

Tugas Kecil 2 IF2211 Strategi Algoritma: Kompresi Gambar dengan Metode Quadtree

Muhammad Ghifary Komara Putra - 13523066¹

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

¹m.ghifary.k.p@gmail.com, 13523066@std.stei.itb.ac.id

Abstrak—Kompresi gambar dengan metode quadtree merupakan suatu metode kompresi gambar bersifat *lossy* yang memanfaatkan struktur data quadtree dan algoritma *divide and conquer* dalam proses kompresi. Dalam konteks ini, suatu *node* quadtree akan menyatakan blok gambar tertentu dan keempat *child* dari *node* tersebut akan menyatakan blok gambar yang merupakan segmen hasil pembagian blok gambar *parent*-nya menjadi empat bagian sama besar. Pembagian dilakukan hingga mencapai ambang batas tertentu, baik ambang batas secara matematis maupun berdasarkan ukuran blok minimum. Penulis mengembangkan alternatif solusi program kompresi gambar dengan metode quadtree berbasis CLI dengan bahasa C++. Pada program ini, pengguna dapat memasukkan gambar yang hendak dikompresi, memilih metode kompresi yang tersedia: variansi, Mean Absolute Deviation (MAD), Max Pixel Difference, entropi, dan Structural Similarity Index (SSIM), kemudian melakukan kompresi berdasarkan ambang batas tertentu. Selain itu, pengguna pun dapat melakukan kompresi berdasarkan target persentase kompresi yang diinginkan serta dapat melihat proses kompresi dalam bentuk file GIF. Kompresi berdasarkan target persentase diimplementasikan dengan algoritma *binary search* terhadap *threshold* sedangkan kompresi GIF dilakukan dengan menelusuri setiap *node* pada *quadtree* hasil kompresi.

Kata Kunci: *divide and conquer, kompresi gambar, quadtree*

I. PENDAHULUAN



Gambar 1. Hasil Kompresi Gambar dengan Metode Quadtree

Kompresi gambar merupakan suatu proses atau algoritma yang digunakan untuk mengurangi harga penyimpanan dari suatu *file* gambar, yaitu mengurangi ukuran *file*. Secara umum, kompresi gambar dapat diklasifikasikan menjadi dua kategori, yaitu *lossy* dan *lossless*. Kompresi gambar *lossy* merupakan metode kompresi yang berusaha mengurangi ukuran *file* dengan cara menghapus/menggantikan sebagian data dari suatu gambar. Di sisi lain, kompresi gambar *lossless* berusaha mengurangi ukuran *file* dengan menghilangkan metadata yang tidak perlu.

Kompresi gambar dengan quadtree merupakan suatu metode kompresi gambar *lossy* yang memanfaatkan struktur data quadtree, yaitu sebuah *tree* yang dapat memiliki empat buah *child*, dan algoritma *divide and conquer* untuk menentukan struktur quadtree yang terbentuk. Pada metode ini, program akan meninjau suatu blok gambar tertentu (diawali dengan seluruh bagian gambar), mengecek apakah blok tersebut telah melewati ambang batas

tertentu, kemudian terus membagi blok tersebut menjadi empat buah bagian hingga ambang batas tersebut berhasil dilewati.

II. DESKRIPSI SOLUSI

A. Implementasi Algoritma *Divide and Conquer*

Solusi dengan pendekatan algoritma *divide and conquer* yang telah dikembangkan adalah sebagai berikut:

1. Buatlah sebuah node dari quadtree. Node ini akan meninjau gambar pada matriks dengan indeks ($x_idx = 0$, $y_idx = 0$) hingga ($width = \text{lebar gambar}$, $height = \text{tinggi gambar}$)
2. Tinjau node, hitung ukuran blok dan hasil pengukuran error sesuai metode yang dipilih. Jika dimensi blok kurang dari atau sama dengan *minimum block size* atau nilai pengukuran error kurang dari atau sama dengan nilai ambang batas (*threshold*), hentikan pembagian. Node ini merupakan node daun. (terdapat kasus khusus untuk SSIM, dijelaskan pada bab VII)
3. Jika node saat ini adalah node daun, hitung rerata setiap channel r, g, b pada blok yang sedang ditinjau. Isi nilai r, g, b gambar hasil kompresi pada bagian blok tersebut dengan rerata channel yang bersesuaian, dengan melakukan traversal pada seluruh pixel dalam blok tersebut, membuatnya menjadi terisi dengan satu warna
4. Jika node saat ini bukan node daun, buat 4 *child node* baru (atau 2, dalam kasus khusus). Proses ini akan membagi node menjadi 4 bagian, berturut-turut kiri atas, kanan atas, kiri bawah, dan kanan bawah. Secara matematis, pembagian indeks matriks yang akan ditinjau adalah sebagai berikut:
 - (x_idx , y_idx) hingga ($half_width$, $half_height$)
 - ($x_idx + half_width$, y_idx) hingga ($half_width + width_increment$, $half_height$), jika $half_width$ tidak bernilai nol
 - (x_idx , $y_idx + half_height$) hingga ($half_width$, $half_height + height_increment$), jika $half_height$ tidak bernilai nol
 - ($x_idx + half_width$, $y_idx + half_height$) hingga ($half_width + width_increment$, $half_height + height_increment$)

Keterangan:

- $Width = \text{lebar blok gambar}$
 - $Height = \text{tinggi block gambar}$
 - $half_width = width/2$
 - $half_height = height/2$
 - $width_increment = width \bmod 2$
 - $height_increment = height \bmod 2$
5. Ulangi langkah 2-4 kepada setiap child node dengan nilai x_idx , y_idx , $width$, dan $height$ yang telah diperbarui

Implementasi algoritma *divide and conquer* dalam bahasa C++ tertera pada bab V.

Catatan: Gambar sebenarnya direpresentasikan sebagai array satu dimensi. Penyebutannya sebagai matriks dilakukan hanya untuk mempermudah visualisasi (agar terdapat dimensi “lebar” dan “tinggi” selayaknya gambar pada umumnya).

B. Validasi Masukan Pengguna

Beberapa validasi masukan pengguna yang diimplementasikan dalam program ini adalah sebagai berikut:

- Validasi terhadap kebenaran masukan file gambar yang hendak dikompresi (absolute path dengan ekstensi png, jpg, atau jpeg pada sistem operasi windows atau linux)
- Validasi terhadap ketidaktersediaan file gambar yang hendak dikompresi
- Validasi terhadap kode angka untuk metode pengukuran error
- Validasi terhadap nilai ambang batas (*threshold*) untuk setiap metode pengukuran error
- Validasi terhadap ukuran blok minimum (*minimum block size*)
- Validasi terhadap target persentase kompresi
- Validasi terhadap kebenaran masukan path file gambar hasil kompresi (absolute path dengan ekstensi png, jpg, atau jpeg pada sistem operasi windows atau linux)

- Validasi terhadap kebenaran masukan path file GIF proses kompresi (absolute path dengan ekstensi gif pada sistem operasi windows atau linux)

III. PANDUAN PENGGUNAAN PROGRAM

1. Program dikembangkan dengan GCC versi 6.3.0. Silakan lakukan instalasi versi tersebut atau versi C++ setelahnya yang kompatibel
2. Clone repositori pada lampiran ke dalam perangkat Anda
3. Pada CLI, pastikan anda berada pada *root directory*
4. Jalankan perintah berikut: `./bin/main`
5. Program siap dijalankan



Jika hendak melakukan kompilasi kembali terhadap file .exe, silakan ikuti langkah berikut:

1. Buka CLI pada root directory
2. Jalankan perintah berikut untuk melakukan kompilasi
`g++ src/error_measurement/variance.cpp src/quadtrees/quadtrees.cpp src/main.cpp
src/error_measurement/entropy.cpp src/error_measurement/mad.cpp src/error_measurement/mpd.cpp
src/bonus/gif_generator.cpp src/bonus/ssim.cpp src/bonus/compression_target.cpp
src/inputs/input_data.cpp -o bin/main`

IV. CONTOH MASUKAN DAN KELUARAN

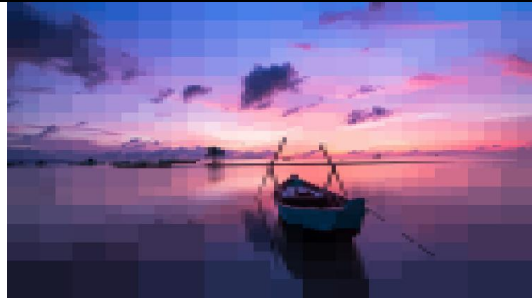
Bab ini akan memaparkan tangkapan layar contoh masukan dan keluaran pada program yang telah dikembangkan. File image asal, image hasil kompresi, serta GIF proses kompresi dapat diakses pada repositori yang tertera pada lampiran, tepatnya pada folder bin/img_input dan folder test. Masukan bernomor ke-i pada Tabel 1 memiliki format nama tc-i-namaGambar.ekstensi untuk masukan, tc-i-namaGambar-cmp.ekstensi untuk hasil kompresi, dan tc-i-namaGambar-gif.gif untuk file GIF. Masing-masing contoh masukan berturut-turut mewakili kasus metode variansi, Mean Absolute Deviation, Max Pixel Difference, Entropy, SSIM, 0 threshold, dan bonus target kompresi.

Tabel 1. Contoh Masukan dan Keluaran Program

No.	Masukan Program	Keluaran Program
1.	 <pre> [INPUT] Absolute Path to Image : C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/bin/img_input/tc1-panorama.jpg Error Measurement Method : 1 Threshold : 2500 Minimum Block Size : 128 Compression Percentage Target : 0 Absolute Path to Result : C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/test/tc1-panorama-cmp.jpg Absolute Path to GIF : C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/test/tc1-panorama-gif.gif </pre>	 <pre> [OUTPUT] Execution Time (ms) : 15594 Input File Size : 1457 Output File Size : 768 Compression Percentage : 47.289% Tree Depth : 9 Node Count : 104201 </pre>
2.		



```
[INPUT]
Absolute Path to Image      :
C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/bin/img_input/tc2-boat.jpg
Error Measurement Method    : 2
Threshold                   : 7
Minimum Block Size          : 32
Compression Percentage Target : 0
Absolute Path to Result     :
C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/test/tc2-boat-cmp.jpg
Absolute Path to GIF        :
C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/test/tc2-boat-gif.gif
```



```
[OUTPUT]
Execution Time (ms)        : 34
Input File Size            : 43
Output File Size           : 25
Compression Percentage      : 41.8605%
Tree Depth                 : 7
Node Count                 : 4921
```

3.



```
[INPUT]
Absolute Path to Image      :
C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/bin/img_input/tc3-home.png
Error Measurement Method    : 3
Threshold                   : 15
Minimum Block Size          : 16
Compression Percentage Target : 0
Absolute Path to Result     :
C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/test/tc3-home-cmp.png
Absolute Path to GIF        :
C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/test/tc3-home-gif.gif
```



```
[OUTPUT]
Execution Time (ms)        : 35
Input File Size            : 691
Output File Size           : 75
Compression Percentage      : 89.1462%
Tree Depth                 : 8
Node Count                 : 16553
```

4.



[INPUT]
Absolute Path to Image :
C:/Users/HP/Documents/Semester4/stima/Tucil12_13523066/bin/img_input/tc4-ikebana.png
Error Measurement Method : 4
Threshold : 2.7
Minimum Block Size : 50
Compression Percentage Target : 0
Absolute Path to Result :
C:/Users/HP/Documents/Semester4/stima/Tucil12_13523066/test/tc4-ikebana-cmp.png
Absolute Path to GIF :
C:/Users/HP/Documents/Semester4/stima/Tucil12_13523066/test/tc4-ikebana-gif.gif

[OUTPUT]
Execution Time (ms) : 1198
Input File Size : 525
Output File Size : 61
Compression Percentage: 88.381%
Tree Depth : 8
Node Count : 21845

5.



[INPUT]
Absolute Path to Image :
C:/Users/HP/Documents/Semester4/stima/Tucil12_13523066/bin/img_input/tc5-hokusai.jpg
Error Measurement Method : 5
Threshold : 0.75
Minimum Block Size : 8
Compression Percentage Target : 0
Absolute Path to Result :
C:/Users/HP/Documents/Semester4/stima/Tucil12_13523066/test/tc5-hokusai-cmp.jpg
Absolute Path to GIF :
C:/Users/HP/Documents/Semester4/stima/Tucil12_13523066/test/tc5-hokusai-gif.gif



[OUTPUT]
Execution Time (ms) : 342
Input File Size : 79
Output File Size : 53
Compression Percentage: 32.9114%
Tree Depth : 8
Node Count : 56053

6.



[INPUT]
Absolute Path to Image :
C:/Users/HP/Documents/Semester4/stima/Tucil12_13523066/bin/img_input/tc6-bocchi.png
Error Measurement Method : 2
Threshold : 0
Minimum Block Size : 32
Compression Percentage Target : 0
Absolute Path to Result :
C:/Users/HP/Documents/Semester4/stima/Tucil12_13523066/test/tc6-bocchi-cmp.png
Absolute Path to GIF :
C:/Users/HP/Documents/Semester4/stima/Tucil12_13523066/test/tc6-bocchi-gif.gif



[OUTPUT]
Execution Time (ms) : 29
Input File Size : 118
Output File Size : 49
Compression Percentage: 58.4746%
Tree Depth : 7
Node Count : 19157

7.



```
[INPUT]
Absolute Path to Image      :
C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/bin/img_input/tc7-emu.jpeg
Error Measurement Method    : 1
Threshold                   : 10
Minimum Block Size          : 16
Compression Percentage Target : 0.35
Absolute Path to Result     :
C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/test/tc7-emu-cmp.jpeg
Absolute Path to GIF        :
C:/Users/HP/Documents/Semester4/stima/Tucil2_13523066/test/tc7-emu-gif.gif
```



```
[OUTPUT]
Execution Time (ms) : 24560
Input File Size     : 135
Output File Size    : 89
Compression Percentage: 33.3333%
Tree Depth          : 11
Node Count          : 131362
```

V. POTONGAN SOURCE CODE

Berikut merupakan potongan kode program yang telah dikembangkan. Perhatikan bahwa beberapa kode sebenarnya memiliki file header. File header dan Kode keseluruhan dapat diakses melalui repositori yang tersedia pada lampiran. Sebagai tambahan, implementasi program dilakukan dengan menggunakan beberapa *library public domain* bersifat *single header file*, di antaranya `stb_image.h`, `stb_image_write.h`, dan `gif.h` karya Charlie Tangora.

Quadtree.cpp

```
1  #include "../header/quadtree.h"
2  #include "../header/stb_image_write.h"
3  #include <string>
4  #include <regex>
5  #include <iostream>
6  using namespace std;
7
8  unsigned char* QuadTree::img = nullptr;
9  int QuadTree::original_width = -1;
10 int QuadTree::original_height = -1;
11 unsigned char* QuadTree::compressed_img = nullptr;
12 int QuadTree::error_measurement_method = -1;
13 float QuadTree::threshold = -1;
14 int QuadTree::min_block_size = -1;
15 int QuadTree::node_count = -1;
16 int QuadTree::max_depth = -1;
17 int QuadTree::channel = -1;
```

```
19 QuadTree::QuadTree(int depth, int x_idx, int y_idx, int width, int height){
20     this->depth = depth;
21     this->x_idx = x_idx;
22     this->y_idx = y_idx;
23     this->width = width;
24     this->height = height;
25     this->block_size = width*height;
26     this->error_result = -1;
27     this->is_leaf = false;
28
29     for(int i=0;i<4;i++){
30         this->children[i] = nullptr;
31     }
32
33     node_count++;
34
35     if(this->depth > max_depth){
36         max_depth = this->depth;
37     }
38
39     this->r_avg = -1;
40     this->g_avg = -1;
41     this->b_avg = -1;
42 }
43
44 void QuadTree::setStatic(unsigned char* img, unsigned char* compressed_img, int error_measurement_method, float threshold, int min_size,
45 int channel){
46     this->img = img;
47     this->original_width = width;
48     this->original_height = height;
49     this->compressed_img = compressed_img;
50     this->error_measurement_method = error_measurement_method;
51     this->threshold = threshold;
52     this->min_block_size = min_size;
53     this->node_count = 1;
54     this->max_depth = 1;
55     this->channel = channel;
56 }
```

```

57 void QuadTree::compressImage(){
58     is_leaf = false;
59     // basis: tinjau block size dan threshold
60     if(block_size<=min_block_size){
61         is_leaf = true;
62     } else if(error_measurement_method==1){ // variance
63         error_result = this->variance();
64         if(error_result<=threshold){
65             is_leaf = true;
66         }
67     } else if(error_measurement_method==2){ // MAD
68         error_result = this->MAD();
69         if(error_result<=threshold){
70             is_leaf = true;
71         }
72     } else if(error_measurement_method==3){ // MPD
73         error_result = this->MPD();
74         if(error_result<=threshold){
75             is_leaf = true;
76         }
77     } else if(error_measurement_method==4){ // entropy
78         error_result = this->entropy();
79         if(error_result<=threshold){
80             is_leaf = true;
81         }
82     } else if(error_measurement_method==5){ // entropy
83         error_result = this->SSIM();
84         if(error_result>=threshold){
85             is_leaf = true;
86         }
87     }
88
89     // DIVIDE
90     if(!is_leaf){
91         int width_increment = 0;
92         int height_increment = 0;
93         if(width%2==1) width_increment = 1;
94         if(height%2==1) height_increment = 1;
95
96         int half_width = width/2;
97         int half_height = height/2;
98         // top left
99         children[0] = new QuadTree(depth+1, x_idx, y_idx, half_width, half_height);
100        // top right
101        if(half_width!=0) children[1] = new QuadTree(depth+1, x_idx + half_width, y_idx, half_width + width_increment, half_height);
102        // bottom left
103        if(half_height!=0) children[2] = new QuadTree(depth+1, x_idx, y_idx + half_height, half_width, half_height + height_increment);
104        // bottom right
105        children[3] = new QuadTree(depth+1, x_idx + half_width, y_idx + half_height, half_width + width_increment, half_height
106        + height_increment);
107
108        for(int i=0;i<4;i++){
109            // DIVIDE
110            if(children[i] != nullptr) children[i]->compressImage();
111        }

```

```

111     } else{ // CONQUER
112         // isi blok yang ditinjau dengan rerata channel r, g, b
113         r_avg = this->channelAverage(0);
114         g_avg = this->channelAverage(1);
115         b_avg = this->channelAverage(2);
116
117
118         int pixel_index;
119         for (int x = x_idx; x < x_idx+width;x++){
120             for (int y = y_idx; y < y_idx+height; y++){
121                 pixel_index = (y * original_width + x) * channel;
122                 compressed_img[pixel_index] = r_avg;
123                 compressed_img[pixel_index+1] = g_avg;
124                 compressed_img[pixel_index+2] = b_avg;
125             }
126         }
127     }
128 }
129
130 void QuadTree::saveCompressedImage(const char* path){
131     regex path_png(".*\\.png$");
132     if(regex_match(path, path_png)){
133         saveCompressedImagePNG(path);
134     } else{
135         saveCompressedImageJPG(path);
136     }
137 }
138
139 void QuadTree::saveCompressedImageJPG(const char* path){
140     stbi_write_jpg(path, width, height, channel, this->compressed_img, 75);
141 }
142
143 void QuadTree::saveCompressedImagePNG(const char* path){
144     stbi_write_png(path, width, height, channel, this->compressed_img, width * channel)
145 }
146
147 int QuadTree::getMaxDepth(){
148     return this->max_depth;
149 }
150 int QuadTree::getNodeCount(){
151     return this->node_count;
152 }

```

```

1  #include "../header/quadtree.h"
2  #include "../header/stb_image.h"
3  #include <cmath>
4  #include <iostream>
5  using namespace std;
6
7  float QuadTree::channelAverage(int rgb){
8      float avg = 0;
9      int pixel_index;
10
11      for(int x=x_idx;x<x_idx+width;x++){
12          for(int y=y_idx;y<y_idx+height;y++){
13              pixel_index = (y * original_width + x) * channel;
14              avg += img[pixel_index + rgb];
15          }
16      }
17      avg /= block_size;
18      return avg;
19  }
20
21  // Menghitung variansi suatu channel R, G, B, dengan nilai int rgb berturut turut 0, 1, 2
22  float QuadTree::channelVariance(int rgb){
23      float result = 0;
24      float avg = 0;
25      float sum_of_squared = 0;
26      int pixel_index;
27
28      for(int x=x_idx;x<x_idx+width;x++){
29          for(int y=y_idx;y<y_idx+height;y++){
30              pixel_index = (y * original_width + x) * channel;
31              avg += img[pixel_index + rgb];
32              sum_of_squared += pow(img[pixel_index + rgb], 2);
33          }
34      }
35
36      // variansi = E(X^2) + (E(X))^2
37      result = sum_of_squared/block_size - pow(avg/block_size, 2);
38      return result;
39  }
40
41  float QuadTree::variance(){
42      float result = pow(this->channelVariance(0), 2) + pow(this->channelVariance(1), 2) + pow(this->channelVariance(2), 2);
43      result/=3;
44      return result;
45  }

```

```
1  #include "../header/quadtree.h"
2  #include "../header/stb_image.h"
3  #include <cmath>
4  #include <iostream>
5  using namespace std;
6
7
8  // Menghitung MAD suatu channel R, G, B, dengan nilai int rgb berturut turut 0, 1, 2
9  float QuadTree::channelMAD(int rgb){
10     float result = 0;
11     float avg = this->channelAverage(rgb);
12
13     int pixel_index;
14     for(int x=x_idx;x<x_idx+width;x++){
15         for(int y=y_idx;y<y_idx+height;y++){
16             pixel_index = (y * original_width + x) * channel;
17             result += abs(img[pixel_index + rgb] - avg);
18         }
19     }
20     result = result/block_size;
21     return result;
22 }
23
24 float QuadTree::MAD(){
25     float result = this->channelMAD(0) + this->channelMAD(1) + this->channelMAD(2);
26     result/=3;
27     return result;
28 }
```

mpd.cpp

```
1  #include "../header/quadtree.h"
2  #include "../header/stb_image.h"
3  #include <cmath>
4  #include <iostream>
5  using namespace std;
6
7
8  // Menghitung MPD suatu channel R, G, B, dengan nilai int rgb berturut turut 0, 1, 2
9  float QuadTree::channelMPD(int rgb){
10     float channel_max = -1;
11     float channel_min = 300;
12
13     int pixel_index;
14     for(int x=x_idx;x<x_idx+width;x++){
15         for(int y=y_idx;y<y_idx+height;y++){
16             pixel_index = (y * original_width + x) * channel;
17             channel_max = fmax(img[pixel_index + rgb], channel_max);
18             channel_min = fmin(img[pixel_index + rgb], channel_min);
19         }
20     }
21
22     return (channel_max - channel_min);
23 }
24
25 float QuadTree::MPD(){
26     float result = this->channelMAD(0) + this->channelMAD(1) + this->channelMAD(2);
27     result/=3;
28     return result;
29 }
```

```

1  #include "../header/quadtree.h"
2  #include "../header/stb_image.h"
3  #include <cmath>
4  #include <iostream>
5  using namespace std;
6
7
8  // Menghitung entropy suatu channel R, G, B, dengan nilai int rgb berturut turut 0, 1, 2
9  float QuadTree::channelEntropy(int rgb){
10     int count[256] = {0};
11     float result = 0;
12
13     int pixel_index;
14     // Banyak kemunculan nilai RGB tertentu
15     for(int x=x_idx; x<x_idx+width; x++){
16         for(int y=y_idx; y<y_idx+height; y++){
17             pixel_index = (y * original_width + x) * channel;
18             count[img[pixel_index + rgb]]++;
19         }
20     }
21
22     // Entropi
23     float probability;
24     for(int i=0; i<256; i++){
25         probability = count[i]/block_size;
26         result += (probability*log2(probability));
27     }
28
29     return -1*result;
30 }
31
32 float QuadTree::entropy(){
33     float result = this->channelEntropy(0) + this->channelEntropy(1) + this->channelEntropy(2);
34     result/=3;
35     return result;
36 }

```

compression_target.cpp

```

1  #include "../header/quadtree.h"
2  #include "../header/stb_image.h"
3  #include <cmath>
4  #include <iostream>
5  #include <fstream>
6  using namespace std;
7
8  void QuadTree::setThreshold(float thresh){
9      this->threshold = thresh;
10 }
11 void QuadTree::setMinBlockSize(float mbs){
12     this->min_block_size = mbs;
13 }
14 void QuadTree::resetRelevantData(int depth, int node_count){
15     this->depth = depth; this->node_count = node_count;
16 }
17 void QuadTree::compressImageByFileSize(float compression_pct, int emm, const char* img_input_path, const char* img_output_path)
18 {
19     // Hitung ukuran file input
20     ifstream file(img_input_path, ios::binary | ios::ate);
21     streamsize size = file.tellg();
22     float input_file_size = static_cast<float>(size / 1024);
23     float current_pct;
24     float output_file_size;
25     float current_min = 0;
26     float current_max;
27
28     if(emm==1){
29         current_max = 16256.25;
30     } else if(emm==2){
31         current_max = 127.5;
32     } else if(emm==3){
33         current_max = 255;
34     } else if(emm==4){
35         current_max = 8;
36     } else if(emm==5){
37         current_max = 1;
38     }
39
40     // Asumsikan min block size bernilai 1
41     this->setMinBlockSize(1);
42
43     // Mulai binary search
44     for(int i=0;i<20;i++){
45         // Ambil bagian tengah dari range nilai
46         this->setThreshold((current_min+current_max)/2);
47         // Kompresi
48         this->resetRelevantData(1,1);
49         this->compressImage();
50         this->saveCompressedImage(img_output_path);
51
52         // Hitung ukuran file output, bandingkan
53         ifstream file2(img_output_path, ios::binary | ios::ate);
54         size = file2.tellg();
55         output_file_size = static_cast<float>(size / 1024);
56
57         current_pct = (input_file_size-output_file_size)/input_file_size;
58         final_compression_pct = current_pct;
59
60         // Evaluasi range binary search
61         if(abs(current_pct - compression_pct) < 0.01){
62             return;
63         } else if(current_pct > compression_pct){
64             if(error_measurement_method==5){
65                 current_min = this->threshold;
66             } else{
67                 current_max = this->threshold;
68             }
69         } else{
70             if(error_measurement_method==5){
71                 current_max = this->threshold;
72             } else{
73                 current_min = this->threshold;
74             }
75         }
76     }
77 }
78
79 }
80
81 }
82

```

ssim.cpp

```
1  #include "../header/quadtree.h"
2  #include "../header/stb_image.h"
3  #include <cmath>
4  #include <iostream>
5  using namespace std;
6
7  // Penurunan rumus tertera pada laporan
8
9  float QuadTree::channelSSIM(int rgb){
10     float C2 = pow(0.03*255, 2);
11     return C2/(this->channelVariance(rgb)+C2);
12 }
13
14 float QuadTree::SSIM(){
15     return 0.299*this->channelSSIM(0) + 0.587*this->channelSSIM(1) + 0.114*this->channelSSIM(2)
16 }
17
```

input_data.cpp

```
1  #include <iostream>
2  #include <string>
3  #include <regex>
4  #include <fstream>
5  using namespace std;
6
7  #include "../header/stb_image.h"
8  #include "../header/stb_image_write.h"
9  #include "../header/gif_generator.h"
10 #include "../header/input_data.hpp"
11
12 bool validateImageFile(const char* filename){
13     regex windows_absolute_path("^[A-Za-z]:[\\/.]*\\.\\.(jpg|jpeg|png)$");
14     regex linux_absolute_path("^[^\\./]*\\.\\.(jpg|jpeg|png)$");
15
16     if(!(regex_match(filename, windows_absolute_path) || regex_match(filename, linux_absolute_path))){
17         cout << "Format file gambar tidak valid!" << endl;
18         return false;
19     }
20     return true;
21 }
22
23 bool validateGIFFile(const char* filename){
24     regex windows_absolute_path("^[A-Za-z]:[\\/.]*\\.\\.(gif)$");
25     regex linux_absolute_path("^[^\\./]*\\.\\.(gif)$");
26
27     if(!(regex_match(filename, windows_absolute_path) || regex_match(filename, linux_absolute_path))){
28         cout << "Format file GIF tidak valid!" << endl;
29         return false;
30     }
31     return true;
32 }
33
34 bool inputPathExist(const char* filename){
35     if(!validateImageFile(filename)) return false;
36     int w, h, channels;
37     unsigned char* img = stbi_load(filename, &w, &h, &channels, 0);
38     if(img==nullptr){
39         cout << "File tidak ditemukan!" << endl;
40         return false;
41     }
42     return true;
43 }
```

```
45 bool validateErrorMethod(int emm){
46     if(emm<1 || emm>5){
47         cout << "Harap masukkan angka dalam rentang 1-5 untuk metode pengukuran error" << endl;
48         return false;
49     }
50     return true;
51 }
52
53 bool validateThreshold(int emm, float thresh, float pct){
54     if(pct>0 && pct<=1){
55         return true;
56     }
57
58     if(!validateErrorMethod) return false;
59     if(emm==1){
60         if(thresh>=0 && thresh<=16256.25){
61             return true;
62         }
63     }
64     else if(emm==2){
65         if(thresh>=0 && thresh<=127.5){
66             return true;
67         }
68     }
69     else if(emm==3){
70         if(thresh>=0 && thresh<=255){
71             return true;
72         }
73     }
74     else if(emm==4){
75         if(thresh>=0 && thresh<=8){
76             return true;
77         }
78     }
79     else if(emm==5){
80         if(thresh>=0 && thresh<=1){
81             return true;
82         }
83     }
84     else{
85         cout << "Nilai threshold tidak valid!" << endl;
86         return false;
87     }
88 }
89
90 bool validateMinBlockSize(const char* filename, int mbs, float pct){
91     if(pct>0 && pct<=1){
92         return true;
93     }
94
95     if(!inputPathExist(filename)) return false;
96     int w, h, channels;
97     unsigned char* img = stbi_load(filename, &w, &h, &channels, 0);
98
99     if(mbs>=1 && mbs <= w*h){
100         return true;
101     } else{
102         cout << "Nilai minimum block size tidak valid!" << endl;
103         return false;
104     }
105 }
```

```

107 bool validateCompressionPct(float pct){
108     if(pct>=0 && pct<=1){
109         return true;
110     } else{
111         cout << "Nilai target persentase kompresi tidak valid!" << endl;
112         return false;
113     }
114 }
115
116 bool validateOutputPath(const char* input_filename, const char* filename){
117     if(!inputPathExist(input_filename)) return false;
118     if(!validateImageFile(filename)) return false;
119
120     int w, h, channels;
121     unsigned char* img = stbi_load(input_filename, &w, &h, &channels, 0);
122
123     regex path_png(".*\\.png$");
124     regex path_jpg(".*\\.jpg$");
125     regex path_jpeg(".*\\.jpeg$");
126     if(
127         !(regex_match(filename, path_png) && regex_match(input_filename, path_png) ||
128           regex_match(filename, path_jpg) && regex_match(input_filename, path_jpg) ||
129           regex_match(filename, path_jpeg) && regex_match(input_filename, path_jpeg)))
130     {
131         cout << "Ekstensi input dan output berbeda" << endl;
132         return false;
133     }
134
135     // mencoba save dan load gambar ke path output
136     if(regex_match(filename, path_png)){
137         stbi_write_png(filename, w, h, channels, img, w*channels);
138     } else{
139         stbi_write_jpg(filename, w, h, channels, img, 75);
140     }
141
142     unsigned char* img2 = stbi_load(filename, &w, &h, &channels, 0);
143     if(img2==nullptr){
144         cout << "Path output tidak valid!" << endl;
145         return false;
146     }
147     return true;
148 }
149
150 bool validateGIFPath(const char* input_filename, const char* filename){
151     if(!inputPathExist(input_filename)) return false;
152     if(!validateGIFFile(filename)) return false;
153
154     return saveGIFTemplate(input_filename, filename);
155 }
156
157 bool InputData::validate(){
158     bool isValid = true;
159     cout << endl;
160     isValid = isValid & validateImageFile(img_input_path);
161     isValid = isValid & inputPathExist(img_input_path);
162     isValid = isValid & validateErrorMethod(error_measurement_method);
163     isValid = isValid & validateThreshold(error_measurement_method, threshold, compression_pct);
164     isValid = isValid & validateMinBlockSize(img_input_path, min_block_size, compression_pct);
165     isValid = isValid & validateCompressionPct(compression_pct);
166     isValid = isValid & validateOutputPath(img_input_path, img_output_path);
167     isValid = isValid & validateGIFPath(img_input_path, gif_path);
168     cout << endl;
169     return isValid;
170 }

```

```

1  #include <iostream>
2  #include <string>
3  #include <chrono>
4  #include <fstream>
5  using namespace std;
6  using namespace chrono;
7
8  #define STB_IMAGE_IMPLEMENTATION
9  #include "header/stb_image.h"
10 #define STB_IMAGE_WRITE_IMPLEMENTATION
11 #include "header/stb_image_write.h"
12
13 #include "header/quadtree.h"
14 #include "header/input_data.hpp"
15
16 int main(){
17     int w, h, channels;
18     float input_file_size;
19     float output_file_size;
20
21     unsigned char *img;
22     unsigned char *compressed_img;
23     bool run = true;
24     bool is_valid = true;
25     string cont;
26
27     cout << "===== " << endl;
28     cout << "Selamat Datang di Program Quadtree Image Compression! :D" << endl;
29     cout << "Disusun oleh: Muhammad Ghifary Komara Putra (13523066)" << endl;
30     cout << "===== " << endl << endl;
31
32     while(run){
33         cout << "[INPUT]" << endl;
34         InputData data;
35
36         cout << "Absolute Path to Image      : " << endl;
37         getline(cin, data.img_input_string);
38         data.img_input_path = data.img_input_string.c_str();
39
40         // cout << endl;
41         // cout << "===== " << endl;
42         // cout << "Error measurement method:" << endl;
43         // cout << "1. Variance" << endl;
44         // cout << "2. Mean Absolute Deviation (MAD)" << endl;
45         // cout << "3. Max Pixel Difference" << endl;
46         // cout << "4. Entropy" << endl;
47         // cout << "5. Structural simmilarity Index (SSIM)" << endl;
48         // cout << "===== " << endl << endl;
49
50         cout << "Error Measurement Method      : ";
51         cin >> data.error_measurement_method;
52         while(cin.fail()){
53             cin.clear();
54             cin.ignore(numeric_limits<streamsize>::max(), '\n');
55             cout << "Harap masukkan angka dalam rentang 1-5" << endl;
56
57             cout << "Error Measurement Method      : ";
58             cin >> data.error_measurement_method;
59         }

```

```

62 // cout << "===== " << endl;
63 // cout << "Threshold (x)" << endl;
64 // cout << "Variance : 0 <= x <= 16256.25" << endl;
65 // cout << "Mean Absolute Deviation (MAD) : 0 <= x <= 127.5" << endl;
66 // cout << "Max Pixel Difference : 0 <= x <= 255" << endl;
67 // cout << "Entropy : 0 <= x <= 8" << endl;
68 // cout << "Structural simmilarity Index (SSIM) : 0 <= x <= 1" << endl;
69 // cout << "===== " << endl << endl;
70 cout << "Threshold : ";
71 cin >> data.threshold;
72 while(cin.fail()){
73     cin.clear();
74     cin.ignore(numeric_limits<streamsize>::max(), '\n');
75     cout << "Harap masukkan angka dalam rentang yang valid" << endl;
76
77     cout << "Threshold : ";
78     cin >> data.threshold;
79 }
80
81 cout << "Minimum Block Size : ";
82 cin >> data.min_block_size;
83 while(cin.fail()){
84     cin.clear();
85     cin.ignore(numeric_limits<streamsize>::max(), '\n');
86     cout << "Harap masukkan angka dalam rentang yang valid" << endl;
87
88     cout << "Minimum Block Size : ";
89     cin >> data.min_block_size;
90 }
91
92 cout << "Compression Percentage Target : ";
93 cin >> data.compression_pct;
94 while(cin.fail()){
95     cin.clear();
96     cin.ignore(numeric_limits<streamsize>::max(), '\n');
97     cout << "Harap masukkan angka dalam rentang yang valid" << endl;
98
99     cout << "Compression Percentage Target : ";
100    cin >> data.compression_pct;
101 }
102
103 cout << "Absolute Path to Result : " << endl;
104 cin.ignore();
105 getline(cin, data.img_output_string);
106 data.img_output_path = data.img_output_string.c_str();
107
108 cout << "Absolute Path to GIF : " << endl;
109 getline(cin, data.gif_string);
110 data.gif_path = data.gif_string.c_str();
111
112 is_valid = data.validate();

```

```

113     if(is_valid){
114         cout << "Generating image ..." << endl;
115         // Compression
116         img = stbi_load(data.img_input_path, &w, &h, &channels, 0);
117         compressed_img = stbi_load(data.img_input_path, &w, &h, &channels, 0);
118
119         // Catatan: faktor koreksi terhadap depth diberikan saat pencetakan output
120         QuadTree qt = QuadTree(1,0,0,w,h);
121         qt.setStatic(img, compressed_img, data.error_measurement_method, data.threshold, data.min_block_size, channels);
122
123         auto start = high_resolution_clock::now();
124         if(data.compression_pct == 0){
125             qt.compressImage();
126         } else{
127             qt.compressImageByFileSize(data.compression_pct, data.error_measurement_method, data.img_input_path, data.img_output_path)
128         }
129         auto end = high_resolution_clock::now();
130         auto exec_time = duration_cast<milliseconds>(end-start);
131
132         cout << "Saving image ..." << endl;
133         if(data.compression_pct == 0){
134             qt.saveCompressedImage(data.img_output_path);
135         }
136         cout << "Saving GIF ..." << endl;
137         qt.generateGIF(data.img_output_path, data.gif_path);
138
139         ifstream file(data.img_input_path, ios::binary | ios::ate);
140         streamsize size = file.tellg();
141         input_file_size = static_cast<float>(size / 1024);
142
143         ifstream file2(data.img_output_path, ios::binary | ios::ate);
144         size = file2.tellg();
145         output_file_size = static_cast<float>(size / 1024);
146
147         if(data.compression_pct == 0){
148             qt.final_compression_pct = (input_file_size - output_file_size)/input_file_size;
149         }
150
151         cout << endl;
152         cout << "[OUTPUT]" << endl;
153         cout << "Execution Time (ms)   : " << exec_time.count() << endl;
154         cout << "Input File Size       : " << input_file_size << endl;
155         cout << "Output File Size      : " << output_file_size << endl;
156         cout << "Compression Percentage: " << qt.final_compression_pct*100 << "%" << endl;
157         cout << "Tree Depth            : " << qt.getMaxDepth() - 1 << endl;
158         cout << "Node Count            : " << qt.getNodeCount() << endl;
159
160         cout << endl << "Continue?" << endl;
161         cout << "(Input Y to continue, anything else to quit)" << endl;
162         getline(cin, cont);
163         if(cont != "Y"){
164             run = false;
165         }
166     }
167 }
168
169 cout << endl;
170 cout << "Sampai jumpa di lain waktu! :D" << endl;
171 return 0;
172 }

```

VI. HASIL DAN ANALISIS

A. Analisis Kompleksitas Algoritma

Pada kasus terbaik, program akan berhenti pada node pertama (depth paling rendah). Jika hal ini terjadi karena ukuran blok minimum terlewati, kompleksitas algoritma adalah $O(1)$. Jika hal ini terjadi karena nilai *threshold* terlewati, kompleksitas algoritma adalah $O(n)$, berasal dari kompleksitas waktu penghitungan error (sama untuk setiap metode). Lalu, Secara umum, kompleksitas waktu algoritma *divide and conquer* yang telah dikembangkan dapat dinotasikan sebagai berikut:

$$T(n) = \begin{cases} x, n \leq \text{min_block_size} \vee \text{threshold telah dilewati} \vee \text{child node tidak ditinjau} \\ 4T\left(\frac{n}{4}\right) + cn, & \text{lainnya} \end{cases}$$

Pada persamaan di atas, n menyatakan banyak *pixel* yang ditinjau oleh suatu node, diperoleh dari hasil perkalian lebar dan tinggi bagian gambar yang sedang ditinjau. Variabel x dapat bernilai 1 untuk mengecek apakah ukuran blok telah melewati minimum, atau 0 pada kasus khusus seperti *child node* yang tidak ditinjau ketika quadtree hanya terbagi 2. Koefisien 4 berasal dari banyak node anak yang diciptakan, dan bahwa keempat node anak tersebut akan ditinjau, menghasilkan fungsi yang rekursif. Koefisien $1/4$ menandakan bahwa banyak pixel yang ditinjau adalah seperempat dari tahap sebelumnya. Terakhir, variabel c menandakan perhitungan-perhitungan lainnya bergantung dari faktor apa saja yang hendak diperhitungkan. hal ini dapat mencakup: perhitungan error, perhitungan rerata setiap *channel* warna, serta pengisian blok gambar dengan hasil rerata.

Perhatikan bahwa kompleksitas waktu dengan notasi big-O dapat didapatkan dari persamaan di atas dengan memanfaatkan Teorema Master sebagai berikut:

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

memenuhi bentuk:

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

dengan

$$a = 4, b = 4, d = 1$$

Nilai tersebut memenuhi kasus kedua Teorema Master, yaitu:

$$a = b^d$$

Sehingga diperoleh kompleksitas waktu algoritma tersebut dalam notasi big-O adalah $T(n) = O(n \log n)$.

B. Analisis Rentang Nilai Ambang Batas

Bagian ini akan memaparkan rentang nilai yang valid untuk metode pengukuran error, kecuali SSIM (tertera pada bab VII). Rentang nilai ini digunakan untuk keperluan validasi program, memastikan pengguna memasukkan nilai ambang batas yang benar, sesuai dengan metode yang dipilih.

B.1 Variansi

Rentang nilai yang valid untuk variansi adalah $[0, 16256.25]$. Nilai minimum dapat diperoleh pada gambar monoton satu warna sedangkan nilai maksimum dapat diperoleh pada gambar dengan 2 warna, dengan setengah dari nilai piksel pada gambar tersebut adalah 0 dan setengah lainnya adalah 255. Atau secara matematis:

- Nilai minimum

$$\mu = x_i$$

x_i menyatakan nilai pixel gambar monoton tersebut

$$\sigma^2 = \sum \frac{(P_{i,c} - \mu)^2}{n} = \sum \frac{(x_i - x_i)^2}{n} = 0$$

- Nilai maksimum

$$\mu = (0.5)(0) + (0.5)(255) = 127.5$$

$$\sigma^2 = \sum \frac{(P_{i,c} - \mu)^2}{n} = (0.5)(0 - 127.5)^2 + (0.5)(255 - 127.5)^2 = 127.5^2 = 16256.25$$

B.2 Mean Absolute Deviation (MAD)

Rentang nilai yang valid untuk MAD adalah [0, 127.5]. Serupa dengan variansi, nilai minimum dapat diperoleh pada gambar monoton satu warna sedangkan nilai maksimum dapat diperoleh pada gambar dengan 2 warna, dengan setengah dari nilai piksel pada gambar tersebut adalah 0 dan setengah lainnya adalah 255. Secara matematis hal tersebut dapat dinotasikan:

- Nilai minimum

$$\mu = x_i$$

x_i menyatakan nilai pixel gambar monoton tersebut

$$MAD = \sum \frac{|P_{i,c} - \mu|}{n} = \sum \frac{|x_i - x_i|}{n} = 0$$

- Nilai maksimum

$$\mu = (0.5)(0) + (0.5)(255) = 127.5$$

$$MAD = \sum \frac{|P_{i,c} - \mu|}{n} = (0.5) \times |0 - 127.5| + (0.5) \times |255 - 127.5| = 127.5$$

B.3 Max Pixel Difference (MPD)

Rentang nilai yang valid untuk MPD adalah [0, 255]. Berbeda dengan dua metode sebelumnya, nilai minimum dapat diperoleh pada gambar monoton satu warna sedangkan nilai maksimum dapat diperoleh pada gambar dengan yang memiliki pixel dengan nilai 0 dan pixel dengan nilai 255. Secara matematis hal tersebut dapat dinotasikan:

- Nilai minimum

$$\max(P_{i,c}) = \min(P_{i,c}) = x_i$$

x_i menyatakan nilai pixel gambar monoton tersebut

$$MAD = \max(P_{i,c}) - \min(P_{i,c}) = x_i - x_i = 0$$

- Nilai maksimum

$$MAD = \max(P_{i,c}) - \min(P_{i,c}) = 255 - 0 = 255$$

B.4 Entropi

Rentang nilai yang valid untuk MPD adalah [0, 8]. Nilai minimum dapat diperoleh pada gambar monoton satu warna, sedangkan nilai maksimum dapat diperoleh pada gambar dengan distribusi kemunculan yang sama untuk setiap nilai pixel yang memungkinkan (0-255). Secara matematis hal tersebut dapat dinotasikan (catatan, seluruh operasi logaritma memiliki basis 2, bukan 10):

- Nilai minimum

$$H_c = - \sum P_c(i) \log(P_c(i)) = - (1 \times \log(1) + 255 \times 0 \times \log(0)) = 0$$

- Nilai maksimum

Pada kasus ini, setiap untuk setiap i berlaku $p_i = \frac{1}{256}$, sehingga diperoleh

$$H_c = - \sum P_c(i) \log(P_c(i)) = - \left(256 \times \frac{1}{256} \times \log\left(\frac{1}{256}\right) \right) = - \left(\log\left(\frac{1}{256}\right) \right) = 8$$

VII. IMPLEMENTASI BONUS

A. Target Kompresi

Pada bonus ini, pengguna dapat memberikan masukan target persentase kompresi yang diinginkan (berada dalam rentang nilai $[0,1]$), dan program akan menyesuaikan nilai *minimum block size* dan *threshold* untuk mencapai tujuan tersebut sebaik mungkin. Pada program ini, penulis memutuskan untuk mengimplementasikan hal tersebut dengan memanfaatkan algoritma *binary search* terhadap nilai *threshold*, sedangkan nilai *minimum block size* diubah menjadi 1. Detail algoritma *binary search* yang diimplementasikan adalah sebagai berikut:

1. Atur nilai *minimum block size* menjadi 1
2. Atur nilai *threshold* menjadi nilai rata-rata dari nilai minimum dan maksimum rentang nilai yang ditinjau. Pada iterasi pertama, sesuaikan dengan metode pengukuran error yang dipilih (misal, jika memilih entropi yang memiliki rentang nilai $[0, 8]$, $threshold = (0+8)/2=4.5$)
3. Lakukan kompresi gambar dengan *threshold* tersebut, kemudian simpan gambar hasil kompresi
4. Hitung persentase kompresi yang diperoleh, bandingkan dengan target.
5. Jika persentase melebihi target, perbarui nilai rentang yang ditinjau menjadi $[min, threshold]$. Namun, jika persentase lebih rendah dari target, perbarui nilai rentang yang ditinjau menjadi $[threshold, max]$. (Logika ini dibalik untuk metode SSIM)
6. Ulangi langkah 2-5 hingga selisih persentase lebih kecil dari 0.01 atau sudah dilakukan 20 kali iterasi

Perhatikan bahwa selisih persentase dan jumlah iterasi merupakan murni pilihan pribadi penulis. Selisih nilai tersebut dirasa merupakan pembulatan yang cukup dan jumlah iterasi dirasa cukup untuk menghasilkan nilai yang cukup sesuai tanpa menghabiskan waktu terlalu lama untuk melakukan kompresi. Selain itu, Penulis melakukan perubahan hanya terhadap *threshold* untuk menjaga esensi dari kompresi quadtree, yaitu membagi blok ketika melewati ambang batas tertentu. Jika perubahan dilakukan hanya terhadap *minimum block size*, blok akan terus terbagi ke dalam ukuran yang sama, tidak peduli apakah setiap pixel dalam blok gambar sudah cukup seragam atau belum. Di sisi lain, memperhitungkan kedua faktor tersebut dapat meningkatkan waktu kalkulasi secara cukup signifikan dengan perbedaan hasil yang tidak sepadan.

B. GIF Proses Kompresi

Pada bonus ini, pengguna dapat melihat proses pembentukan gambar hasil kompresi pada setiap *depth* dari *quadtree* dalam bentuk file GIF. Implementasi pembuatan GIF adalah sebagai berikut:

1. Lakukan kompresi terhadap gambar
2. Simpan gambar hasil kompresi
3. Load gambar hasil kompresi sebanyak *depth* dari *quadtree*. Setiap gambar ini akan mewakili satu frame dari GIF
4. Traversal quadtree secara *depth-first*, dimulai dari *parent node* (*depth* pertama).
5. Untuk setiap *node*, hitung rata-rata nilai pixel setiap channel r, g, b pada blok gambar yang ditinjau oleh *node* tersebut, kemudian ubah warna blok gambar *frame* dengan hasil rata-rata tersebut, pada bagian yang ditinjau oleh *node*. Untuk *node* pada *depth* ke- i , ubah warna *frame* ke- i hingga *frame* terakhir.
6. Ulangi langkah 5 untuk setiap *child* dari *node* tersebut jika *node* yang sedang ditinjau bukanlah *node* daun
7. Gabungkan setiap frame berurut menjadi suatu GIF kemudian simpan

Perhatikan bahwa langkah 5 dan 6 akan membuat frame seolah-olah “ditimpa” saat pewarnaan. Contohnya, *frame* terakhir akan diubah warnanya oleh *node* pada *depth* ke-1 hingga terakhir. Hal ini menghabiskan jauh lebih banyak waktu dari seharusnya, namun membuat program menjadi *robust* dan tidak rentan terkena *bug* atau *error* saat *library* GIF yang digunakan berperilaku tidak dapat diprediksi.

C. Structural Similarity Index (SSIM)

Pada bonus ini, pengguna dapat memilih satu metode penghitungan error tambahan, yaitu SSIM. Formula dari SSIM suatu channel warna c adalah sebagai berikut:

$$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}; C_1, C_2 > 0$$

x menyatakan blok tertentu pada gambar masukan pengguna sedangkan y menyatakan blok gambar tersebut jika dikompresi. Perhatikan bahwa warna blok gambar hasil kompresi adalah rata-rata dari blok yang sama pada gambar masukan pengguna. Artinya, $\mu_{x,c} = \mu_{y,c}$. Selain itu, karena y merupakan gambar monoton 1 warna, variansi dan kovariansi akan bernilai 0. Sehingga, formula SSIM dapat disederhanakan menjadi:

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

Berdasarkan referensi *paper* SSIM, penulis memilih nilai $C_2 = (0.03 \times 255)^2 = 58.5225$

Lalu, hasil SSIM keseluruhan adalah sebagai berikut

$$SSIM_{RGB} = w_R \times SSIM_R + w_G \times SSIM_G + w_B \times SSIM_B$$
$$SSIM_{RGB} = 0.299 \times SSIM_R + 0.587 \times SSIM_G + 0.114 \times SSIM_B$$

SSIM sebagai metode pengukuran error memiliki rentang nilai $[0,1]$. Nilai minimum diperoleh pada variansi maksimum, sedangkan nilai maksimum diperoleh pada variansi minimum untuk setiap *channel* warna.

- Nilai minimum

$$SSIM_c(x, y) = \frac{C_2}{\sigma_{x,c}^2 + C_2} = \frac{C_2}{16256.25 + C_2} \approx 0$$

- Nilai maksimum

$$SSIM_c(x, y) = \frac{C_2}{\sigma_{x,c}^2 + C_2} = \frac{C_2}{0 + C_2} = 1$$

Selain itu, SSIM memiliki perilaku yang unik. Pada metode pengukuran error lainnya, semakin kecil error, semakin baik hasil yang diperoleh. Namun, karena SSIM merupakan indeks kesamaan struktur, nilai SSIM yang baik adalah nilai SSIM setinggi mungkin. Artinya, pada proses kompresi, jika metode ini dipilih, pembagian blok berhenti jika *threshold* melebihi nilai ambang batas SSIM yang dipilih. Selain itu, logika *binary search* pada bonus target kompresi pun menjadi terbalik pada kasus ini.

LAMPIRAN

Tautan menuju repositori: https://github.com/Sanesasaha/Tucil2_13523066

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil dijalankan	✓	
3. Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	✓	

4. Mengimplementasi seluruh metode perhitungan error wajib	✓	
5. [Bonus] Implementasi persentase kompresi sebagai parameter tambahan	✓	
6. [Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	✓	
7. [Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar	✓	
8. Program dan laporan dibuat sendiri	✓	