

Laporan Tugas Kecil 2 IF 2211

Strategi Algoritma

Penerapan Algoritma Divide and Conquer dalam Compressing Image



Oleh:

Richard Christian - 13523024

Kenneth Poenadi - 13523040

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132**

2025

Daftar Isi

BAB 1	3
Latar Belakang	3
BAB 2	4
Notasi Pseudocode	4
2.1 Notasi Algoritmik image.cpp dan image.hpp	4
2.1.1 Image.hpp	4
2.1.2 Image.cpp	4
2.2 Notasi Algoritmik quadtree.hpp dan quadtree.cpp	7
2.2.1 Quadtree.hpp	7
2.2.2 Quadtree.cpp	8
2.3 Notasi Algoritmik utils.hpp dan utils.cpp	12
2.3.1 utils.cpp	12
2.4 Notasi Algoritmik Bonus	18
2.4.1 Bonus.cpp	18
2.5 Notasi Algoritmik Gifer.cpp	21
2.5.1 Gifer.cpp	21
BAB 3	23
Penjelasan Algoritma	23
3.1 Penjelasan Algoritma Divide and Conquer yang digunakan	23
Gambar 1. Data Type dan Keterangan Min dan MAX Value pada C++	24
Gambar 2. Visualisasi QuadTree	25
3.2 Penjelasan Algoritma BONUS	27
1. SSIM (Bonus)	27
2. Persentase Kompresi	28
3. GIF	30
3.3. Source Code Program	30
BAB 4	57
Testing	57
4.1 Hasil Kompresi dan Output Program	57
4.2 Analisis dan Pembahasan kompleksitas algoritma	75
BAB 5	76
KESIMPULAN & SARAN	76
LAMPIRAN	77

BAB 1

Latar Belakang

Gambar dalam jumlah besar dapat menghabiskan banyak memori. Ini menjadi masalah ketika gambar-gambar tersebut mungkin tidak dilihat dalam skala penuh atau jarang digunakan, karena membuang-buang memori dan daya pemrosesan untuk mempertahankannya dalam kualitas penuh. Selain itu, gambar memiliki area besar dengan piksel yang hanya merupakan duplikasi dari banyak tetangganya, sehingga menyebabkan pemborosan memori untuk menyimpan informasi yang sebenarnya berulang. Masalah ini dapat diatasi dengan kompresi gambar.

Sebagian besar metode kompresi gambar membantu mengatasi masalah ini dengan mengurangi ukuran gambar, sehingga mengurangi jumlah piksel dan memori yang digunakan. Metode ini memang efektif, tetapi jika kita ingin mendapatkan kembali gambar dengan kualitas tinggi, sering kali itu menjadi tidak mungkin. Untungnya, ada opsi lain yang lebih baik, salah satunya adalah quadtree.

Quadtree dapat mengatasi masalah yang disebutkan di atas tanpa kehilangan kualitas gambar yang signifikan seperti metode kompresi lainnya. Selain itu, quadtree juga mempercepat proses pembelajaran mesin dalam deteksi tepi gambar.

BAB 2

Notasi Pseudocode

2.1 Notasi Algoritmik image.cpp dan image.hpp

2.1.1 Image.hpp

```
struct Pixel
    r: unsigned char
    g: unsigned char
    b: unsigned char
    a: unsigned char

struct Image
    width: integer
    height: integer
    channels: integer
    pixels: Matriks (vector<vector<Pixel>>)
```

2.1.2 Image.cpp

```
function loadImage(filename: string) → image
    KAMUS LOKAL:
        img: Image
        data: pointer ke array byte
        data ← stbi_load(filename, img.width, img.height, img.channels, 4)
        if(data = NULL) then
            output("Error: Tidak dapat memuat gambar " + filename)
            img.width ← 0, img.height ← 0, img.channels ← 0
            → img
        img.channels ← 3
        ALOKASI img.pixels dengan ukuran (img.height * img.width)
        y traversal 0...img.height - 1
            x traversal 0... img.width - 1
            index ← (y * img.width + x) * img.channels
            img.pixels[y][x] ← {data[index], data[index+1], data[index+2], }
        free(data (stbi_image_free(data)))
        → img
```

```
function checkFile(filename: string) → boolean
```

```

file ← fopen(filename, "rb")
  if(file ≠ NULL) then
    fclose(file)
    → true
  → false

function saveReconstructedImage(filename: string, image: vector<vector<Pixel>>, format: string)
KAMUS LOKAL:
  height, width, channels, index: integer
  isGrayscale: boolean
  data: pointer ke array byte
  p: Pixel
  x, y: integer

  height ← image.size()
  width ← image[0].size()

  isGrayscale ← true
  for y ← 0 to height - 1 and isGrayscale do
    for x ← 0 to width - 1 do
      p ← image[y][x]
      if (p.r ≠ p.g or p.g ≠ p.b) then
        isGrayscale ← false
        break
      endif
    endfor
  endfor

  if (isGrayscale) then
    channels ← 1
  else
    channels ← 3
  endif

  ALOKASI data dengan ukuran width * height * channels

  Y traversal 0... height - 1
    x traversal 0 ... width - 1
      p ← image[y][x]
      index ← (y * width + x) * channels

      if (isGrayscale) then
        data[index] ← p.r
      else

```

```

        data[index] ← p.r
        data[index + 1] ← p.g
        data[index + 2] ← p.b
    endif
endfor
endfor

if (format = "png") then
    if (not stbi_write_png(filename + ".png", width, height, channels, data, width * channels)) then
        output("Gagal save PNG")
    else
        output("=====")
        output("")
        output("PNG berhasil disave")
        output("")
        output("=====")
    endif
else if (format = "jpg") then
    if (not stbi_write_jpg(filename + ".jpg", width, height, channels, data, 50)) then
        output("Gagal save JPG")
    else
        output("=====")
        output("")
        output("JPG berhasil disave")
        output("")
    endif
else if (format = "jpeg") then
    if (not stbi_write_jpg(filename + ".jpeg", width, height, channels, data, 50)) then
        output("Gagal save JPEG")
    else
        output("=====")
        output("")
        output("JPEG berhasil disave")
        output("")
    endif
endif
endif

delete[] data

```

function getFileExtension(filename: string) → string

KAMUS LOKAL:

dotPos: integer

dotPos ← posisi terakhir karakter "." dalam filename

```
if (dotPos = -1) then
    return ""
endif
```

→ substring dari filename mulai dotPos + 1 hingga akhir

```
function getFileSize(fileName: string) → double
```

KAMUS LOKAL:

file: ifstream

```
file ← buka file(fileName, binary mode + ate mode)
```

→ file.tellg() / 1024

```
function calculateCompression(originalSize: double, compressedSize: double) → double
```

→ (1 - (compressedSize / originalSize)) * 100

2.2 Notasi Algoritmik quadtree.hpp dan quadtree.cpp

2.2.1 Quadtree.hpp

```
class QuadTree {
private:
    const vector<vector<Pixel>>* matrix;
    double x, y;
    double sizeX, sizeY;
    double minBlockSize;
    QuadTree *GambarKiriAtas, *GambarKananAtas, *GambarKiriBawah, *GambarKananBawah;
    Pixel averageColor;

public:
    QuadTree(const vector<vector<Pixel>>* mat, double x, double y, double sizeX, double sizeY, int minBlockSize);
    const vector<vector<Pixel>>* getMatrix() const;
    double getX() const;
    double getY() const;
    double getSizeX() const;
    double getSizeY() const;
    double getMinBlockSize() const;
    QuadTree* getGambarKiriAtas() const;
```

```

QuadTree* getGambarKananAtas() const;
QuadTree* getGambarKiriBawah() const;
QuadTree* getGambarKananBawah() const;
void setGambarKiriAtas(QuadTree* node);
void setGambarKananAtas(QuadTree* node);
void setGambarKiriBawah(QuadTree* node);
void setGambarKananBawah(QuadTree* node);
static QuadTree* buildQuadTree(const vector<vector<Pixel>>* mat, double x, double y, double
sizeX, double sizeY, double minBlockSize, double threshold, bool useVariance, bool useMPD, bool
useMAD, bool useEntropy, bool useSSIM);
bool isLeaf(QuadTree* node);
void reconstructImage(QuadTree* node, vector<vector<Pixel>>& pixelMatrix, double x, double y);
static int depthTree(QuadTree* node, int depth=0);
static long countNode(QuadTree* node, long nodes=0);
void reconstructImageFrame(QuadTree* node, vector<vector<Pixel>>& pixelMatrix, double
offsetX, double offsetY, int depth, int maxDepth);
};


```

2.2.2 Quadtree.cpp

Constructor QuadTree(mat: pointer ke vector<vector<Pixel>>, x: double, y: double, sizeX: double, sizeY: double, minBlockSize: integer)

```

matrix ← mat
x ← x
y ← y
sizeX ← sizeX
sizeY ← sizeY
minBlockSize ← minBlockSize
GambarKiriAtas, GambarKananAtas, GambarKiriBawah, GambarKananBawah ← NULL

```

Function getMatrix() → pointer ke vector<vector<Pixel>>
→ matrix

Function getX() → double
→ x

Function getY() → double
→ y

Function getSizeX() → double
→ sizeX

Function getSizeY() → double
→ sizeY

Function getMinBlockSize() → double
→ minBlockSize

Function getGambarKiriAtas() → pointer ke QuadTree
→ GambarKiriAtas

Function getGambarKananAtas() → pointer ke QuadTree
→ GambarKananAtas

Function getGambarKiriBawah() → pointer ke QuadTree
→ GambarKiriBawah

Function getGambarKananBawah() → pointer ke QuadTree
→ GambarKananBawah

Procedure setGambarKiriAtas(node: pointer ke QuadTree)
GambarKiriAtas ← node

Procedure setGambarKananAtas(node: pointer ke QuadTree)
GambarKananAtas ← node

Procedure setGambarKiriBawah(node: pointer ke QuadTree)
GambarKiriBawah ← node

Procedure setGambarKananBawah(node: pointer ke QuadTree)
GambarKananBawah ← node

Function buildQuadTree(
mat: pointer ke vector<vector<Pixel>>,
x, y, sizeX, sizeY: double,
minBlockSize, threshold: double,
useVariance, useMPD, useMAD, useEntropy, useSSIM: boolean
) → pointer ke QuadTree

KAMUS
error: double
avgBlock: matrix 2D Pixel
avgColor: Pixel
ssim: double

ALGORITMA

```

error ← 0.0

if useVariance then
    error ← calculateRGBAVariance(mat, x, y, sizeX, sizeY)
else if useMPD then
    error ← calculateMPD(mat, x, y, sizeX, sizeY)
else if useMAD then
    error ← calculateRGBAMad(mat, x, y, sizeX, sizeY)
else if useEntropy then
    error ← calculateRGBAEentropyTotal(mat, x, y, sizeX, sizeY)
else if useSSIM then
    avgColor ← getAverageColor(mat, x, y, sizeX, sizeY)
    for j ← 0 to sizeY - 1 do
        for i ← 0 to sizeX - 1 do
            avgBlock[j][i] ← avgColor
    ssim ← calculateSSIM_RGB(mat, &avgBlock, x, y, sizeX, sizeY)
    error ← 1.0 - ssim

blockArea ← sizeX * sizeY
if blockArea / 2 < minBlockSize OR blockArea < minBlockSize OR error <= threshold then
    leaf ← new QuadTree(mat, x, y, sizeX, sizeY, minBlockSize)
    leaf.averageColor ← getAverageColor(mat, x, y, sizeX, sizeY)
    → leaf

// Bagi blok
midX ← sizeX / 2
midY ← sizeY / 2
sizeX1 ← midX, sizeX2 ← sizeX - midX
sizeY1 ← midY, sizeY2 ← sizeY - midY

node ← new QuadTree(mat, x, y, sizeX, sizeY, minBlockSize)
node.averageColor ← getAverageColor(mat, x, y, sizeX, sizeY)

node.setGambarKiriAtas(buildQuadTree(mat, x, y, sizeX1, sizeY1, minBlockSize, threshold,
useVariance, useMPD, useMAD, useEntropy, useSSIM))
node.setGambarKananAtas(buildQuadTree(mat, x + sizeX1, y, sizeX2, sizeY1, minBlockSize,
threshold, useVariance, useMPD, useMAD, useEntropy, useSSIM))
node.setGambarKiriBawah(buildQuadTree(mat, x, y + sizeY1, sizeX1, sizeY2, minBlockSize,
threshold, useVariance, useMPD, useMAD, useEntropy, useSSIM))
node.setGambarKananBawah(buildQuadTree(mat, x + sizeX1, y + sizeY1, sizeX2, sizeY2,
minBlockSize, threshold, useVariance, useMPD, useMAD, useEntropy, useSSIM))

→ node

```

Function isLeaf(node: pointer ke QuadTree) → boolean
→ (semua anak node bernilai NULL)

Procedure reconstructImage(node: pointer ke QuadTree, pixelMatrix: matriks Pixel, offsetX: offsetY: double)

if node = NULL then return

if isLeaf(node) then
for i ← 0 to node.sizeY - 1 do
for j ← 0 to node.sizeX - 1 do
pixelMatrix[offsetY + i][offsetX + j] ← node.averageColor
return

reconstructImage(GambarKiriAtas, pixelMatrix, offsetX, offsetY)
reconstructImage(GambarKananAtas, pixelMatrix, offsetX + sizeX / 2, offsetY)
reconstructImage(GambarKiriBawah, pixelMatrix, offsetX, offsetY + sizeY / 2)
reconstructImage(GambarKananBawah, pixelMatrix, offsetX + sizeX / 2, offsetY + sizeY / 2)

Procedure reconstructImageFrame(node: pointer ke QuadTree, pixelMatrix: matriks Pixel, offsetX, offsetY: double, depth, maxDepth: integer)

if node = NULL then return

if depth < maxDepth then
for i ← 0 to node.sizeY - 1 do
for j ← 0 to node.sizeX - 1 do
pixelMatrix[offsetY + i][offsetX + j] ← node.averageColor

reconstructImageFrame(GambarKiriAtas, ..., depth + 1)
reconstructImageFrame(GambarKananAtas, ..., depth + 1)
reconstructImageFrame(GambarKiriBawah, ..., depth + 1)
reconstructImageFrame(GambarKananBawah, ..., depth + 1)

Function depthTree(node: pointer ke QuadTree, depth: integer) → integer

if node = NULL then → 0
if isLeaf(node) then → depth

leftTop ← depthTree(GambarKiriAtas, depth + 1)
rightTop ← depthTree(GambarKananAtas, depth + 1)
leftBottom ← depthTree(GambarKiriBawah, depth + 1)
rightBottom ← depthTree(GambarKananBawah, depth + 1)

→ maksimum dari keempat depth

Function countNode(node: pointer ke QuadTree, nodes: long) → long

```

if node = NULL then → nodes

count ← nodes + 1

count ← countNode(GambarKiriAtas, count)
count ← countNode(GambarKananAtas, count)
count ← countNode(GambarKiriBawah, count)
count ← countNode(GambarKananBawah, count)

→ count

```

2.3 Notasi Algoritmik utils.hpp dan utils.cpp

2.3.1 utils.cpp

function calculateMean(matrix: pointer to vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer, sizeY: integer, colorChannel: integer) → double

KAMUS LOKAL:

```

sum: double ← 0.0
count: integer ← 0
traversal j y ... (y + sizeY - 1), j < matrix.size()
    traversal i x ... (x + sizeX - 1), i < (*matrix)[j].size()
        if(colorChannel == 0) then
            sum ← sum + (*matrix)[j][i].r
        else if(colorChannel == 1) then
            sum ← sum + (*matrix)[j][i].g
        else if (colorChannel == 2) then
            sum ← sum + (*matrix)[j][i].b
        count ← count + 1
    →(if count > 0 then (sum / count) else 0.0)

```

function calculateVariance(matrix: pointer to vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer, sizeY: integer, colorChannel: integer) → double

KAMUS LOKAL:

```

mean: double ← calculateMean(matrix, x, y, sizeX, sizeY, colorChannel)
sum: double ← 0.0
count: integer ← 0
traversal j y ... (y + sizeY - 1), j < matrix.size()
    traversal i x ... (x + sizeX - 1), i < (*matrix)[j].size()

```

```

if(colorChannel == 0) then
    pixelValue ← (*matrix)[j][i].r
else if(colorChannel == 1) then
    pixelValue ← (*matrix)[j][i].g
else
    pixelValue ← (*matrix)[j][i].b
sum ← sum + (pixelValue - mean) * (pixelValue - mean)
count ← count + 1
→ (if count > 0 then (sum / count) else 0.0)

```

```

function calculateRGBVariance(matrix: pointer to vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer, sizeY: integer) → double
    varianceR ← calculateVariance(matrix, x, y, sizeX, sizeY, 0)
    varianceG ← calculateVariance(matrix, x, y, sizeX, sizeY, 1)
    varianceB ← calculateVariance(matrix, x, y, sizeX, sizeY, 2)
    →(varianceR + varianceG + varianceB) / 3.0

```

```

function calculateMad(matrix: pointer to vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer, sizeY: integer, colorChannel: integer) → double

```

KAMUS LOKAL:

```

mean: double ← calculateMean(matrix, x, y, sizeX, sizeY, colorChannel)
sum: double ← 0.0
count: integer ← 0
traversal j from y... (y + sizeY - 1), j < matrix.size()
    traversal i from x... (x + sizeX - 1), t i < (*matrix)[j].size()
        if(colorChannel == 0) then
            pixelValue ← (*matrix)[j][i].r
        else if(colorChannel == 1) then
            pixelValue ← (*matrix)[j][i].g
        else
            pixelValue ← (*matrix)[j][i].b
        diff ← pixelValue - mean
        if(diff < 0) then
            diff ← -diff
        sum ← sum + diff
        count ← count + 1
    → (if count > 0 then (sum / count) else 0.0)

```

```

function calculateRGBMad(matrix: pointer to vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer, sizeY: integer) → double

```

```

    madR ← calculateMad(matrix, x, y, sizeX, sizeY, 0)

```

```

madG ← calculateMad(matrix, x, y, sizeX, sizeY, 1)
madB ← calculateMad(matrix, x, y, sizeX, sizeY, 2)
→ (madR + madG + madB) / 3.0

```

function calculateMPDMaxMin(matrix: pointer to vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer, sizeY: integer, colorChannel: integer) → double

kamus lokal:

```

maxVal: double ← -1
minVal: double ← 256
traversal j from y to (y + sizeY - 1), dengan syarat j < matrix.size()
traversal i from x to (x + sizeX - 1), dengan syarat i < (*matrix)[j].size()
  if(colorChannel == 0) then
    pixelValue ← (*matrix)[j][i].r
  else if(colorChannel == 1) then
    pixelValue ← (*matrix)[j][i].g
  else
    pixelValue ← (*matrix)[j][i].b
  if(pixelValue > maxVal) then
    maxVal ← pixelValue
  if(pixelValue < minVal) then
    minVal ← pixelValue
→ maxVal - minVal

```

function calculateMPD(matrix: pointer to vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer, sizeY: integer) → double

```

diffR ← calculateMPDMaxMin(matrix, x, y, sizeX, sizeY, 0)
diffG ← calculateMPDMaxMin(matrix, x, y, sizeX, sizeY, 1)
diffB ← calculateMPDMaxMin(matrix, x, y, sizeX, sizeY, 2)
→ (diffR + diffG + diffB) / 3.0

```

function calculateEntropy(matrix: pointer to vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer, sizeY: integer, colorChannel: integer) → double

KAMUS LOKAL:

```

histogram: array[0..255] of integer, semua elemen ← 0
totalPixels: integer ← 0
entropy: double ← 0.0
traversal i y... (y + sizeY - 1), j < matrix.size()
  Traversal j x ... (x + sizeX - 1), i < (*matrix)[j].size()
    if(colorChannel == 0) then
      pixelValue ← (*matrix)[j][i].r
    else if(colorChannel == 1) then

```

```

pixelValue ← (*matrix)[j][i].g
else
    pixelValue ← (*matrix)[j][i].b
    histogram[pixelValue] ← histogram[pixelValue] + 1
    totalPixels ← totalPixels + 1
traversal k 0... 255
    if(histogram[k] > 0) then
        probability ← histogram[k] / totalPixels
        entropy ← entropy - (probability * log2(probability))
    → entropy

```

```

function calculateRGBEntropyTotal(matrix: pointer to vector<vector<Pixel>>, x: integer, y: integer,
sizeX: integer, sizeY: integer) → double
    entropyR ← calculateEntropy(matrix, x, y, sizeX, sizeY, 0)
    entropyG ← calculateEntropy(matrix, x, y, sizeX, sizeY, 1)
    entropyB ← calculateEntropy(matrix, x, y, sizeX, sizeY, 2)
    return (entropyR + entropyG + entropyB) / 3.0

```

/* SSIM (BONUS) */

```

// Konstanta untuk SSIM
const double C1 ← (0.01 * 255) * (0.01 * 255)
const double C2 ← (0.03 * 255) * (0.03 * 255)

```

```

// Fungsi untuk menghitung kovarians antara dua matriks
function calculateCovariance(mat1: pointer ke vector<vector<Pixel>>, mat2: pointer ke
vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer, sizeY: integer, colorChannel: integer) →
double
KAMUS LOKAL:
mean1, mean2, sum ← double
count ← integer

```

ALGORITMA:

```

mean1 ← calculateMean(mat1, x, y, sizeX, sizeY, colorChannel)
mean2 ← calculateMean(mat2, x, y, sizeX, sizeY, colorChannel)
sum ← 0.0
count ← 0

```

```

for j ← y to y + sizeY - 1 do
    if j ≥ size(mat1) or j ≥ size(mat2) then break
    for i ← x to x + sizeX - 1 do
        if i ≥ size(mat1[j]) or i ≥ size(mat2[j]) then break
        if colorChannel = 0 then // Red

```

```

pixelValue1 ← mat1[j][i].r
pixelValue2 ← mat2[j][i].r
else if colorChannel = 1 then // Green
    pixelValue1 ← mat1[j][i].g
    pixelValue2 ← mat2[j][i].g
else if colorChannel = 2 then // Blue
    pixelValue1 ← mat1[j][i].b
    pixelValue2 ← mat2[j][i].b
else// Alpha
    pixelValue1 ← mat1[j][i].a
    pixelValue2 ← mat2[j][i].a

sum ← sum + (pixelValue1 - mean1) * (pixelValue2 - mean2)
count ← count + 1

```

```

if count > 0 then
    → sum / count
else
    → 0.0

```

// Fungsi untuk menghitung nilai SSIM

```

function calculateSSIM(original: pointer ke vector<vector<Pixel>>, compressed: pointer ke
vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer, sizeY: integer, colorChannel: integer) →
double
KAMUS LOKAL:
mu1, mu2, sigma1_sq, sigma2_sq, sigma12, numerator, denominator, ssim ← double

```

ALGORITMA:

```

mu1 ← calculateMean(original, x, y, sizeX, sizeY, colorChannel)
mu2 ← calculateMean(compressed, x, y, sizeX, sizeY, colorChannel)

sigma1_sq ← calculateVariance(original, x, y, sizeX, sizeY, colorChannel)
sigma2_sq ← calculateVariance(compressed, x, y, sizeX, sizeY, colorChannel)

sigma12 ← calculateCovariance(original, compressed, x, y, sizeX, sizeY, colorChannel)

```

```

numerator ← (2 * mu1 * mu2 + C1) * (2 * sigma12 + C2)
denominator ← (mu1 * mu1 + mu2 * mu2 + C1) * (sigma1_sq + sigma2_sq + C2)

```

```

if denominator < 1e-10 then
    → 1.0

```

```

ssim ← numerator / denominator
→ max(0.0, min(1.0, ssim))

```

```
// Fungsi untuk menghitung nilai SSIM untuk gambar RGBA
function calculateSSIM_RGB(original: pointer ke vector<vector<Pixel>>, compressed: pointer ke
vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer, sizeY: integer) → double
KAMUS LOKAL:
```

```
wR, wG, wB, wA, totalWeight, SSIM_R, SSIM_G, SSIM_B ← double
```

ALGORITMA:

```
// Bobot yang dimodifikasi untuk RGBA
```

```
wR ← 0.2125
```

```
wG ← 0.7154
```

```
wB ← 0.0721
```

```
totalWeight ← wR + wG + wB
```

```
if(x < 0 or y < 0 or sizeX ≤ 0 or sizeY ≤ 0 or y ≥ size(original) or y ≥ size(compressed)) then
→ 0.0
```

```
SSIM_R ← calculateSSIM(original, compressed, x, y, sizeX, sizeY, 0)
```

```
SSIM_G ← calculateSSIM(original, compressed, x, y, sizeX, sizeY, 1)
```

```
SSIM_B ← calculateSSIM(original, compressed, x, y, sizeX, sizeY, 2)
```

```
→ (wR * SSIM_R + wG * SSIM_G + wB * SSIM_B)/ totalWeight
```

```
function getAverageColor(mat: pointer to vector<vector<Pixel>>, x: integer, y: integer, sizeX: integer,
sizeY: integer) → Pixel
```

kamus lokal:

```
totalR, totalG, totalB: long long ← 0
```

```
totalPixels: integer ← sizeX * sizeY
```

```
avgColor: Pixel
```

```
traversal y ... (y + sizeY - 1)
```

```
    traversal x ... (x + sizeX - 1)
```

```
        totalR ← totalR + (*mat)[i][j].r
```

```
        totalG ← totalG + (*mat)[i][j].g
```

```
        totalB ← totalB + (*mat)[i][j].b
```

```
avgColor.r ← totalR / totalPixels (konversi ke unsigned char)
```

```
avgColor.g ← totalG / totalPixels (konversi ke unsigned char)
```

```
avgColor.b ← totalB / totalPixels (konversi ke unsigned char)
```

```
→ avgColor
```

2.4 Notasi Algoritmik Bonus

2.4.1 Bonus.cpp

Fungsi testCompression(img: Image, blockSize: double, threshold: double,
inputFileSize: double, filename: string, outputBase: string,
useSSIM: boolean) → double

KAMUS

tempOutputFilename: string
pixelMatrix: pointer ke vector<vector<Pixel>>
root: pointer ke QuadTree
reconstructImageMatrix: matriks Pixel
ext: string
outputFileSize: double
compressionRate: double

ALGORITMA

```
tempOutputFilename ← outputBase + "_temp"  
pixelMatrix ← &img.pixels  
  
root ← buildQuadTree(pixelMatrix, 0.0, 0.0, img.width, img.height,  
blockSize, threshold, false, false, false, false, useSSIM)  
  
reconstructImageMatrix ← matriks kosong berukuran [img.height][img.width]  
root.reconstructImage(root, reconstructImageMatrix, 0.0, 0.0)  
  
ext ← getFileExtension(filename)  
saveReconstructedImage(tempOutputFilename, reconstructImageMatrix, ext)  
  
outputFileSize ← getFileSize(tempOutputFilename + "." + ext)  
compressionRate ← calculateCompression(inputFileSize, outputFileSize)  
  
hapus file (tempOutputFilename + "." + ext)  
hapus root  
  
→ compressionRate
```

Fungsi findOptimalParameters(img: Image, targetCompressionRate: double,
filename: string, outputBase: string,
useSSIM: boolean) → pasangan<double, double>

KAMUS

inputFileSize: double

```
minBlockSize, bestThreshold, closestCompressionRate, minDifference: double
minDimension, i: integer
maxBlockSize: double
cache: array of CacheEntry (struct berisi blockSize, threshold, compressionRate)
blockSize, compressionRate, endCompressionRate, currentCompressionRate: double
startThreshold, endThreshold, step, currentThreshold, difference: double
shouldSearch: boolean
```

ALGORITMA

```
inputFileSize ← getFileSize(filename)
minBlockSize ← 4.0
bestThreshold ← 0.0
closestCompressionRate ← 0.0
minDifference ← 100.0

minDimension ← minimum dari img.width dan img.height
maxBlockSize ← 1.0
While maxBlockSize * 8 ≤ minDimension do:
    maxBlockSize ← maxBlockSize * 8

untuk setiap blockSize dari maxBlockSize sampai  $\geq 4.0$ , dikali 1/4:
    output: "Block size: ", blockSize

If useSSIM then
    startThreshold ← 0.1
    endThreshold ← 0.9
else
    startThreshold ← 5.0
    endThreshold ← 50.0

compressionRate ← testCompression(img, blockSize, startThreshold, inputFileSize, filename,
outputBase, useSSIM)
simpan {blockSize, startThreshold, compressionRate} ke cache

output: " Threshold: ", startThreshold, ", Compression: ", compressionRate, "%"

if|compressionRate - targetCompressionRate| > 10.0 then
    output: " Skip. Masi Jauh dari target"
    lanjut ke iterasi berikutnya

difference ← |compressionRate - targetCompressionRate|
If difference < minDifference then
    perbarui minDifference, closestCompressionRate, minBlockSize, bestThreshold
```

```

endCompressionRate ← testCompression(img, blockSize, endThreshold, inputFileSize, filename,
outputBase, useSSIM)
    simpan {blockSize, endThreshold, endCompressionRate} ke cache

    output: " End test - Threshold: ", endThreshold, ", Compression: ", endCompressionRate, "%"

    difference ← |endCompressionRate - targetCompressionRate|
    If difference < minDifference then
        perbarui minDifference, closestCompressionRate, minBlockSize, bestThreshold

    shouldSearch ← FALSE
    If (compressionRate < targetCompressionRate dan endCompressionRate > targetCompressionRate)
or
        (compressionRate > targetCompressionRate dan endCompressionRate < targetCompressionRate)
then
    shouldSearch ← TRUE

    If not shouldSearch then
        if minDifference < 5.0 then
            output: " Dapat yang toleransinya <= 5%"
            → (minBlockSize, bestThreshold)
        lanjut ke iterasi berikutnya

    step ← jika useSSIM maka 0.1 else 5.0

    If |endCompressionRate - targetCompressionRate| < |compressionRate - targetCompressionRate|
then
    currentThreshold ← endThreshold - step
else
    currentThreshold ← startThreshold + step

    While currentThreshold > startThreshold dan currentThreshold < endThreshold do:
        currentCompressionRate ← testCompression(img, blockSize, currentThreshold, inputFileSize,
filename, outputBase, useSSIM)
        simpan {blockSize, currentThreshold, currentCompressionRate} ke cache

        output: " Testing - Threshold: ", currentThreshold, ", Compression: ", currentCompressionRate,
"%"

        difference ← |currentCompressionRate - targetCompressionRate|
        If difference < minDifference then
            perbarui minDifference, closestCompressionRate, minBlockSize, bestThreshold

        If difference < 5.0 then

```

```

        output: " Found very good result (< 5% difference)"
        → (minBlockSize, bestThreshold)

    if(useSSIM dan currentCompressionRate < targetCompressionRate) or
        (!useSSIM dan currentCompressionRate > targetCompressionRate) then
            currentThreshold ← currentThreshold + step
    else
        currentThreshold ← currentThreshold - step

    output: "Best approximation found: ", closestCompressionRate, "% (difference: ", minDifference, "%)"
    → (minBlockSize, bestThreshold)

```

2.5 Notasi Algoritmik Gifer.cpp

2.5.1 Gifer.cpp

Fungsi gifMaker(width: integer, height: integer, gifOutput: pointer ke char,
 maxDepth: integer, root: pointer ke QuadTree,
 reconstructImageMatrix: referensi ke matriks Pixel) → void

KAMUS

writer: GifWriter
 i, x, y, idx: integer
 frameData: array of uint8_t

ALGORITMA

If GifBegin(writer, gifOutput, width, height, 10) gagal then
 output: "Gagal membuat GIF"
 →

i traversal maxDepth.... 0
 panggil root.reconstructImageFrame(root, reconstructImageMatrix, 0, 0, i, maxDepth + 1)
 frameData ← array ukuran width * height * 4

y traversal 0... height-1:
 x traversal 0 ...< width-1:
 idx ← (y * width + x) * 4
 p ← reconstructImageMatrix[y][x]
 frameData[idx + 0] ← p.r
 frameData[idx + 1] ← p.g
 frameData[idx + 2] ← p.b
 frameData[idx + 3] ← 255

```
panggil GifWriteFrame(writer, frameData, width, height, 100)
```

```
panggil GifEnd(writer)
```

```
output: (newline)
```

```
output: "GIF berhasil dibuat"
```

BAB 3

Penjelasan Algoritma

3.1 Penjelasan Algoritma Divide and Conquer yang digunakan

1. Pengubahan Gambar menjadi Matriks 2D

Untuk tahap memproses menjadi Quadtree dan memproses gambar, tentunya kita butuh mengubah gambar menjadi matriks yang berisi nilai dari RGB dan ini menggunakan library stb yang lengkap berisi tentang yang memungkinkan pembacaan berbagai format gambar seperti **PNG dan JPEG**.

Langkah-langkahnya adalah sebagai berikut:

1. Membaca gambar dari file menggunakan `stbi_load()` untuk mendapatkan nilai RGB setiap piksel.
2. Menyimpan setiap piksel dalam struktur data `vector<vector<Pixel>>`, di mana `Pixel` adalah struktur yang menyimpan nilai R (red), G (green), B (blue), dan A (alpha, bila ada) dan kami menggunakan `unsigned char` untuk menyimpan nilai ini karena pas menggunakan 0-255 sesuai dengan warna RGB agar hemat memori.

DATA TYPE	MIN_VALUE	MAX_VALUE
unsigned char	0	255
signed char	-128	127
unsigned short int	0	65535
signed short int	-32768	32767
unsigned int	0	65535
signed int	-32768	32767
unsigned long int	0	4294967295
signed long int	-2147483648	2147483647
float	3.4 *10^-38	3.4 *10^38
double	1.7*10^-308	1.7*10^308
long double	3.4*10^-4932	1.1*10^-4932
enum	-32768	32767

Gambar 1. Data Type dan Keterangan Min dan Max Value pada C++

3. Menyimpan informasi dimensi gambar (lebar, tinggi, dan jumlah kanal warna) dalam Image yang sudah kita struct pada image.hpp
2. Pengkonversian Matriks 2D menjadi Quadtree

Tahap ini merupakan langkah utama dalam proses kompresi gambar menggunakan QuadTree. Setelah gambar dikonversi menjadi matriks 2D berisi nilai RGB, matriks tersebut akan diproses untuk membangun struktur QuadTree berdasarkan aturan Divide and Conquer.

Quadtree sendiri terdiri atas:

```
const vector<vector<Pixel>>*> matrix;  
double x, y;  
double sizeX, sizeY;  
double minBlockSize;  
QuadTree *GambarKiriAtas, *GambarKananAtas, *GambarKiriBawah, *GambarKananBawah;  
Pixel averageColor;
```

X → mencatat koordinat X agar dapat digunakan untuk rekonstruksi

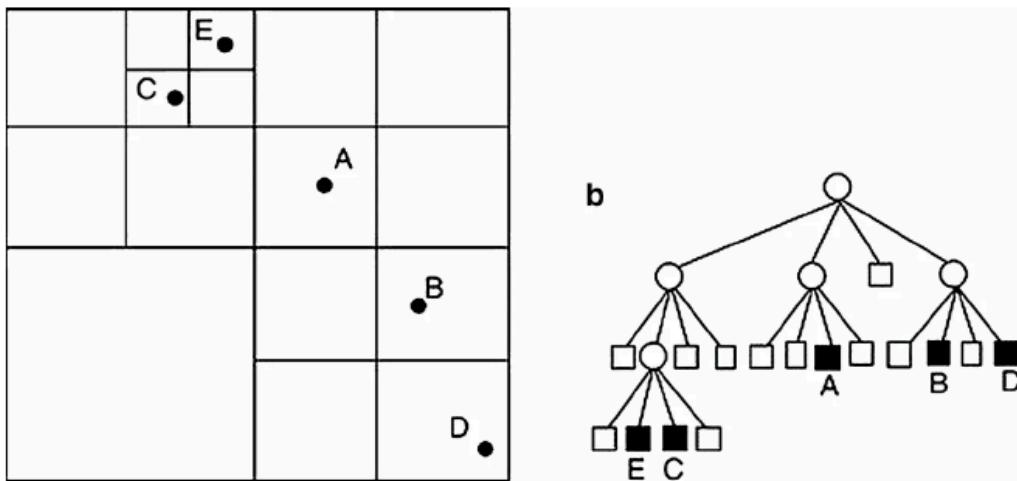
Y → mencatat koordinat Y agar dapat digunakan untuk rekonstruksi

sizeX → mencatat ukuran di sumbu X

sizeY → mencatat ukuran di sumbu Y

minBlockSize → jumlah blok minimum (input dari pengguna)

QuadTree *GambarKiriAtas, KananAtas, KiriBawah, KananBawah ← menunjuk pada alamat QuadTree anaknya



Gambar 2. Visualisasi QuadTree

Langkah-langkah Konversi Matriks ke QuadTree:

1. Cek apakah matriks sudah homogen
 - Sebuah blok piksel dianggap homogen jika perbedaan warnanya dalam batas threshold error yang telah ditentukan pengguna.
 - Jika homogen, maka blok tersebut disimpan sebagai satu node dalam QuadTree.
2. Jika tidak homogen, bagi menjadi 4 bagian
 - Kiri Atas (Top-Left)
 - Kanan Atas (Top-Right)
 - Kiri Bawah (Bottom-Left)
 - Kanan Bawah (Bottom-Right)
 - Lakukan rekursi pada setiap bagian hingga mencapai ukuran minimum yang ditentukan pengguna atau hingga semua blok menjadi homogen.
3. Setiap node dalam QuadTree menyimpan informasi berikut:
 - Koordinat awal (x, y) dan ukuran blok (sizeX, sizeY).
 - Rata-rata warna blok tersebut jika menjadi leaf node.
 - Pointer ke empat child node, jika blok masih dapat dibagi lagi
4. Menghentikan pembagian blok berdasarkan beberapa kondisi:
 - Ukuran blok sudah lebih kecil dari ukuran minimum block size dan apabila dibagi 2 juga masih lebih besar dibandingkan minimum block size..

- Perbedaan warna dalam blok kurang dari threshold error yang sesuai dengan metode yang diukur menggunakan rumus error yang dipilih.

3. Rekonstruksi dari QuadTree menjadi matriks array 2D

Tahap ini bertujuan untuk mengubah kembali struktur QuadTree menjadi array 2D yang merepresentasikan gambar awal sebelum kompresi. Proses ini dilakukan dengan melakukan traversal pada QuadTree dan mengisi matriks dengan warna yang sesuai.

Traversal QuadTree

- Jika node adalah leaf, maka seluruh blok yang dicakup oleh node tersebut akan diisi dengan warna rata-rata dari area tersebut.
- Jika node bukan leaf, maka rekonstruksi dilakukan secara rekursif untuk keempat anaknya.

Saat merekonstruksi gambar dari QuadTree, kita mengisi kembali array 2D yang merepresentasikan gambar menggunakan nilai rata-rata warna (RGB) dari setiap leaf node.

→ QuadTree dapat dikembalikan ke array karena dalam setiap QuadTree kita menyimpan koordinat x, y, size x, dan size y nya asal dia saat masih menjadi array, oleh karena itu kita dapat mengembalikannya menjadi matriks kembali.

4. Matriks 2D menjadi gambar png/jpg/jpeg

Tahap ini mengubah matriks 2D kembali menjadi gambar, hal ini menggunakan library stb_image_write yang tentunya membutuhkan data RGB yang ada pada matriks 2D kita.

5. Gambar berhasil terbentuk dengan ukuran file yang lebih rendah dibandingkan sebelumnya

3.2 Penjelasan Algoritma BONUS

1. SSIM (Bonus)

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

SSIM (Structural Similarity Index):

- Menggunakan rata-rata (μ) untuk menghitung intensitas piksel.
- Varians untuk mengukur sebaran nilai piksel.
- Kovarian untuk menangkap struktur bersama antara dua gambar.
- Tiga komponen ini (luminansi, kontras, dan struktur) digabungkan untuk menilai kesamaan gambar.

Alasan membuat fungsi kovarian baru:

- Karena MAD tidak bisa menggantikan peran kovarian.
- Kovarian penting untuk mengukur hubungan struktural antara gambar asli dan hasil kompresi.

Nilai konstan C1 dan C2:

- Digunakan untuk menjaga kestabilan perhitungan SSIM.
- Mencegah pembagian dengan nol saat nilai mean atau varians sangat kecil.

Alur program calculateSSIM_RGB:

- Memanggil calculateSSIM untuk setiap kanal warna (Red, Green, Blue, Alpha).
- Menghitung rata-rata, varian, dan kovarian untuk masing-masing kanal.

Penggabungan SSIM tiap kanal warna:

- Digabung menggunakan rumus $(wR * SSIM_R + wG * SSIM_G + wB * SSIM_B + wA * SSIM_A) / totalWeight$.
- Bobot ini mencerminkan sensitivitas mata manusia terhadap warna:

$wR = 0.299$ (mata cukup sensitif terhadap merah),

$wG = 0.587$ (paling sensitif terhadap hijau),

$wB = 0.114$ (paling tidak sensitif terhadap biru).

Alasan penggunaan bobot dalam SSIM adalah untuk menyesuaikan perhitungan dengan persepsi visual manusia, karena mata kita tidak merespons semua warna secara setara. Dalam banyak aplikasi pengolahan citra, termasuk konversi ke grayscale, digunakan bobot seperti $wR = 0.299$, $wG = 0.587$, dan $wB = 0.114$ karena mata manusia paling sensitif terhadap warna hijau, lalu merah, dan paling sedikit terhadap biru. Pendekatan ini umum digunakan karena saat menilai kualitas atau kesamaan struktur gambar, warna RGB dalam SSIM bukanlah aspek utama yang diperhatikan yang lebih penting adalah struktur dan kontras gambar tersebut. Itulah mengapa dalam perhitungan SSIM program ini mengonversi menjadi grayscale, gambar bisa dikonversi ke grayscale terlebih dahulu untuk fokus pada informasi struktural, bukan informasi warna, karena cukup dengan grayscale saja untuk menentukan struktur dari gambar.

SSIM pada program ini, apabila kita menggunakan threshold tinggi seperti 0.9, maka errornya menjadi $1 - ssim$ menjadi 0.1, yang mana 0.1 pada ssim sangatlah jauh dari gambar aslinya, karena semakin dekat ke 1 maka SSIM akan semakin dekat ke gambar aslinya, namun karena $1 - ssim$ maka semakin dekat 0 maka semakin dekat kepada gambar aslinya. Mengapa $1 - ssim$ kalau begitu?

-> di dalam algoritma Quadtree, kita konsisten menggunakan error threshold artinya semakin besar nilai error, maka semakin tidak homogen blok tersebut dan harus dibagi menjadi 4 blok anak, maka karena ssim terbalik dengan yang lain (kalau semakin gede ke 1 semakin mirip).

2. Persentase Kompresi

Fungsi `findOptimalParameters` bertujuan untuk mencari kombinasi optimal antara ukuran blok (`blockSize`) dan nilai threshold agar hasil kompresi menggunakan algoritma Quadtree mendekati compression rate yang ditargetkan.

1. Ambil Ukuran File Asli

Pertama-tama, ukuran file input dihitung menggunakan `getFileSize()`. Nilai ini digunakan sebagai acuan dasar untuk menghitung persentase kompresi dari hasil Quadtree.

2. Tentukan Ukuran Blok Maksimum

Ukuran blok maksimum dihitung dengan mencari pangkat dua terbesar (kelipatan 8) yang masih muat di dimensi terkecil gambar (`min(width, height)`). Ini untuk memastikan blok dapat terbagi merata tanpa keluar dari batas gambar.

3. Iterasi Ukuran Blok

Dimulai dari `blockSize` terbesar ke terkecil, dibagi empat setiap iterasi (`blockSize /= 4`). Untuk setiap ukuran blok, program menguji berbagai `threshold` untuk menemukan kombinasi parameter yang paling mendekati target kompresi.

4. Uji Threshold Awal dan Akhir

Untuk setiap `blockSize`, dilakukan uji coba dua nilai `threshold` awal:

- SSIM → mulai dari 0.1 hingga 0.9

Hasil kompresi untuk masing-masing `threshold` diuji lewat `testCompression()` dan disimpan dalam cache untuk menghindari duplikasi komputasi.

5. Pencarian dengan Linear Search

Pertama, kita akan menggunakan SSIM yang paling akurat yaitu errornya 0.1 sehingga kita mendapatkan mendapatkan compression rate terendah, apabila kita mendapatkan bahwa target masih jauh yaitu $> 10\%$ dibawah yang didapatkan oleh compression rate maka akan langsung membagi blocksize menjadi 4 lalu mencoba lagi apabila masih dibawah , bagi lagi, apabila menemukan yang berada di sekitar range 5%, maka hasil tersebut langsung kita jadikan hasil untuk parameter optimal.

6. Pemilihan Parameter Terbaik

Selama proses pencarian, setiap kombinasi `blockSize` dan `threshold` yang menghasilkan kompresi paling mendekati target disimpan sebagai parameter terbaik sementara. Jika tidak ditemukan hasil yang benar-benar sesuai, maka kombinasi terbaik sejauh ini akan dikembalikan.

3. GIF

Implementasi GIF dibuat opsional dalam program utama - pengguna dapat memilih untuk membuat gif dari proses kompresi quadtree atau tidak. Bila pengguna memilih untuk melakukan save gif, pengguna akan ditanyakan address untuk melakukan save gif setelahnya. GIF dibuat dengan bantuan library gif.h, dan proses pembuatan gif juga dihitung dalam waktu eksekusi program.

Bila pengguna memilih untuk melakukan save GIF, maka alur program akan masuk ke dalam fungsi gifMaker dari gifer.hpp setelah gambar kompresi selesai disave. Fungsi ini menerima lebar dan tinggi gambar, address output gif, maxDepth dari quadtree, quadtree itu sendiri, dan vector untuk menyimpan matrix pixel.

gifMaker memanfaatkan maxDepth yang sudah dimiliki, untuk melakukan iterasi sesuai maxDepth dari quadtree. gifMaker menggunakan fungsi bantuan reconstructImageFrame, yang mirip seperti reconstructImage, dengan perbedaan pixel pada vector diisi tidak dengan melihat leaf, tetapi menyesuaikan dengan depth iterasi pada saat ini.

reconstructImageFrame lalu akan membuat frame setiap gif satu per satu, dimulai dari kedalaman terendah, dimana seluruh gambar hanya satu rata-rata warna. Gif.h lalu akan menerima data ini dengan GifWriteFrame, dengan delay 100 agar perkembangan algoritma quadtree dapat terlihat dengan jelas.

3.3. Source Code Program

Program utama dibuat menggunakan bahasa c++, dengan bantuan library stb_image.h, stb_image_write.h, dan gif.h.

main.cpp

```
#include "include/image.hpp"
#include "include/quadtree.hpp"
#include "include/utils.hpp"
#include "include/bonus.hpp"
#include "include/gifer.hpp"
#include <iostream>
#include <string>
#include <chrono>

using namespace std;

int main() {
    //buat waktu
```

```

chrono::high_resolution_clock::time_point start;

//input file
string filename;
cout << "======" << endl;
cout << "WELCOME TO COMPRESS QUADTREE TERGACOR!" << endl;
cout << "======" << endl;
cout << "Silahkan masukkan path file gambar: ";
getline(cin, filename);

if (!checkFile(filename)) {
    cerr << "Error: File tidak ditemukan!" << endl;
    return 1;
}

Image img = loadImage(filename);
if (img.width == 0 || img.height == 0) {
    cerr << "Error: Gagal memuat gambar!" << endl;
    return 1;
}

double inputFileSize = getFileSize(filename);
cout << "======" << endl;
cout << "      Dimensi Gambar: " << img.width << " x " << img.height << "" << endl;
cout << "======" << endl;

double targetCompressionRate;
cout << "Masukkan target kompresi (0.01 - 1): ";
cin >> targetCompressionRate;

while (targetCompressionRate < 0 || targetCompressionRate > 1) {
    cout << "Target kompresi harus antara 0-1, silahkan ulangi input: ";
    cin >> targetCompressionRate;
}

string outputFilename;
bool useVariance = false, useMPD = false, useMAD = false, useEntropy = false, useSSIM = false;
double min;
double threshold;
char togifornottogif;
string gifOutput;

//jika targetCompressionRate 0, maka user input threshold dan min block size
if (targetCompressionRate == 0) {
    int methodInput;
    cout << "======" << endl;
    cout << "Pilih metode error:" << endl;
    cout << "1: Variance\n2: MPD\n3: MAD\n4: Entropy\n5: SSIM" << endl;
    cout << "======" << endl;
    cout << "Pilihan: ";
    cin >> methodInput;
}

```

```

while (methodInput < 1 || methodInput > 5) {
    cout << "Input antara 1 - 5, silahkan ulangi input: ";
    cin >> methodInput;
}
cout << "===== " << endl;

cout << "Masukkan threshold error: ";
cin >> threshold;
while (threshold <= 0) {
    cout << "Threshold tidak boleh 0 atau dibawah 0, silahkan ulangi input: ";
    cin >> threshold;
}

cout << "Masukkan minimum block size: ";
cin >> min;
while (min < 4 || min > (img.height * img.width)) {
    cout << "Ulangi input, luas blok minimum tidak boleh kurang dari 4 (minimal 2 x 2) atau
melebihi luas gambar: ";
    cin >> min;
}
cout << "===== " << endl;

cout << "Masukkan path file gambar output: ";
cin.ignore();
getline(cin, outputFilename);

cout << "Apakah anda ingin membuat GIF? (y/n): ";
cin >> togifornottogif;

if (togifornottogif == 'y' || togifornottogif == 'Y') {
    cout << "Masukkan path file GIF output: ";
    cin.ignore();
    getline(cin, gifOutput);

    gifOutput += ".gif";
} else if (togifornottogif == 'n' || togifornottogif == 'N'){
    cout << "Tidak membuat gif" << endl;
}
//waktu dimulai

start = chrono::high_resolution_clock::now();

switch (methodInput) {
    case 1: useVariance = true; break;
    case 2: useMPD = true; break;
    case 3: useMAD = true; break;
    case 4: useEntropy = true; break;
    case 5: useSSIM = true; break;
    default:
        cout << "Input tidak valid. Default ke Varians.\n";
        useVariance = true;
}

```

```

        break;
    }
} else { //jika targetCompressionRate 1, maka masuk ke alur target %
    targetCompressionRate *= 100;

    useSSIM = true;

    cout << "Masukkan nama file gambar output: ";
    cin.ignore();
    getline(cin, outputFilename);
    cout << "======" << endl;
    cout << endl;

    cout << "Apakah anda ingin membuat GIF? (y/n): ";
    cin >> togiforNotGif;

    if (togiforNotGif == 'y' || togiforNotGif == 'Y') {
        cout << "Masukkan path file GIF output: ";
        cin.ignore();
        getline(cin, gifOutput);

        gifOutput += ".gif";
    } else if (togiforNotGif == 'n' || togiforNotGif == 'N'){
        cout << "Tidak membuat gif" << endl;
    }

    cout << "\nMencari parameter optimal untuk target kompresi " << targetCompressionRate <<
    "%...\n";

    //waktu dimulai
    start = chrono::high_resolution_clock::now();

    auto optimalParams = Kompresi::findOptimalParameters(
        img, targetCompressionRate, filename, outputFilename, useSSIM
    );
    min = optimalParams.first;
    threshold = optimalParams.second;

    cout << "\nParameter optimal ditemukan:\n";
    cout << "- Block size: " << min << endl;
    cout << "- Threshold: " << threshold << endl;
}

const vector<vector<Pixel>>* pixelMatrix = &img.pixels;
QuadTree* root = QuadTree::buildQuadTree(
    pixelMatrix, 0, 0, img.width, img.height,
    min, threshold,
    useVariance, useMPD, useMAD, useEntropy, useSSIM
);

```

```

//untuk menyimpan hasil rekonstruksi
vector<vector<Pixel>> reconstructImageMatrix(img.height, vector<Pixel>(img.width));
root->reconstructImage(root, reconstructImageMatrix, 0, 0);

string ext = getFileExtension(filename);
int maxDepth = QuadTree::depthTree(root);

saveReconstructedImage(outputFilename, reconstructImageMatrix, ext);

//kalau buat gif (masuk di timer)
if (togifornottogif == 'y' || togifornottogif == 'Y') {
    gifMaker(img.width, img.height, gifOutput.c_str(), maxDepth, root, reconstructImageMatrix);
}

//buat timer ms
auto end = chrono::high_resolution_clock::now();
auto duration = chrono::duration_cast<chrono::milliseconds>(end-start);

//output terminal

double outputFileSize = getFileSize(outputFilename + "." + ext);

double compressionPerc = calculateCompression(inputFileSize, outputFileSize);
cout << "\n======" << endl;
cout << "          HASIL KOMPRESI      " << endl;
cout << "======" << endl;

cout << " Waktu eksekusi: " << duration.count() << " ms" << endl;
cout << " Ukuran file input: " << inputFileSize << " KB" << endl;
cout << " Ukuran file output: " << outputFileSize << " KB" << endl;
cout << " Persentase kompresi: " << compressionPerc << "%" << endl;

cout << "-----" << endl;
cout << " Kedalaman Quadtree: " << QuadTree::depthTree(root) << endl;
cout << " Jumlah simpul pada Tree: " << QuadTree::countNode(root) << endl;
cout << "======" << endl;

delete root;
return 0;
}

```

quadtree.hpp

```

#ifndef QUADTREE_HPP
#define QUADTREE_HPP

#include "image.hpp"
#include "utils.hpp"

using namespace std;

```

```

class QuadTree {
private:
    const vector<vector<Pixel>>* matrix;
    double x, y;
    double sizeX, sizeY;
    double minBlockSize;
    QuadTree *GambarKiriAtas, *GambarKananAtas, *GambarKiriBawah, *GambarKananBawah;
    Pixel averageColor;

public:
    QuadTree(const vector<vector<Pixel>>* mat, double x, double y, double sizeX, double sizeY, int minBlockSize);
    const vector<vector<Pixel>>* getMatrix() const;
    double getX() const;
    double getY() const;
    double getSizeX() const;
    double getSizeY() const;
    double getMinBlockSize() const;
    QuadTree* getGambarKiriAtas() const;
    QuadTree* getGambarKananAtas() const;
    QuadTree* getGambarKiriBawah() const;
    QuadTree* getGambarKananBawah() const;
    void setGambarKiriAtas(QuadTree* node);
    void setGambarKananAtas(QuadTree* node);
    void setGambarKiriBawah(QuadTree* node);
    void setGambarKananBawah(QuadTree* node);
    static QuadTree* buildQuadTree(const vector<vector<Pixel>>* mat, double x, double y, double sizeX, double sizeY, double minBlockSize, double threshold, bool useVariance, bool useMPD, bool useMAD, bool useEntropy, bool useSSIM);
    bool isLeaf(QuadTree* node);
    void reconstructImage(QuadTree* node, vector<vector<Pixel>>& pixelMatrix, double x, double y);
    static int depthTree(QuadTree* node, int depth=0);
    static long countNode(QuadTree* node, long nodes=0);
    void reconstructImageFrame(QuadTree* node, vector<vector<Pixel>>& pixelMatrix, double offsetX, double offsetY, int depth, int maxDepth);
};

#endif

```

quadtree.cpp

```

#include "include/quadtree.hpp"
#include <iostream>
#include "include/image.hpp"
using namespace std;

QuadTree::QuadTree(const vector<vector<Pixel>>* mat, double x, double sizeX, double sizeY, int minBlockSizeYes)
    : matrix(mat), x(x), y(y), sizeX(sizeX), sizeY(sizeY),
      minBlockSize(minBlockSizeYes),

```

```
        GambarKiriAtas(nullptr), GambarKananAtas(nullptr), GambarKiriBawah(nullptr),
GambarKananBawah(nullptr) {}

const vector<vector<Pixel>>* QuadTree::getMatrix() const {
    return matrix;
}

double QuadTree::getX() const {
    return x;
}

double QuadTree::getY() const {
    return y;
}

double QuadTree::getSizeX() const {
    return sizeX;
}

double QuadTree::getSizeY() const {
    return sizeY;
}

double QuadTree::getMinBlockSize() const {
    return minBlockSize;
}

QuadTree* QuadTree::getGambarKiriAtas() const {
    return GambarKiriAtas;
}

QuadTree* QuadTree::getGambarKananAtas() const {
    return GambarKananAtas;
}

QuadTree* QuadTree::getGambarKiriBawah() const {
    return GambarKiriBawah;
}

QuadTree* QuadTree::getGambarKananBawah() const {
    return GambarKananBawah;
}

void QuadTree::setGambarKiriAtas(QuadTree* node) {
    GambarKiriAtas = node;
}

void QuadTree::setGambarKananAtas(QuadTree* node) {
    GambarKananAtas = node;
}
```

```

void QuadTree::setGambarKiriBawah(QuadTree* node) {
    GambarKiriBawah = node;
}

void QuadTree::setGambarKananBawah(QuadTree* node) {
    GambarKananBawah = node;
}

QuadTree* QuadTree::buildQuadTree(const vector<vector<Pixel>>* mat, double x, double y, double sizeX,
double sizeY, double minBlockSize, double threshold, bool useVariance, bool useMPD, bool useMAD, bool
useEntropy, bool useSSIM)
{
    double error = 0.0;

    if (useVariance) {
        error = calculateRGBAVariance(mat, x, y, sizeX, sizeY);
    } else if (useMPD) {
        error = calculateMPD(mat, x, y, sizeX, sizeY);
    } else if (useMAD) {
        error = calculateRGBAMad(mat, x, y, sizeX, sizeY);
    } else if (useEntropy) {
        error = calculateRGBAEentropyTotal(mat, x, y, sizeX, sizeY);
    } else if (useSSIM) {
        int startX = static_cast<int>(x);
        int startY = static_cast<int>(y);
        int blockSizeX = static_cast<int>(sizeX);
        int blockSizeY = static_cast<int>(sizeY);

        vector<vector<Pixel>> avgBlock(blockSizeY, vector<Pixel>(blockSizeX));

        Pixel avgColor = getAverageColor(mat, startX, startY, blockSizeX, blockSizeY);

        for (int j = 0; j < blockSizeY; j++) {
            for (int i = 0; i < blockSizeX; i++) {
                avgBlock[j][i] = avgColor;
            }
        }
        double ssim = calculateSSIM_RGB(mat, &avgBlock, startX, startY, blockSizeX, blockSizeY);
        error = 1.0 - ssim;
    }

    int block_area = static_cast<int>(sizeX * sizeY);
    int min_block_size = static_cast<int>(minBlockSize);

    if ((block_area / 2) < min_block_size || block_area < min_block_size || error <= threshold) {
        QuadTree* leaf = new QuadTree(mat, x, y, sizeX, sizeY, minBlockSize);
        leaf->averageColor = getAverageColor(mat, static_cast<int>(x), static_cast<int>(y),
                                             static_cast<int>(sizeX), static_cast<int>(sizeY));
        return leaf;
    }
}

```

```

        double midX = sizeX / 2;
        double midY = sizeY / 2;
        double sizeX1 = midX, sizeX2 = sizeX - midX;
        double sizeY1 = midY, sizeY2 = sizeY - midY;

        QuadTree* node = new QuadTree(mat, x, y, sizeX, sizeY, minBlockSize);
        node->averageColor = getAverageColor(mat, static_cast<int>(x), static_cast<int>(y),
                                              static_cast<int>(sizeX), static_cast<int>(sizeY));

        node->setGambarKiriAtas(buildQuadTree(mat, x, y, sizeX1, sizeY1, minBlockSize, threshold,
                                              useVariance, useMPD, useMAD, useEntropy, useSSIM));
        node->setGambarKananAtas(buildQuadTree(mat, x + sizeX1, y, sizeX2, sizeY1, minBlockSize, threshold,
                                              useVariance, useMPD, useMAD, useEntropy, useSSIM));
        node->setGambarKiriBawah(buildQuadTree(mat, x, y + sizeY1, sizeX1, sizeY2, minBlockSize, threshold,
                                              useVariance, useMPD, useMAD, useEntropy, useSSIM));
        node->setGambarKananBawah(buildQuadTree(mat, x + sizeX1, y + sizeY1, sizeX2, sizeY2, minBlockSize,
                                                 threshold,
                                                 useVariance, useMPD, useMAD, useEntropy, useSSIM));

        return node;
    }

    bool QuadTree::isLeaf(QuadTree* node) {
        if ((node->getGambarKiriAtas() == nullptr) && (node->getGambarKananAtas() == nullptr) &&
            (node->getGambarKiriBawah() == nullptr) && (node->getGambarKananBawah() == nullptr)) return true;
        else return false;
    }

    void QuadTree::reconstructImage(QuadTree* node, vector<vector<Pixel>>& pixelMatrix, double offsetX,
                                    double offsetY) {
        if (!node) return;

        if (isLeaf(node)) {
            for (int i = 0; i < node->getSizeY(); i++) {
                for (int j = 0; j < node->getSizeX(); j++) {
                    double imgX = offsetX + j;
                    double imgY = offsetY + i;
                    pixelMatrix[imgY][imgX] = node->averageColor;
                }
            }
            return;
        }
        if (node->getGambarKiriAtas()) {
            reconstructImage(node->getGambarKiriAtas(), pixelMatrix, offsetX, offsetY);
        }
        if (node->getGambarKananAtas()) {
            reconstructImage(node->getGambarKananAtas(), pixelMatrix, offsetX + node->getSizeX() / 2,
                            offsetY);
        }
        if (node->getGambarKiriBawah()) {
            reconstructImage(node->getGambarKiriBawah(), pixelMatrix, offsetX, offsetY + node->getSizeY());
        }
    }
}

```

```

    /2);
}
if (node -> getGambarKananBawah()) {
    reconstructImage(node -> getGambarKananBawah(), pixelMatrix, offsetX + node->getSizeX() /2,
offsetY + node->getSizeY() /2);
}
}

void QuadTree::reconstructImageFrame(QuadTree* node, vector<vector<Pixel>>& pixelMatrix, double offsetX,
double offsetY, int depth, int maxDepth) {
    if(!node) return;

    if (depth < maxDepth) {
        for (int i = 0; i < node -> getSizeY(); i++) {
            for (int j = 0; j < node -> getSizeX(); j++) {
                double imgX = offsetX + j;
                double imgY = offsetY + i;
                pixelMatrix[imgY][imgX] = node->averageColor;
            }
        }
        if (node -> getGambarKiriAtas()) {
            reconstructImageFrame(node -> getGambarKiriAtas(), pixelMatrix, offsetX, offsetY, depth + 1,
maxDepth);
        }
        if (node -> getGambarKananAtas()) {
            reconstructImageFrame(node -> getGambarKananAtas(), pixelMatrix, offsetX + node->getSizeX() /2,
offsetY, depth + 1, maxDepth);
        }
        if (node -> getGambarKiriBawah()) {
            reconstructImageFrame(node -> getGambarKiriBawah(), pixelMatrix, offsetX, offsetY +
node->getSizeY() /2, depth + 1, maxDepth);
        }
        if (node -> getGambarKananBawah()) {
            reconstructImageFrame(node -> getGambarKananBawah(), pixelMatrix, offsetX + node->getSizeX() /2,
offsetY + node->getSizeY() /2, depth + 1, maxDepth);
        }
    }
}

int QuadTree::depthTree(QuadTree* node, int depth) {
    if (!node) return 0;

    if (node->isLeaf(node)) {
        return depth;
    }

    int leftTopDepth = depthTree(node->getGambarKiriAtas(), depth + 1);
    int rightTopDepth = depthTree(node->getGambarKananAtas(), depth + 1);
    int leftBottomDepth = depthTree(node->getGambarKiriBawah(), depth + 1);
    int rightBottomDepth = depthTree(node->getGambarKananBawah(), depth + 1);

    return max(leftTopDepth, rightTopDepth, leftBottomDepth, rightBottomDepth);
}

```

```

int rightBottomDepth = depthTree(node->getGambarKananBawah(), depth + 1);

int maxDepth = max(max(leftTopDepth, rightTopDepth), max(leftBottomDepth, rightBottomDepth));

return maxDepth;
}

long QuadTree::countNode(QuadTree* node, long nodes) {
    if (!node) return nodes;

    long count = nodes + 1;

    count = countNode(node->getGambarKiriAtas(), count);
    count = countNode(node->getGambarKananAtas(), count);
    count = countNode(node->getGambarKiriBawah(), count);
    count = countNode(node->getGambarKananBawah(), count);

    return count;
}

```

image.hpp

```

//di include/image.hpp
#ifndef IMAGE_HPP
#define IMAGE_HPP

#include <vector>
#include <string>
#include "stb_image.h"
#include "stb_image_write.h"

using namespace std;

struct Pixel {
    unsigned char r, g, b, a;
};

struct Image {
    int width;
    int height;
    int channels;

    vector<std::vector<Pixel>> pixels;
};

Image loadImage(const string& filename);
bool checkFile(const string& filename);
void saveReconstructedImage(const std::string &filename, const std::vector<vector<Pixel>>& image, const string& format);
string getFileExtension(const string& filename);

```

```
double getFileSize(const string& fileName);
double calculateCompression(double originalSize, double compressedSize);

#endif
```

image.cpp

```
#include "include/image.hpp"
#include <iostream>
#define STB_IMAGE_IMPLEMENTATION
#include "include/stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "include/stb_image_write.h"
#include <fstream>

using namespace std;

Image loadImage(const string& filename) {
    Image img;
    unsigned char* data = stbi_load(filename.c_str(), &img.width, &img.height, &img.channels, 0);

    if (!data) {
        cerr << "Error: Tidak dapat memuat gambar " << filename << endl;
        img.width = img.height = img.channels = 0;
        return img;
    }

    img.pixels.resize(img.height, vector<Pixel>(img.width));

    for (int y = 0; y < img.height; ++y) {
        for (int x = 0; x < img.width; ++x) {
            int index = (y * img.width + x) * img.channels;
            Pixel& p = img.pixels[y][x];

            if (img.channels == 1){
                p.r = data[index];
                p.g = data[index];
                p.b = data[index];
                p.a = 255;
            }
            else if(img.channels == 3) {
                p.r = data[index];
                p.g = data[index + 1];
                p.b = data[index + 2];
                p.a = 255;
            } else if (img.channels == 4) {
                p.r = data[index];
                p.g = data[index + 1];
                p.b = data[index + 2];
                p.a = data[index + 3];
            }
        }
    }
}
```

```

        }
    }

    stbi_image_free(data);
    return img;
}

bool checkFile(const string& filename) {
    FILE* file = fopen(filename.c_str(), "rb");
    if (file) {
        fclose(file);
        return true;
    }
    return false;
}

void saveReconstructedImage(const string &filename, const vector<vector<Pixel>>& image, const string& format) {
    int height = image.size();
    int width = image[0].size();

    bool isGrayscale = true;
    bool Alpha = false;
    for (int y = 0; y < height && (isGrayscale || !Alpha); ++y) {
        for (int x = 0; x < width; ++x) {
            const Pixel& p = image[y][x];
            if (p.r != p.g || p.g != p.b) {
                isGrayscale = false;
                break;
            }
            if (p.a != 255) {
                Alpha = true;
            }
        }
    }

    bool isJPEG = (format == "jpg" || format == "jpeg");
    int channels;

    if (isJPEG) {
        channels = 3;
    } else {
        channels = Alpha ? 4 : (isGrayscale ? 1 : 3);
    }

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

    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            const Pixel& p = image[y][x];

```

```

int index = (y * width + x) * channels;

if (isJPEG ) {
    if (isGrayscale) {
        data[index] = p.r;
        data[index + 1] = p.g;
        data[index + 2] = p.b;
    } else {
        data[index] = p.r;
        data[index + 1] = p.g;
        data[index + 2] = p.b;
    }
}
else if (isGrayscale) {
    data[index] = p.r;
} else if (!Alpha) {
    data[index] = p.r;
    data[index + 1] = p.g;
    data[index + 2] = p.b;
} else {
    data[index] = p.r;
    data[index + 1] = p.g;
    data[index + 2] = p.b;
    data[index + 3] = p.a;
}
}

if (format == "png") {
    if (!stbi_write_png((filename + ".png").c_str(), width, height, channels, data, width * channels)) {
        cout << "Gagal save PNG" << endl;
    } else {
        cout << "======" << endl;
        cout << endl;
        cout << "PNG berhasil disave" << endl;
        cout << endl;

        cout << "======" << endl;
    }
} else if (format == "jpg") {
    if (!stbi_write_jpg((filename + ".jpg").c_str(), width, height, channels, data, 50)) {
        cout << "Gagal save JPG" << endl;
    } else {
        cout << "======" << endl;
        cout << endl;
        cout << "JPG berhasil disave" << endl;
        cout << endl;

    }
} else if (format == "jpeg") {

```

```

        if (!stbi_write_jpg((filename + ".jpeg").c_str(), width, height, channels, data, 50)) {
            cout << "Gagal save JPEG" << endl;
        } else {
            cout << "======" << endl;
            cout << endl;
            cout << "JPEG berhasil disave" << endl;
            cout << endl;

        }
    }

    delete[] data;
}

string getFileExtension(const string& filename) {
    size_t dotPos = filename.find_last_of(".");
    if (dotPos == string::npos) return "";
    return filename.substr(dotPos + 1);
}

double getFileSize(const string& fileName) {
    ifstream file(fileName, ios::binary | ios::ate);
    return file.tellg()/1024;
}

double calculateCompression(double originalSize, double compressedSize) {
    return (1 - (compressedSize / originalSize)) * 100;
}

```

utils.hpp

```

#ifndef VARIANS_HPP
#define VARIANS_HPP

#include "quadtree.hpp"

using namespace std;

double calculateMean(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY, int
colorChannel);
double calculateVariance(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY, int
colorChannel);
double calculateRGBAVariance(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY);

double calculateMad(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY, int
colorChannel);
double calculateRGBAMad(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY);

double calculateMPDMaxMin(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY, int
colorChannel);
double calculateMPD(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY);

```

```

double calculateEntropy(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY, int
colorChannel);
double calculateRGBAEentropyTotal(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int
sizeY);

Pixel getAverageColor(const vector<vector<Pixel>>* mat, int x, int y, int sizeX, int sizeY);

double calculateCovariance(const vector<vector<Pixel>>* matrix1, const vector<vector<Pixel>>* matrix2,
int x, int y, int sizeX, int sizeY, int colorChannel);
double calculateSSIM(const vector<vector<Pixel>>* original, const vector<vector<Pixel>>* compressed, int
x, int y, int sizeX, int sizeY, int colorChannel);
double calculateSSIM_RGB(const vector<vector<Pixel>>* original, const vector<vector<Pixel>>* compressed,
int x, int y, int sizeX, int sizeY);

#endif

```

utils.cpp

```

#include "include/quadtreenode.hpp"
#include <cmath>

double calculateMean(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY, int
colorChannel) {
    double sum = 0.0;
    int count = 0;

    if (x < 0 || y < 0 || sizeX <= 0 || sizeY <= 0 || y >= static_cast<int>(matrix->size())) {
        return 0.0;
    }

    for (int j = y; j < y + sizeY && j < static_cast<int>(matrix->size()); j++) {
        for (int i = x; i < x + sizeX && i < static_cast<int>((*matrix)[j].size()); i++) {
            double pixelValue = 0.0;
            if (colorChannel == 0) // Red
                pixelValue = static_cast<double>((*matrix)[j][i].r);
            else if (colorChannel == 1) // Green
                pixelValue = static_cast<double>((*matrix)[j][i].g);
            else if (colorChannel == 2) // Blue
                pixelValue = static_cast<double>((*matrix)[j][i].b);
            else if (colorChannel == 3) // Alpha
                pixelValue = static_cast<double>((*matrix)[j][i].a);

            sum += pixelValue;
            count++;
        }
    }

    return (count > 0) ? (sum / count) : 0.0;
}

/* Varians */

```

```

double calculateVariance(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY, int
colorChannel) {
    double mean = calculateMean(matrix, x, y, sizeX, sizeY, colorChannel);

    double sum = 0.0;
    int count = 0;

    for (int j = y; j < y + sizeY && j < static_cast<int>(matrix->size()); j++) {
        for (int i = x; i < x + sizeX && i < static_cast<int>((*matrix)[j].size()); i++) {
            double pixelValue;
            if (colorChannel == 0) // Red
                pixelValue = (*matrix)[j][i].r;
            else if (colorChannel == 1) // Green
                pixelValue = (*matrix)[j][i].g;
            else if (colorChannel == 2) // Blue
                pixelValue = (*matrix)[j][i].b;
            else if (colorChannel == 3) // Alpha
                pixelValue = (*matrix)[j][i].a;

            sum += (pixelValue - mean) * (pixelValue - mean);
            count++;
        }
    }

    return (count > 0) ? (sum / count) : 0.0;
}

double calculateRGBAVariance(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY) {
    double varianceR = calculateVariance(matrix, x, y, sizeX, sizeY, 0);
    double varianceG = calculateVariance(matrix, x, y, sizeX, sizeY, 1);
    double varianceB = calculateVariance(matrix, x, y, sizeX, sizeY, 2);
    double varianceA = calculateVariance(matrix, x, y, sizeX, sizeY, 3);

    return (varianceR + varianceG + varianceB + varianceA) / 4.0;
}

/* Varians */

/* MAD */

double calculateMad(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY, int
colorChannel) {
    double mean = calculateMean(matrix, x, y, sizeX, sizeY, colorChannel);

    double sum = 0.0;
    int count = 0;

    for (int j = y; j < y + sizeY && j < static_cast<int>(matrix->size()); j++) {
        for (int i = x; i < x + sizeX && i < static_cast<int>((*matrix)[j].size()); i++) {
            double pixelValue;
            if (colorChannel == 0) // Red

```

```

        pixelValue = (*matrix)[j][i].r;
    else if (colorChannel == 1) // Green
        pixelValue = (*matrix)[j][i].g;
    else if (colorChannel == 2) // Blue
        pixelValue = (*matrix)[j][i].b;
    else if (colorChannel == 3) // Alpha
        pixelValue = (*matrix)[j][i].a;

        double diff = pixelValue - mean;
        sum += (diff < 0) ? -diff : diff;
        count++;
    }
}

return (count > 0) ? (sum / count) : 0.0;
}

double calculateRGBAMad(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY) {
    double MadR = calculateMad(matrix, x, y, sizeX, sizeY, 0);
    double MadG = calculateMad(matrix, x, y, sizeX, sizeY, 1);
    double MadB = calculateMad(matrix, x, y, sizeX, sizeY, 2);
    double MadA = calculateMad(matrix, x, y, sizeX, sizeY, 3);

    return (MadR + MadG + MadB + MadA) / 4.0;
}

/* MAD */

/* Max Pixel Difference (MPD) */

double calculateMPDMaxMin(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY, int
colorChannel) {
    double maxnya = -1;
    double minnya = 256;

    for (int j = y; j < y + sizeY && j < static_cast<int>(matrix->size()); j++) {
        for (int i = x; i < x + sizeX && i < static_cast<int>((*matrix)[j].size()); i++) {
            double pixelValue;
            if (colorChannel == 0) // Red
                pixelValue = (*matrix)[j][i].r;
            else if (colorChannel == 1) // Green
                pixelValue = (*matrix)[j][i].g;
            else if (colorChannel == 2) // Blue
                pixelValue = (*matrix)[j][i].b;
            else if (colorChannel == 3) // Alpha
                pixelValue = (*matrix)[j][i].a;

            if (pixelValue > maxnya) {
                maxnya = pixelValue;
            }
        }
    }
}
```

```

        if (pixelValue < minnya) {
            minnya = pixelValue;
        }
    }

    return (maxnya - minnya);
}

double calculateMPD(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY) {
    double diffR = calculateMPDMaxMin(matrix, x, y, sizeX, sizeY, 0);
    double diffG = calculateMPDMaxMin(matrix, x, y, sizeX, sizeY, 1);
    double diffB = calculateMPDMaxMin(matrix, x, y, sizeX, sizeY, 2);
    double diffA = calculateMPDMaxMin(matrix, x, y, sizeX, sizeY, 3);

    return (diffR + diffG + diffB + diffA) / 4.0;
}

/* Max Pixel Difference (MPD) */

/* Entropy */

double calculateEntropy(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY, int colorChannel) {
    //pertama, bikin dl histogram isinya pixel value yang possible
    vector<int> histogram(256, 0);
    int totalPixels = 0;

    for (int j = y; j < y + sizeY && j < static_cast<int>(matrix->size()); j++) {
        for (int i = x; i < x + sizeX && i < static_cast<int>((*matrix)[j].size()); i++) {
            int pixelValue;
            if (colorChannel == 0) // Red
                pixelValue = (*matrix)[j][i].r;
            else if (colorChannel == 1) // Green
                pixelValue = (*matrix)[j][i].g;
            else if (colorChannel == 2) // Blue
                pixelValue = (*matrix)[j][i].b;
            else if (colorChannel == 3) // Alpha
                pixelValue = (*matrix)[j][i].a;

            histogram[pixelValue]++;
            totalPixels++;
        }
    }

    // H = -sum(p(i) * log2(p(i)))
    double entropy = 0.0;

    for (int i = 0; i < 256; i++) {
        if (histogram[i] > 0) {
            double probability = static_cast<double>(histogram[i]) / totalPixels;

```

```

        entropy -= probability * log2(probability);
    }

    return entropy;
}

double calculateRGBAEentropyTotal(const vector<vector<Pixel>>* matrix, int x, int y, int sizeX, int sizeY)
{
    double entropyR = calculateEntropy(matrix, x, y, sizeX, sizeY, 0);
    double entropyG = calculateEntropy(matrix, x, y, sizeX, sizeY, 1);
    double entropyB = calculateEntropy(matrix, x, y, sizeX, sizeY, 2);
    double entropyA = calculateEntropy(matrix, x, y, sizeX, sizeY, 3);

    return (entropyR + entropyG + entropyB + entropyA) / 4.0;
}

/* Entropy */

/* Normalisasi */

Pixel getAverageColor(const vector<vector<Pixel>>* mat, int x, int y, int sizeX, int sizeY) {
    long long totalR = 0, totalG = 0, totalB = 0, totalA = 0;
    int totalPixels = sizeX * sizeY;

    for (int i = y; i < y + sizeY; ++i) {
        for (int j = x; j < x + sizeX; ++j) {
            totalR += (*mat)[i][j].r;
            totalG += (*mat)[i][j].g;
            totalB += (*mat)[i][j].b;
            totalA += (*mat)[i][j].a;
        }
    }

    Pixel avgColor;
    avgColor.r = static_cast<unsigned char>(totalR / totalPixels);
    avgColor.g = static_cast<unsigned char>(totalG / totalPixels);
    avgColor.b = static_cast<unsigned char>(totalB / totalPixels);
    avgColor.a = static_cast<unsigned char>(totalA / totalPixels);

    return avgColor;
}

/* Normalisasi */

/* SSIM (BONUS) */

const double C1 = (0.01 * 255) * (0.01 * 255);
const double C2 = (0.03 * 255) * (0.03 * 255);

double calculateCovariance(const vector<vector<Pixel>>* matrix1, const vector<vector<Pixel>>* matrix2,

```

```

        int x, int y, int sizeX, int sizeY, int colorChannel) {
    double mean1 = calculateMean(matrix1, x, y, sizeX, sizeY, colorChannel);
    double mean2 = calculateMean(matrix2, x, y, sizeX, sizeY, colorChannel);
    double sum = 0.0;
    int count = 0;

    for (int j = y; j < y + sizeY && j < static_cast<int>(matrix1->size()) && j <
static_cast<int>(matrix2->size()); j++) {
        for (int i = x; i < x + sizeX && i < static_cast<int>((*matrix1)[j].size()) && i <
static_cast<int>((*matrix2)[j].size()); i++) {
            double pixelValue1, pixelValue2;

            if (colorChannel == 0) {
                pixelValue1 = static_cast<double>((*matrix1)[j][i].r);
                pixelValue2 = static_cast<double>((*matrix2)[j][i].r);
            } else if (colorChannel == 1) {
                pixelValue1 = static_cast<double>((*matrix1)[j][i].g);
                pixelValue2 = static_cast<double>((*matrix2)[j][i].g);
            } else if (colorChannel == 2) {
                pixelValue1 = static_cast<double>((*matrix1)[j][i].b);
                pixelValue2 = static_cast<double>((*matrix2)[j][i].b);
            } else {
                pixelValue1 = static_cast<double>((*matrix1)[j][i].a);
                pixelValue2 = static_cast<double>((*matrix2)[j][i].a);
            }

            sum += (pixelValue1 - mean1) * (pixelValue2 - mean2);
            count++;
        }
    }

    return (count > 0) ? (sum / count) : 0.0;
}

double calculateSSIM(const vector<vector<Pixel>>* original, const vector<vector<Pixel>>* compressed,
                     int x, int y, int sizeX, int sizeY, int colorChannel) {
    if (x < 0 || y < 0 || sizeX <= 0 || sizeY <= 0 || y >= static_cast<int>(original->size()) || y >= static_cast<int>(compressed->size())) {
        return 0.0;
    }

    double mu1 = calculateMean(original, x, y, sizeX, sizeY, colorChannel);
    double mu2 = calculateMean(compressed, x, y, sizeX, sizeY, colorChannel);

    double sigma1_sq = calculateVariance(original, x, y, sizeX, sizeY, colorChannel);
    double sigma2_sq = calculateVariance(compressed, x, y, sizeX, sizeY, colorChannel);

    double sigma12 = calculateCovariance(original, compressed, x, y, sizeX, sizeY, colorChannel);

    double numerator = (2 * mu1 * mu2 + C1) * (2 * sigma12 + C2);
    double denominator = (mu1 * mu1 + mu2 * mu2 + C1) * (sigma1_sq + sigma2_sq + C2);
}

```

```

if (denominator < 1e-10) {
    return 1.0;
}

double ssim = numerator / denominator;

return max(0.0, min(1.0, ssim));
}

double calculateSSIM_RGB(const vector<vector<Pixel>>* original, const vector<vector<Pixel>>* compressed,
                         int x, int y, int sizeX, int sizeY) {
const double wR = 0.2125, wG = 0.7154, wB = 0.0721;
double totalWeight = wR + wG + wB;

double SSIM_R = calculateSSIM(original, compressed, x, y, sizeX, sizeY, 0);
double SSIM_G = calculateSSIM(original, compressed, x, y, sizeX, sizeY, 1);
double SSIM_B = calculateSSIM(original, compressed, x, y, sizeX, sizeY, 2);

return (wR * SSIM_R + wG * SSIM_G + wB * SSIM_B) / totalWeight;
}

/* SSIM (BONUS) */

```

bonus.hpp

```

#ifndef BONUS_HPP
#define BONUS_HPP

#include <string>
#include <utility>
#include "quadtree.hpp"

class Kompresi {
public:
    static std::pair<double, double> findOptimalParameters(
        const Image& img,
        double targetCompressionRate,
        const std::string& filename,
        const std::string& outputBase,
        bool useSSIM
    );

    static double testCompression(
        const Image& img,
        double blockSize,
        double threshold,
        double inputFileSize,
        const std::string& filename,
        const std::string& outputBase,

```

```
    bool useSSIM
);
};

#endif // BONUS_HPP
```

bonus.cpp

```
#include "include/bonus.hpp"
#include "include/quadtree.hpp"
#include "include/utils.hpp"
#include <iostream>
#include <algorithm>
#include <cmath>

using namespace std;

double Kompresi::testCompression(
    const Image& img,
    double blockSize,
    double threshold,
    double inputFileSize,
    const string& filename,
    const string& outputBase,
    bool useSSIM
) {
    string tempOutputFilename = outputBase + "_temp";
    const vector<vector<Pixel>>* pixelMatrix = &img.pixels;

    QuadTree* root = QuadTree::buildQuadTree(
        pixelMatrix, 0.0, 0.0, static_cast<double>(img.width), static_cast<double>(img.height),
        blockSize, threshold,
        false, false, false, false, useSSIM
    );

    vector<vector<Pixel>> reconstructImageMatrix(img.height, vector<Pixel>(img.width));
    root->reconstructImage(root, reconstructImageMatrix, 0.0, 0.0);

    string ext = getFileExtension(filename);
    saveReconstructedImage(tempOutputFilename, reconstructImageMatrix, ext);

    double outputFileSize = getFileSize(tempOutputFilename + "." + ext);
    double compressionRate = calculateCompression(inputFileSize, outputFileSize);

    remove((tempOutputFilename + "." + ext).c_str());
    delete root;
}

pair<double, double> Kompresi::findOptimalParameters()
```

```

const Image& img,
double targetCompressionRate,
const string& filename,
const string& outputBase,
bool useSSIM
) {
    double inputFileSize = getFileSize(filename);

    double minBlockSize = 4.0;
    double bestThreshold = 0.0;
    double closestCompressionRate = 0.0;
    double minDifference = 100.0;

    int minDimension = min(img.width, img.height);
    double maxBlockSize = 1.0;
    while (maxBlockSize * 8 <= minDimension) {
        maxBlockSize *= 8;
    }

    //caching hasil kompresi untuk menghindari perhitungan berulang
    struct CacheEntry {
        double blockSize;
        double threshold;
        double compressionRate;
    };
    vector<CacheEntry> cache;

    //loop dari ukuran blok paling gede ke paling kecil (dibagi 44 tiap iterasi)
    for (double blockSize = maxBlockSize; blockSize >= 4.0; blockSize /= 4) {
        cout << "Block size: " << blockSize << endl;

        double startThreshold = 0.1;
        double endThreshold = 0.9;

        double compressionRate = testCompression(img, blockSize, startThreshold,
                                                inputFileSize, filename, outputBase, useSSIM);
        cache.push_back({blockSize, startThreshold, compressionRate});

        cout << " Threshold: " << startThreshold
            << ", Compression: " << compressionRate << "%" << endl;

        //jika hasil jauh dari target (lebih dari 10%), skip block size ini
        if (abs(compressionRate - targetCompressionRate) > 10.0) {
            cout << " Skip. Masi Jauh dari target" << endl;
            continue;
        }

        double difference = abs(compressionRate - targetCompressionRate);
        if (difference < minDifference) {
            minDifference = difference;
            closestCompressionRate = compressionRate;
        }
    }
}

```

```

        minBlockSize = blockSize;
        bestThreshold = startThreshold;
    }

    double endCompressionRate = testCompression(img, blockSize, endThreshold,
                                                inputFileSize, filename, outputBase, useSSIM);
    cache.push_back({blockSize, endThreshold, endCompressionRate});

    cout << "  End test - Threshold: " << endThreshold
        << ", Compression: " << endCompressionRate << "%" << endl;

    difference = abs(endCompressionRate - targetCompressionRate);
    if (difference < minDifference) {
        minDifference = difference;
        closestCompressionRate = endCompressionRate;
        minBlockSize = blockSize;
        bestThreshold = endThreshold;
    }

    // Jika kedua threshold menghasilkan kompresi di sisi yang berbeda dari target,
    // maka kita bisa menemukan nilai di antaranya
    bool shouldSearch = false;

    if ((compressionRate < targetCompressionRate && endCompressionRate > targetCompressionRate) ||
        (compressionRate > targetCompressionRate && endCompressionRate < targetCompressionRate)) {
        shouldSearch = true;
    }

    if (!shouldSearch) {
        if (minDifference < 5.0) {
            cout << "  Dapat yang toleransinya <= 5%" << endl;
            return make_pair(minBlockSize, bestThreshold);
        }
        continue;
    }

    double step = 0.1;

    double currentThreshold = startThreshold;
    if (abs(endCompressionRate - targetCompressionRate) < abs(compressionRate -
targetCompressionRate)) {
        currentThreshold = endThreshold - step;
    } else {
        currentThreshold = startThreshold + step;
    }

    while (currentThreshold > startThreshold && currentThreshold < endThreshold) {
        double currentCompressionRate = testCompression(img, blockSize, currentThreshold,
                                                        inputFileSize, filename, outputBase, useSSIM);
        cache.push_back({blockSize, currentThreshold, currentCompressionRate});
    }
}

```

```

        cout << " Testing - Threshold: " << currentThreshold
        << ", Compression: " << currentCompressionRate << "%" << endl;

        difference = abs(currentCompressionRate - targetCompressionRate);
        if (difference < minDifference) {
            minDifference = difference;
            closestCompressionRate = currentCompressionRate;
            minBlockSize = blockSize;
            bestThreshold = currentThreshold;

            if (difference < 5.0) {
                cout << " Didapatkan yang percentagenya (< 5% difference)" << endl;
                return make_pair(minBlockSize, bestThreshold);
            }
        }

        //nentuin arah pencarian berikutnya
        if ((useSSIM && currentCompressionRate < targetCompressionRate) ||
            (!useSSIM && currentCompressionRate > targetCompressionRate)) {
            currentThreshold += step;
        } else {
            currentThreshold -= step;
        }
    }

    cout << "Aproksimasi Terbaik: " << closestCompressionRate
    << "% (difference: " << minDifference << "%)" << endl;

    return make_pair(minBlockSize, bestThreshold);
}

```

gifer.hpp

```

#ifndef GIF_HPP
#define GIF_HPP

#include "image.hpp"

class QuadTree;

void gifMaker(int width, int height, const char* gifOutput, int maxDepth, QuadTree* root,
std::vector<std::vector<Pixel>>& reconstructImageMatrix);

#endif

```

gifer.cpp

```
#include "include/gifer.hpp"
```

```
#include "include/quadtree.hpp"
#include "gif.h"

#include <iostream>

void gifMaker(int width, int height, const char* gifOutput, int maxDepth, QuadTree* root,
std::vector<std::vector<Pixel>>& reconstructImageMatrix) {
    GifWriter writer;

    if (!GifBegin(&writer, gifOutput, width, height, 10)) {
        std::cerr << "Gagal membuat GIF\n";
        return;
    }

    for (int i = maxDepth; i >= 0; i--) {
        root->reconstructImageFrame(root, reconstructImageMatrix, 0, 0, i, maxDepth + 1);

        std::vector<uint8_t> frameData(width * height * 4);
        for (int y = 0; y < height; ++y) {
            for (int x = 0; x < width; ++x) {
                int idx = (y * width + x) * 4;
                const Pixel& p = reconstructImageMatrix[y][x];
                frameData[idx + 0] = p.r;
                frameData[idx + 1] = p.g;
                frameData[idx + 2] = p.b;
                frameData[idx + 3] = 255;
            }
        }

        GifWriteFrame(&writer, frameData.data(), width, height, 100);
    }

    GifEnd(&writer);

    cout << endl;
    cout << "GIF berhasil dibuat" << endl;
}
```

BAB 4

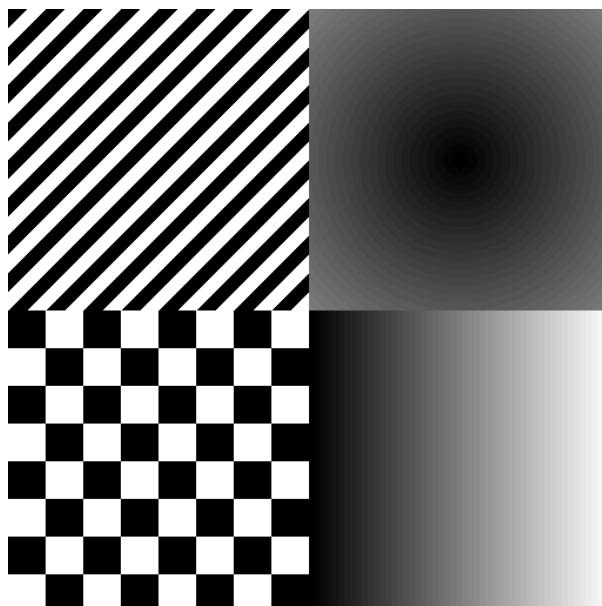
Testing

4.1 Hasil Kompresi dan Output Program

1. Pengujian Threshold Sintetis

Pengujian untuk melihat behaviour dari setiap threshold kompresi yang digunakan, dengan block size minimum = 4

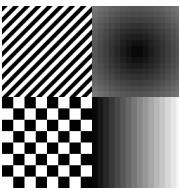
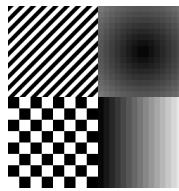
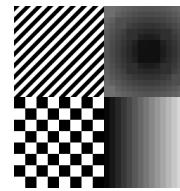
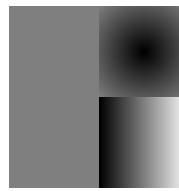
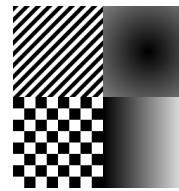
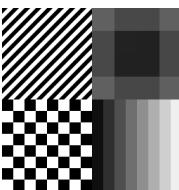
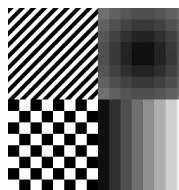
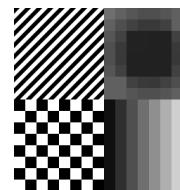
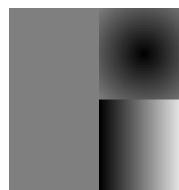
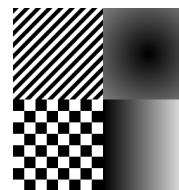
Gambar yang digunakan:



Menguji kompresi pada garis lurus, diagonal, pola checker, gradien lurus, gradien lingkaran.

Spesifikasi gambar: PNG, 8 bit, 7,89KB, 512x512px.

Test	Variance	MPD	MAD	Entropy	SSIM
High Quality					
Threshold, Kompresi	2 57,14%	4 64,28%	2 71,42%	0,5 50%	0,1 50%

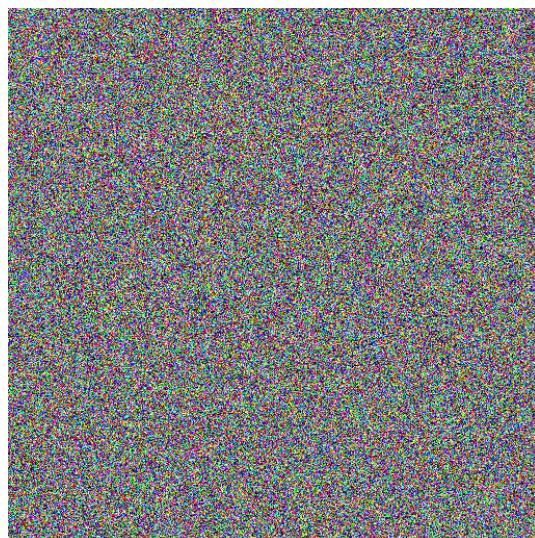
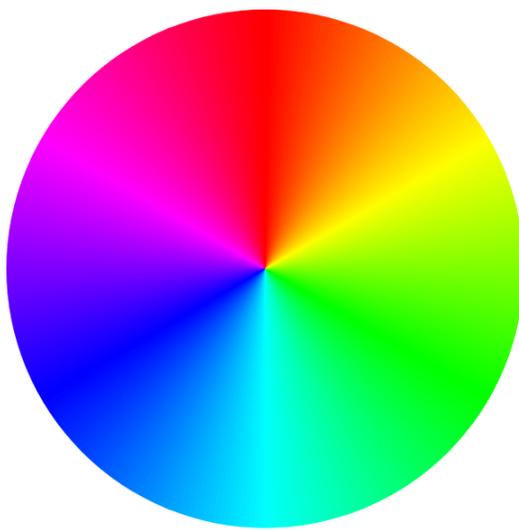
Medium Quality					
Threshold, Kompresi	16 78,57%	16 78,57%	4 78,57%	1 50%	0.5 50%
Low Quality					
Threshold, Kompresi	128 78,57%	32 78,57%	8 78,57%	2 64,28%	0.9 50%

Pada pengujian ini, Variansi, MAD, dan MPD menunjukkan performa yang baik pada pola diagonal dan checker, dengan tingkat kompresi yang cukup baik. Akan tetapi, ketiga metode kompresi ini kehilangan detail yang cukup banyak pada gradien, khususnya pada threshold error yang lebih tinggi.

Entropy menunjukkan performa yang lebih baik dalam mempertahankan detail pada kedua gradien pada bagian kanan gambar, tetapi detail pada garis diagonal dan pola checker hilang.

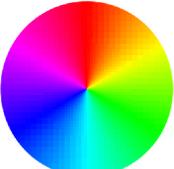
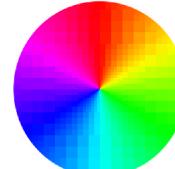
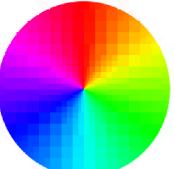
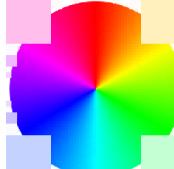
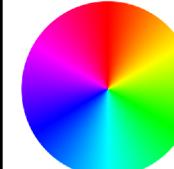
Metode SSIM tidak menunjukkan perbedaan kualitas yang signifikan antara ketiga threshold pada pengujian ini, dan ketiga gambar memiliki output yang sama.

2. Pengujian Sintetis Lain



Pengujian sintetis untuk melihat behaviour berbagai threshold kompresi dengan warna dan worst case scenario (noise), dengan spesifikasi kompresi medium quality pada percobaan sebelumnya.

Spesifikasi gambar 1: PNG, 24 bit, 90,4KB, 600x600px. gambar 2: PNG, 24 bit, 657KB, 512x512px.

Test	Variance	MPD	MAD	Entropy	SSIM
Gambar 1					
Kompresi	72,22%	75,55%	76,67%	87,77%	32,22%
Gambar 2					
Kompresi	53,88%	53,88%	53,88%	53,88%	53,88%

Pengujian ini menguji kemali kelima metode dengan warna dan noise. Threshold menggunakan pendekatan yang dianggap cukup dekat, mengikuti threshold pada medium quality pada tabel sebelumnya. Min. block size juga menggunakan 4 pada pengujian ini.

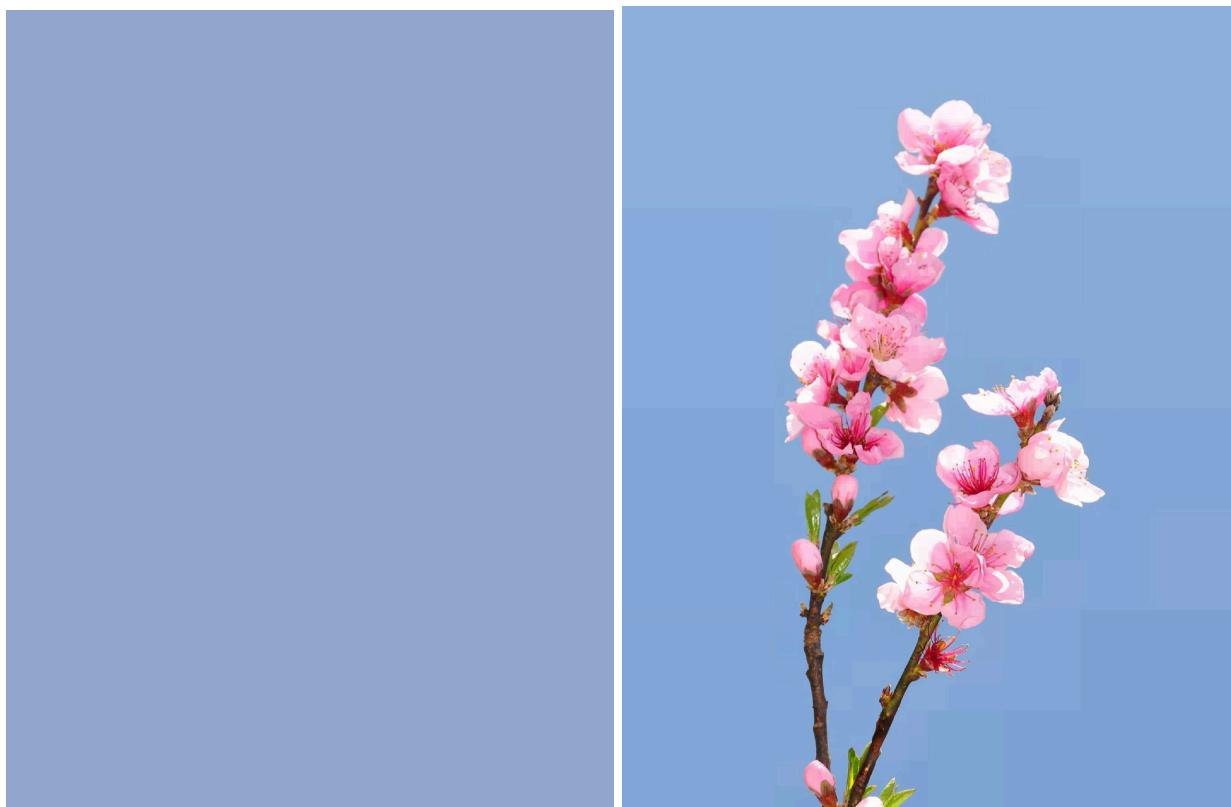
Pada pengujian warna, SSIM memberikan hasil yang paling baik, tetapi dengan tingkat kompresi yang paling rendah diantara metode lainnya. Variansi memberikan trade-off antara tingkat kompresi dan pertahanan detail warna pada gradien yang paling baik. MPD dan MAD menunjukkan artifact yang lebih besar dari variansi, dengan peningkatan kompresi yang kecil. Entropy memberikan tingkat kompresi tertinggi, tetapi dengan artifact yang paling besar diantara metode lainnya.

Pengujian noise merupakan worst case scenario, dimana setiap metode menghasilkan gambar yang sama.

3. Pengujian Variasi Gambar - Utama (menyeluruh)



Spesifikasi gambar: JPG, 24 bit, 3.1MB, 2406x3164px.



vbunga.gif

vbunga.jpg

Spesifikasi kompresi: block size 4, threshold variansi 16,

```
=====
WELCOME TO COMPRESS QUADTREE TERGACOR!
=====
Silahkan masukkan path file gambar: D:\Skool\univ\tucan\bunga.jpg
=====
Dimensi Gambar: 2406 x 3164
=====
Masukkan target kompresi (0.01 - 1): 0
=====
Pilih metode error:
1: Variance
2: MPD
3: MAD
4: Entropy
5: SSIM
=====
Pilihan: 1
=====
Masukkan threshold error: 16
Masukkan minimum block size: 4
=====
Masukkan path file gambar output: vbunga
Apakah anda ingin membuat GIF? (y/n): n
Tidak membuat gif
=====
JPG berhasil disave

=====
HASIL KOMPRESI
=====
Waktu eksekusi: 4881 ms
Ukuran file input: 3177 KB
Ukuran file output: 223 KB
Persentase kompresi: 92.9808 %

Kedalaman Quadtree: 10
Jumlah simpul pada Tree: 138685
```

Tanpa GIF

```
=====
WELCOME TO COMPRESS QUADTREE TERGACOR!
=====
Silahkan masukkan path file gambar: D:\Skool\univ\tucan\bunga.jpg
=====
Dimensi Gambar: 2406 x 3164
=====
Masukkan target kompresi (0.01 - 1): 0
=====
Pilih metode error:
1: Variance
2: MPD
3: MAD
4: Entropy
5: SSIM
=====
Pilihan: 1
=====
Masukkan threshold error: 16
Masukkan minimum block size: 4
=====
Masukkan path file gambar output: vbunga
Apakah anda ingin membuat GIF? (y/n): y
Masukkan path file GIF output: vbunga
=====

JPG berhasil disave

GIF berhasil dibuat

=====
HASIL KOMPRESI
=====
Waktu eksekusi: 22629 ms
Ukuran file input: 3177 KB
Ukuran file output: 223 KB
Persentase kompresi: 92.9808 %

Kedalaman Quadtree: 10
Jumlah simpul pada Tree: 138685
```

Dengan GIF

Pengujian dengan memvariasiakan block size pada threshold



bunga64.jpg

Spesifikasi kompresi gambar 1: block size 64, threshold variansi 16

```
=====
          HASIL KOMPRESI
=====
Waktu eksekusi: 4836 ms
Ukuran file input: 3177 KB
Ukuran file output: 205 KB
Persentase kompresi: 93.5474 %

Kedalaman Quadtree: 8
Jumlah simpul pada Tree: 15973
=====
```

Spesifikasi kompresi gambar 2: block size 256, threshold variansi 16

```
=====
          HASIL KOMPRESI
=====
Waktu eksekusi: 4611 ms
Ukuran file input: 3177 KB
Ukuran file output: 184 KB
Persentase kompresi: 94.2084 %

Kedalaman Quadtree: 7
Jumlah simpul pada Tree: 4925
=====
```



bunga256.jpg

Pengujian dengan threshold lain, acuan threshold sama dengan quality medium pada percobaan pertama.



MPD - bungampd.jpg

Output MPD:

```
=====
          HASIL KOMPRESI
=====
Waktu eksekusi: 3078 ms
Ukuran file input: 3177 KB
Ukuran file output: 224 KB
Persentase kompresi: 92.9493 %

Kedalaman Quadtree: 10
Jumlah simpul pada Tree: 123329
=====
```

Output MAD:

```
=====
          HASIL KOMPRESI
=====
Waktu eksekusi: 5055 ms
Ukuran file input: 3177 KB
Ukuran file output: 221 KB
Persentase kompresi: 93.0438 %

Kedalaman Quadtree: 10
Jumlah simpul pada Tree: 101829
=====
```



Entropy - bungae.jpg

Output Entropy:

```
=====
HASIL KOMPRESI
=====
Waktu eksekusi: 13322 ms
Ukuran file input: 3177 KB
Ukuran file output: 227 KB
Persentase kompresi: 92.8549 %

Kedalaman Quadtree: 10
Jumlah simpul pada Tree: 1356289
=====
```

Output SSIM:

```
=====
HASIL KOMPRESI
=====
Waktu eksekusi: 10180 ms
Ukuran file input: 3177 KB
Ukuran file output: 227 KB
Persentase kompresi: 92.8549 %

Kedalaman Quadtree: 10
Jumlah simpul pada Tree: 1310721
=====
```



SSIM - bungassim.jpg



Auto - target 85% - bungaauto.jpg

```
=====
WELCOME TO COMPRESS QUADTREE TERGACOR!
=====
Silahkan masukkan path file gambar: D:\Skool\univ\tucil2test2\bungaauto.jpg
=====
Dimensi Gambar: 2406 x 3164
=====
Masukkan target kompresi (0.01 - 1): 0.85
Masukkan nama file gambar output: bungaauto
=====

Apakah anda ingin membuat GIF? (y/n): n
Tidak membuat gif

Mencari parameter optimal untuk target kompresi 85%...
Block size: 512
=====

JPG berhasil disave
Threshold: 0.1, Compression: 94.1454%
=====

JPG berhasil disave
End test - Threshold: 0.9, Compression: 94.1454%
Block size: 128
=====

JPG berhasil disave
Threshold: 0.1, Compression: 93.4529%
=====

JPG berhasil disave
End test - Threshold: 0.9, Compression: 93.4529%
Block size: 32
=====

JPG berhasil disave
Threshold: 0.1, Compression: 92.9808%
```

```
=====
JPG berhasil disave

End test - Threshold: 0.9, Compression: 92.9808%
Block size: 8
=====

JPG berhasil disave
Threshold: 0.1, Compression: 92.8549%
=====

JPG berhasil disave
Threshold: 0.1, Compression: 92.8549%
Aproksimasi Terbaik: 92.8549% (difference: 7.8548%)

Parameter optimal ditemukan:
- Block size: 8
- Threshold: 0.1
=====

JPG berhasil disave

=====
HASIL KOMPRESI
=====
Waktu eksekusi: 76614 ms
Ukuran file input: 3177 KB
Ukuran file output: 227 KB
Percentase kompresi: 92.8549 %

Kedalaman Quadtree: 10
Jumlah simpul pada Tree: 1310721
```

4. Tes Variasi Gambar Lain (hanya diujikan sebagian)



Spesifikasi gambar: JPEG, 24 bit, 39,4 KB, 500x750px. Output:trukvariansi.jpeg

Keunikan : JPEG, ukuran file kecil

```
=====
WELCOME TO COMPRESS QUADTREE TERGACOR!
=====
Silahkan masukkan path file gambar: D:\skool\univ\
=====
Dimensi Gambar: 500 x 750
=====
Masukkan target kompresi (0.01 - 1): 0
=====
Pilih metode error:
1: Variance
2: MPD
3: MAD
4: Entropy
5: SSIM
=====
Pilihan: 1
=====
Masukkan threshold error: 16
Masukkan minimum block size: 4
=====
Masukkan path file gambar output: trukvariansi
Apakah anda ingin membuat GIF? (y/n): n
Tidak membuat gif
=====
```

```
=====
JPEG berhasil disave
=====
HASIL KOMPRESI
=====
Waktu eksekusi: 376 ms
Ukuran file input: 61 KB
Ukuran file output: 39 KB
Persentase kompresi: 36.0656 %
=====
Kedalaman Quadtree: 8
Jumlah simpul pada Tree: 55633
=====
```



*Background biru bukan bagian dari PNG, untuk menunjukkan transparansi.

Spesifikasi gambar: PNG, 32bit, 3,87MB, 2000x1331px. Output: 32pngcprs.png

Keunikan : PNG 32 bit, Transparansi

```
=====
WELCOME TO COMPRESS QUADTREE TERGACOR!
=====

Silahkan masukkan path file gambar: D:\Skool\

=====
Dimensi Gambar: 2000 x 1331

=====
Masukkan target kompresi (0.01 - 1): 0

=====
Pilih metode error:
1: Variance
2: MPD
3: MAD
4: Entropy
5: SSIM

=====
Pilihan: 2

=====
Masukkan threshold error: 16
Masukkan minimum block size: 4

=====
```

```
=====
Masukkan path file gambar output: 32pngcprs
Apakah anda ingin membuat GIF? (y/n): n
Tidak membuat gif

=====

PNG berhasil disave

=====

=====
HASIL KOMPRESI
=====

Waktu eksekusi: 2343 ms
Ukuran file input: 3970 KB
Ukuran file output: 639 KB
Persentase kompresi: 83.9843 %

=====
Kedalaman Quadtree: 10
Jumlah simpul pada Tree: 146517

=====
```



Pengujian alur threshold otomatis + pembuatan gif secara bersamaan, pengujian gif pada png dengan transparency. Output: 32pngauto.png, 32pngauto.gif

```
=====
WELCOME TO COMPRESS QUADTREE TERGACOR!
=====
Silahkan masukkan path file gambar: D:\Skool\univ\tucil2t
=====
Dimensi Gambar: 2000 x 1331
=====
Masukkan target kompresi (0.01 - 1): 0.8
Masukkan nama file gambar output: 32pngauto
=====

Apakah anda ingin membuat GIF? (y/n): y
Masukkan path file GIF output: 32pngauto

Mencari parameter optimal untuk target kompresi 80%...
Block size: 512
=====

PNG berhasil disave

=====
Threshold: 0.1, Compression: 97.0781%
Skip. Masi Jauh dari target
Block size: 128
=====

PNG berhasil disave

=====
Threshold: 0.1, Compression: 96.2469%
Skip. Masi Jauh dari target
Block size: 32
=====

PNG berhasil disave

=====
Threshold: 0.1, Compression: 93.3753%
Skip. Masi Jauh dari target
Block size: 8
=====
```

```
=====
PNG berhasil disave

=====
Threshold: 0.1, Compression: 83.9295%
=====

PNG berhasil disave

=====
End test - Threshold: 0.9, Compression: 84.2317%
Dapat yang toleransinya <= 5%

Parameter optimal ditemukan:
- Block size: 8
- Threshold: 0.1
=====

PNG berhasil disave

=====
GIF berhasil dibuat

=====
HASIL KOMPRESI
=====

Waktu eksekusi: 29753 ms
Ukuran file input: 3970 KB
Ukuran file output: 638 KB
Persentase kompresi: 83.9295 %

Kedalaman Quadtree: 9
Jumlah simpul pada Tree: 327681
=====
```



Spesifikasi gambar: JPG, 24 bit, 17,7 MB, 6916x4616px. Output: pohoncprs.jpg

Keunikan : JPG, ukuran file sangat besar

```
=====
WELCOME TO COMPRESS QUADTREE TERGACOR!
=====
Silahkan masukkan path file gambar: D:\Skool\univ\tuci
=====
Dimensi Gambar: 6916 x 4616
=====
Masukkan target kompresi (0.01 - 1): 0
=====
Pilih metode error:
1: Variance
2: MPD
3: MAD
4: Entropy
5: SSIM
=====
Pilihan: 3
=====
Masukkan threshold error: 4
Masukkan minimum block size: 4
=====
```

```
=====
Masukkan path file gambar output: pohoncprs
Apakah anda ingin membuat GIF? (y/n): n
Tidak membuat gif
=====
JPG berhasil disave
=====
HASIL KOMPRESI
=====
Waktu eksekusi: 34351 ms
Ukuran file input: 18145 KB
Ukuran file output: 1834 KB
Percentase kompresi: 89.8925 %
=====
Kedalaman Quadtree: 11
Jumlah simpul pada Tree: 1123493
=====
```



Spesifikasi gambar: JPG, 24 bit, 3.5 MB, 3840x2160px. Output: sunflow.jpg

Keunikan : Kompleksitas & detail cukup tinggi

```
=====
WELCOME TO COMPRESS QUADTREE TERGACOR!
=====
silahkan masukkan path file gambar: D:\skool\ur
=====
Dimensi Gambar: 3840 x 2160
=====
Masukkan target kompresi (0.01 - 1): 0
=====
Pilih metode error:
1: Variance
2: MPD
3: MAD
4: Entropy
5: SSIM
=====
Pilihan: 4
=====
Masukkan threshold error: 1
Masukkan minimum block size: 4
=====
```

```
=====
Masukkan path file gambar output: sunflow
Apakah anda ingin membuat GIF? (y/n): n
Tidak membuat gif
=====
JPG berhasil disave
=====
HASIL KOMPRESI
=====
Waktu eksekusi: 14953 ms
Ukuran file input: 3585 KB
Ukuran file output: 840 KB
Percentase kompresi: 76.569 %
=====
Kedalaman Quadtree: 10
Jumlah simpul pada Tree: 1168365
=====
```

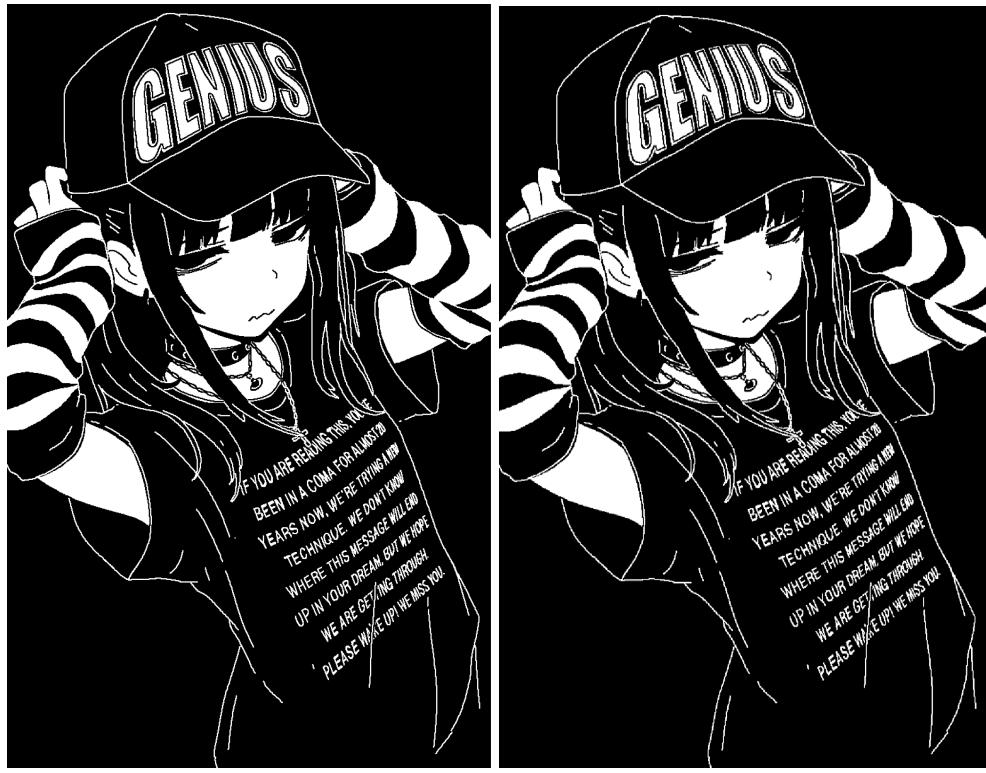


Spesifikasi gambar: PNG, 24bit, 1.27MB, 1200x800px. Output: manusiacprs.jpg

Keunikan : Citra mengandung wajah manusia

```
=====
WELCOME TO COMPRESS QUADTREE TERGACOR!
=====
Silahkan masukkan path file gambar: D:\Skool\univ\tu
=====
Dimensi Gambar: 1200 x 800
=====
Masukkan target kompresi (0.01 - 1): 0
=====
Pilih metode error:
1: Variance
2: MPD
3: MAD
4: Entropy
5: SSIM
=====
Pilihan: 5
=====
Masukkan threshold error: 0.5
Masukkan minimum block size: 4
=====
```

```
=====
Masukkan path file gambar output: manusiacprs
Apakah anda ingin membuat GIF? (y/n): n
Tidak membuat gif
=====
PNG berhasil disave
=====
=====
HASIL KOMPRESI
=====
Waktu eksekusi: 1908 ms
Ukuran file input: 1309 KB
Ukuran file output: 759 KB
Persentase kompresi: 42.0168 %
=====
Kedalaman Quadtree: 9
Jumlah simpul pada Tree: 344065
=====
```

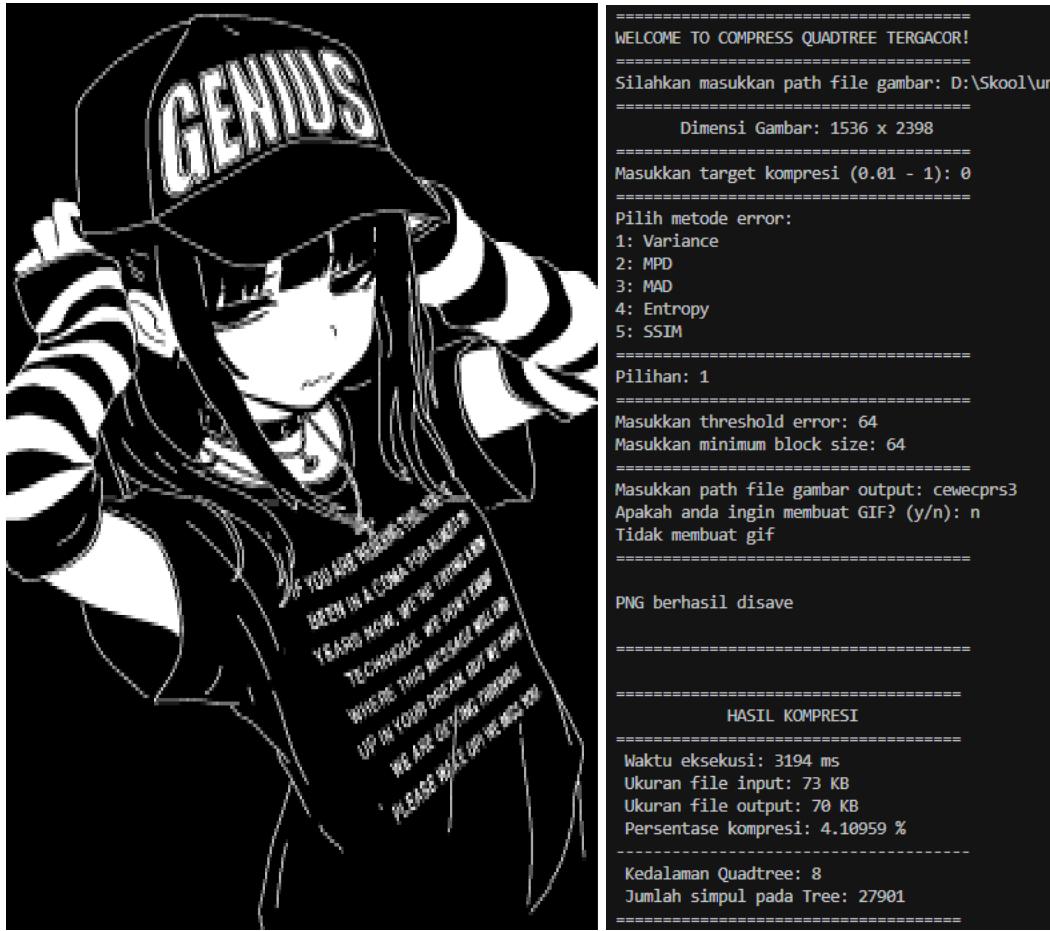


Spesifikasi gambar: PNG, 8bit, 73,4KB,1536x2398px. Output:cewecprs.jpg

Keunikan : **Edge Case**, PNG 8 bit, hanya mengandung 2 warna (hitam/putih, tidak ada abu-abu)

```
=====
WELCOME TO COMPRESS QUADTREE TERGACOR!
=====
Silahkan masukkan path file gambar: D:\Skod
=====
Dimensi Gambar: 1536 x 2398
=====
Masukkan target kompresi (0.01 - 1): 0
=====
Pilih metode error:
1: Variance
2: MPD
3: MAD
4: Entropy
5: SSIM
=====
Pilihan: 1
=====
Masukkan threshold error: 16
Masukkan minimum block size: 4
=====
```

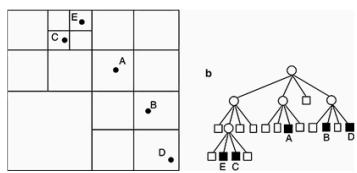
```
=====
Masukkan path file gambar output: cewecprs
Apakah anda ingin membuat GIF? (y/n): n
Tidak membuat gif
=====
PNG berhasil disave
=====
=====
HASTL KOMPREST
=====
Waktu eksekusi: 3646 ms
Ukuran file input: 73 KB
Ukuran file output: 95 KB
Persentase kompresi: -30.137 %
=====
Kedalaman Quadtree: 10
Jumlah simpul pada Tree: 160321
=====
```



Output: cewecprs3.png

Test ini merupakan edge case, dimana kompresi quadtree malah berdampak negatif pada ukuran gambar akhir. Ini dikarenakan metode kompresi png, khususnya bagian deflate (LZSS + Huffman) yang efisiensinya sangat diuntungkan saat gambar hanya memiliki 2 warna saja. Karena kompresi quadtree yang menghasilkan ukuran blok-blok pixel lebih besar dengan mengambil rata-rata warna dari blok yang dihasilkan, blok yang dihasilkan pada pertemuan antara warna putih dan hitam seringkali menjadi abu-abu. Warna abu-abu ini meningkatkan color depth dari gambar, yang meningkatkan kompleksitas dari gambar.

Penyederhanaan geometri yang diberikan oleh kompresi quadtree sebenarnya berdampak positif terhadap ukuran gambar, tetapi penambahan kompleksitas dari bit depth ini melebihi penghematan yang diberikan oleh kompresi quadtree. Ini menyebabkan kompresi quadtree memerlukan parameter threshold dan/atau block size yang lebih tinggi untuk dapat mengkompensasi inefisiensi ini pada kompresi PNG. Bila gambar dikonversikan terlebih dahulu menjadi bentuk JPG/JPEG (dengan catatan konversi menimbulkan ukuran file jauh lebih besar dari PNG + artifacting yang dapat ditimbulkan kompresi JPG/JPEG), metode quadtree akan tetap efektif.



Gambar 3. Struktur Data Quadtree dalam Kompresi Gambar
(Sumber: <https://medium.com/@tannerwyrick/quadtrees-for-image-processing-302536c95c00>.)

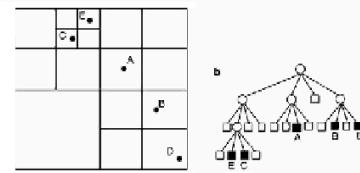
Parameter:

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
Variance	$\sigma_e^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$
	$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$
	$\sigma_c^2 = \text{Variansi tiap kanal warna } c (R, G, B) \text{ dalam satu blok}$
	$P_{i,c} = \text{Nilai piksel pada posisi } i \text{ untuk kanal warna } c$
	$\mu_c = \text{Nilai rata-rata tiap piksel dalam satu blok}$
	$N = \text{Banyaknya piksel dalam satu blok}$



Gambar 4. Struktur Data Quadtree dalam Kompresi Gambar
(Sumber: <https://medium.com/@tannerwyrick/quadtrees-for-image-processing-103536c95c00>.)

Parameter:

1. Error Measurement Methods (Metode Pengukuran Error)

Metode yang dipakai 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 2. Metode Pengukuran Error

Metode	Formula
Variance	$\sigma_e^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$
	$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$
	$\sigma_c^2 = \text{Variansi tiap kanal warna } c (R, G, B) \text{ dalam satu blok}$
	$P_{i,c} = \text{Nilai piksel pada posisi } i \text{ untuk kanal warna } c$
	$\mu_c = \text{Nilai rata-rata tiap piksel dalam satu blok}$
	$N = \text{Banyaknya piksel dalam satu blok}$

Spesifikasi gambar: PNG, 32bit, 60,8KB, 718x886px. Output: testeks.png

Keunikan : Edge Case, Screenshot Windows, Bentuk dokumen/teks

```
=====
WELCOME TO COMPRESS QUADTREE TERGACOR!
=====
Silahkan masukkan path file gambar: D:\Skool\un
=====
Dimensi Gambar: 718 x 886
=====
Masukkan target kompresi (0.01 - 1): 0
=====
Pilih metode error:
1: Variance
2: MPD
3: MAD
4: Entropy
5: SSIM
=====
Pilihan: 1
=====
```

```
=====
Masukkan threshold error: 16
Masukkan minimum block size: 4
=====
Masukkan path file gambar output: testeks
Apakah anda ingin membuat GIF? (y/n): n
Tidak membuat gif
=====
PNG berhasil disave
=====
=====
HASIL KOMPRESI
=====
Waktu eksekusi: 627 ms
Ukuran file input: 60 KB
Ukuran file output: 48 KB
Persentase kompresi: 20 %
=====
Kedalaman Quadtree: 9
Jumlah simpul pada Tree: 41573
=====
```

Kompresi quadtree walaupun efektif untuk mengurangi ukuran file pada gambar dokumen/teks, tetapi kurang ideal karena penghematan kompresi yang minimal dan banyaknya informasi yang hilang pada garis halus/teks.

4.2 Analisis dan Pembahasan kompleksitas algoritma

Untuk menganalisis kompleksitas algoritma program ini, pertama kita harus mulai dari awal dimana program mengubah Image menjadi sebuah matriks vector yang mana tentunya untuk mengisi nilai dari matriks, dibutuhkan 2 iterasi terhadap baris dan kolom sehingga kompleksitas untuk loadImage sendiri adalah $O(N^2)$.

Setelah itu, matriks vektor akan dipecah menjadi quadtree yang mana bila kita liat, quadtree di build dengan menggunakan rekursif ke empat anaknya yang menggunakan algoritma divide and conquer, Karena setiap kali kita membagi blok menjadi empat sub-blok, kedalaman pohon kira-kira adalah $\log_4(N)$, sehingga memiliki $O(N) = O(N \log N)$.

Setelah menjadi quadtree, kita akan melakukan reconstruct, untuk reconstruct sendiri sama juga menggunakan rekursif yang mana setiap iterasinya memasuki 2 loop sehingga kompleksitasnya adalah $O(N^2)$

Save menjadi gambar dari matriks vector juga menggunakan big-o ($O(N^2)$) untuk mendapatkan setiap value dari matriks.

Sehingga overall, program kami memiliki kompleksitas $O(N^2)$

Namun, pada percentage compress, kita menggunakan algoritma binary search yang mana fungsi findOptimalParameter memiliki big-O $O(\log N \times T \times N^2) = O(T \times N^2 \times \log N)$. Nilai T di sini merepresentasikan jumlah langkah threshold yang kita uji untuk menemukan kompresi optimal, yang ditentukan oleh besarnya rentang threshold (dari startThreshold hingga endThreshold) dibagi dengan step size (0.1 untuk SSIM). Sedangkan N berasal dari dimensi gambar (width atau height), di mana kompleksitas membangun dan merekonstruksi quadtree adalah $O(N^2)$. Komponen $\log N$ muncul dari loop yang mengurangi ukuran blok dengan membaginya dengan 4 setiap iterasi, yang menghasilkan maksimal $\log_4(N)$ iterasi. Dengan demikian, algoritma pencarian parameter optimal ini memiliki kompleksitas yang lebih tinggi dibandingkan dengan hanya membangun quadtree tunggal, karena harus melakukan multiple testing dengan berbagai kombinasi ukuran blok dan nilai threshold untuk menemukan parameter yang menghasilkan kompresi paling mendekati target yang diinginkan.

Untuk GIF sendiri kompleksitas algoritmanya adalah $T(N) = N^2 * \text{depth} = O(N^2)$.

BAB 5

KESIMPULAN & SARAN

A. Kesimpulan

Algoritma Divide and Conquer merupakan salah satu strategi algoritma yang dapat digunakan untuk memecahkan berbagai masalah. Algoritma Divide and Conquer secara umum bekerja dengan membagi suatu masalah yang besar menjadi bagian yang lebih kecil. Pada tulis ini, algoritma divide and conquer diaplikasikan dalam pemrosesan image compression dengan metode QuadTree. Metode ini memecah-mecah gambar menjadi 4 buah kuadran yang lebih kecil dan memeriksa error pada gambar dengan berbagai cara sesuai dengan threshold yang ditentukan. Bila melewati batas threshold, maka gambar akan dipecah kembali hingga mencapai block size terkecil sesuai input pengguna. Metrik yang digunakan untuk menentukan error pada tulis ini adalah variansi, MAD, MPD, entropy, dan SSIM. Proyek ini berhasil dalam mengaplikasikan algoritma divide and conquer, dan membantu kami untuk dapat lebih memahami penerapannya dalam aplikasi nyata. Kami juga mendapatkan wawasan tambahan tentang pemrosesan citra, dan berbagai metode kompresi yang sering digunakan.

B. Saran

Beberapa saran yang dapat kami berikan untuk pengembangan selanjutnya adalah:

- Bagian bonus program masih dapat dikembangkan agar lebih efisien dan mempersingkat waktu eksekusi program.
- Perhatikan spesifikasi alur program dengan lebih teliti, agar tidak perlu penyesuaian ulang alur program setelah program selesai.
- Pastikan tipe data yang digunakan sudah sesuai dengan kebutuhan sejak tahap awal, untuk menghindari revisi kode karena tipe data tidak sesuai/ tidak cukup untuk kebutuhan program.

LAMPIRAN

Source Code Program: https://github.com/KennethhPoenadi/Tucil2_13523024_13523040

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	✓	