

TUGAS KECIL 2
KOMPRESI GAMBAR DENGAN METODE QUADTREE
IF-2211
STRATEGI ALGORITMA



13523099

Daniel Pedrosa Wu

13523109

Haegen Quinston

Dosen Pengampu:

Dr. Nur Ulfa Maulidevi, S.T, M.Sc.

Dr. Ir. Rinaldi Munir, M.T.

Monterico Adrian, S.T, M.T.

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

JL. GANESA 10, BANDUNG 40132

2025

DAFTAR ISI

DAFTAR ISI.....	2
BAB 1 ALGORITMA.....	3
1.1 Algoritma Divide & Conquer.....	3
1.2 Implementasi Algoritma Divide & Conquer pada Program.....	3
BAB 2 SOURCE CODE PROGRAM.....	8
2.1 Tabel Screenshot Source Code Program.....	8
BAB 3 PERCOBAAN.....	31
3.1 Tabel Tampilan Input/Output pada Percobaan.....	31
BAB 4 ANALISIS HASIL PERCOBAAN.....	39
4.1 Kompleksitas Algoritma.....	39
4.2 Pengaruh Nilai Threshold.....	40
4.3 Pengaruh Minimal Block Size.....	40
4.4 Membandingkan 5 Error Measurement Method.....	41
4.5 Implikasi dan Rekomendasi.....	41
BAB 5 IMPLEMENTASI BONUS.....	41
5.1 Structural Similarity Index (SSIM).....	41
5.2 Target Compression Percentage.....	42
5.3 GIF Visualization.....	45
LAMPIRAN.....	47
Tautan Repository GitHub.....	47
Tabel Checklist.....	47
Referensi Tambahan.....	47

BAB 1 ALGORITMA

1.1 Algoritma Divide & Conquer

Algoritma Divide and Conquer adalah pendekatan rekursif untuk menyelesaikan masalah yang kompleks dengan cara membagi masalah tersebut menjadi 2 atau lebih bagian yang lebih kecil dan lebih mudah diselesaikan. Proses ini terdiri dari tiga langkah utama:

1. Divide (Pembagian):

Masalah besar dipecah menjadi beberapa sub-masalah yang lebih kecil. Sub-masalah ini biasanya memiliki sifat yang sama dengan masalah aslinya sehingga solusinya dapat diintegrasikan nanti.

2. Conquer (Penyelesaian):

Setiap sub-masalah diselesaikan secara terpisah. Jika sub-masalah tersebut sudah cukup sederhana, penyelesaiannya dapat dilakukan secara langsung. Jika tidak, algoritma yang sama diterapkan secara rekursif untuk memecahnya lebih lanjut.

3. Combine (Penggabungan):

Setelah setiap sub-masalah diselesaikan, hasil-hasilnya digabungkan untuk membentuk solusi akhir dari masalah yang lebih besar. Penggabungan inilah yang menyatukan solusi solusi parsial menjadi satu kesatuan yang utuh.

Contoh penerapan algoritma Divide and Conquer adalah Merge Sort, di mana daftar data dipecah menjadi potongan-potongan kecil yang terurut, kemudian potongan-potongan tersebut digabung kembali secara terstruktur untuk menghasilkan daftar yang tersusun secara keseluruhan.

1.2 Implementasi Algoritma Divide & Conquer pada Program

Pseudocode di bawah ini hanya meninjau bagian kode yang mengimplementasikan algoritma Divide & Conquer pada sebuah quadtree. Aplikasi algoritma Divide & Conquer bisa dilihat pada pseudocode berikut:

```
BEGIN QUADTREE
    image      : Image
    frame      : Image
    errorMethod : EMM
    threshold   : double
    minBlockSize : integer
    animation   : Animation (optional)
    root        : Node
```

```

nodeCount      : integer
leafCount      : integer
depth         : integer

FUNCTION construct():

    SET root = new Node(x = 0, y = 0, width = image.width, height = image.height)

        SET root.avgColor = CALL meanColorBlock(root.x, root.y, root.width,
root.height)

        INITIALIZE queue as empty

        PUSH (root, 0) into queue

        SET nodeCount = 1

        SET depth = 0

        WHILE queue is not empty DO:

            (node, currentDepth) = POP from queue

            SET depth = max(depth, currentDepth)

            SET error = errorMethod.computeBlockError(image, node.x, node.y, node.width,
node.height)

                IF      errorMethod.isWithinThreshold(error, threshold)      OR      NOT
node.canSplit(minBlockSize) THEN:

                    INCREMENT leafCount

                    CONTINUE to next iteration

                ENDIF

                SET halfW = node.width / 2

                SET halfH = node.height / 2

                CREATE child0 = new Node(node.x, node.y, halfW, halfH)

// Top-Left

                CREATE child1 = new Node(node.x + halfW, node.y, node.width - halfW, halfH)

// Top-Right

                CREATE child2 = new Node(node.x, node.y + halfH, halfW, node.height - halfH)

// Bottom-Left

                CREATE child3 = new Node(node.x + halfW, node.y + halfH, node.width - halfW,

```

```

node.height - halfH)

// Bottom-Right

FOR EACH child IN [child0, child1, child2, child3] DO:

    SET child.avgColor = CALL meanColorBlock(child.x, child.y, child.width,
child.height)

    PUSH (child, currentDepth + 1) into queue

    INCREMENT nodeCount

END FOR

END WHILE

```

Pseudocode ini dapat disederhanakan lagi menjadi notasi yang hanya mengandung algoritma Divide & Conquer-nya sebagai berikut:

```

FUNCTION Construct():

    INITIALIZE root NODE

    INITIALIZE queue

    ENQUEUE root INTO queue

    WHILE queue is not empty DO:

        SET currentNode AS top of queue

        DEQUEUE FROM queue

        IF currentNode's error < threshold or currentNode cannot split
        further THEN:

            ADD currentNode AS LEAF

        ELSE:

            SPLIT currentNode into 4 child NODE

            ENQUEUE ALL child INTO queue

    END WHILE

```

Algoritma pada fungsi Construct ini menerapkan pendekatan Divide & Conquer secara iteratif untuk membangun suatu quadtree yang merepresentasikan gambar yang ingin dikompresi. Strategi tersebut terbagi menjadi tahap berikut:

1. Inisialisasi root Node dan Queue

Langkah pertama adalah membuat node akar yang akan menjadi node pertama dalam quadtree ini. Node ini akan merepresentasikan gambar dengan ukuran asli dari gambar yang diinput ke dalam objek. Karena implementasi ini menggunakan pendekatan yang iteratif, maka suatu queue diperlukan untuk menyimpan node-node yang akan diproses. Karena root node adalah node pertama, maka ia juga akan menjadi node yang pertama dimasukkan ke dalam queue. Queue adalah struktur yang bekerja pada prinsip FIFO (First-In, First-Out) atau dalam kata lain node yang pertama kali masuk ke dalam queue, akan juga menjadi node yang pertama kali keluar dari queue. Pada implementasi, node ini akan disimpan dengan informasi kedalaman pohonnya yang pada situasi ini masih 0 (akar).

2. Memproses Node dalam Queue

Selama queue tidak kosong, maka node yang berada di dalam queue akan diproses secara iteratif. Pada iterasi pertama, hanyalah root node yang tersimpan dalam queue sehingga itulah menjadi node pertama yang diproses. Di sinilah masuk ke dalam tahapan Divide & Conquer. Berdasarkan metode pengukuran error yang dipilih, maka program akan menentukan error yang terdapat pada node atau dalam konteks ini lebih tepatnya blok tersebut. Karena root node adalah blok pertama, maka ia memiliki ukuran blok yang setara dengan ukuran gambar aslinya. Jika error ini lebih besar daripada ambang batas yang ditentukan, maka fungsi Construct() akan men-divide blok ini menjadi 4 sub blok yang sama besar (jika ukuran blok awal tidaklah persegi, maka ukuran tidak akan sama walaupun perbedaannya tidak terlalu signifikan).

Namun, ada parameter lain yang juga dapat mempengaruhi apakah suatu blok akan di-divide menjadi 4 sub blok, yaitu parameter ukuran blok terkecil. Parameter ini menentukan ukuran blok terkecil yang bisa tersimpan dalam pohon. Sehingga, kondisi lain yang digunakan mengecek apakah blok tersebut boleh di-divide. Pengecekan ini mengecek apakah ukuran sub blok terkecil (jika ukuran blok bukan persegi) ukurannya akan melebihi parameter ukuran blok terkecil. Tentunya blok juga harus bisa dipartisi menjadi 4 bagian. Fungsi ini membagi blok menjadi 4 dengan partisi atas-kiri, atas-kanan, bawah-kiri, dan bawah-kanan. Oleh karena itu, ukuran dari blok juga harus mendukung partisi berikut.

Tahapan Conquer terjadi saat error dari blok sudah lebih kecil dibandingkan ambang batas yang ditentukan ataupun jika sudah tidak bisa dibagi menjadi 4 partisi lagi. Jika ini terjadi, maka node dari blok tersebut akan ditandai sebagai leaf dan tidak akan diproses lebih lagi.

3. Pemrosesan Sub Blok hasil Divide

Jika suatu blok sudah diputuskan untuk di-divide, maka fungsi Construct() akan membentuk 4 node baru. Keempat node yang merepresentasikan sub blok dari blok yang di-divide dimasukkan ke dalam queue dan disertai dengan informasi kedalaman yang merupakan inkrement dari nilai

kedalaman node induknya. Dengan demikian, node-node ini akan diproses oleh queue pada iterasi selanjutnya. Proses ini berlangsung secara iteratif hingga queue kosong. Suatu queue yang kosong artinya seluruh node yang memungkinkan telah diproses dan pohon quadtree selesai dibangun, dengan seluruh sub pohon mencapai leaf node.

Algoritma pada fungsi Construct() menerapkan pendekatan Divide & Conquer menggunakan strategi traversal Breadth-First Search (BFS). Strategi tersebut terbagi menjadi berbagai tahap berikut:

1. Inisialisasi Root Node dan Queue

Langkah pertama adalah membuat node akar (root node) yang mencakup seluruh dimensi gambar. Node ini kemudian dimasukkan ke dalam sebuah queue bersama dengan informasi kedalaman (depth) awal, yaitu 0. Meskipun fungsi Construct() hanya dipanggil sekali, operasi push dan pop dilakukan secara iteratif melalui antrian untuk memproses tiap node secara sistematis.

2. Memproses Node dalam Queue

Selama antrian tidak kosong, setiap iterasi akan mengambil node pertama dari queue. Program menghitung warna rata-rata pada area blok yang diwakili oleh node tersebut, dan hasilnya disimpan dalam atribut avgColor. Selanjutnya, fungsi computeBlockError digunakan untuk mengukur error pada blok tersebut berdasarkan metode error yang dipilih (seperti MAD, MPD, Variance, Entropy, atau SSIM).

3. Keputusan Basis atau Pemecahan Node

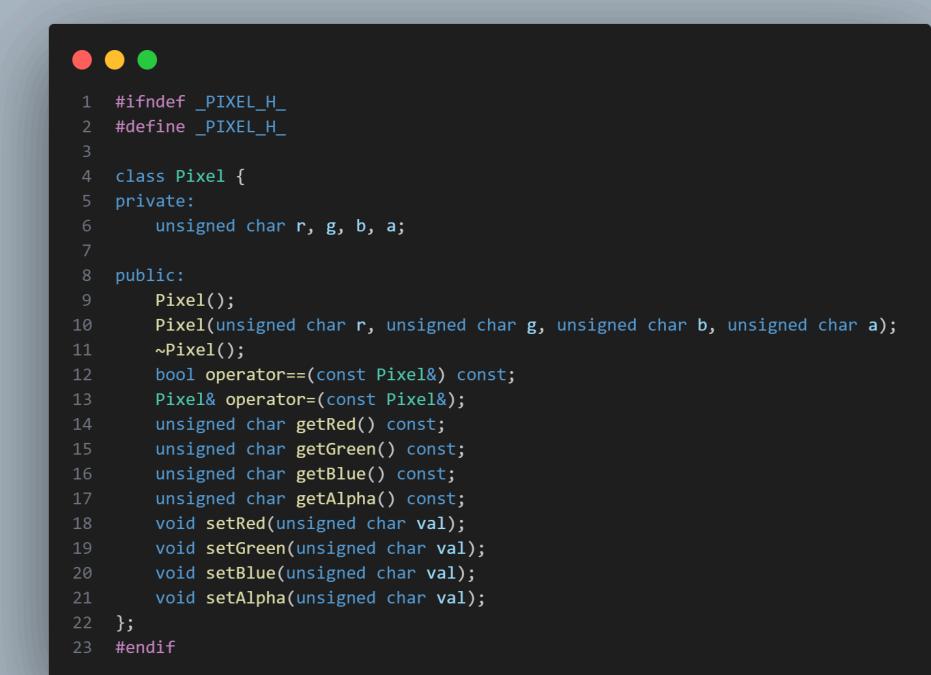
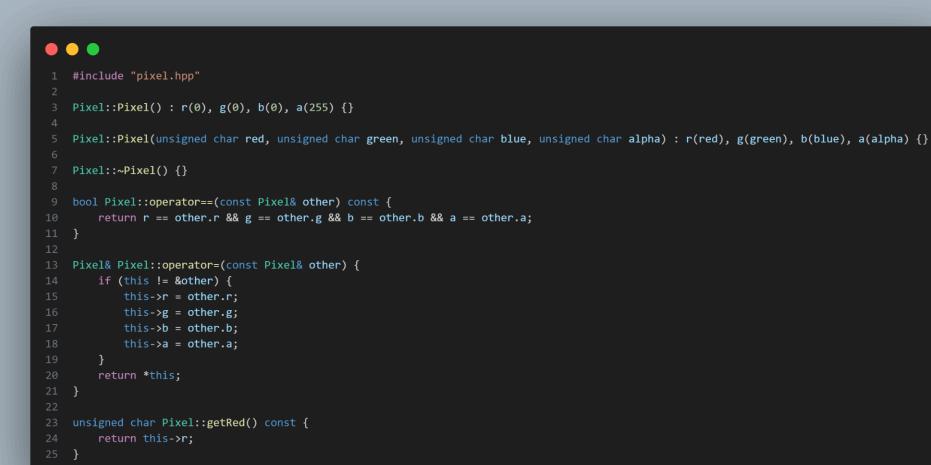
Jika error yang diukur kurang dari atau sama dengan threshold, atau jika node tidak dapat dibagi lebih lanjut karena sudah mencapai ukuran minimum, maka node tersebut dianggap sebagai daun (leaf) dan tidak dipecah lagi. Ini merupakan kondisi basis dari rekursi.

Jika error melebihi threshold, node tersebut dibagi menjadi empat sub-blok (child nodes) yang mewakili kuadran atas kiri, atas kanan, bawah kiri, dan bawah kanan. Masing-masing sub-blok kemudian dimasukkan kembali ke dalam antrian dengan nilai kedalaman yang bertambah (current_depth + 1) untuk diproses selanjutnya.

Proses ini berlanjut sampai semua node dalam antrian telah diproses. Hasil akhirnya adalah sebuah struktur pohon (quad-tree) di mana setiap daun mewakili blok gambar dengan error yang dapat diterima.

BAB 2 SOURCE CODE PROGRAM

2.1 Tabel Screenshot Source Code Program

Nama File	Screenshot Source Code
pixel.hpp, pixel.cpp	 <pre>1 #ifndef _PIXEL_H_ 2 #define _PIXEL_H_ 3 4 class Pixel { 5 private: 6 unsigned char r, g, b, a; 7 8 public: 9 Pixel(); 10 Pixel(unsigned char r, unsigned char g, unsigned char b, unsigned char a); 11 ~Pixel(); 12 bool operator==(const Pixel&) const; 13 Pixel& operator=(const Pixel&); 14 unsigned char getRed() const; 15 unsigned char getGreen() const; 16 unsigned char getBlue() const; 17 unsigned char getAlpha() const; 18 void setRed(unsigned char val); 19 void setGreen(unsigned char val); 20 void setBlue(unsigned char val); 21 void setAlpha(unsigned char val); 22 }; 23 #endif</pre>  <pre>1 #include "pixel.hpp" 2 3 Pixel::Pixel() : r(0), g(0), b(0), a(255) {} 4 5 Pixel::Pixel(unsigned char red, unsigned char green, unsigned char blue, unsigned char alpha) : r(red), g(green), b(blue), a(alpha) {} 6 7 Pixel::~Pixel() {} 8 9 bool Pixel::operator==(const Pixel& other) const { 10 return r == other.r && g == other.g && b == other.b && a == other.a; 11 } 12 13 Pixel& Pixel::operator=(const Pixel& other) { 14 if (this != &other) { 15 this->r = other.r; 16 this->g = other.g; 17 this->b = other.b; 18 this->a = other.a; 19 } 20 return *this; 21 } 22 23 unsigned char Pixel::getRed() const { 24 return this->r; 25 }</pre>



```
1 unsigned char Pixel::getGreen() const {
2     return this->g;
3 }
4
5 unsigned char Pixel::getBlue() const {
6     return this->b;
7 }
8
9 unsigned char Pixel::getAlpha() const {
10    return this->a;
11 }
12
13 void Pixel::setRed(unsigned char val) {
14     this->r = val;
15 }
16
17 void Pixel::setGreen(unsigned char val) {
18     this->g = val;
19 }
20
21 void Pixel::setBlue(unsigned char val) {
22     this->b = val;
23 }
24
25 void Pixel::setAlpha(unsigned char val) {
26     this->a = val;
27 }
```

image.hpp, image.cpp

```
 1  #ifndef _IMAGE_H_
 2  #define _IMAGE_H_
 3
 4  #include <filesystem>
 5  #include <cstring>
 6  #include <iostream>
 7  #include "../pixel/pixel.hpp"
 8
 9  namespace fs = std::filesystem;
10
11 class Image {
12 private:
13     fs::path path;
14     std::string extension;
15     int width;
16     int height;
17     int channels;
18     unsigned char* data;
19     std::uintmax_t fileSize;
20
21 public:
22     Image(const fs::path& path);
23     Image(const Image&);
24     ~Image();
25     bool load(const fs::path& filePath);
26     bool save(const fs::path& outPath) const;
27     fs::path getPath() const;
28     std::string getExtension() const;
29     int getWidth() const;
30     int getHeight() const;
31     int getChannels() const;
32     unsigned char* getData() const;
33     std::uintmax_t getFileSize() const;
34     Pixel getPixelAt(int x, int y) const;
35     void setPixelAt(int x, int y, Pixel p);
36     static void countBytes(void *context, void *data, int size);
37     void setFileSize(std::uintmax_t size);
38     uintmax_t estimateNewFileSize() const;
39 };
40
41 #endif
```

```
 1 #define STB_IMAGE_IMPLEMENTATION
 2 #define STB_IMAGE_WRITE_IMPLEMENTATION
 3 #include "../include/stb_image.h"
 4 #include "../include/stb_image_write.h"
 5 #include "image.hpp"
 6
 7 Image::Image(const fs::path& filePath) : data(nullptr), width(0), height(0), channels(0), fileSize(0) {
 8     load(filePath);
 9 }
10
11 Image::Image(const Image& other) : width(other.width), height(other.height), channels(other.channels), path(other.path), extension(other.extension), fileSize(other.fileSize) {
12     int dataSize = width * height * channels;
13     data = static_cast<unsigned char*>(malloc(dataSize));
14     if (data && other.data) {
15         std::memcpy(data, other.data, dataSize);
16     }
17 }
18
19 Image::~Image() {
20     if (data) {
21         stbi_image_free(data);
22         data = nullptr;
23     }
24 }
25
26 bool Image::load(const fs::path& filePath) {
27     if (data) {
28         stbi_image_free(data);
29         data = nullptr;
30     }
31
32     path = filePath;
33     extension = path.extension().string();
34
35     data = stbi_load(path.string().c_str(), &width, &height, &channels, 0);
36     if (!data) {
37         std::cerr << "Failed to load image: " << path << std::endl;
38         return false;
39     }
40
41     try {
42         fileSize = fs::file_size(path);
43     } catch (const fs::filesystem_error e) {
44         std::cerr << "Error getting file size: " << e.what() << std::endl;
45         fileSize = 0;
46     }
47
48     return true;
49 }
50
51 bool Image::save(const fs::path& outPath) const {
52     if (!data) {
53         std::cerr << "Error: No image data to save.\n";
54         return false;
55     }
56
57     std::string ext = outPath.extension().string();
58     int stride = width * channels;
59     bool success = false;
60
61     if (ext == ".png") {
62         stbi_write_png_compression_level = 9;
63         stbi_write_force_png_filter = 4;
64         success = stbi_write_png(outPath.string().c_str(), width, height, channels, data, stride);
65     } else if (ext == ".jpg" || ext == ".jpeg") {
66         success = stbi_write_jpg(outPath.string().c_str(), width, height, channels, data, 75);
67     } else {
68         std::cerr << "Unsupported file format for saving: " << ext << std::endl;
69         return false;
70     }
71
72     if (!success) {
73         std::cerr << "Failed to save image to: " << outPath << std::endl;
74     }
75
76     return success;
77 }
78
79 fs::path Image::getPath() const {
80     return path;
81 }
82
83 std::string Image::getExtension() const {
84     return extension;
85 }
```

```
● ○ ●
1 std::string Image::getExtension() const {
2     return extension;
3 }
4
5 int Image::getWidth() const {
6     return width;
7 }
8
9 int Image::getHeight() const {
10    return height;
11 }
12
13 int Image::getChannels() const {
14    return channels;
15 }
16
17 unsigned char* Image::getData() const {
18    return data;
19 }
20
21 std::uintmax_t Image::getFileSize() const {
22    return fileSize;
23 }
24
25 Pixel Image::getPixelAt(int x, int y) const {
26    if (!data || x < 0 || x >= width || y < 0 || y >= height) {
27        return Pixel();
28    }
29
30    int index = (y * width + x) * channels;
31    unsigned char r = data[index];
32    unsigned char g = (channels >= 2) ? data[index + 1] : r;
33    unsigned char b = (channels >= 3) ? data[index + 2] : r;
34    unsigned char a = (channels == 4) ? data[index + 3] : 255;
35
36    return Pixel(r, g, b, a);
37 }
38
39 void Image::setPixelAt(int x, int y, Pixel p) {
40    if (!data || x < 0 || x >= width || y < 0 || y >= height) {
41        return;
42    }
43
44    int index = (y * width + x) * channels;
45    data[index] = p.getRed();
46    if (channels >= 2) {
47        data[index + 1] = p.getGreen();
48    }
49    if (channels >= 3) {
50        data[index + 2] = p.getBlue();
51    }
52    if (channels == 4) {
53        data[index + 3] = p.getAlpha();
54    }
55 }
56
57 void Image::countBytes(void* context, void* data, int size) {
58    size_t* byteCount = static_cast<size_t*>(context);
59    *byteCount += static_cast<size_t>(size);
60 }
61
62 void Image::setFileSize(std::uintmax_t size) {
63    fileSize = size;
64 }
65
66 std::uintmax_t Image::estimateNewFileSize() const {
67    std::uintmax_t byteCount = 0;
68
69    if (extension == ".png") {
70        int stride = width * channels;
71        stbi_write_png_compression_level = 9;
72        stbi_write_force_png_filter = 4;
73        stbi_write_png_to_func(countBytes, &byteCount, width, height, channels, data, stride);
74    } else if (extension == ".jpeg" || extension == ".jpg") {
75        stbi_write_jpg_to_func(countBytes, &byteCount, width, height, channels, data, 75);
76    } else {
77        std::cerr << "Unsupported file format for saving: " << extension << std::endl;
78        return 0;
79    }
80
81    return byteCount;
82 }
```

quadtree.hpp, quadtree.cpp

```
1 #ifndef _QUADTREE_H_
2 #define _QUADTREE_H_
3
4 #include <queue>
5 #include "../error_measurement/emm.hpp"
6 #include "../image/image.hpp"
7 #include "../image/animation.hpp"
8 #include "node.hpp"
9 #include <algorithm>
10 #include <functional>
11
12 class QuadTree {
13 private:
14     const Image& image;
15     Image frame;
16     const EMM& errorMethod;
17     double threshold;
18     int minBlockSize;
19     Node* root;
20     int depth = 0;
21     int leafCount = 0;
22     int nodeCount = 0;
23
24     Animation* animation = nullptr;
25
26     Pixel meanColorBlock(int x, int y, int width, int height);
27
28 public:
29     QuadTree(const Image& image, const EMM& errorMethod, double threshold, int minBlockSize);
30     ~QuadTree();
31     void setThreshold(double threshold);
32     double getThreshold() const;
33     void clearTree();
34     void construct();
35     void generateAnimation(const std::string& gifPath);
36     void render(Image& output);
37     int getTotalNodes() const;
38     int getTotalLeaves() const;
39     int getDepth() const;
40 };
41
42 #endif
```

```
1 #include "quadtree.hpp"
2
3 QuadTree::QuadTree(const Image& image, const EMM& errorMethod, double threshold, int minBlockSize) : image(image),
4 frame(image), errorMethod(errorMethod), threshold(threshold), minBlockSize(minBlockSize), root(nullptr) { }
5
6 QuadTree::~QuadTree() {
7     clearTree();
8     if (animation) {
9         delete animation;
10        animation = nullptr;
11    }
12 }
13
14 void QuadTree::setThreshold(double threshold) {
15     this->threshold = threshold;
16 }
17
18 double QuadTree::getThreshold() const {
19     return threshold;
20 }
```

```
 1 void QuadTree::clearTree() {
 2     if (!root) return;
 3
 4     std::queue<Node*> q;
 5     q.push(root);
 6
 7     while (!q.empty()) {
 8         Node* current = q.front();
 9         q.pop();
10
11         for (int i = 0; i < 4; ++i) {
12             if (current->childrenNode[i]) {
13                 q.push(current->childrenNode[i]);
14                 current->childrenNode[i] = nullptr;
15             }
16         }
17
18         delete current;
19     }
20
21     root = nullptr;
22     depth = 0;
23     nodeCount = 0;
24     leafCount = 0;
25 }
26
27 void QuadTree::construct() {
28     if (root) {
29         clearTree();
30     }
31     root = new Node(0, 0, image.getWidth(), image.getHeight());
32     root->avgColor = meanColorBlock(root->x, root->y, root->width, root->height);
33
34     std::queue<std::pair<Node*, int>> q;
35     q.push({root, 0});
36
37     nodeCount = 1;
38     depth = 0;
39
40     while (!q.empty()) {
41         auto [node, currentDepth] = q.front(); q.pop();
42         depth = std::max(depth, currentDepth);
43
44         double error = errorMethod.computeBlockError(image, node->x, node->y, node->width, node->height);
45
46         if (errorMethod.isWithinThreshold(error, threshold) || !node->canSplit(minBlockSize)) {
47             leafCount++;
48             continue;
49         }
50
51         int halfW = node->width / 2;
52         int halfH = node->height / 2;
53
54         node->childrenNode[0] = new Node(node->x, node->y, halfW, halfH); // TL
55         node->childrenNode[1] = new Node(node->x + halfW, node->y, node->width - halfW, halfH); // TR
56         node->childrenNode[2] = new Node(node->x, node->y + halfH, halfW, node->height - halfH); // BL
57         node->childrenNode[3] = new Node(node->x + halfW, node->y + halfH, node->width - halfW, node->height - halfH); // BR
58
59         for (auto* child : node->childrenNode) {
60             child->avgColor = meanColorBlock(child->x, child->y, child->width, child->height);
61             q.push({child, currentDepth + 1});
62             nodeCount++;
63         }
64     }
65 }
66
67 void QuadTree
```

```
● ● ●

1 void QuadTree::generateAnimation(const std::string& gifPath) {
2     if (gifPath.empty() || !root) return;
3
4     if (!animation) {
5         animation = new Animation(gifPath, image.getWidth(), image.getHeight());
6     }
7
8     std::queue<const Node*> q;
9     q.push(root);
10
11    while (!q.empty()) {
12        int size = q.size();
13
14        for (int i = 0; i < size; i++) {
15            const Node* node = q.front(); q.pop();
16
17            for (int j = node->x; j < node->x + node->width && j < frame.getWidth(); j++) {
18                for (int k = node->y; k < node->y + node->height && k < frame.getHeight(); k++) {
19                    frame.setPixelAt(j, k, node->avgColor);
20                }
21            }
22            for (auto* child : node->childrenNode) {
23                if (child) q.push(child);
24            }
25        }
26
27        animation->addFrame(frame);
28    }
29 }
30
31 void QuadTree::render(Image& output) {
32     if (!root) return;
33
34     std::queue<const Node*> q;
35     q.push(root);
36
37     while (!q.empty()) {
38         const Node* node = q.front();
39         q.pop();
40
41         bool hasChildren = node->childrenNode[0] != nullptr;
42
43         if (!hasChildren) {
44             for (int j = node->x; j < node->x + node->width && j < output.getWidth(); j++) {
45                 for (int k = node->y; k < node->y + node->height && k < output.getHeight(); k++) {
46                     output.setPixelAt(j, k, node->avgColor);
47                 }
48             }
49         } else {
50             for (auto* child : node->childrenNode) {
51                 if (child != nullptr) {
52                     q.push(child);
53                 }
54             }
55         }
56     }
57 }
58
59 Pixel QuadTree::meanColorBlock(int x, int y, int width, int height){
60     long long r = 0, g = 0, b = 0;
61     int count = width * height;
62
63     for (int i = x; i < x + width; i++) {
64         for (int j = y; j < y + height; j++) {
65             Pixel p = image.getPixelAt(i, j);
66             r += p.getRed();
67             g += p.getGreen();
68             b += p.getBlue();
69         }
70     }
71     return Pixel(r / count, g / count, b / count, 255);
72 }
73
74 int QuadTree::getTotalNodes() const { return nodeCount; }
75 int QuadTree::getTotalLeaves() const { return leafCount; }
76 int QuadTree::getDepth() const { return depth; }
```

animation.hpp, animation.cpp

```
● ● ●  
1 #ifndef _ANIMATION_H_  
2 #define _ANIMATION_H_  
3  
4 #include <string>  
5 #include <memory>  
6 #include "image.hpp"  
7  
8 class Animation {  
9 private:  
10     struct Impl;  
11     std::unique_ptr<Impl> impl;  
12  
13 public:  
14     Animation(const std::string& path, int width, int height, int delay = 70);  
15     ~Animation();  
16  
17     void addFrame(const Image& image);  
18     void close();  
19 };  
20  
21 #endif  
22
```

```
● ● ●  
1 #include "animation.hpp"  
2 #include "../include/gif.h"  
3 #include <iostream>  
4  
5 struct Animation::Impl {  
6     GifWriter writer;  
7     bool init;  
8     int width, height, delay;  
9  
10    Impl(const std::string& path, int w, int h, int d)  
11        : init(false), width(w), height(h), delay(d)  
12    {  
13        init = GifBegin(&writer, path.c_str(), width, height, delay);  
14    }  
15  
16    ~Impl() {  
17        if (init) GifEnd(&writer);  
18    }
```

```
1     void addFrame(const Image& image) {
2         if (!init) return;
3
4         int imgChannels = image.getChannels();
5         unsigned char* imgData = image.getData();
6
7         if (imgChannels == 4) {
8             GifWriteFrame(&writer, imgData, width, height, delay);
9             return;
10        }
11
12        unsigned char* rgba = new unsigned char[width * height * 4];
13        for (int y = 0; y < height; y++) {
14            for (int x = 0; x < width; x++) {
15                int idx_src = (x + y * width) * imgChannels;
16                int idx_dest = (x + y * width) * 4;
17
18                unsigned char r = 0, g = 0, b = 0, a = 255;
19                if (imgChannels == 1) {
20                    r = g = b = imgData[idx_src];
21                } else if (imgChannels == 3) {
22                    r = imgData[idx_src];
23                    g = imgData[idx_src + 1];
24                    b = imgData[idx_src + 2];
25                } else {
26                    std::cerr << "Unsupported image channel count: " << imgChannels << std::endl;
27                    delete[] rgba;
28                    return;
29                }
30
31                rgba[idx_dest + 0] = r;
32                rgba[idx_dest + 1] = g;
33                rgba[idx_dest + 2] = b;
34                rgba[idx_dest + 3] = a;
35            }
36        }
37
38        GifWriteFrame(&writer, rgba, width, height, delay);
39        delete[] rgba;
40    }
41
42    void close() {
43        if (init) {
44            GifEnd(&writer);
45            init = false;
46        }
47    }
48 };
49
50 Animation::Animation(const std::string& path, int width, int height, int delay)
51     : impl(std::make_unique<Impl>(path, width, height, delay))
52 {}
53
54 Animation::~Animation() = default;
55
56 void Animation::addFrame(const Image& image) {
57     impl->addFrame(image);
58 }
59
60 void Animation::close() {
61     impl->close();
62 }
```

emm.hpp, emm.cpp

```
1 #ifndef _EMM_H_
2 #define _EMM_H_
3
4 #include <string>
5 #include "../image/image.hpp"
6
7 class EMM {
8 public:
9     enum class Channel {
10         RED,
11         GREEN,
12         BLUE
13     };
14
15     virtual double computeBlockError(const Image& image, int x, int y, int width, int height) const = 0;
16     virtual bool isWithinThreshold(double error, double threshold) const;
17     virtual long long sumColorBlock(const Image& image, int x, int y, int width, int height, Channel c) const;
18     virtual long long sumColorSquaredBlock(const Image& image, int x, int y, int width, int height, Channel c) const;
19     virtual double meanColorBlock(const Image& image, int x, int y, int width, int height, Channel c) const;
20     virtual std::string identify() const = 0;
21     virtual double getLowerBound() const = 0;
22     virtual double getUpperBound() const = 0;
23     virtual ~EMM() = default;
24 };
25
26
27 #endiff
```

```
1 #include "emm.hpp"
2
3 bool EMM::isWithinThreshold(double error, double threshold) const {
4     return error <= threshold;
5 }
6
7 long long EMM::sumColorBlock(const Image& image, int x, int y, int width, int height, Channel c) const {
8     long long sum = 0;
9
10    for (int i = x; i < x + width; i++) {
11        for (int j = y; j < y + height; j++) {
12            Pixel p = image.getPixelAt(i, j);
13            switch(c) {
14                case Channel::RED:
15                    sum += p.getRed();
16                    break;
17                case Channel::GREEN:
18                    sum += p.getGreen();
19                    break;
20                case Channel::BLUE:
21                    sum += p.getBlue();
22                    break;
23            }
24        }
25    }
26    return sum;
27 }
28
29 long long EMM::sumColorSquaredBlock(const Image& image, int x, int y, int width, int height, Channel c) const {
30     long long sum = 0;
31
32    for (int i = x; i < x + width; i++) {
33        for (int j = y; j < y + height; j++) {
34            Pixel p = image.getPixelAt(i, j);
35            switch(c) {
36                case Channel::RED:
37                    sum += p.getRed() * p.getRed();
38                    break;
39                case Channel::GREEN:
40                    sum += p.getGreen() * p.getGreen();
41                    break;
42                case Channel::BLUE:
43                    sum += p.getBlue() * p.getBlue();
44                    break;
45            }
46        }
47    }
48    return sum;
49 }
```

```
1 double EMM::meanColorBlock(const Image& image, int x, int y, int width, int height, Channel c) const {
2     long long sum = 0;
3     const int count = width * height;
4
5     for (int j = y; j < y + height; j++) {
6         for (int i = x; i < x + width; i++) {
7             Pixel p = image.getPixelAt(i, j);
8             switch(c) {
9                 case Channel::RED:
10                     sum += p.getRed();
11                     break;
12                 case Channel::GREEN:
13                     sum += p.getGreen();
14                     break;
15                 case Channel::BLUE:
16                     sum += p.getBlue();
17                     break;
18             }
19         }
20     }
21     return static_cast<double>(sum) / count;
22 }
```

mad.hpp, mad.cpp

```
1 #ifndef _MAD_H_
2 #define _MAD_H_
3
4 #include "emm.hpp"
5 #include <cmath>
6
7 class MAD : public EMM {
8
9 public:
10     double computeBlockError(const Image& image, int x, int y, int width, int height) const override;
11     bool ThresholdWithinBound(double threshold) override;
12     std::string identify() const override;
13     double getLowerBound() const override;
14     double getUpperBound() const override;
15 };
16
17 #endif
```

```
1 #include "mad.hpp"
2
3 double MAD::computeBlockError(const Image& image, int x, int y, int width, int height) const {
4     int count = width * height;
5
6     double meanRed = meanColorBlock(image, x, y, width, height, Channel::RED);
7     double meanGreen = meanColorBlock(image, x, y, width, height, Channel::GREEN);
8     double meanBlue = meanColorBlock(image, x, y, width, height, Channel::BLUE);
9
10    double absDifSumRed = 0;
11    double absDifSumGreen = 0;
12    double absDifSumBlue = 0;
```

```
● ● ●

1   for (int i = x; i < x + width; i++) {
2       for (int j = y; j < y + height; j++) {
3           Pixel p = image.getPixelAt(i, j);
4           absDifSumRed += std::abs(p.getRed() - meanRed);
5           absDifSumGreen += std::abs(p.getGreen() - meanGreen);
6           absDifSumBlue += std::abs(p.getBlue() - meanBlue);
7       }
8   }
9
10  double madRed = absDifSumRed / count;
11  double madGreen = absDifSumGreen / count;
12  double madBlue = absDifSumBlue / count;
13
14  return (madRed + madGreen + madBlue) / 3;
15 }
16
17 bool MAD::ThresholdWithinBound(double threshold) {
18     return threshold >= 0 && threshold <= 127.5;
19 }
20
21 std::string MAD::identify() const {
22     return "MAD";
23 }
24
25 double MAD::getLowerBound() const {
26     return 0;
27 }
28
29 double MAD::getUpperBound() const {
30     return 127.5;
31 }
```

mpd.hpp, mpd.cpp

```
● ● ●

1 #ifndef _MPD_H_
2 #define _MPD_H_
3
4 #include "emm.hpp"
5 #include <cmath>
6
7 class MPD : public EMM {
8 public:
9     double computeBlockError(const Image& image, int x, int y, int width, int height) const override;
10    bool ThresholdWithinBound(double threshold) override;
11    std::string identify() const override;
12    double getLowerBound() const override;
13    double getUpperBound() const override;
14 };
15
16 #endif
```

```
1 #include "mpd.hpp"
2
3 double MPD::computeBlockError(const Image& image, int x, int y, int width, int height) const {
4     Pixel p = image.getPixelAt(x, y);
5
6     int minRed = p.getRed();
7     int minGreen = p.getGreen();
8     int minBlue = p.getBlue();
9
10    int maxRed = p.getRed();
11    int maxGreen = p.getGreen();
12    int maxBlue = p.getBlue();
13
14    for (int i = x; i < x + width; i++) {
15        for (int j = y; j < y + height; j++) {
16            p = image.getPixelAt(i, j);
17            minRed = std::min(minRed, static_cast<int>(p.getRed()));
18            minGreen = std::min(minGreen, static_cast<int>(p.getGreen()));
19            minBlue = std::min(minBlue, static_cast<int>(p.getBlue()));
20            maxRed = std::max(maxRed, static_cast<int>(p.getRed()));
21            maxGreen = std::max(maxGreen, static_cast<int>(p.getGreen()));
22            maxBlue = std::max(maxBlue, static_cast<int>(p.getBlue()));
23        }
24    }
25
26    int redDif = maxRed - minRed;
27    int greenDif = maxGreen - minGreen;
28    int blueDif = maxBlue - minBlue;
29
30    return static_cast<double>(redDif + greenDif + blueDif) / 3;
31 }
32
33 bool MPD::ThresholdWithinBound(double threshold) {
34     return threshold >= 0 && threshold <= 255;
35 }
36
37 std::string MPD::identify() const {
38     return "MPD";
39 }
40
41 double MPD::getLowerBound() const {
42     return 0;
43 }
44
45 double MPD::getUpperBound() const {
46     return 255;
47 }
```

entropy.hpp, entropy.cpp

```
1 #ifndef _ENTROPY_H_
2 #define _ENTROPY_H_
3
4 #include "emm.hpp"
5 #include <cmath>
6 #include <array>
7
8 class Entropy : public EMM {
9 public:
10     double computeBlockError(const Image& image, int x, int y, int width, int height) const override;
11     bool ThresholdWithinBound(double threshold) override;
12     std::string identify() const override;
13     double getLowerBound() const override;
14     double getUpperBound() const override;
15 };
16
17 #endif
```

```
1 #include "entropy.hpp"
2
3 double Entropy::computeBlockError(const Image& image, int x, int y, int width, int height) const {
4     int count = width * height;
5     if (count == 0) {
6         return 0;
7     }
8     std::array<int, 256> histRed = {0};
9     std::array<int, 256> histGreen = {0};
10    std::array<int, 256> histBlue = {0};
11
12    for (int i = x; i < x + width; i++) {
13        for (int j = y; j < y + height; j++) {
14            Pixel p = image.getPixelAt(i, j);
15            histRed[p.getRed()]++;
16            histGreen[p.getGreen()]++;
17            histBlue[p.getBlue()]++;
18        }
19    }
20
21    double entropyRed = 0;
22    double entropyGreen = 0;
23    double entropyBlue = 0;
24
25    for (int i = 0; i < 256; i++) {
26        if (histRed[i] > 0) {
27            double probability = static_cast<double>(histRed[i]) / count;
28            entropyRed -= probability * std::log2(probability);
29        }
30        if (histGreen[i] > 0) {
31            double probability = static_cast<double>(histGreen[i]) / count;
32            entropyGreen -= probability * std::log2(probability);
33        }
34        if (histBlue[i] > 0) {
35            double probability = static_cast<double>(histBlue[i]) / count;
36            entropyBlue -= probability * std::log2(probability);
37        }
38    }
39
40    return (entropyRed + entropyGreen + entropyBlue) / 3;
41 }
```



```
1  bool Entropy::ThresholdWithinBound(double threshold) {
2      return threshold >= 0 && threshold <= 8;
3  }
4
5  std::string Entropy::identify() const {
6      return "Entropy";
7  }
8
9  double Entropy::getLowerBound() const {
10     return 0;
11 }
12 double Entropy::getUpperBound() const {
13     return 8;
14 }
```

variance.hpp, variance.cpp



```
1  #ifndef _VARIANCE_H_
2  #define _VARIANCE_H_
3
4  #include "emm.hpp"
5
6  class Variance : public EMM {
7
8  public:
9      double computeBlockError(const Image& image, int x, int y, int width, int height) const override;
10     bool ThresholdWithinBound(double threshold) override;
11     std::string identify() const override;
12     double getLowerBound() const override;
13     double getUpperBound() const override;
14 };
15
16 #endif
```

```
● ● ●
1 #include "variance.hpp"
2
3 double Variance::computeBlockError(const Image& image, int x, int y, int width, int height) const {
4     int count = width * height;
5
6     long long sumRed = sumColorBlock(image, x, y, width, height, Channel::RED);
7     long long sumGreen = sumColorBlock(image, x, y, width, height, Channel::GREEN);
8     long long sumBlue = sumColorBlock(image, x, y, width, height, Channel::BLUE);
9
10    long long sumRedSquared = sumColorSquaredBlock(image, x, y, width, height, Channel::RED);
11    long long sumGreenSquared = sumColorSquaredBlock(image, x, y, width, height, Channel::GREEN);
12    long long sumBlueSquared = sumColorSquaredBlock(image, x, y, width, height, Channel::BLUE);
13
14    double meanRed = static_cast<double> (sumRed) / count;
15    double meanGreen = static_cast<double> (sumGreen) / count;
16    double meanBlue = static_cast<double> (sumBlue) / count;
17
18    double varianceRed = static_cast<double> (sumRedSquared) / count - meanRed * meanRed;
19    double varianceGreen = static_cast<double> (sumGreenSquared) / count - meanGreen * meanGreen;
20    double varianceBlue = static_cast<double> (sumBlueSquared) / count - meanBlue * meanBlue;
21
22    return (varianceRed + varianceGreen + varianceBlue) / 3;
23 }
24
25 bool Variance::ThresholdWithinBound(double threshold) {
26     return threshold >= 0 && threshold <= 16256.25;
27 }
28
29 std::string Variance::identify() const {
30     return "Variance";
31 }
32
33 double Variance::getLowerBound() const {
34     return 0;
35 }
36
37 double Variance::getUpperBound() const {
38     return 16256.25;
39 }
```

ssim.hpp, ssim.cpp

```
● ● ●
1 #ifndef SSIM_HPP
2 #define SSIM_HPP
3
4 #include "emm.hpp"
5 #include "../image/image.hpp"
6 #include <cmath>
7
8 class SSIM : public EMM {
9 private:
10     static double computeChannelSSIM(double sumRef, double sumTest, double sumRef2, double sumTest2,
11                                     double sumRefTest, int count);
12 public:
13     double computeBlockError(const Image& refImg, int x, int y, int width, int height) const override;
14     bool ThresholdWithinBound(double threshold) override;
15     bool isWithinThreshold(double error, double threshold) const override;
16     std::string identify() const override;
17     double getLowerBound() const override;
18     double getUpperBound() const override;
19 };
20
21 #endif
```

```

1 #include "ssim.hpp"
2
3 static constexpr double k1 = 0.01;
4 static constexpr double k2 = 0.03;
5 static constexpr double L = 255.0;
6
7 double SSIM::computeChannelSSIM(double sumRef, double sumTest,
8                                 double sumRef2, double sumTest2,
9                                 double sumRefTest, int count)
10 {
11     double meanRef = sumRef / count;
12     double meanTest = sumTest / count;
13     double varRef = (sumRef2 / count) - (meanRef * meanRef);
14     double varTest = (sumTest2 / count) - (meanTest * meanTest);
15     double cov = (sumRefTest / count) - (meanRef * meanTest);
16
17     double c1 = (k1 * L) * (k1 * L);
18     double c2 = (k2 * L) * (k2 * L);
19
20     double numerator = (2.0 * meanRef * meanTest + c1) * (2.0 * cov + c2);
21     double denominator = (meanRef * meanRef + meanTest * meanTest + c1) * (varRef + varTest + c2);
22
23     if (denominator < 1e-12)
24         return 1.0;
25     return numerator / denominator;
26 }
27
28 double SSIM::computeBlockError(const Image& refImg, int x, int y, int width, int height) const
29 {
30     const int count = width * height;
31     if (count == 0)
32         return 1.0;
33
34     double meanRed = meanColorBlock(refImg, x, y, width, height, Channel::RED);
35     double meanGreen = meanColorBlock(refImg, x, y, width, height, Channel::GREEN);
36     double meanBlue = meanColorBlock(refImg, x, y, width, height, Channel::BLUE);
37
38     Pixel meanPixel(static_cast<unsigned char>(meanRed),
39                      static_cast<unsigned char>(meanGreen),
40                      static_cast<unsigned char>(meanBlue),
41                      255);
42
43     double sumRRef = 0.0, sumRTest = 0.0, sumRRef2 = 0.0, sumRTest2 = 0.0, sumRRefTest = 0.0;
44     double sumGRef = 0.0, sumGTest = 0.0, sumGRef2 = 0.0, sumGTest2 = 0.0, sumGRefTest = 0.0;
45     double sumBRef = 0.0, sumBTest = 0.0, sumBRef2 = 0.0, sumBTest2 = 0.0, sumBRefTest = 0.0;
46
47     for (int j = y; j < y + height; j++) {
48         for (int i = x; i < x + width; i++) {
49             Pixel pref = refImg.getPixelAt(i, j);
50             double rRef = static_cast<double>(pref.getRed());
51             double rTest = static_cast<double>(meanPixel.getRed());
52             double gRef = static_cast<double>(pref.getGreen());
53             double gTest = static_cast<double>(meanPixel.getGreen());
54             double bRef = static_cast<double>(pref.getBlue());
55             double bTest = static_cast<double>(meanPixel.getBlue());
56
57             sumRRef += rRef;
58             sumRTest += rTest;
59             sumRRef2 += rRef * rRef;
60             sumRTest2 += rTest * rTest;
61             sumRRefTest += rRef * rTest;
62
63             sumGRef += gRef;
64             sumGTest += gTest;
65             sumGRef2 += gRef * gRef;
66             sumGTest2 += gTest * gTest;
67             sumGRefTest += gRef * gTest;
68
69             sumBRef += bRef;
70             sumBTest += bTest;
71             sumBRef2 += bRef * bRef;
72             sumBTest2 += bTest * bTest;
73             sumBRefTest += bRef * bTest;
74         }
75     }
76
77     double ssimR = computeChannelSSIM(sumRRef, sumRTest, sumRRef2, sumRTest2, sumRRefTest, count);
78     double ssimG = computeChannelSSIM(sumGRef, sumGTest, sumGRef2, sumGTest2, sumGRefTest, count);
79     double ssimB = computeChannelSSIM(sumBRef, sumBTest, sumBRef2, sumBTest2, sumBRefTest, count);
80
81     return ((ssimR + ssimG + ssimB) / 3.0);
82 }

```

```
● ● ●

1  bool SSIM::ThresholdWithinBound(double threshold)
2  {
3      return (threshold >= 0.0 && threshold <= 1.0);
4  }
5
6  bool SSIM::isWithinThreshold(double error, double threshold) const {
7      return error >= threshold;
8  }
9
10 std::string SSIM::identify() const {
11     return "SSIM";
12 }
13
14 double SSIM::getLowerBound() const {
15     return 0;
16 }
17
18 double SSIM::getUpperBound() const {
19     return 1;
20 }
```

compressor.hpp, compressor.cpp

```
● ● ●

1 #ifndef _COMPRESSOR_H_
2 #define _COMPRESSOR_H_
3
4 #include "../error_measurement/errm.hpp"
5 #include "../image/animation.hpp"
6 #include "../image/image.hpp"
7 #include "../quadtree/quadtreenode.hpp"
8 #include "../quadtree/quadtreet.hpp"
9 #include <iostream>
10 #include <cmath>
11
12 class Compressor {
13 public:
14     std::string outputPath;
15     std::string gifPath;
16     double targetCompression;
17
18     const EMR& errorMethod;
19     Image inputImage;
20     Image outputImage;
21     Animation* animation = nullptr;
22     uintmax_t originalSize;
23     Quadtree quadtree;
24
25     double timeStart;
26     double timeEnd;
27
28     double bestThreshold;
29
30     Compressor(const Image& image, const std::string& outputPath, const std::string& gifPath, double threshold, int minBlockSize, double targetCompression, const EMR& errorMethod);
31     ~Compressor();
32     void compress();
33     void targetCompress();
34 };
35
36 #endif
```

```
1 #include "compressor.hpp"
2
3 Compressor::Compressor(const Image& image, const std::string& outputPath, const std::string& gifPath, double threshold, int minBlockSize, double targetCompression, const ErrorMethod errorMethod)
4 : outputPath(outputPath), gifPath(gifPath), targetCompression(targetCompression), errorMethod(errorMethod), inputImage(image),
5   outputImage(image), originalSize(image.getFileSize()), animation(nullptr),
6   quadtree(inputImage, errorMethod, threshold, minBlockSize) {}
7
8 Compressor::~Compressor() {
9     if (animation != nullptr) {
10         delete animation;
11         animation = nullptr;
12     }
13 }
14
15 void Compressor::compress() {
16     timeStart = clock();
17     if (targetCompression != 0) {
18         targetCompress();
19     }
20     quadtree.construct();
21     quadtree.render(outputImage);
22     outputImage.save(outputPath);
23     if (!gifPath.empty()) {
24         quadtree.generateAnimation(gifPath);
25     }
26     outputImage.setFileSize(outputImage.estimateNewFileSize());
27     timeEnd = clock();
28 }
```

```
● ● ●
```

```
1 void Compressor::targetCompress() {
2     double leftBound, rightBound;
3     if (errorMethod.identify() == "SSIM") {
4         leftBound = errorMethod.getUpperBound();
5         rightBound = errorMethod.getLowerBound();
6     } else {
7         leftBound = errorMethod.getLowerBound();
8         rightBound = errorMethod.getUpperBound();
9     }
10
11    double targetSize = originalSize * (1 - targetCompression);
12    quadtree.setThreshold(leftBound);
13    quadtree.construct();
14    quadtree.render(outputImage);
15    outputImage.setFileSize(outputImage.estimateNewFileSize());
16    if (outputImage.getFileSize() < targetSize) {
17        quadtree.setThreshold(leftBound);
18        return;
19    }
20
21    quadtree.setThreshold(rightBound);
22    quadtree.construct();
23    quadtree.render(outputImage);
24    outputImage.setFileSize(outputImage.estimateNewFileSize());
25    if (outputImage.getFileSize() > targetSize) {
26        quadtree.setThreshold(rightBound);
27        return;
28    }
29    double bestError = std::numeric_limits<double>::max();
30    double bestThreshold = quadtree.getThreshold();
31
32    for (int i = 0; i < 30; i++) {
33        double mid = (leftBound + rightBound) / 2;
34        quadtree.setThreshold(mid);
35        quadtree.construct();
36        quadtree.render(outputImage);
37        outputImage.setFileSize(outputImage.estimateNewFileSize());
38        double currentSize = outputImage.getFileSize();
39        double currentError = std::abs(currentSize - targetSize);
40
41        if (currentError < bestError) {
42            bestError = currentError;
43            bestThreshold = mid;
44        }
45
46        if (currentSize > targetSize) {
47            leftBound = mid;
48        } else {
49            rightBound = mid;
50        }
51
52        if (currentError / targetSize < 0.0001) {
53            break;
54        }
55    }
56    quadtree.setThreshold(bestThreshold);
57}
58
```

dpw_quadtree_program.cpp

```
1 #include "quadtree/quadtree.hpp"
2 #include "error_measurement/mad.hpp"
3 #include "error_measurement/mpd.hpp"
4 #include "error_measurement/variance.hpp"
5 #include "error_measurement/entropy.hpp"
6 #include "error_measurement/ssim.hpp"
7 #include "compressor/compressor.hpp"
8 #include <regex>
9
10 using namespace std;
11
12 int main() {
13     string inputFilePath, outputPath, gifFilePath;
14     double threshold;
15     int minBlockSize;
16     double targetCompression;
17     int method;
18     string ext;
19
20     Image* inputImage = nullptr;
21
22     auto trim = [] (std::string& s) {
23         s = std::regex_replace(s, std::regex("^\\s+|\\s+$"), "");
24     };
25
26     while (true) {
27         cout << "Enter input file path: ";
28         getline(cin, inputFilePath);
29         trim(inputFilePath);
30
31         ext = fs::path(inputFilePath).extension().string();
32         if (ext != ".png" && ext != ".jpg" && ext != ".jpeg") {
33             cerr << "Error: Unsupported input file extension. Only .png, .jpg, and .jpeg are allowed.\n";
34             continue;
35         }
36
37         Image* temp = new Image(inputFilePath);
38         if (temp->getData() == nullptr) {
39             delete temp;
40             continue;
41         }
42
43         inputImage = temp;
44         break;
45     }
46
47
48     cout << "1. QuadTree Compression with Mean Absolute Deviation" << endl;
49     cout << "2. QuadTree Compression with Max Pixel Difference" << endl;
50     cout << "3. QuadTree Compression with Variance" << endl;
51     cout << "4. QuadTree Compression with Entropy" << endl;
52     cout << "5. QuadTree Compression with Structural Similarity Index" << endl;
53
54     while (true) {
55         cout << "Enter a compression method (1-5): ";
56         cin >> method;
57         if (!cin.fail() && method >= 1 && method <= 5) break;
58
59         cerr << "Error: Invalid method number. Please enter a number between 1 and 5.\n";
60         cin.clear();
61         cin.ignore(numeric_limits<streamsize>::max(), '\n');
62     }
```

```
● ● ●
1 EMM* errorMethod = nullptr;
2 switch (method) {
3     case 2: errorMethod = new MPP(); break;
4     case 3: errorMethod = new Variance(); break;
5     case 4: errorMethod = new Entropy(); break;
6     case 5: errorMethod = new SSIM(); break;
7     default: errorMethod = new MAD(); break;
8 }
9
10 while (true) {
11     cout << "Enter threshold [" << errorMethod->getLowerBound() << " - " << errorMethod->getUpperBound() << "]: ";
12     cin >> threshold;
13
14     if (!cin.fail() && errorMethod->ThresholdWithinBound(threshold)) break;
15
16     cerr << "Error: Threshold must be within the valid range for this error metric.\n";
17     cin.clear();
18     cin.ignore(numeric_limits<streamsize>::max(), '\n');
19 }
20
21 while (true) {
22     cout << "Enter minimum block size: ";
23     cin >> minBlockSize;
24     if (!cin.fail() && minBlockSize > 0) break;
25
26     cerr << "Error: Minimum block size must be positive.\n";
27     cin.clear();
28     cin.ignore(numeric_limits<streamsize>::max(), '\n');
29 }
30
31 while (true) {
32     cout << "Enter target compression ratio [0.0 - 1.0]: ";
33     cin >> targetCompression;
34     if (!cin.fail() && targetCompression >= 0 && targetCompression <= 1) {
35         cin.ignore(numeric_limits<streamsize>::max(), '\n');
36         break;
37     }
38
39     cerr << "Error: Compression ratio must be between 0 and 1.\n";
40     cin.clear();
41     cin.ignore(numeric_limits<streamsize>::max(), '\n');
42 }
```

```

1  while (true) {
2      cout << "Enter output file path: ";
3      getline(cin, outputPath);
4      trim(outputPath);
5
6      fs::path outputExt = fs::path(outputPath).extension();
7      fs::path outputDir = fs::path(outputPath).parent_path();
8
9      if (outputExt != ".png" && outputExt != ".jpg" && outputExt != ".jpeg") {
10         cerr << "Error: Unsupported output file extension. Only .png, .jpg, and .jpeg are allowed.\n";
11         continue;
12     }
13
14     if (outputExt != ext) {
15         cerr << "Error: Output file extension must match the input file extension (" << ext << ").\n";
16         continue;
17     }
18
19     if (!outputDir.empty() && !fs::exists(outputDir)) {
20         cerr << "Error: Output directory does not exist: " << outputDir << endl;
21         continue;
22     }
23
24     break;
25 }
26
27 while (true) {
28     cout << "Enter GIF file path (or leave blank): ";
29     getline(cin, gifFilePath);
30     trim(gifFilePath);
31
32     if (gifFilePath.empty()) break;
33
34     if (fs::path(gifFilePath).extension() != ".gif") {
35         cerr << "Error: GIF path must end with '.gif'.\n";
36         continue;
37     }
38
39     fs::path gifDir = fs::path(gifFilePath).parent_path();
40     if (!gifDir.empty() && !fs::exists(gifDir)) {
41         cerr << "Error: GIF output directory does not exist: " << gifDir << endl;
42         continue;
43     }
44
45     break;
46 }
47
48 Compressor compressor(*inputImage, outputPath, gifFilePath, threshold, minBlockSize, targetCompression, *errorMethod);
49 compressor.compress();
50
51 double timeElapsed = (compressor.timeEnd - compressor.timeStart) / CLOCKS_PER_SEC;
52 cout << "Time taken for compression: " << timeElapsed << " seconds" << endl;
53 cout << "Initial image size: " << compressor.inputImage.getFileSize() << " bytes" << endl;
54 cout << "Compressed image size: " << compressor.outputImage.getFileSize() << " bytes" << endl;
55 if (compressor.originalSize == 0) {
56     cout << "Compression percentage: N/A (original size is 0)" << endl;
57 } else {
58     double ratio = static_cast<double>(compressor.outputImage.getFileSize()) / compressor.originalSize;
59     cout << "Compression percentage: " << (1.0 - ratio) * 100 << "%" << endl;
60 }
61 cout << "Depth of QuadTree: " << compressor.quadtree.getDepth() << endl;
62 cout << "Total nodes: " << compressor.quadtree.getTotalNodes() << endl;
63 cout << "Image saved to: " << compressor.outputPath << endl;
64 if (compressor.animation != nullptr) {
65     cout << "GIF saved to: " << compressor.gifPath << endl;
66 }
67
68 delete errorMethod;
69 delete inputImage;
70 return 0;
71 }

```

BAB 3 PERCOBAAN

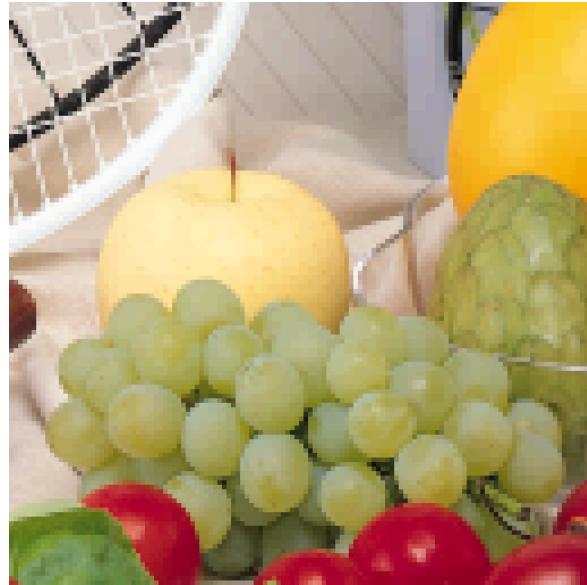
3.1 Tabel Tampilan Input/Output pada Percobaan

No.	CLI + Image Input	CLI + Image Output
1	<p>Input image jpeg berukuran 1600 x 900 menggunakan metode MAD, threshold 5, MBS 16, tanpa target compression ratio.</p> 	<pre> Enter input file path: ../test/wa.jpeg 1. QuadTree Compression with Mean Absolute Deviation 2. QuadTree Compression with Max Pixel Difference 3. QuadTree Compression with Variance 4. QuadTree Compression with Entropy 5. QuadTree Compression with Structural Similarity Index Enter a compression method (1-5): 1 Enter threshold [0 - 127.5]: 5 Enter minimum block size: 16 Enter target compression ratio [0.0 - 1.0]: 0 Enter output file path: ../test/wamad.jpeg Enter GIF file path (or leave blank): ../test/wamad.gif Time taken for compression: 2.633 seconds Initial image size: 112961 bytes Compressed image size: 54075 bytes Compression percentage: 52.1295% Depth of QuadTree: 8 Total nodes: 11037 Image saved to: ../test/wamad.jpeg </pre>  

- 2 Input image png berukuran 512 x 512 menggunakan metode MPD, threshold 20, MBS 16, tanpa target compression ratio.



```
Enter input file path: ../test/fruits.png
1. QuadTree Compression with Mean Absolute Deviation
2. QuadTree Compression with Max Pixel Difference
3. QuadTree Compression with Variance
4. QuadTree Compression with Entropy
5. QuadTree Compression with Structural Similarity Index
Enter a compression method (1-5): 2
Enter threshold [0 - 255]: 20
Enter minimum block size: 16
Enter target compression ratio [0.0 - 1.0]: 0
Enter output file path: ..test/fruitsmpd.png
Error: Output directory does not exist: "..test"
Enter output file path: ../test/fruitsmpd.png
Enter GIF file path (or leave blank): ../test/fruitsmpd.gif
Time taken for compression: 0.889 seconds
Initial image size: 472100 bytes
Compressed image size: 61598 bytes
Compression percentage: 86.9523%
Depth of QuadTree: 7
Total nodes: 16849
Image saved to: ../test/fruitsmpd.png
```

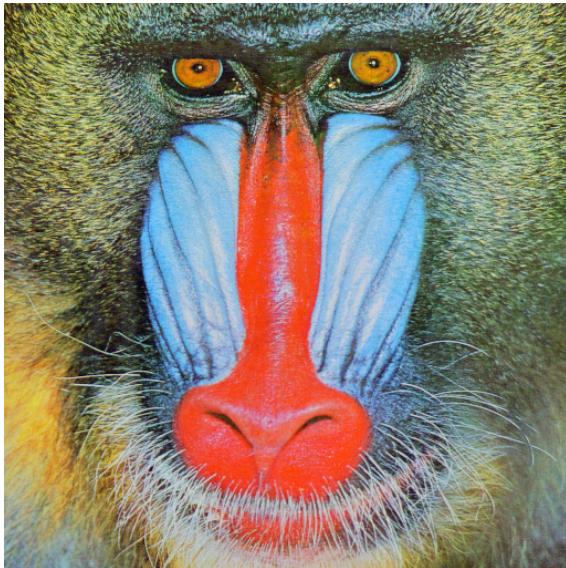


		
3	<p>Input image png berukuran 768 x 512 menggunakan metode Variance, threshold 20, MBS 16, tanpa target compression ratio.</p> 	<pre> Enter a compression method (1-5): 3 Enter threshold [0 - 16256.2]: 0.5 Enter minimum block size: 16 Enter target compression ratio [0.0 - 1.0]: 0 Enter output file path: ../test/tulipsvar.png Enter GIF file path (or leave blank): ../test/tulipsvar.gif Time taken for compression: 1.493 seconds Initial image size: 679233 bytes Compressed image size: 83081 bytes Compression percentage: 87.7684% Depth of QuadTree: 7 Total nodes: 21845 Image saved to: ../test/tulipsvar.png </pre> 

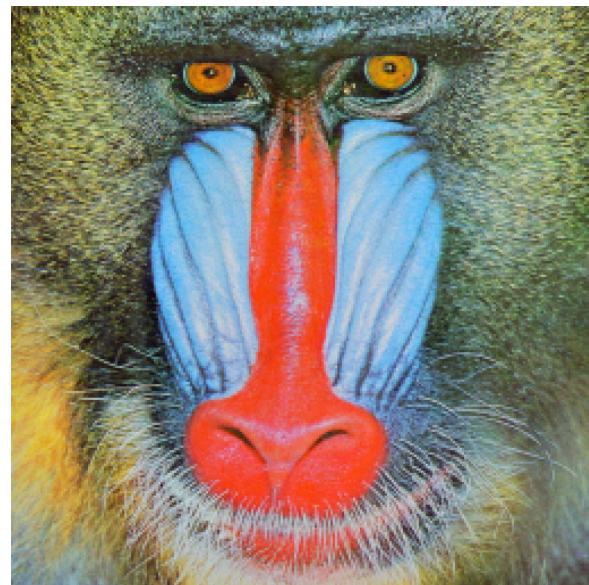
4	<p>Input image jpg berukuran 256 x 256 menggunakan metode Entropy, threshold 0.5, MBS 4, tanpa target compression ratio.</p> 	<pre> Enter input file path: ../test/anomaly.jpg 1. QuadTree Compression with Mean Absolute Deviation 2. QuadTree Compression with Max Pixel Difference 3. QuadTree Compression with Variance 4. QuadTree Compression with Entropy 5. QuadTree Compression with Structural Similarity Index Enter a compression method (1-5): 4 Enter threshold [0 - 8]: 0.5 Enter minimum block size: 4 Enter target compression ratio [0.0 - 1.0]: 0 Enter output file path: ../test/anomaly.jpg Enter GIF file path (or leave blank): ../test/anomaly.gif Time taken for compression: 0.268 seconds Initial image size: 20401 bytes Compressed image size: 11355 bytes Compression percentage: 44.341% Depth of QuadTree: 7 Total nodes: 21845 Image saved to: ../test/anomaly.jpg </pre> 

5	<p>Input image jpeg berukuran 8192 x 5494 menggunakan metode SSIM, threshold 0.75, MBS 16, tanpa target compression ratio.</p> 	<pre> Enter input file path: ../test/ferrari.jpeg 1. QuadTree Compression with Mean Absolute Deviation 2. QuadTree Compression with Max Pixel Difference 3. QuadTree Compression with Variance 4. QuadTree Compression with Entropy 5. QuadTree Compression with Structural Similarity Index Enter a compression method (1-5): 5 Enter threshold [0 - 1]: 0.75 Enter minimum block size: 64 Enter target compression ratio [0.0 - 1.0]: 0 Enter output file path: ../test/ferrarissim.jpeg Enter GIF file path (or leave blank): ../test/ferrarissim.gif Time taken for compression: 242.476 seconds Initial image size: 5426604 bytes Compressed image size: 1663026 bytes Compression percentage: 69.3542% Depth of QuadTree: 9 Total nodes: 175349 Image saved to: ../test/ferrarissim.jpeg </pre>  <p>GIF</p>

- 6 Input image png berukuran 512 x 512 menggunakan metode SSIM, threshold 0.9, MBS 4, dengan target compression ratio 0.5.



```
Enter input file path: ../test/baboon.png
1. QuadTree Compression with Mean Absolute Deviation
2. QuadTree Compression with Max Pixel Difference
3. QuadTree Compression with Variance
4. QuadTree Compression with Entropy
5. QuadTree Compression with Structural Similarity Index
Enter a compression method (1-5): 5
Enter threshold [0 - 1]: 0.9
Enter minimum block size: 4
Enter target compression ratio [0.0 - 1.0]: 0.5
Enter output file path: ../test/babooncom.png
Enter GIF file path (or leave blank): ../test/babooncom.gif
Time taken for compression: 1.515 seconds
Initial image size: 637192 bytes
Compressed image size: 279222 bytes
Compression percentage: 56.1793%
Depth of QuadTree: 8
Total nodes: 87381
Image saved to: ../test/babooncom.png
```



7	<p>Input image jpg berukuran 5011 x 7517 dengan metode Variance, dengan threshold 75, MBS 50, tanpa target compression ratio.</p>  <pre> Enter a compression method (1-5): 3 Enter threshold [0 - 16256.2]: 75 Enter minimum block size: 50 Enter target compression ratio [0.0 - 1.0]: 0 Enter output file path: ../test/buildingtest.jpg Enter GIF file path (or leave blank): ../test/buildingtest.gif Time taken for compression: 173.807 seconds Initial image size: 4529386 bytes Compressed image size: 1173708 bytes Compression percentage: 74.0864% Depth of QuadTree: 9 Total nodes: 65329 Image saved to: ../test/buildingtest.jpg </pre>	

		 GIF
8	Input pilihan compression method di luar constraint seharusnya.	<pre> Enter input file path: ../wa.jpeg Failed to load image: "../wa.jpeg" Enter input file path: ../test/wa.jpeg 1. QuadTree Compression with Mean Absolute Deviation 2. QuadTree Compression with Max Pixel Difference 3. QuadTree Compression with Variance 4. QuadTree Compression with Entropy 5. QuadTree Compression with Structural Similarity Index Enter a compression method (1-5): 6 Error: Invalid method number. Please enter a number between 1 and 5. Enter a compression method (1-5): 0 Error: Invalid method number. Please enter a number between 1 and 5. Enter a compression method (1-5): a Error: Invalid method number. Please enter a number between 1 and 5. Enter a compression method (1-5): </pre>
9	Input path file awal yang null atau tak terdefinisi.	<pre> Enter input file path: Error: Unsupported input file extension. Only .png, .jpg, and .jpeg are allowed. Enter input file path: ./test/abs.png Failed to load image: "./test/abs.png" Enter input file path: </pre>

10	Input output path yang tak terdefinisi.	<pre> Enter input file path: ./test/ferrari.jpeg 1. QuadTree Compression with Mean Absolute Deviation 2. QuadTree Compression with Max Pixel Difference 3. QuadTree Compression with Variance 4. QuadTree Compression with Entropy 5. QuadTree Compression with Structural Similarity Index Enter a compression method (1-5): 5 Enter threshold [0 - 1]: 0.8 Enter minimum block size: 4 Enter target compression ratio [0.0 - 1.0]: 0 Enter output file path: ../step Error: Unsupported output file extension. Only .png, .jpg, and .jpeg are allowed. Enter output file path: ../step/ferrari.jpeg Error: Output directory does not exist: "../step" Enter output file path: </pre>
----	---	--

BAB 4 ANALISIS HASIL PERCOBAAN

4.1 Kompleksitas Algoritma

Kompleksitas algoritma adalah suatu cara untuk mengukur efisiensi suatu algoritma dari sisi waktu eksekusi atau penggunaan memori berdasarkan ukuran masukan. Yang akan dibahas pada bagian ini adalah kompleksitas waktu dari algoritma Divide & Conquer yang digunakan dalam implementasi kompresi image melalui quadtree. Notasi yang paling umum digunakan untuk menentukan kompleksitas suatu algoritma adalah notasi Big-O, yang menyatakan batas atas pertumbuhan waktu eksekusi seiring membesarnya input.

Teorema Master adalah metode cepat untuk menentukan kompleksitas algoritma yang bersifat divide and conquer. Jika algoritma memiliki bentuk rekurens sebagai berikut:

$$T(n) = aT\left(\frac{n}{b}\right) + cn^d$$

Maka kompleksitas $T(n)$ dapat dinyatakan sebagai:

$$T(n) = O(n^d) \text{ jika } a < b^d$$

$$T(n) = O(n^d \log_b(n)) \text{ jika } a = b^d$$

$$T(n) = O(n^{\log_b a}) \text{ jika } a > b^d$$

Algoritma kompresi gambar dengan **quadtree** bekerja dengan memecah gambar menjadi blok-blok yang lebih, jika blok tersebut memiliki tingkat error yang melebihi ambang batas. Ini jelas merupakan kasus *divide and conquer*.

Relasi rekurens untuk proses ini dapat ditulis sebagai:

$$T(n) = 4T(n/4) + cn$$

dengan:

$$a = 4 \text{ (karena 1 blok dibagi menjadi 4)}$$

$b = 4$ (ukuran blok dibagi menjadi 1/4)

$d = 1$ (karena error dan rata-rata linear terhadap jumlah piksel dalam blok)

Karena $a = b^d \Rightarrow 4 = 4^1$, maka berdasarkan **kasus kedua** dari Teorema Master:

$$T(n) = O(n * \log n)$$

Karena $n = W * H$ (jumlah pixel pada gambar), maka kompleksitas akhirnya adalah:

$$O(W * H * \log_4(W * H))$$

Kompleksitas ini menunjukkan bahwa meskipun proses bersifat rekursif, total kerja yang dilakukan per level tetap linear terhadap ukuran gambar, dan hanya terjadi selama logaritmik banyaknya level pembagian blok.

4.2 Pengaruh Nilai Threshold

Peningkatan nilai threshold (dengan pengecualian metode SSIM) menghasilkan tingkat simplifikasi yang lebih tinggi pada gambar. Semakin besar simplifikasi yang terjadi pada warna piksel, maka kemungkinan terjadi kehilangan informasi penting juga meningkat. Dengan demikian, meskipun persentase kompresi yang lebih tinggi dapat dicapai, biaya yang harus dibayar adalah penurunan kualitas gambar secara signifikan.

4.3 Pengaruh Minimal Block Size

Selain nilai threshold, parameter ukuran blok minimal (minimal block size) juga sangat berpengaruh terhadap ukuran berkas akhir. Apabila nilai ukuran blok minimal ditetapkan terlalu besar, proses overriding pada blok gambar akan berlangsung tanpa mempertimbangkan metode pengukuran kesalahan yang ada. Hal ini dapat menyebabkan hilangnya informasi secara drastis pada tiap blok, yang pada akhirnya berdampak negatif terhadap kualitas citra yang dihasilkan.

4.4 Membandingkan 5 Error Measurement Method

Berdasarkan urutan kualitas gambar yang dihasilkan dengan mata telanjang, kelima metode pengukuran kesalahan bisa di-*ranking* sebagai berikut:

SSIM

Entropy

MPD

MAD

Variance

Keunggulan metode SSIM terutama terletak pada formulasi yang mampu mencerminkan kemiripan struktural dalam gambar secara lebih efektif. Metode ini menjaga integritas informasi visual dengan lebih baik sehingga kehilangan detail yang signifikan dapat dihindari, berbeda dengan metode lainnya.

4.5 Implikasi dan Rekomendasi

Dalam penerapan teknik kompresi citra, pemilihan parameter threshold dan ukuran blok minimal harus dilakukan dengan sangat cermat. Keseimbangan antara kompresi yang tinggi dan kualitas gambar yang tetap terjaga dapat dicapai dengan:

- Menyesuaikan nilai threshold secara tepat, mengingat nilai threshold yang tinggi (kecuali pada SSIM) cenderung menghasilkan simplifikasi berlebihan.
- Menetapkan ukuran blok minimal yang tidak terlalu besar, guna menghindari overriding yang mengabaikan mekanisme pengukuran kesalahan, sehingga informasi detail gambar dapat dipertahankan.
- Menggunakan metode pengukuran kesalahan yang handal seperti SSIM, agar kualitas gambar tetap optimal meskipun terjadi proses kompresi yang agresif.

BAB 5 IMPLEMENTASI BONUS

5.1 Structural Similarity Index (SSIM)

SSIM (Structural Similarity Index Measure) adalah metrik kuantitatif yang digunakan untuk mengukur kesamaan perceptual antara dua gambar. Metrik ini didasarkan pada asumsi bahwa informasi struktural dalam gambar merupakan komponen kunci dalam persepsi visual manusia. Dengan mengukur perbedaan dalam luminansi, kontras, dan struktur antara gambar referensi dan gambar yang diuji, SSIM memberikan nilai yang berkisar dari 0 hingga 1. Nilai mendekati 1 menunjukkan bahwa kedua gambar memiliki kesamaan yang tinggi dalam aspek visual, sehingga kompresi yang dilakukan tidak mengorbankan detail penting.

Secara matematis, SSIM antara dua gambar X dan Y dapat dihitung dengan rumus:

$$SSIM(X, Y) = \frac{(2\mu_X\mu_Y + C_1)(2\sigma_{XY} + C_2)}{(\mu_X^2 + \mu_Y^2 + C_1)(\sigma_X^2 + \sigma_Y^2 + C_2)}$$

dimana

- μ_X dan μ_Y adalah rata-rata intensitas gambar X dan Y ,
- σ_X^2 dan σ_Y^2 adalah variansi dari masing-masing gambar,
- σ_{XY} adalah kovarians antara X dan Y ,
- C_1 dan C_2 adalah konstanta stabilisasi untuk menghindari pembagian dengan nilai nol.

Dalam implementasi kode, konstanta tersebut ditetapkan sebagai:

$$k1 = 0.01,$$

$$k2 = 0.03,$$

$L = 255$ (nilai maksimum intensitas piksel pada gambar 8-bit).

Sehingga, $C1 = (k1 \times L)^2$ dan $C2 = (k2 \times L)^2$.

Pada program, fungsi **computeChannelSSIM** bertugas menghitung SSIM untuk satu saluran warna (misalnya merah, hijau, atau biru) dengan langkah-langkah sebagai berikut:

- **Penghitungan Mean:** $\mu = \frac{\text{sum}}{\text{count}}$
- **Penghitungan Variansi:** $\sigma^2 = \frac{\text{sum of squares}}{\text{count}} - \mu^2$
- **Penghitungan Kovarians:** $\sigma_{XY} = \frac{\text{sum of products}}{\text{count}} - \mu_X \mu_Y$
- **Stabilisasi Numerik:** Penggunaan $C1$ dan $C2$ memastikan bahwa nilai pembagi tidak mendekati nol, yang jika terjadi akan menyebabkan ketidakstabilan dalam perhitungan.

Jika penyebut (denominator) dalam perhitungan mendekati angka yang sangat kecil (misalnya, kurang dari $1 * 10^{-12}$), fungsi akan mengembalikan nilai 1.0 untuk menghindari pembagian dengan nol.

Fungsi **computeBlockError** menerapkan perhitungan SSIM pada setiap channel dengan beberapa tahap. Pertama, ia melakukan iterasi pada blok piksel yang ditentukan untuk menghitung total nilai dan kuadratnya untuk masing-masing channel. Lalu, ia menghitung SSIM untuk setiap channel (merah, hijau, dan biru) secara terpisah menggunakan fungsi **computeChannelSSIM**. Terakhir, rata-rata nilai SSIM dari ketiga channel dihitung untuk memberikan satu nilai representatif, sehingga nilai akhir berada dalam rentang [0..1].

5.2 Target Compression Percentage

Parameter target compression percentage merupakan nilai yang ditetapkan oleh pengguna untuk menentukan seberapa besar pengurangan ukuran file yang diinginkan dari gambar asli. Dalam algoritma, nilai ini memandu proses penyesuaian ambang batas (threshold) pada struktur quadtree. Nilai ini digunakan dalam perhitungan ukuran target, dengan persamaan:

$$\text{target size} = \text{initial file size} \times (1 - \text{target compression})$$

Implementasi dari fitur ini terdapat sepenuhnya di file compressor.cpp, terkhususnya di fungsi `targetCompress`. Secara teknik, proses pencarian ambang batas secara dinamis ini dilakukan dengan metode pencarian binary search. Binary search adalah metode pencarian berbasis Divide & Conquer yang efisien pada suatu list yang sesudah terurut menarik. Binary search berulang kali membagi ruang pencarian menjadi dua bagian setiap iterasi hingga menemukan targetnya. Jika menggunakan metode pengukuran error yang sama, threshold yang lebih rendah (lebih tinggi dalam kasus SSIM) akan selalu menghasilkan lebih mendetail dan sebaliknya. Karena itu, metode pencarian binary search dapat digunakan untuk mencari threshold yang cocok untuk persentase kompresi yang diinginkan.

```
FUNCTION targetCompress():

    INITIALIZE leftBound

    INITIALIZE rightBound

    IF errorMethod is "SSIM" THEN:

        leftBound = errorMethod's upper bound

        rightBound = errorMethod's lower bound

    ELSE:

        leftBound = errorMethod's lower bound

        rightBound = errorMethod's upper bound

    SET targetSize = originalSize * (1 - targetCompression)

    SET quadtree's threshold AS leftBound;

    Construct(quadtree)

    Render(outputImage)

    IF outputImage's file size < targetSize THEN:

        return

    SET quadtree's threshold AS rightBound;

    Construct(quadtree)

    Render(outputImage)

    IF outputImage's file size > targetSize THEN:

        return

    SET bestError = INT_MAX

    SET bestThreshold = quadtree's threshold

    FOR i ← 1 TO 30 DO:

        SET mid = (leftBound + rightBound) / 2;

        SET quadtree's threshold AS mid;
```

```

Construct(quadtree)

    Render(outputImage)

        SET currentSize = outputImage's file size

        SET currentError = |currentSize - targetSize|

        IF currentError < bestError THEN:

            SET bestError = currentError

            SET bestThreshold = mid

        IF currentSize > targetSize THEN:

            SET leftBound = mid

        ELSE:

            SET rightBound = mid

        IF currentError / targetSize < 0.0001 THEN:

            break

    END FOR

    SET quadtree's threshold as bestThreshold

```

Fungsi targetCompress bertujuan untuk menentukan nilai threshold optimal yang kemudian akan digunakan pada fungsi compress utama guna menghasilkan file output dengan ukuran yang mendekati target. Pendekatan pencarian threshold menggunakan metode binary search, dengan penentuan batas sebagai berikut:

- Secara umum, batas kiri diatur sebagai nilai terendah (lower bound) dan batas kanan sebagai nilai tertinggi (upper bound) dari metode pengukuran error.
- Pada metode SSIM, yang menghasilkan indeks similaritas, penentuan batas dibalik sehingga nilai terendah dijadikan batas kanan dan nilai tertinggi dijadikan batas kiri.

Selanjutnya, fungsi akan menghitung persentase kompresi yang dihasilkan pada kedua ambang tersebut untuk memastikan tidak terjadi pencarian redundant. Contohnya, jika penggunaan threshold paling ringan menghasilkan ukuran file yang masih melebihi target, maka pencarian lebih lanjut tidak diperlukan karena threshold yang lebih ketat akan menghasilkan output yang tidak optimal, dan sebaliknya.

Apabila kedua nilai ambang tidak memenuhi target kompresi, fungsi akan menjalankan binary search dengan maksimal 30 iterasi—batas ini ditetapkan agar proses pencarian memiliki titik henti yang definitif dan menghindari loop tak berhingga. Pada setiap iterasi, langkah-langkah berikut dilakukan:

1. Menghitung nilai tengah antara batas kiri dan kanan.
2. Mencoba threshold tersebut dalam quadtree untuk melakukan kompresi.
3. Mengukur perbedaan antara ukuran file hasil kompresi dengan ukuran file target.
4. Jika perbedaan tersebut cukup mendekati target, proses pencarian dihentikan dan threshold optimal disimpan.
5. Jika belum memenuhi, nilai batas kiri atau kanan diperbarui sesuai dengan hasil pengukuran, dan iterasi berlanjut.

Hasil akhirnya adalah gambar yang ukurannya mendekati target, dengan perhitungan persentase kompresi aktual menggunakan formula $(1.0 - (\text{ukuran file hasil kompresi} / \text{ukuran file asli})) \times 100\%$. Pendekatan ini tidak hanya menjamin efisiensi dalam pengurangan ukuran file tetapi juga memastikan kualitas gambar tetap terjaga sesuai parameter yang diinginkan pengguna.

5.3 GIF Visualization

Komponen visualisasi GIF berfungsi untuk merepresentasikan proses pembangunan struktur quadtree secara bertahap berdasarkan level kedalaman pohon. Setiap frame pada GIF menggambarkan kondisi pembagian blok gambar pada tingkat kedalaman tertentu secara berurutan. Oleh karena itu, hasil akhir menghasilkan animasi yang menunjukkan bagaimana program membagi gambar menjadi blok-blok yang lebih kecil dari akar hingga simpul daun. Algoritma implementasinya adalah sebagai berikut:

```
FUNCTION generateAnimation(string gifPath):  
    INITIALIZE queue  
    ENQUEUE root INTO queue  
    WHILE queue is not empty DO:  
        SET size = size of queue  
        FOR i ← 0 TO size DO:  
            SET currentNode = top of queue  
            DEQUEUE queue
```

```

FOR j ← currentNode's x TO currentNode's x + currentNode's
width AND j < frame's width DO:

    FOR k ← currentNode's y TO currentNode's y + currentNode's
height AND k < frame's height DO:

        SET frame's pixel AT (j, k) = node's average color

    FOR EACH child in childrenNode DO:

        IF child DO:

            ENQUEUE child

    ADD FRAME TO ANIMATION

```

Pada implementasi quadtree, node akar (root node) disimpan sebagai titik awal untuk melakukan traversal seluruh pohon, mengingat setiap node menyimpan referensi ke anak-anaknya. Untuk menambahkan frame ke dalam GIF secara level per level, digunakanlah teknik Breadth-First Search (BFS) dengan memanfaatkan sebuah queue. Berikut adalah penjelasan rinci mekanisme yang diterapkan:

- Node akar dimasukkan ke dalam queue sebagai titik awal.
- Selama queue tidak kosong, dilakukan iterasi dengan membaca jumlah node yang ada pada queue pada iterasi tersebut. Hal ini memastikan bahwa semua node pada tingkat (level) saat itu diproses terlebih dahulu sebelum pindah ke level berikutnya.
- Untuk setiap node yang di dequeue, fungsi mengambil warna rata-rata blok yang direpresentasikan oleh node tersebut. Warna rata-rata inilah yang digunakan untuk mempopulasikan sebuah frame. Karena setiap level quadtree, secara kolektif, mencakup jumlah piksel yang sama dengan gambar asli, maka setiap frame yang dihasilkan merefleksikan kondisi visual gambar pada kedalaman tertentu.
- Setelah seluruh node pada satu level selesai diproses, child node dari setiap node tersebut ditambahkan ke queue. Proses inilah yang memastikan bahwa traversal berlanjut secara mendalam, level per level, hingga seluruh quadtree telah dijelajahi.
- Hasil akhirnya adalah sebuah GIF yang dihasilkan dari rangkaian frame, di mana setiap frame menggambarkan kondisi kompresi gambar pada tingkat yang berbeda, memberikan gambaran evolusi proses kompresi.

Pendekatan ini memungkinkan visualisasi progresif dari kompresi gambar melalui quadtree, dengan setiap frame secara efektif merepresentasikan keadaan gambar pada kedalaman tertentu dalam struktur tree.

LAMPIRAN

Tautan Repository GitHub

Tabel Checklist

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil dijalankan	✓	
3. Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	✓	
4. Mengimplementasi seluruh metode perhitungan error wajib	✓	
5. [Bonus] Implementasi persentase kompresi sebagai parameter tambahan	✓	
6. [Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	✓	
7. [Bonus] Output berupa GIF Visualisasi Proses Pembentukan Quadtree dalam Kompresi Gambar	✓	
8. Program dan laporan dibuat kelompok sendiri	✓	

Referensi Tambahan

Zhou Wang, A.C. Bovik, H.R. Sheikh, E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity; 30 April 2004.

Imatest Documentation: SSIM. Structure Similarity Index;
<https://www.imatest.com/docs/ssim/#:~:text=Introduction%20%E2%80%94%20The%20Structural%20Similarity%20Index,by%20losses%20in%20data%20transmission>.

York, Tanner W. QuadTrees for Image Compression; 13 May 2020;
<https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>