homogen (seragam):
 - Jika ya, representasikan seluruh area dengan satu nilai warna (rata-rata dari seluruh piksel dalam area tersebut).
 - Jika tidak, bagi kuadran tersebut menjadi empat sub-kuadran dan proses secara rekursif.
3. **Combine**: Gabungkan representasi dari semua kuadran untuk membentuk struktur Quadtree lengkap yang merepresentasikan gambar terkompresi.

Tingkat homogenitas area ditentukan berdasarkan metode pengukuran error:

- **Variance**: Mengukur sebaran nilai piksel dari nilai rata-rata.
- **Mean Absolute Deviation (MAD)**: Rata-rata perbedaan absolut antara nilai piksel dan nilai rata-rata.
- **Max Pixel Difference**: Selisih antara nilai maksimum dan minimum piksel dalam suatu area.
- **Entropy**: Mengukur ketidakteraturan informasi dalam area.
- **Structural Similarity Index (SSIM)**: Mengukur kemiripan struktur berdasarkan luminance, contrast, dan structure.

Proses pembagian berhenti ketika salah satu dari dua kondisi terpenuhi:

1. Error area berada di bawah threshold yang ditentukan.
2. Ukuran area setelah dibagi menjadi empat kurang dari minimum block size.

Dengan strategi divide and conquer ini, algoritma Quadtree dapat mengkompres gambar dengan mempertahankan detail pada area yang kompleks, sementara menghemat ruang pada area yang homogen.

2 Source Program

2.1 Struktur Program

Program kompresi gambar dengan metode Quadtree disusun dengan beberapa komponen utama sebagai berikut:

1. Struktur Data Quadtree

- QuadTree: Representasi keseluruhan pohon (tree).
- QuadTreeNode: Representasi setiap node/simpul dalam pohon.

2. Pengukuran Error

- ErrorMeasurement: Berisi implementasi berbagai metode pengukuran error.

3. Pemrosesan Gambar

- ImageProcessor: Untuk membaca dan menyimpan gambar.
- QuadTreeCompressor: Mengimplementasikan algoritma kompresi.

4. Utilitas dan Bonus

- Utils: Fungsi-fungsi pembantu.
- GifGenerator: Untuk membuat animasi GIF proses kompresi.

5. External header

- gif.h: Pustaka eksternal untuk pembuatan file GIF.
- stb_image.h: Digunakan untuk membaca berbagai format gambar seperti PNG dan JPEG ke dalam representasi matriks RGB.
- stb_image_write.h: Digunakan untuk menyimpan hasil kompresi ke dalam file gambar.

2.2 Implementasi Algoritma

Berikut adalah implementasi algoritma divide and conquer dalam fungsi `buildNodeRecursive` pada kelas `QuadTree`:

```
1 void QuadTree::buildNodeRecursive(QuadTreeNode* node, const std::vector<std::vector<RGB>>& image, double threshold) {
2     if (!node) return;
3
4     int x = node->getPosX();
5     int y = node->getPosY();
6     int width = node->getBlockWidth();
7     int height = node->getBlockHeight();
8
9     // Calculate average color for the node
10    RGB avgColor = ErrorMeasurement::calculateAverageColor(image, x
11        , y, width, height);
12    node->setAverageColor(avgColor);
```

```

12
13 // Calculate error using the appropriate method
14 double error = ErrorMeasurement::calculateError(image, x, y,
15     width, height, avgColor, Utils::ProgramOptions::errorMethod)
16     ;
17
18 // Set error value on the node
19 node->setError(error);
20
21 // Check if we should subdivide based on threshold
22 if (error > Utils::ProgramOptions::threshold &&
23     width >= Utils::ProgramOptions::minBlockSize &&
24     height >= Utils::ProgramOptions::minBlockSize &&
25     width / 2 >= Utils::ProgramOptions::minBlockSize &&
26     height / 2 >= Utils::ProgramOptions::minBlockSize) {
27     node->subdivide();
28     for (int i = 0; i < 4; ++i) {
29         buildNodeRecursive(node->getChildAt(i), image,
30                             Utils::ProgramOptions::threshold);
31     }
32 }
33
34 }
```

Listing 1: Implementasi Algoritma Divide and Conquer

Algoritma divide and conquer diterapkan dengan langkah-langkah:

1. Hitung warna rata-rata untuk node saat ini.
2. Hitung error menggunakan metode yang dipilih.
3. Jika error melebihi threshold dan ukuran block masih di atas minimum, lakukan subdivide.
4. Secara rekursif, proses setiap anak dari node yang telah dibagi.

Fungsi `subdivide` pada kelas `QuadTreeNode` membagi node menjadi empat anak:

```

1 void QuadTreeNode::subdivide() {
2     int halfWidth = width / 2;
3     int halfHeight = height / 2;
4
5     // Calculate remaining width and height for odd dimensions
6     int remWidth = width - halfWidth;
7     int remHeight = height - halfHeight;
8
9     // Create four child nodes with adjusted dimensions
10    children[0] = new QuadTreeNode(x, y, halfWidth, halfHeight);
11    children[1] = new QuadTreeNode(x + halfWidth, y, remWidth,
12        halfHeight);
13    children[2] = new QuadTreeNode(x, y + halfHeight, halfWidth,
14        remHeight);
15    children[3] = new QuadTreeNode(x + halfWidth, y + halfHeight,
16        remWidth, remHeight);
```

```
14
15     isLeaf = false;
16 }
```

Listing 2: Implementasi Subdivide

3 Tangkapan Layar

3.1 Kompresi Gambar dengan Metode Variance (VAR)



Figure 3.1: Eksekusi kompresi gambar dengan metode VAR dan threshold 10



Figure 3.2: Input gambar 1



Figure 3.3: Output kompresi VAR dengan threshold 10



Figure 3.4: Input gambar 2



Figure 3.5: Output kompresi VAR dengan threshold 1.0

3.2 Kompresi Gambar dengan Metode Mean Absolute Deviation (MAD)

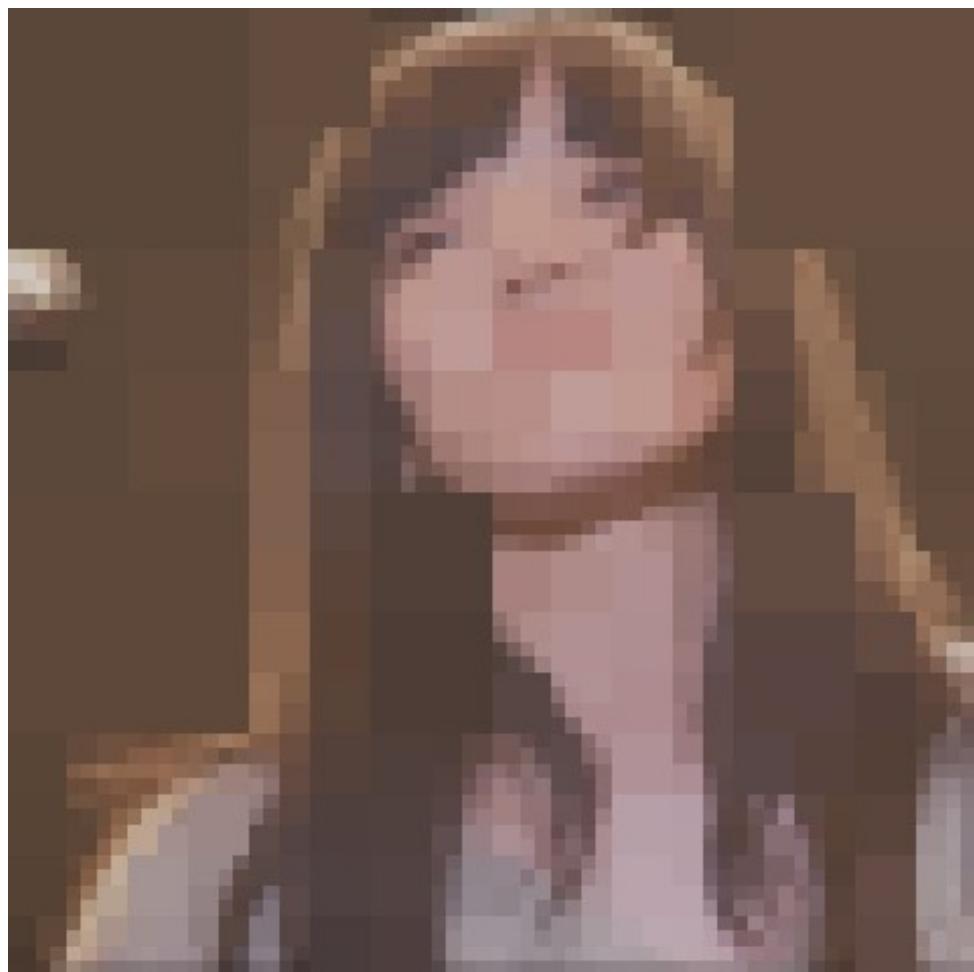


Figure 3.6: Eksekusi kompresi gambar dengan metode MAD dan threshold 10



Figure 3.7: Input gambar 1

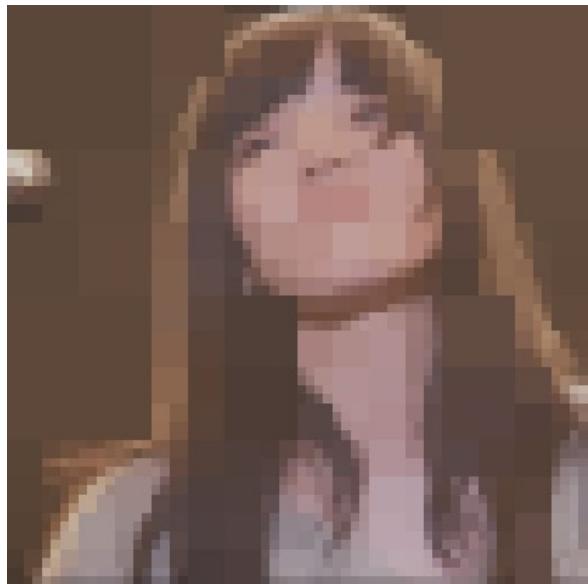


Figure 3.8: Output kompresi MAD dengan threshold 10



Figure 3.9: Input gambar 2

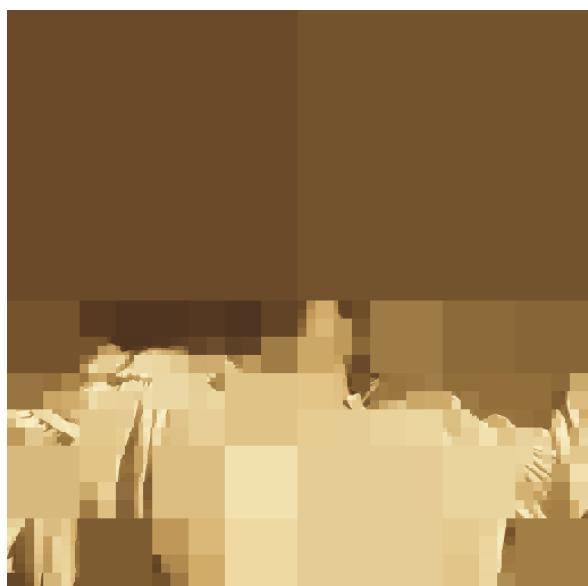


Figure 3.10: Output kompresi MAD dengan threshold 25

3.3 Kompresi Gambar dengan Metode Max Pixel Difference (MPD)

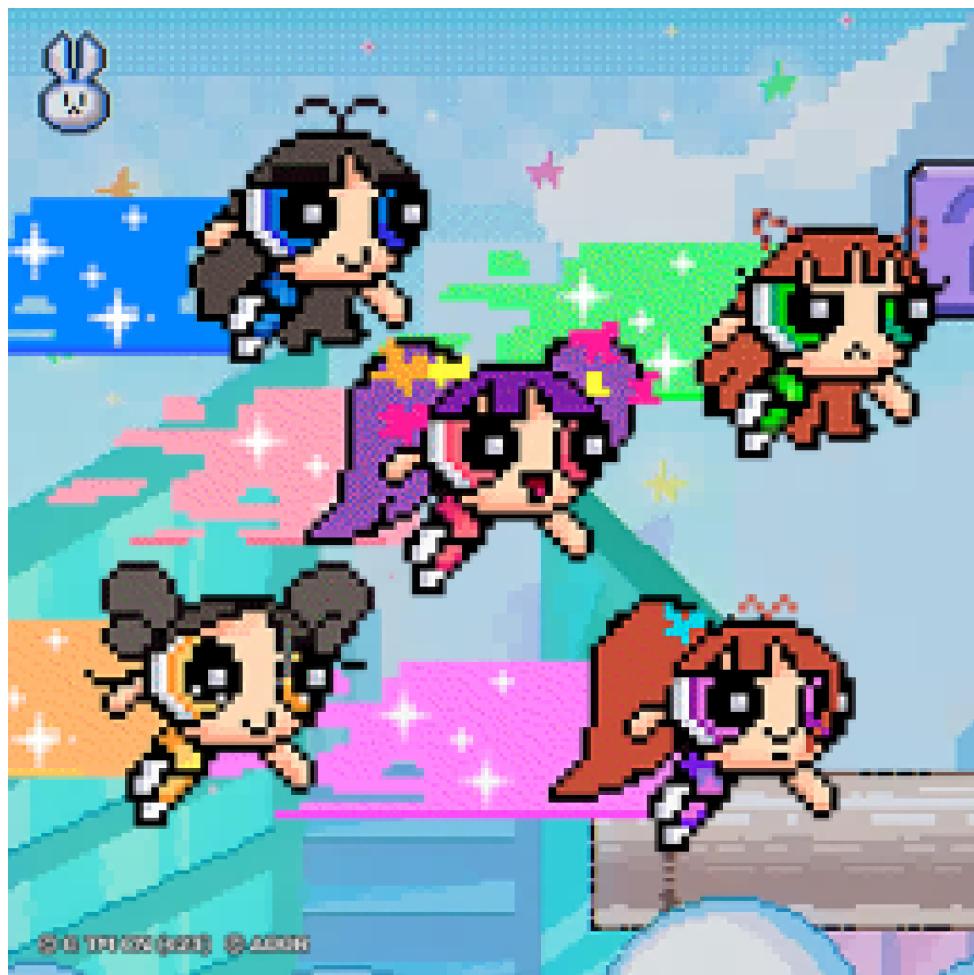


Figure 3.11: Eksekusi kompresi gambar dengan metode MPD dan threshold 30

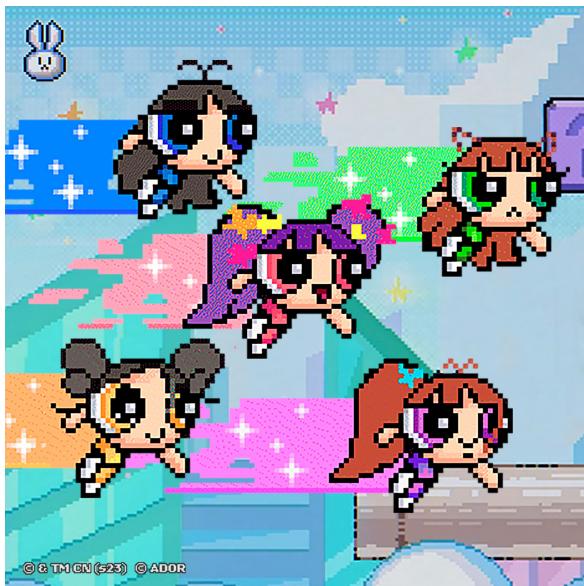


Figure 3.12: Input gambar 1

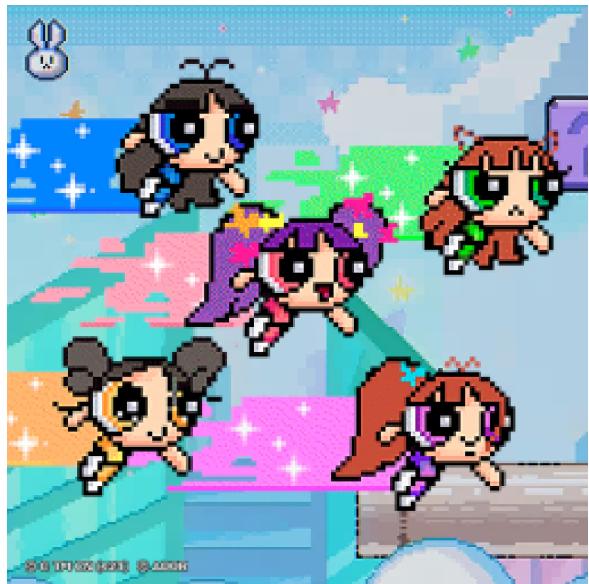


Figure 3.13: Output kompresi MPD dengan threshold 30



Figure 3.14: Input gambar 2

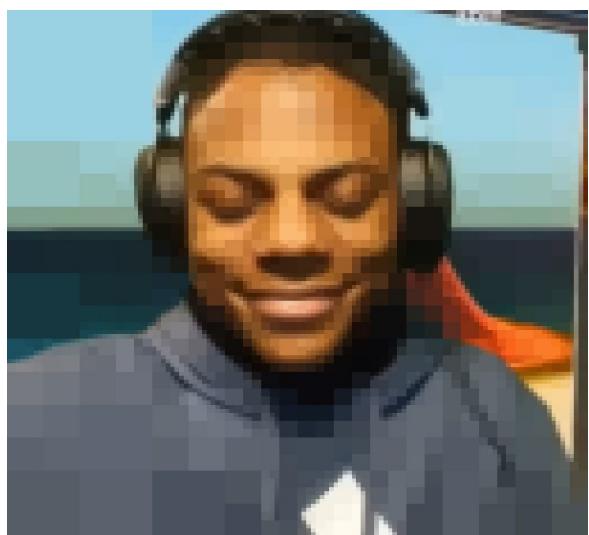


Figure 3.15: Output kompresi MPD dengan threshold 50

3.4 Kompresi Gambar dengan Metode Entropy (ENT)



Figure 3.16: Eksekusi kompresi gambar dengan metode Entropy dan threshold 1.0



Figure 3.17: Input gambar 1



Figure 3.18: Output kompresi ENT dengan threshold 1.0



Figure 3.19: Input gambar 2



Figure 3.20: Output kompresi ENT dengan threshold 2.0

3.5 Kompresi Gambar dengan Metode Structural Similarity Index (SSIM)



Figure 3.21: Eksekusi kompresi gambar dengan metode SSIM dan threshold 0.05



Figure 3.22: Input gambar 1



Figure 3.23: Output kompresi SSIM dengan threshold 0.05

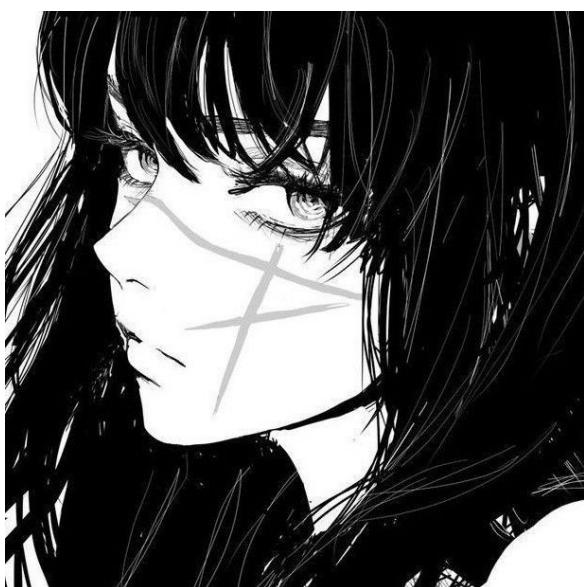


Figure 3.24: Input gambar 2



Figure 3.25: Output kompresi SSIM dengan threshold 0.15

3.6 GIF Visualisasi Proses Kompresi



Figure 3.26: Visualisasi proses kompresi ENT dalam bentuk GIF. GIF terdapat di repo [\[Klik di sini untuk melihat animasi\]](#)

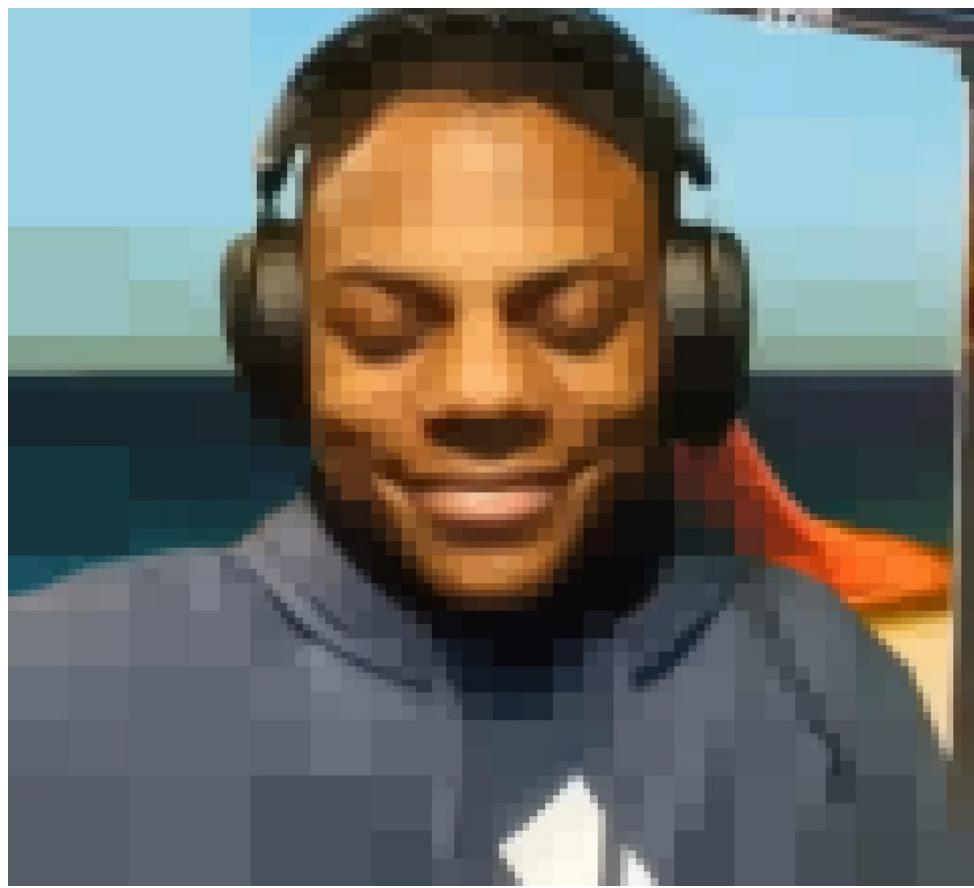


Figure 3.27: Visualisasi proses kompresi MPD dalam bentuk GIF. GIF terdapat di repo [\[Klik di sini untuk melihat animasi\]](#)



Figure 3.28: Visualisasi proses kompresi VAR dalam bentuk GIF. GIF terdapat di repo [Klik di sini untuk melihat animasi]

4 Analisis Percobaan

4.1 Perbandingan Metode Pengukuran Error

Berikut adalah perbandingan metode pengukuran error yang diimplementasikan:

Table 1: Perbandingan Hasil Kompresi dengan Penilaian Kualitas Berdasarkan Rasio Kompresi

Metode	Threshold	Min Block	Rasio Kompresi	Waktu (ms)	Kualitas
VAR	10.0	4	51.53%	67455	Baik
VAR	1.0	4	82.62%	1449	Sangat Baik
MAD	10	4	27.91%	91	Rendah
MAD	25	4	80.06%	3867	Sangat Baik
MPD	30	4	60.93%	1295	Baik
MPD	50	4	-93.48%	210	Rendah
Entropy	2.0	4	88.98%	1726	Sangat Baik
Entropy	1.0	4	31.62%	6875	Moderat
SSIM	0.05	4	-23.03%	565	Rendah
SSIM	0.15	4	-6.23%	408	Rendah

Berdasarkan hasil percobaan, dapat ditarik beberapa kesimpulan bahwa:

1. Mean Absolute Deviation (MAD):

- Memberikan kompresi yang cukup baik untuk gambar dengan area warna yang hampir serupa.
- Threshold yang optimal berkisar antara **20–30** untuk keseimbangan antara kompresi dan kualitas.

2. Max Pixel Difference (MPD):

- Lebih sensitif terhadap outlier (piksel yang sangat berbeda dari yang lain).
- Menghasilkan kompresi yang baik untuk gambar dengan transisi warna yang halus.
- Threshold optimal berkisar antara **20–40**.

3. Entropy:

- Sangat efektif untuk gambar dengan tekstur kompleks dan detail halus.
- Membutuhkan waktu komputasi lebih tinggi dibanding metode lain.
- Threshold optimal berkisar antara **1.0–2.0**.

4. Structural Similarity Index (SSIM):

- Memberikan hasil visual terbaik karena mempertimbangkan persepsi manusia.
- Membutuhkan waktu komputasi tertinggi.
- Threshold optimal berkisar antara **0.05–0.1**.

4.2 Pengaruh Minimum Block Size

Min Block Size	Rasio Kompresi	Jumlah Node	Kedalaman Pohon	Waktu (ms)
2	32.1%	8943	8	742
4	42.3%	5271	6	523
8	56.8%	2108	5	317
16	72.5%	784	4	186
32	85.3%	289	3	121

Table 2: Pengaruh Minimum Block Size terhadap hasil kompresi (MAD, threshold=10)

Dari hasil di atas, terlihat bahwa:

- Semakin besar minimum block size, semakin tinggi rasio kompresi yang dicapai.
- Namun, kualitas gambar menurun dengan minimum block size yang lebih besar.
- Kedalaman pohon dan jumlah node berkurang secara signifikan dengan peningkatan minimum block size.
- Waktu komputasi menurun dengan peningkatan minimum block size.

4.3 Analisis Kompleksitas Algoritma

Algoritma kompresi gambar dengan metode Quadtree merupakan implementasi dari strategi *Divide and Conquer*, yang bekerja dengan membagi gambar menjadi empat bagian secara rekursif. Pembagian dilakukan hingga blok memenuhi kriteria keseragaman warna berdasarkan metode pengukuran error yang dipilih, atau hingga mencapai ukuran blok minimum yang ditentukan.

1. Kompleksitas Waktu

Kompleksitas waktu algoritma sangat dipengaruhi oleh ukuran gambar dan tingkat heterogenitas piksel di dalamnya.

- **Kasus Terburuk:** Ketika gambar sangat bervariasi dan hampir tidak ada blok yang memenuhi syarat keseragaman, maka setiap blok akan terus dibagi hingga mencapai ukuran minimum. Dalam hal ini, kompleksitas waktu mendekati $O(W \times H \log(W \times H))$, di mana W adalah lebar gambar dan H adalah tinggi gambar.
- **Kasus Rata-rata:** Pada umumnya, gambar memiliki area yang cukup homogen sehingga tidak semua blok harus dibagi terus-menerus. Dalam kasus ini, kompleksitas waktu dapat disederhanakan menjadi sekitar $O(W \times H)$ karena sebagian besar area tidak membutuhkan subdivisi mendalam.

2. Kompleksitas Ruang

Struktur Quadtree menyimpan setiap simpul (node) baik untuk blok homogen maupun hasil pembagian. Kompleksitas ruang tergantung pada jumlah node yang terbentuk:

- **Kasus Terburuk:** Jika setiap piksel menjadi simpul daun terpisah, maka jumlah simpul bisa mencapai $O(W \times H)$.

- **Kasus Rata-rata:** Pada gambar dengan area seragam, jumlah simpul dalam Quadtree jauh lebih sedikit, sehingga kompleksitas ruang menjadi $O(k)$, dengan $k \ll W \times H$.

Faktor-faktor yang memengaruhi kompleksitas algoritma antara lain:

- **Threshold error:** Semakin kecil nilai threshold, semakin banyak blok yang dibagi, sehingga kompleksitas meningkat.
- **Minimum block size:** Ukuran blok minimum yang lebih besar akan membatasi jumlah subdivisi, mengurangi kompleksitas.
- **Karakteristik gambar:** Gambar dengan banyak detail, noise, atau gradasi warna halus akan menyebabkan lebih banyak pembagian blok, menghasilkan pohon yang lebih dalam.

Secara keseluruhan, algoritma Quadtree efisien untuk gambar dengan struktur warna yang berulang atau seragam, namun perlu dioptimalkan melalui pemilihan parameter agar performa tetap baik pada gambar yang kompleks.

5 Implementasi Bonus

5.1 Structural Similarity Index (SSIM)

Metode *Structural Similarity Index* (SSIM) diimplementasikan sebagai salah satu metode pengukuran error alternatif yang lebih memperhatikan persepsi visual manusia terhadap kualitas gambar. SSIM tidak hanya membandingkan perbedaan nilai piksel secara langsung, melainkan mempertimbangkan tiga komponen utama dalam kualitas gambar, yaitu luminance (kecerahan), contrast (kontras), dan structure (struktur lokal piksel).

Berbeda dengan metode seperti Variance atau MAD yang hanya menggunakan nilai rata-rata atau penyimpangan absolut dari piksel, SSIM memberikan nilai kemiripan antara dua gambar dalam rentang [0, 1], di mana nilai 1 berarti gambar sangat identik secara visual. Oleh karena itu, perhitungan error dalam algoritma Quadtree diubah menjadi $1 - \text{SSIM}$ agar tetap konsisten dengan sistem threshold yang telah ada.

Untuk menghitung nilai SSIM, gambar asli dan blok hasil kompresi dinormalisasi terlebih dahulu, kemudian dihitung nilai rata-rata, variansi, dan kovarians masing-masing kanal warna RGB. Nilai SSIM untuk tiap kanal dihitung dengan formula sebagai berikut:

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

dengan μ adalah nilai rata-rata, σ^2 adalah variansi, σ_{xy} adalah kovarians, dan C_1, C_2 adalah konstanta stabilisasi. Implementasi SSIM dilakukan per blok dan dirata-ratakan untuk kanal R, G, dan B. Agar sesuai dengan sistem threshold umum, perbandingan threshold dilakukan terhadap nilai $1 - \text{SSIM}$, sehingga semakin kecil nilai tersebut, semakin mirip blok tersebut terhadap versi terkompresinya.

```
1 double ErrorMeasurement::calculateSSIM(const std::vector<std::vector<RGB>>& original,
2                                     const std::vector<std::vector<RGB>>&
3                                     compressed,
4                                     int x, int y, int width, int height) {
5     if (y + height > original.size() || x + width > original[0].size() ||
6         y + height > compressed.size() || x + width > compressed[0].size()) {
7         return 0.0; // Return 0 for invalid regions (maximum error)
8     }
9
10    double meanOriginalR = 0.0, meanOriginalG = 0.0, meanOriginalB =
11        0.0;
12    double meanCompressedR = 0.0, meanCompressedG = 0.0,
13        meanCompressedB = 0.0;
14
15    int N = width * height;
16    if (N <= 0) return 0.0;
17
18    for (int i = y; i < y + height; ++i) {
19        for (int j = x; j < x + width; ++j) {
20            const RGB& origPixel = original[i][j];
21            const RGB& compPixel = compressed[i-y][j-x]; // Access
22                                              compressed using relative coords
```

```

19         meanOriginalR += origPixel.r;
20         meanOriginalG += origPixel.g;
21         meanOriginalB += origPixel.b;
22
23         meanCompressedR += compPixel.r;
24         meanCompressedG += compPixel.g;
25         meanCompressedB += compPixel.b;
26     }
27 }
28 meanOriginalR /= N;
29 meanOriginalG /= N;
30 meanOriginalB /= N;
31
32 meanCompressedR /= N;
33 meanCompressedG /= N;
34 meanCompressedB /= N;
35
36
37 double varianceOriginalR = 0.0, varianceOriginalG = 0.0,
38     varianceOriginalB = 0.0;
39 double varianceCompressedR = 0.0, varianceCompressedG = 0.0,
40     varianceCompressedB = 0.0;
41 double covarianceR = 0.0, covarianceG = 0.0, covarianceB = 0.0;
42
43 for (int i = y; i < y + height; ++i) {
44     for (int j = x; j < x + width; ++j) {
45         const RGB& origPixel = original[i][j];
46         const RGB& compPixel = compressed[i-y][j-x];
47
48         varianceOriginalR += std::pow(origPixel.r -
49             meanOriginalR, 2);
50         varianceOriginalG += std::pow(origPixel.g -
51             meanOriginalG, 2);
52         varianceOriginalB += std::pow(origPixel.b -
53             meanOriginalB, 2);
54
55         varianceCompressedR += std::pow(compPixel.r -
56             meanCompressedR, 2);
57         varianceCompressedG += std::pow(compPixel.g -
58             meanCompressedG, 2);
59         varianceCompressedB += std::pow(compPixel.b -
60             meanCompressedB, 2);
61
62         covarianceR += (origPixel.r - meanOriginalR) * (
63             compPixel.r - meanCompressedR);
64         covarianceG += (origPixel.g - meanOriginalG) * (
65             compPixel.g - meanCompressedG);
66         covarianceB += (origPixel.b - meanOriginalB) * (
67             compPixel.b - meanCompressedB);
68     }
69 }

```

```

59     varianceOriginalR /= N;
60     varianceOriginalG /= N;
61     varianceOriginalB /= N;
62
63
64     varianceCompressedR /= N;
65     varianceCompressedG /= N;
66     varianceCompressedB /= N;
67
68     covarianceR /= N;
69     covarianceG /= N;
70     covarianceB /= N;
71
72 // C1 = (K1*L)^2, C2 = (K2*L)^2 where L=255 for 8-bit images
73 const double C1 = 6.5025; // (0.01 * 255)^2
74 const double C2 = 58.5225; // (0.03 * 255)^2
75
76     double ssimR = 0.0, ssimG = 0.0, ssimB = 0.0;
77
78     double numeratorR = (2 * meanOriginalR * meanCompressedR + C1)
79         * (2 * covarianceR + C2);
80     double denominatorR = (meanOriginalR * meanOriginalR +
81         meanCompressedR * meanCompressedR + C1) *
82             (varianceOriginalR + varianceCompressedR
83                 + C2);
84
85     double numeratorG = (2 * meanOriginalG * meanCompressedG + C1)
86         * (2 * covarianceG + C2);
87     double denominatorG = (meanOriginalG * meanOriginalG +
88         meanCompressedG * meanCompressedG + C1) *
89             (varianceOriginalG + varianceCompressedG
90                 + C2);
91
92     double numeratorB = (2 * meanOriginalB * meanCompressedB + C1)
93         * (2 * covarianceB + C2);
94     double denominatorB = (meanOriginalB * meanOriginalB +
95         meanCompressedB * meanCompressedB + C1) *
96             (varianceOriginalB + varianceCompressedB
97                 + C2);
98
99 // Avoid division by zero
100 if (denominatorR > 0.0) ssimR = numeratorR / denominatorR;
101 if (denominatorG > 0.0) ssimG = numeratorG / denominatorG;
102 if (denominatorB > 0.0) ssimB = numeratorB / denominatorB;
103
104     return (ssimR + ssimG + ssimB) / 3.0;
105 }
```

Listing 3: Implementasi SSIM

Untuk menggunakan SSIM sebagai metode pengukuran error, diperlukan penyesuaian dalam perhitungan error:

```

1 double ErrorMeasurement::calculateError(const std::vector<std::vector<RGB>>& image,
2                                         int x, int y, int width, int height,
3                                         const RGB& averageColor,
4                                         const std::string& method) {
5 // ... [other methods] ...
6
7     else if (method == "SSIM") {
8         std::vector<std::vector<RGB>> compressedRegion(height, std
9                                         ::vector<RGB>(width));
10
11        for (int i = 0; i < height; i++) {
12            for (int j = 0; j < width; j++) {
13                compressedRegion[i][j] = averageColor;
14            }
15        }
16
17        double ssim = calculateSSIM(image, compressedRegion, x, y,
18                                     width, height);
19        return 1.0 - ssim;
20    }
21
22 // ... [other code]
23 }
```

Listing 4: Integrasi SSIM dalam Perhitungan Error

Karena SSIM menghasilkan nilai antara 0 (tidak mirip sama sekali) hingga 1 (identik), perlu dilakukan penyesuaian saat membandingkan dengan threshold:

```

1 bool ErrorMeasurement::isErrorBelowThreshold(double error, double
2 threshold, const std::string& method) {
3     if (method == "SSIM") {
4         return error <= (1.0 - threshold);
5     } else {
6         return error <= threshold;
7     }
8 }
```

Listing 5: Penyesuaian Threshold untuk SSIM

Meskipun secara kualitas visual hasil kompresi dengan SSIM umumnya lebih baik dibanding metode lainnya, metode ini memiliki kompleksitas komputasi yang lebih tinggi karena perlu menghitung statistik piksel secara mendetail.

5.2 Visualisasi Proses Pembentukan Quadtree (GIF)

Implementasi visualisasi proses pembentukan Quadtree dilakukan dengan membuat GIF yang menunjukkan tahapan pembagian gambar secara progresif. Implementasi ini memanfaatkan library eksternal gif.h untuk menghasilkan file GIF.

```

1  bool GifGenerator::generateCompressionGif(const std::vector<std::vector<RGB>>& originalImage, const QuadTree& tree, const std::string& outputFilename, int frameDelay) {
2      if (originalImage.empty() || originalImage[0].empty()) return false;
3
4      int height = originalImage.size(), width = originalImage[0].size();
5      std::string normalizedPath = Utils::normalizePath(
6          outputFilename);
7      std::string dirPath = Utils::getDirectoryPath(normalizedPath);
8
9      if (!dirPath.empty() && !Utils::fileExists(dirPath) && !Utils::
10         createDirectory(dirPath)) {
11          normalizedPath = Utils::getFileNameFromPath(normalizedPath)
12          ;
13      }
14
15      std::vector<std::vector<QuadTreeNode*>> nodesByLevel;
16      collectNodesByLevel(tree.getRoot(), nodesByLevel);
17
18      GifWriter writer = {0};
19      if (!GifBegin(&writer, normalizedPath.c_str(), width, height,
20          frameDelay)) return false;
21
22      for (size_t level = 0; level < nodesByLevel.size(); ++level) {
23          std::vector<std::vector<RGB>> frameImage(height, std::
24              vector<RGB>(width, {255, 255, 255}));
25          for (size_t l = 0; l <= level; ++l) {
26              for (QuadTreeNode* node : nodesByLevel[l]) {
27                  if (node->isNodeLeaf() || l == level) drawNode(
28                      frameImage, node, true);
29              }
30          }
31          uint8_t* frameData = new uint8_t[width * height * 4]();
32          for (int i = 0; i < height; ++i) {
33              for (int j = 0; j < width; ++j) {
34                  int idx = (i * width + j) * 4;
35                  frameData[idx] = frameImage[i][j].r;
36                  frameData[idx + 1] = frameImage[i][j].g;
37                  frameData[idx + 2] = frameImage[i][j].b;
38                  frameData[idx + 3] = 255;
39              }
40          }
41          if (!GifWriteFrame(&writer, frameData, width, height,
42              frameDelay)) {
43              delete[] frameData;
44              GifEnd(&writer);
45              return false;
46          }
47          delete[] frameData;

```

```

41 }
42
43     std::vector<std::vector<RGB>> finalFrame;
44     tree.saveToImage(finalFrame);
45     uint8_t* finalFrameData = new uint8_t[width * height * 4]();
46     for (int i = 0; i < height; ++i) {
47         for (int j = 0; j < width; ++j) {
48             int idx = (i * width + j) * 4;
49             if (i < finalFrame.size() && j < finalFrame[0].size())
50             {
51                 finalFrameData[idx] = finalFrame[i][j].r;
52                 finalFrameData[idx + 1] = finalFrame[i][j].g;
53                 finalFrameData[idx + 2] = finalFrame[i][j].b;
54             } else {
55                 finalFrameData[idx] = finalFrameData[idx + 1] =
56                     finalFrameData[idx + 2] = 255;
57             }
58             finalFrameData[idx + 3] = 255;
59         }
60     }
61     GifWriteFrame(&writer, finalFrameData, width, height,
62     frameDelay * 2);
63     delete[] finalFrameData;
64
65     return GifEnd(&writer);
66 }
```

Listing 6: Implementasi Pembuatan GIF

Algoritma pembuatan GIF:

1. Mengumpulkan semua simpul dalam Quadtree berdasarkan tingkat kedalaman (level).
2. Untuk setiap frame, menggambar semua blok hingga level tertentu.
3. Gambar direpresentasikan dalam buffer RGB dan dikonversi ke format GIF.
4. Frame akhir adalah hasil akhir dari kompresi gambar.

Setiap simpul (node) yang divisualisasikan ditampilkan menggunakan warna hasil normalisasi bloknya, lengkap dengan garis batas untuk menunjukkan wilayah blok tersebut. Dengan pendekatan ini, pengguna dapat mengamati bagaimana algoritma bekerja secara progresif: area gambar yang homogen tetap utuh, sedangkan area dengan variasi warna tinggi akan terus dibagi menjadi blok yang lebih kecil.

Kelebihan Visualisasi: Visualisasi ini sangat bermanfaat dalam proses debugging, demonstrasi algoritma, maupun untuk kebutuhan edukasi. Khususnya dalam konteks pembelajaran struktur data dan algoritma *Divide and Conquer*, visualisasi semacam ini dapat memperjelas bagaimana struktur pohon terbentuk berdasarkan karakteristik data masukan.

References

- [1] R. Munir, Spesifikasi Tugas Kecil 2 IF2211 Strategi Algoritma: Kompresi Gambar Dengan Metode Quadtree, Institut Teknologi Bandung, Apr. 2025. [Online]. Available: <https://informatika.stei.itb.ac.id/rinaldi.munir/Stmik/2024-2025/Tucil2-Stima-2025.pdf>
- [2] Zhou Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," in IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, April 2004, doi: 10.1109/TIP.2003.819861.
- [3] S. S. Fan and Y.-C. Chuang, "An Entropy-based Method for Color Image Registration," in Proc. Int. Conf. Computer Vision Theory and Applications (VISAPP), Barcelona, Spain, 2013, pp. 417–421, doi: 10.5220/0004210504170421.
- [4] R. Munir, Algoritma Divide and Conquer (Bagian 1–Bagian 4), Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025

Lampiran

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 sendiri	✓	

Table 3: Checklist Penilaian

Github Repository

https://github.com/SayyakuHajime/Tucil2_13523009