Laporan Tugas Kecil 2
IF2211 Strategi Algoritma

Kompresi Gambar dengan Metode Quadtree

Disusun oleh:

Refki Alfarizi (13523002)

Program Studi Teknik Informatika

Sekolah Teknik Elektro Dan Informatika

Institut Teknologi Bandung

11 April 2025

Daftar Isi

1 Deskripsi Tugas **1**

2 Pendekatan Divide and Conquer dalam Pembuatan Quadtree **3**
2.1 Metode Pengukuran Error . . . 4
2.1.1 Variance . . . 4
2.1.2 Mean Absolute Deviation (MAD) . . . 4
2.1.3 Max Pixel Difference . . . 5
2.1.4 Entropy . . . 5
2.1.5 Structural Similarity Index (SSIM) . . . 6
2.2 Threshold pada Metode Pengukuran Error . . . 6
2.2.1 Variance . . . 6
2.2.2 Mean Absolute Deviation (MAD) . . . 7
2.2.3 Max Pixel Difference . . . 7
2.2.4 Entropy . . . 8
2.2.5 Structural Similarity Index (SSIM) . . . 8
2.3 Algoritma Divide and Conquer pada Quadtree . . . 9
2.3.1 Pembuatan Quadtree . . . 9
2.3.2 Penerapan Quadtree menjadi Gambar . . . 10
2.3.3 Penerapan Quadtree menjadi Image Sequence (Animasi GIF) . . . 11
2.4 Optimasi dengan Summed Area Table . . . 12

3 Kode Sumber **15**
3.1 Quadtree . . . 15
3.1.1 `quadtreenode.h` . . . 15
3.1.2 `quadtreenode.cpp` . . . 16
3.1.3 `quadtreeimage.h` . . . 16
3.1.4 `quadtreeimage.cpp` . . . 17
3.2 Error Measurement . . . 21
3.2.1 `error_method.h` . . . 21
3.2.2 `emm_variance.h` . . . 21
3.2.3 `emm_mad.h` . . . 22
3.2.4 `emm_mpd.h` . . . 24
3.2.5 `emm_entropy.h` . . . 25
3.2.6 `emm_ssim.h` . . . 27
3.3 Image . . . 28
3.3.1 `image.h` . . . 28
3.3.2 `image.cpp` . . . 30

3.3.3 image_sequence.h . . . 37
3.3.4 image_sequence.cpp . . . 38
3.4 Controller . . . 39
3.4.1 compression_controller.h . . . 39
3.4.2 compression_controller.cpp . . . 41
3.5 Utils . . . 45
3.5.1 debug.h . . . 45
3.5.2 style.h . . . 46
3.6 CLI . . . 48
3.6.1 cli.h . . . 48
3.6.2 cli.cpp . . . 51
3.7 Main . . . 72
3.7.1 main.cpp . . . 72

4 Uji Kasus **73**
4.1 Pengujian Input . . . 73
4.1.1 Pengujian Path Absolute Linux . . . 73
4.1.2 Pengujian Path Absolute Windows . . . 73
4.1.3 Pengujian Path Relative . . . 74
4.1.4 Pengujian Path Tidak Ada atau Salah . . . 74
4.1.5 Pengujian Threshold di Luar Batas . . . 74
4.1.6 Pengujian Minimum Block Size di Luar Batas . . . 75
4.1.7 Pengujian Compression Target di Luar Batas . . . 75
4.1.8 Pengujian Output File Berbeda Ekstensi . . . 75
4.1.9 Pengujian Output File GIF Salah Ekstensi . . . 76
4.2 Pengujian Metode Eror . . . 76
4.2.1 Metode Variansi . . . 76
4.2.2 Metode Mean Absolute Deviation . . . 77
4.2.3 Metode Max Pixel Difference . . . 78
4.2.4 Metode Entropy . . . 78
4.2.5 Metode SSIM . . . 79
4.3 Pengujian Target Kompresi . . . 80
4.3.1 Pengujian Target Kompresi . . . 80

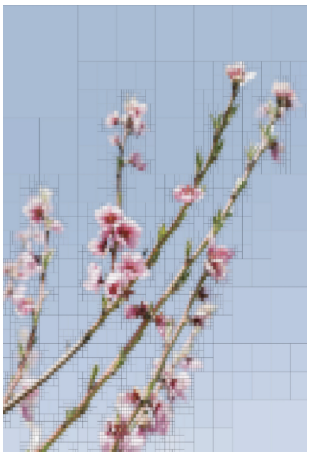
5 Analisis Hasil **81**
5.1 Analisis Kompleksitas . . . 81
5.1.1 Analisis Kompleksitas Waktu . . . 81
5.1.2 Kompleksitas Ruang . . . 82

Daftar Pustaka **84**

Lampiran **85**
Pranala Repository GitHub . . . 85
Tabel Pengerjaan . . . 85

Bab 1

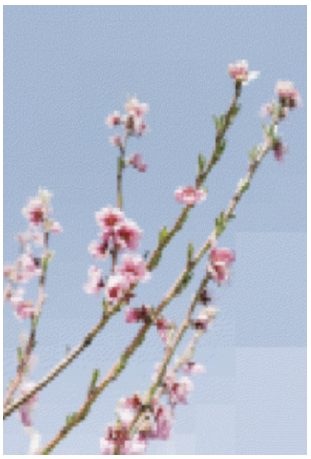
Deskripsi Tugas



Gambar 1.1. Quadtree dalam Kompresi Gambar
(Sumber: https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00)

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

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

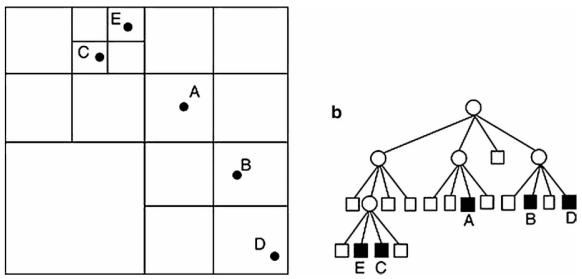


Gambar 1.2. Proses Pembentukan Quadtree dalam Kompresi Gambar
(Sumber:
https://miro.medium.com/v2/resize:fit:640/format:webp/1*LHD7PsbmbgNBFrYkxyG5dA.gif)

Catatan: Cek sumber untuk melihat animasi GIF pada Gambar 2.

Bab 2

Pendekatan Divide and Conquer dalam Pembuatan Quadtree



Gambar 2.1. Struktur Data Quadtree dalam Kompresi Gambar
(Sumber: https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00)

Permasalahan kompresi gambar dengan metode Quadtree yang telah dijelaskan pada Bab 1 dapat diselesaikan dengan pendekatan algoritma *divide and conquer*. Pendekatan ini membagi gambar menjadi beberapa blok untuk diproses menjadi Quadtree. Beberapa parameter yang digunakan dalam proses ini antara lain:

- **Metode Pengukuran Error:** Metode yang digunakan untuk menentukan seberapa besar perbedaan dalam satu blok gambar. Jika nilai error dalam blok melebihi ambang batas (*threshold*), maka blok tersebut akan dibagi menjadi empat bagian yang lebih kecil.
- **Threshold (Ambang Batas):** Nilai batas yang menentukan apakah sebuah blok dianggap cukup seragam untuk disimpan atau harus dibagi lebih lanjut.
- **Ukuran Blok Minimum:** Merupakan ukuran terkecil dari sebuah blok (dalam piksel) yang diizinkan dalam proses kompresi. Jika ukuran blok yang hendak dibagi berada di bawah nilai minimum yang telah ditentukan, maka pembagian tidak dilakukan, meskipun nilai error masih di atas threshold.
- **Persentase Kompresi:** Parameter yang menunjukkan seberapa besar ukuran gambar telah berkurang setelah dikompresi. Perhitungannya adalah:

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

2.1 Metode Pengukuran Error

Berikut adalah metode-metode yang digunakan sebagai metode pengukuran error untuk pendekatan algoritma *divide and conquer* yang digunakan untuk membuat Quadtree.

2.1.1 Variance

Variansi mengukur sebaran nilai piksel di dalam satu blok gambar. Rumus variansi pada satu channel adalah:

$$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^{N} (P_{i,c} - \bar{c})^2$$

Untuk gambar dengan tiga channel (RGB), variansi gabungan dihitung sebagai:

$$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$$

Keterangan:

- σ_c^2 = Variansi kanal warna c (R, G, atau B)
- $P_{i,c}$ = Nilai piksel ke-i pada kanal c
- $\bar{c}$ = Rata-rata piksel pada kanal c
- N = Jumlah piksel dalam satu blok

2.1.2 Mean Absolute Deviation (MAD)

MAD mengukur nilai rata-rata deviasi absolut antara nilai piksel dan rata-rata blok, Rumus MAD pada satu channel adalah:

$$MAD_c = \frac{1}{N} \sum_{i=1}^{N} |P_{i,c} - \bar{c}|$$

Untuk gambar dengan tiga channel (RGB), nilai MAD gabungan dihitung sebagai:

$$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$$

Keterangan:

- MAD_c = Mean Absolute Deviation pada kanal c
- $P_{i,c}$ = Nilai piksel ke-i pada kanal c
- $\bar{c}$ = Rata-rata piksel pada kanal c

- N = Jumlah piksel dalam satu blok

2.1.3 Max Pixel Difference

Max Pixel Difference mengukur perbedaan antara nilai piksel maksimum dan minimum dalam suatu blok. Rumus Max Pixel Difference pada satu channel adalah:

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

Untuk gambar dengan tiga channel (RGB), nilai Max Pixel Difference gabungan dihitung sebagai:

$$D_{RGB} = \frac{D_R + D_G + D_B}{3}$$

Keterangan:

- D_c = Selisih nilai piksel maksimum dan minimum pada kanal c
- $P_{i,c}$ = Nilai piksel ke-i pada kanal c

2.1.4 Entropy

Entropi mengukur kompleksitas atau ketidakpastian distribusi nilai piksel dalam blok. Rumus Entropy pada satu channel adalah:

$$H_c = -\sum_{i=1}^{N} P_c(i) \log_2\Big(P_c(i)\Big)$$

Untuk gambar dengan tiga channel (RGB), nilai entropi gabungan dihitung sebagai:

$$H_{RGB} = \frac{H_R + H_G + H_B}{3}$$

Keterangan:

- H_c = Entropi kanal warna c
- $P_c(i)$ = Probabilitas kemunculan piksel nilai ke-i pada kanal c
- N = Jumlah total piksel atau kombinasi nilai piksel dalam satu blok

2.1.5 Structural Similarity Index (SSIM)

SSIM mengukur perbandingan struktur visual antara blok gambar sebelum dan sesudah proses kompresi. Rumus SSIM pada satu channel adalah:

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

Untuk gambar dengan tiga channel (RGB), nilai SSIM gabungan dihitung sebagai:

$$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$$

Keterangan:

- $SSIM_c(x, y)$ = Indeks SSIM untuk kanal c, dibandingkan antara blok sebelum dan sesudah kompresi
- $\bar{x}_c, \bar{y}_c$ = Rata-rata piksel pada blok x dan y di kanal c
- $\sigma_{xy,c}$ = Kovariansi piksel kanal c antara blok x dan y
- $\sigma_{x,c}^2, \sigma_{y,c}^2$ = Variansi piksel untuk blok x dan y pada kanal c
- C_1, C_2 = Konstanta untuk menghindari pembagian dengan nol

2.2 Threshold pada Metode Pengukuran Error

Dalam proses evaluasi blok pada kompresi gambar, threshold atau ambang batas digunakan untuk menentukan apakah blok gambar dianggap seragam atau perlu dibagi lebih lanjut. Nilai threshold ditentukan berdasarkan nilai maksimum (atau minimum) yang mungkin diperoleh oleh metode pengukuran error pada citra 8-bit. Berikut adalah derivasi dan range threshold untuk tiap metode:

2.2.1 Variance

Diketahui bahwa untuk sebuah channel (misalnya R, G, atau B) nilai piksel berkisar antara 0 hingga 255. Nilai *maksimum* perbedaan pada satu piksel adalah 127.5 yaitu pada kasus ekstrem gambar hanya berwarna hitam dan putih yang memungkinkan rata rata menjadi 127.5 dan nilai pada suatu pixel antara 0 atau 255, sehingga jika dipertimbangkan kuadrat perbedaan:

$$\sigma_c^2 = \frac{1}{N}\sum_{i=1}^{N}(P_{i,c} - \bar{c})^2 \quad \Rightarrow \quad T_{\sigma^2}^{\max} = 127.5^2 = 16256.25.$$

Range Threshold: $\sigma_c^2 \in [0, 16256.25]$.

Keterangan:

- $P_{i,c}$ = Nilai piksel ke-i pada channel c
- $\bar{c}$ = Rata-rata piksel channel c
- N = Jumlah piksel dalam blok

2.2.2 Mean Absolute Deviation (MAD)

Untuk MAD, perhitungan nilai mutlak perbedaan adalah:

$$MAD_c = \frac{1}{N} \sum_{i=1}^{N} |P_{i,c} - \bar{c}| \,.$$

Nilai *maksimum* terjadi saat perbedaan absolut mencapai 127.5 untuk semua piksel (kasus ekstrem yang sama seperti pada metode variance), sehingga:

$$T_{MAD}^{\max} = 127.5.$$

Range Threshold: $MAD_c \in [0, 127.5]$.

Keterangan:

- $P_{i,c}$, $\bar{c}$, N seperti sebelumnya.

2.2.3 Max Pixel Difference

Metode ini menghitung selisih antara nilai piksel maksimum dan minimum dalam sebuah blok:

$$D_c = \max(P_{i,c}) - \min(P_{i,c}).$$

Nilai maksimum perbedaan adalah 255 (misal, apabila $\min(P_{i,c}) = 0$ dan $\max(P_{i,c}) = 255$):

$$T_D^{\max} = 255.$$

Range Threshold: $D_c \in [0, 255]$.

Keterangan:

- $\max(P_{i,c})$ dan $\min(P_{i,c})$ adalah nilai ekstrim pada blok.

2.2.4 Entropy

Entropi untuk satu channel didefinisikan sebagai:

$$H_c = -\sum_{i=1}^{N} P_c(i) \log_2\Big(P_c(i)\Big).$$

Dalam citra 8-bit, nilai maksimum entropi terjadi jika semua 256 tingkat keabuan muncul dengan probabilitas yang sama, sehingga:

$$T_H^{\max} = \log_2(256) = 8.$$

Range Threshold: $H_c \in [0, 8]$.

Keterangan:

- $P_c(i)$ = Probabilitas kemunculan nilai piksel tertentu pada channel c.

2.2.5 Structural Similarity Index (SSIM)

SSIM mengukur kesamaan struktural antara dua blok gambar. Salah satu pendekatan untuk menentukan threshold SSIM adalah dengan membandingkan blok asli x dengan versi blok yang diwakili oleh nilai rata-ratanya $\bar{x}$. Mengingat blok rata-rata memiliki variansi nol ($\sigma_{\bar{x}}^2 = 0$), kita dapat menyederhanakan rumus SSIM untuk satu channel sebagai berikut:

$$SSIM(x, \bar{x}) = \frac{(2\bar{x}^2 + C_1)\, C_2}{(2\bar{x}^2 + C_1)(\sigma_x^2 + C_2)} = \frac{C_2}{\sigma_x^2 + C_2}.$$

Jika diasumsikan nilai variansi σ_x^2 bervariasi dari 0 hingga 255^2, maka:

- Saat $\sigma_x^2 = 0$ (blok seragam), $SSIM(x, \bar{x}) = 1$.
- Saat $\sigma_x^2 = 127.5^2$, nilai SSIM minimum adalah:

$$SSIM_{\min} = \frac{C_2}{127.5^2 + C_2}.$$

Range Threshold: $SSIM_c \in [0, 1]$.

Keterangan:

- C_2 = Konstanta kecil untuk menghindari pembagian dengan nol (biasanya 58.5225).
- σ_x^2 = Variansi blok x.
- Range ini menyiratkan bahwa nilai SSIM antara blok yang sangat berbeda (maksimum variansi) dan blok seragam adalah antara $\frac{C_2}{127.5^2 + C_2}$ hingga 1.

2.3 Algoritma Divide and Conquer pada Quadtree

Algoritma Divide and Conquer pada Quadtree menggunakan pendekatan *Breadth-First Search* (BFS) untuk memproses setiap level dari quadtree. Pada proses pembuatan Quadtree, setiap node pada gambar dievaluasi nilai error-nya (sesuai metode error yang dipilih) dan akan dibagi jika dan hanya jika error melebihi threshold (pengecualian untuk SSIM) dan ukuran node memenuhi syarat. Sedangkan pada proses mengaplikasikan quadtree, tiap node leaf (tidak ter-divide) diwarnai dengan nilai rata-rata pada region-nya sehingga dihasilkan gambar terkompresi. Selain itu, terdapat pula proses untuk menghasilkan deretan gambar (*image sequence*) yang memperlihatkan transformasi quadtree per level sebagai animasi GIF.

2.3.1 Pembuatan Quadtree

```
function buildquadtree(image, threshold, minblocksize):
  // buat node akar yang mencakup seluruh image
  rootnode <- create_node(0, 0, image.width, image.height)
  queue <- empty_queue()
  push(queue, rootnode)
  nodecount <- 1
  depth <- 0

  while queue is not empty do:
    nodes_current_level <- size(queue)
    for i from 1 to nodes_current_level do:
      currentnode <- pop(queue)
      error_value <- evaluate_error(image, currentnode.region)

      if error_value > threshold and (currentnode.area / 4) >=
   minblocksize then:
        if not currentnode.is_divided then:
          currentnode.divide()
        for each child in currentnode.children do:
          if child exists then:
            nodecount <- nodecount + 1
            push(queue, child)
    depth <- depth + 1

  -> rootnode
```

Program 2.1. Pseudocode pembuatan quadtree

Penjelasan Langkah-langkah:

- **Inisialisasi:**
  - Buat node akar (`rootnode`) yang mencakup keseluruhan gambar berdasarkan koordinat (0,0) dengan lebar dan tinggi dari gambar.
  - Inisialisasi sebuah antrian (`queue`) dan masukkan node akar ke dalamnya.

- Set nilai `nodecount` dan `depth` untuk melacak jumlah node dan kedalaman quadtree.

- **Proses Level-by-Level:**
  - Selama antrian tidak kosong, hitung jumlah node di level saat ini.
  - Untuk setiap node di level tersebut, lakukan:
    * Ambil node dari antrian dan evaluasi error-nya menggunakan metode pengukuran error pada region node.
    * Jika error melebihi `threshold` dan area node cukup besar (setidaknya empat kali `minblocksize`), maka:
      · Periksa apakah node belum dibagi, kemudian bagi node tersebut menjadi empat subblok.
      · Masukkan tiap anak node (child) yang ada ke dalam antrian.
  - Setelah semua node pada level saat ini diproses, tingkatkan nilai `depth`.
- **Akhir Proses:**
  - Setelah antrian kosong, struktur quadtree telah lengkap.
  - Kembalikan node akar (`rootnode`) sebagai representasi dari quadtree.

2.3.2 Penerapan Quadtree menjadi Gambar

```
function applyquadtree(quadtree, image):
  resultimage <- copy(image)
  queue <- empty_queue()
  push(queue, quadtree.root)

  while queue is not empty do:
    currentnode <- pop(queue)
    if currentnode is not divided then:
      avgcolor <- compute_average_color(image, currentnode.region)
      set_block_color(resultimage, currentnode.region, avgcolor)
    else:
      for each child in currentnode.children do:
        if child exists then:
          push(queue, child)

  if resultimage.channels == 4 then:
    for each pixel in resultimage do:
      set_alpha(resultimage, pixel, image.alpha(pixel))

  -> resultimage
```

Program 2.2. Pseudocode penerapan quadtree

Penjelasan Langkah-langkah:

- **Inisialisasi:**
  - Salin gambar asli ke dalam `resultimage` sebagai dasar gambar terkompresi.
  - Inisialisasi antrian dan masukkan node akar dari quadtree.
- **Traversing Quadtree:**
  - Selama antrian tidak kosong, lakukan:
    - Ambil node dari antrian.
    - Jika node tersebut bukan merupakan node yang telah dibagi (leaf), maka:
      - Hitung warna rata-rata dari blok yang diwakili oleh node.
      - Terapkan warna tersebut ke seluruh area blok dalam `resultimage`.
    - Jika node telah dibagi, masukkan seluruh anak (child) yang ada ke dalam antrian untuk diproses.
- **Penyesuaian untuk Alpha Channel:**
  - Jika gambar memiliki 4 channel (misalnya, PNG dengan alpha), lakukan penyalinan nilai alpha dari gambar asli ke `resultimage` untuk setiap piksel.
- **Akhir Proses:**
  - Kembalikan `resultimage` yang telah diterapkan kompresi berdasarkan quadtree.

2.3.3 Penerapan Quadtree menjadi Image Sequence (Animasi GIF)

```
function applyanimation(quadtree, image, delay):
  resultsequence <- empty_sequence()
  tempimage <- copy(image)
  queue <- empty_queue()
  push(queue, quadtree.root)

  while queue is not empty do:
    nodes_current_level <- size(queue)
    for i from 1 to nodes_current_level do:
      currentnode <- pop(queue)
      area <- currentnode.width * currentnode.height
      x <- currentnode.posx
      y <- currentnode.posy
      w <- currentnode.width
      h <- currentnode.height
      avgcolor <- compute_average_color(image, currentnode.region)

      // Terapkan warna rata-rata ke blok di tempimage
      set_block_color(tempimage, currentnode.region, avgcolor)

      for each child in currentnode.children do:
        if child exists then:
```

```
23            push(queue, child)
24      add_image(resultsequence, tempimage, delay)
25
26    -> resultsequence
```

Program 2.3. Pseudocode penerapan quadtree pada image sequence

Penjelasan Langkah-langkah:

- **Inisialisasi:**
  - Salin image asli ke `tempimage` yang nantinya akan dimodifikasi per level.
  - Buat objek `resultsequence` untuk menyimpan deretan gambar dan inisialisasi antrian dengan memasukkan `root` node dari quadtree.
- **Proses Level-by-Level (BFS) untuk Animasi:**
  - Selama antrian tidak kosong, tentukan jumlah node pada level saat ini.
  - Untuk setiap node pada level tersebut:
    - Ambil node dari antrian.
    - Hitung nilai warna rata-rata dari region node menggunakan `compute_average_color`.
    - Terapkan warna rata-rata ke blok yang sesuai di `tempimage` menggunakan `set_block_color`.
    - Masukkan semua anak node ke dalam antrian.
  - Setelah memproses seluruh node pada level tersebut, tambahkan `tempimage` beserta delay yang ditentukan ke dalam `resultsequence`.
- **Akhir Proses:**
  - Setelah seluruh level diproses, kembalikan `resultsequence` yang berisi gambar animasi yang memperlihatkan transformasi quadtree per level.

2.4 Optimasi dengan Summed Area Table

Pada proses pembuatan quadtree, terdapat beberapa perhitungan yang bersifat *intensif* jika dilakukan secara langsung pada setiap node, antara lain:

- Perhitungan metode error seperti SSIM, Variance, dan MAD yang memerlukan perhitungan *average color* dari setiap blok gambar.
- Penggunaan *average color* untuk pewarnaan setiap node pada Quadtree, baik pada pembuatan image biasa maupun pada pembuatan image sequence.

Perhitungan rata-rata (average) pada setiap blok melibatkan penjumlahan nilai piksel di dalam blok tersebut. Jika dilakukan secara langsung untuk setiap node, hal ini menjadi pemborosan komputasi mengingat setiap perhitungan dijalankan berulang-ulang untuk area yang tumpang tindih.

Penggunaan Summed Area Table

Diberikan sebuah citra I dengan nilai piksel $I(x, y)$ di setiap koordinat (x, y), *summed area table* (SAT) adalah sebuah matriks S di mana setiap elemen $S(x, y)$ menyimpan jumlah kumulatif dari nilai piksel pada area persegi panjang dari $(1, 1)$ hingga (x, y). Secara matematis, didefinisikan sebagai:

$$S(x, y) = \sum_{i=1}^{x} \sum_{j=1}^{y} I(i, j).$$

Misalkan kita ingin mendapatkan jumlah seluruh piksel dalam area persegi panjang dengan koordinat kiri atas (x_1, y_1) dan kanan bawah (x_2, y_2). Dengan menggunakan SAT, jumlah tersebut dapat dihitung secara instan melalui:

$$\text{Sum}(x_1, y_1, x_2, y_2) = S(x_2, y_2) - S(x_1 - 1, y_2) - S(x_2, y_1 - 1) + S(x_1 - 1, y_1 - 1).$$

Intuisi:

- $S(x_2, y_2)$ memberikan jumlah nilai piksel dari (1,1) hingga (x_2, y_2).
- $S(x_1 - 1, y_2)$ mengurangi jumlah nilai dari area di sebelah kiri area target.
- $S(x_2, y_1 - 1)$ mengurangi jumlah nilai dari area di atas area target.
- $S(x_1 - 1, y_1 - 1)$ ditambahkan kembali untuk mengkompensasi area yang telah dikurangkan dua kali (yaitu, area di atas dan sebelah kiri).

Dengan precompute SAT, setiap penjumlahan untuk area persegi panjang dapat dihitung dengan hanya empat operasi aritmatika, sehingga kompleksitas waktu per akses menjadi $O(1)$, terlepas dari ukuran area tersebut.

Summed Squared Area Table (SSAT): Konsep yang sama berlaku untuk SSAT, di mana tiap elemen menyimpan jumlah kumulatif dari nilai piksel yang telah dikuadratkan:

$$S^2(x, y) = \sum_{i=1}^{x} \sum_{j=1}^{y} I(i, j)^2.$$

Dengan SSAT, perhitungan $E(X^2)$ pada suatu blok dapat dilakukan dengan rumus:

$$E(X^2) = \frac{S^2(x_1, y_1, x_2, y_2)}{N},$$

di mana N adalah jumlah piksel pada blok tersebut, dan $S^2(x_1, y_1, x_2, y_2)$ dihitung mirip dengan SAT.

Aplikasi pada Kompresi Gambar: Seperti dijelaskan pada section sebelumnya, perhitungan beberapa metode error (misalnya, variance, MAD, dan SSIM) membutuhkan

perhitungan rata-rata ($E(X)$) dan rata-rata nilai kuadrat ($E(X^2)$) dari blok piksel. Dengan SAT dan SSAT, perhitungan ini dapat dilakukan secara efisien:

$$\text{Variance} = E(X^2) - (E(X))^2 = \frac{S^2(x_1, y_1, x_2, y_2)}{N} - \left(\frac{S(x_1, y_1, x_2, y_2)}{N}\right)^2.$$

Karena kedua akses SAT dan SSAT memerlukan waktu $O(1)$, maka perhitungan average dan variansi untuk setiap blok dapat dilakukan dengan sangat cepat, mengoptimasi keseluruhan proses algoritma quadtree.

Optimisasi dengan SAT dan SSAT tidak hanya menghemat waktu perhitungan rata-rata dan variansi, tetapi juga mengurangi overhead komputasi pada metode error seperti SSIM, variance, dan MAD yang digunakan dalam pembentukan Quadtree maupun pewarnaan pada gambar terkompresi. Dengan demikian, penerapan optimisasi ini sangat bermanfaat untuk meningkatkan efisiensi dan *compute time* pada algoritma divide and conquer dalam mmembentuk Quadtree dan mengompresi gambar.

Bab 3

Kode Sumber

Selain program yang akan dicantumkan, aplikasi ini menggunakan *dependencies* yang perlu diunduh seperti `stb_image.h`, `stbi_image_writer.h`, dan `gif.h`.

3.1 Quadtree

3.1.1 `quadtreenode.h`

```
1  #ifndef QUADTREENODE_H
2  #define QUADTREENODE_H
3
4  #include <cassert>
5
6  class QuadtreeNode {
7  public:
8    int mPosX;
9    int mPosY;
10
11   int mWidth;
12   int mHeight;
13
14   bool mIsDivided;
15
16   QuadtreeNode *mChildren[4];
17
18   QuadtreeNode(int x, int y, int width, int height);
19   ~QuadtreeNode();
20
21   QuadtreeNode(const QuadtreeNode &) = delete;
22   QuadtreeNode &operator=(const QuadtreeNode &) = delete;
23
24   inline void divide() {
25     if (mIsDivided) {
26       return;
27     }
28
```

```
29      int halfWidth = mWidth / 2;
30      int halfHeight = mHeight / 2;
31      // first quadrant
32      mChildren[0] = new QuadtreeNode(mPosX + halfWidth, mPosY,
33                                      mWidth - halfWidth, halfHeight);
34      // second quadrant
35      mChildren[1] = new QuadtreeNode(mPosX, mPosY, halfWidth,
        halfHeight);
36      // third quadran
37      mChildren[2] = new QuadtreeNode(mPosX, mPosY + halfHeight,
        halfWidth,
38                                      mHeight - halfHeight);
39      // fourth quadran
40      mChildren[3] = new QuadtreeNode(mPosX + halfWidth, mPosY +
        halfHeight,
41                                      mWidth - halfWidth, mHeight -
        halfHeight);
42      mIsDivided = true;
43    }
44  };
45
46  #endif
```

Program 3.1. quadtreenode.h

3.1.2 quadtreenode.cpp

```
1   #include "quadtreenode.h"
2
3   QuadtreeNode::QuadtreeNode(int x, int y, int width, int height)
4       : mPosX(x), mPosY(y), mWidth(width), mHeight(height),
        mIsDivided(false) {
5     for (auto &child : mChildren)
6       child = nullptr;
7   }
8
9   QuadtreeNode::~QuadtreeNode() {
10    for (auto child : mChildren) {
11      if (child != nullptr)
12        delete child;
13    }
14  }
```

Program 3.2. quadtreenode.cpp

3.1.3 quadtreeimage.h

```
1   #ifndef QUADTREEIMAGE_H
2   #define QUADTREEIMAGE_H
```

```
3
4  #include "error_measurement/error_method.h"
5  #include "image/image.h"
6  #include "image/image_sequence.h"
7  #include "quadtreenode.h"
8
9  class QuadtreeImage {
10 private:
11   const Image &mImage;
12   float mThreshold;
13   int mMinBlockSize;
14   ErrorMethod *mErrorMethod;
15
16   int mDepth;
17   int mNodeCount;
18
19   QuadtreeNode *mRoot;
20
21   static constexpr int DEFAULT_SEQUENCE_DELAY = 70;
22
23 public:
24   QuadtreeImage(const Image &image, float threshold, int minBlockSize,
25                 ErrorMethod *errorMethod);
26   ~QuadtreeImage();
27
28   bool build();
29
30   Image apply();
31   ImageSequence applyAnimation();
32
33   void clear();
34
35   int getDepth() const { return mDepth; }
36   int getNodeCount() const { return mNodeCount; }
37   QuadtreeNode *getRoot() const { return mRoot; }
38 };
39
40 #endif
```

Program 3.3. quadtreeimage.h

3.1.4 quadtreeimage.cpp

```
1  #include "quadtreeimage.h"
2  // #include "utils/debug.h"
3  #include <queue>
4
5  QuadtreeImage::QuadtreeImage(const Image &image, float threshold,
6                               int minBlockSize, ErrorMethod *errorMethod)
7      : mImage(image), mThreshold(threshold), mMinBlockSize(minBlockSize),
```

```
 8        mErrorMethod(errorMethod), mDepth(0), mNodeCount(0),
      mRoot(nullptr) {}
 9
10  QuadtreeImage::~QuadtreeImage() { clear(); }
11
12  bool QuadtreeImage::build() {
13    // DEBUG_TIMER("Building tree");
14    mRoot = new QuadtreeNode(0, 0, mImage.getWidth(), mImage.getHeight());
15    std::queue<QuadtreeNode *> nodeQueue;
16    nodeQueue.push(mRoot);
17
18    mNodeCount = 1;
19    mDepth = 0;
20
21    while (!nodeQueue.empty()) {
22      const int nodesThisLevel = nodeQueue.size();
23
24      for (int i = 0; i < nodesThisLevel; ++i) {
25        QuadtreeNode *currentNode = nodeQueue.front();
26        nodeQueue.pop();
27
28        const double nodeError = mErrorMethod->calculateError(
29            mImage, currentNode->mPosX, currentNode->mPosY,
      currentNode->mWidth,
30            currentNode->mHeight);
31
32        const bool isQualityAcceptable =
33            mErrorMethod->isQualityAcceptable(nodeError, mThreshold);
34
35        const bool hasMinimumSizeForDivision =
36            (currentNode->mWidth * currentNode->mHeight) / 4 >=
      mMinBlockSize;
37
38        if (!isQualityAcceptable && hasMinimumSizeForDivision) {
39          if (!currentNode->mIsDivided) {
40            currentNode->divide();
41          }
42
43          for (const auto &child : currentNode->mChildren) {
44            if (child) {
45              ++mNodeCount;
46              nodeQueue.push(child);
47            }
48          }
49        }
50      }
51
52      mDepth++;
53    }
54
55    return mRoot != nullptr;
```

```
56  }
57
58  Image QuadtreeImage::apply() {
59    // DEBUG_TIMER("Applying tree to image");
60    Image resultImage(mImage);
61
62    std::queue<QuadtreeNode *> nodeQueue;
63    nodeQueue.push(mRoot);
64
65    while (!nodeQueue.empty()) {
66      QuadtreeNode *current = nodeQueue.front();
67      nodeQueue.pop();
68
69      if (!current->mIsDivided) {
70        const int area = current->mWidth * current->mHeight;
71        const int x = current->mPosX;
72        const int y = current->mPosY;
73        const int w = current->mWidth;
74        const int h = current->mHeight;
75
76        unsigned char avgR = static_cast<unsigned char>(
77            mImage.getChannelBlockSum(x, y, w, h, 0) / area);
78        unsigned char avgG = static_cast<unsigned char>(
79            mImage.getChannelBlockSum(x, y, w, h, 1) / area);
80        unsigned char avgB = static_cast<unsigned char>(
81            mImage.getChannelBlockSum(x, y, w, h, 2) / area);
82
83        resultImage.setBlockColorAt(x, y, w, h, avgR, avgG, avgB);
84
85      } else {
86        for (auto &child : current->mChildren) {
87          if (child != nullptr) {
88            nodeQueue.push(child);
89          }
90        }
91      }
92    }
93
94    // preserve the alpha channel for png
95    // (setBlockColorAt use some optimization to apply color faster but
       not
96    // maintaining the alpha channel)
97    if (resultImage.getChannels() == 4) {
98      for (int y = 0; y < resultImage.getHeight(); y++) {
99        for (int x = 0; x < resultImage.getWidth(); x++) {
100         resultImage.setAlphaAt(x, y, mImage.getAlphaAt(x, y));
101       }
102     }
103   }
104
105   return resultImage;
```

```
106 }
107
108 ImageSequence QuadtreeImage::applyAnimation() {
109   // DEBUG_TIMER("Applying tree to image sequence");
110   ImageSequence resultSequence;
111   Image tempImage(mImage);
112
113   std::queue<QuadtreeNode *> nodeQueue;
114   nodeQueue.push(mRoot);
115
116   while (!nodeQueue.empty()) {
117     const int nodesThisLevel = nodeQueue.size();
118
119     for (int i = 0; i < nodesThisLevel; ++i) {
120       QuadtreeNode *current = nodeQueue.front();
121       nodeQueue.pop();
122
123       const int area = current->mWidth * current->mHeight;
124       const int x = current->mPosX;
125       const int y = current->mPosY;
126       const int w = current->mWidth;
127       const int h = current->mHeight;
128
129       unsigned char avgR = static_cast<unsigned char>(
130           mImage.getChannelBlockSum(x, y, w, h, 0) / area);
131       unsigned char avgG = static_cast<unsigned char>(
132           mImage.getChannelBlockSum(x, y, w, h, 1) / area);
133       unsigned char avgB = static_cast<unsigned char>(
134           mImage.getChannelBlockSum(x, y, w, h, 2) / area);
135
136       tempImage.setBlockColorAt(x, y, w, h, avgR, avgG, avgB);
137
138       for (auto &child : current->mChildren) {
139         if (child != nullptr) {
140           nodeQueue.push(child);
141         }
142       }
143     }
144     resultSequence.addImage(tempImage, DEFAULT_SEQUENCE_DELAY);
145   }
146
147   return resultSequence;
148 }
149
150 void QuadtreeImage::clear() {
151   if (mRoot) {
152     delete mRoot;
153     mRoot = nullptr;
154   }
155 }
```

Program 3.4. quadtreeimage.cpp

3.2 Error Measurement

3.2.1 error_method.h

```
1  #ifndef ERROR_METHOD_H
2  #define ERROR_METHOD_H
3
4  #include "image/image.h"
5
6  class ErrorMethod {
7  public:
8    virtual ~ErrorMethod() = default;
9    virtual double calculateError(const Image &image, int x, int y, int
       width,
10                                 int height) const = 0;
11   virtual bool isInErrorBound(double error) const = 0;
12
13   virtual double getUpperBound() const = 0;
14   virtual double getLowerBound() const = 0;
15
16   virtual bool isQualityAcceptable(double error, double threshold)
       const = 0;
17   virtual std::string getIdentifier() const = 0;
18 };
19
20 #endif
```

Program 3.5. $\text{error}_{m}ethod.h$

3.2.2 emm_variance.h

```
1  #ifndef EMM_VARIANCE_H
2  #define EMM_VARIANCE_H
3
4  #include "error_method.h"
5  #include <cmath>
6  #include <cstdlib>
7
8  namespace EMM {
9  class Variance : public ErrorMethod {
10 private:
11   static constexpr double kErrorUpperBound = 127.5 * 127.5;
12   static constexpr double kErrorLowerBound = 0;
13
14 public:
```

```
15    double calculateError(const Image &image, int x, int y, int width,
16                          int height) const override {
17      int count = width * height;
18
19      long long sumR = image.getChannelBlockSum(x, y, width, height, 0);
20      long long sumG = image.getChannelBlockSum(x, y, width, height, 1);
21      long long sumB = image.getChannelBlockSum(x, y, width, height, 2);
22
23      long long sumSqR = image.getChannelSquareBlockSum(x, y, width,
        height, 0);
24      long long sumSqG = image.getChannelSquareBlockSum(x, y, width,
        height, 1);
25      long long sumSqB = image.getChannelSquareBlockSum(x, y, width,
        height, 2);
26
27      double meanR = static_cast<double>(sumR) / count;
28      double meanG = static_cast<double>(sumG) / count;
29      double meanB = static_cast<double>(sumB) / count;
30
31      double varianceR = static_cast<double>(sumSqR) / count - meanR *
        meanR;
32      double varianceG = static_cast<double>(sumSqG) / count - meanG *
        meanG;
33      double varianceB = static_cast<double>(sumSqB) / count - meanB *
        meanB;
34
35      return (varianceR + varianceG + varianceB) / 3;
36    }
37
38    inline bool isInErrorBound(double error) const override {
39      return error <= kErrorUpperBound && error >= kErrorLowerBound;
40    };
41
42    double getUpperBound() const override { return kErrorUpperBound; }
43    double getLowerBound() const override { return kErrorLowerBound; }
44
45    bool isQualityAcceptable(double variance, double threshold) const
        override {
46      return variance <= threshold;
47    }
48
49    std::string getIdentifier() const override { return "VAR"; }
50  };
51  } // namespace EMM
52  #endif
```

Program 3.6. emm_variance.h

3.2.3 emm_mad.h

```cpp
#ifndef EMM_MAD_H
#define EMM_MAD_H

#include "error_method.h"
#include <array>
#include <cmath>
#include <cstdlib>

namespace EMM {
class MeanAbsoluteDeviation : public ErrorMethod {
private:
  static constexpr double kErrorUpperBound = 127.5;
  static constexpr double kErrorLowerBound = 0;

public:
  double calculateError(const Image &image, int x, int y, int width,
                        int height) const override {
    std::array<unsigned int, 3> sum = {0, 0, 0};
    int count = width * height;

    double avgR =
        static_cast<double>(image.getChannelBlockSum(x, y, width,
    height, 0)) /
        count;
    double avgG =
        static_cast<double>(image.getChannelBlockSum(x, y, width,
    height, 1)) /
        count;
    double avgB =
        static_cast<double>(image.getChannelBlockSum(x, y, width,
    height, 2)) /
        count;

    for (int i = y; i < y + height; ++i) {
      for (int j = x; j < x + width; ++j) {
        std::array<unsigned char, 3> color = image.getColorAt(j, i);
        sum[0] += std::abs(color[0] - avgR);
        sum[1] += std::abs(color[1] - avgG);
        sum[2] += std::abs(color[2] - avgB);
      }
    }

    double madR = static_cast<double>(sum[0]) / count;
    double madG = static_cast<double>(sum[1]) / count;
    double madB = static_cast<double>(sum[2]) / count;

    return (madR + madG + madB) / 3;
  }

  inline bool isInErrorBound(double error) const override {
    return error <= kErrorUpperBound && error >= kErrorLowerBound;
```

```
49    };
50
51    double getUpperBound() const override { return kErrorUpperBound; }
52    double getLowerBound() const override { return kErrorLowerBound; }
53
54    bool isQualityAcceptable(double mad, double threshold) const override
       {
55      return mad <= threshold;
56    }
57    std::string getIdentifier() const override { return "MAD"; }
58  };
59  } // namespace EMM
60  #endif
```

Program 3.7. emm_mad.h

3.2.4 emm_mpd.h

```
1   #ifndef EMM_MPD_H
2   #define EMM_MPD_H
3
4   #include "error_method.h"
5   #include <algorithm>
6   #include <array>
7   #include <cmath>
8   #include <cstdlib>
9
10  namespace EMM {
11  class MaximumPixelDifference : public ErrorMethod {
12  private:
13    static constexpr double kErrorUpperBound = 255;
14    static constexpr double kErrorLowerBound = 0;
15
16  public:
17    double calculateError(const Image &image, int x, int y, int width,
18                          int height) const override {
19      std::array<unsigned char, 3> firstColor = image.getColorAt(x, y);
20      unsigned char maxR = firstColor[0];
21      unsigned char maxG = firstColor[1];
22      unsigned char maxB = firstColor[2];
23
24      unsigned char minR = firstColor[0];
25      unsigned char minG = firstColor[1];
26      unsigned char minB = firstColor[2];
27
28      for (int i = y; i < y + height; ++i) {
29        for (int j = x; j < x + width; ++j) {
30          std::array<unsigned char, 3> color = image.getColorAt(j, i);
31          maxR = (std::max)(maxR, color[0]);
32          maxG = (std::max)(maxG, color[1]);
```

```
33          maxB = (std::max)(maxB, color[2]);
34
35          minR = (std::min)(minR, color[0]);
36          minG = (std::min)(minG, color[1]);
37          minB = (std::min)(minB, color[2]);
38
39          // early exit if maximum difference is reached for all channels
40          if (maxR == 255 && minR == 0 && maxG == 255 && minG == 0 &&
41              maxB == 255 && minB == 0) {
42            return 255.0;
43          }
44        }
45      }
46
47      return static_cast<double>(maxR - minR + maxG - minG + maxB - minB)
       / 3;
48    }
49
50    inline bool isInErrorBound(double error) const override {
51      return error <= kErrorUpperBound && error >= kErrorLowerBound;
52    };
53
54    double getUpperBound() const override { return kErrorUpperBound; }
55    double getLowerBound() const override { return kErrorLowerBound; }
56
57    bool isQualityAcceptable(double mpd, double threshold) const override
       {
58      return mpd <= threshold;
59    }
60    std::string getIdentifier() const override { return "MPD"; }
61  };
62  } // namespace EMM
63  #endif
```

Program 3.8. emm_mpd.h

3.2.5 emm_entropy.h

```
1  #ifndef EMM_ENTROPY_H
2  #define EMM_ENTROPY_H
3
4  #include "error_method.h"
5  #include <array>
6  #include <cmath>
7
8  namespace EMM {
9
10 class Entropy : public ErrorMethod {
11 private:
12   // 8-bit channel: log2(256) = 8.
```

```
13    static constexpr double kErrorUpperBound = 8.0;
14    static constexpr double kErrorLowerBound = 0.0;
15
16  public:
17    double calculateError(const Image &image, int x, int y, int width,
18                          int height) const override {
19      int count = width * height;
20      std::array<int, 256> histR = {0};
21      std::array<int, 256> histG = {0};
22      std::array<int, 256> histB = {0};
23
24      for (int i = y; i < y + height; ++i) {
25        for (int j = x; j < x + width; ++j) {
26          std::array<unsigned char, 3> color = image.getColorAt(j, i);
27          ++histR[color[0]];
28          ++histG[color[1]];
29          ++histB[color[2]];
30        }
31      }
32
33      double entropyR = 0.0, entropyG = 0.0, entropyB = 0.0;
34      for (int i = 0; i < 256; ++i) {
35        if (histR[i] > 0) {
36          double p = static_cast<double>(histR[i]) / count;
37          entropyR -= p * std::log2(p);
38        }
39        if (histG[i] > 0) {
40          double p = static_cast<double>(histG[i]) / count;
41          entropyG -= p * std::log2(p);
42        }
43        if (histB[i] > 0) {
44          double p = static_cast<double>(histB[i]) / count;
45          entropyB -= p * std::log2(p);
46        }
47      }
48      return (entropyR + entropyG + entropyB) / 3.0;
49    }
50
51    inline bool isInErrorBound(double error) const override {
52      return error <= kErrorUpperBound && error >= kErrorLowerBound;
53    };
54
55    double getUpperBound() const override { return kErrorUpperBound; }
56    double getLowerBound() const override { return kErrorLowerBound; }
57
58    bool isQualityAcceptable(double entropy, double threshold) const
       override {
59      return entropy <= threshold;
60    }
61    std::string getIdentifier() const override { return "ENT"; }
62  };
```

```
63
64 } // namespace EMM
65
66 #endif
```

Program 3.9. emm_entropy.h

3.2.6 emm_ssim.h

```
1  #ifndef EMM_SSIM_H
2  #define EMM_SSIM_H
3
4  #include "error_method.h"
5  #include <cmath>
6  #include <cstdlib>
7
8  namespace EMM {
9  class StructuralSimilarityIndexMeasure : public ErrorMethod {
10 private:
11   static constexpr double kErrorUpperBound = 1;
12   static constexpr double kErrorLowerBound = 0;
13   static constexpr double kC2 = 58.5225;
14
15 public:
16   double calculateError(const Image &image, int x, int y, int width,
17                         int height) const override {
18     // because the ssim compares with the image with its average
19     // the equation can be simplified as:
20     // (C2) / (Var_x ^ 2 + C2)
21
22     int count = width * height;
23
24     long long sumR = image.getChannelBlockSum(x, y, width, height, 0);
25     long long sumG = image.getChannelBlockSum(x, y, width, height, 1);
26     long long sumB = image.getChannelBlockSum(x, y, width, height, 2);
27
28     long long sumSqR = image.getChannelSquareBlockSum(x, y, width,
       height, 0);
29     long long sumSqG = image.getChannelSquareBlockSum(x, y, width,
       height, 1);
30     long long sumSqB = image.getChannelSquareBlockSum(x, y, width,
       height, 2);
31
32     double meanR = static_cast<double>(sumR) / count;
33     double meanG = static_cast<double>(sumG) / count;
34     double meanB = static_cast<double>(sumB) / count;
35
36     double varianceR = static_cast<double>(sumSqR) / count - meanR *
       meanR;
```

```
37      double varianceG = static_cast<double>(sumSqG) / count - meanG *
        meanG;
38      double varianceB = static_cast<double>(sumSqB) / count - meanB *
        meanB;
39
40      double ssimR = kC2 / (varianceR + kC2);
41      double ssimG = kC2 / (varianceG + kC2);
42      double ssimB = kC2 / (varianceB + kC2);
43
44      return (ssimR + ssimG + ssimB) / 3;
45    }
46
47    inline bool isInErrorBound(double error) const override {
48      return error <= kErrorUpperBound && error >= kErrorLowerBound;
49    };
50
51    double getUpperBound() const override { return kErrorUpperBound; }
52    double getLowerBound() const override { return kErrorLowerBound; }
53
54    bool isQualityAcceptable(double ssim, double threshold) const
        override {
55      return ssim >= threshold;
56    }
57    std::string getIdentifier() const override { return "SIM"; }
58  };
59  } // namespace EMM
60  #endif
```

Program 3.10. emm_ssim.h

3.3 Image

3.3.1 image.h

```
1  #ifndef IMAGE_H
2  #define IMAGE_H
3
4  #include <array>
5  #include <cstring>
6  #include <string>
7
8  class Image {
9  public:
10   Image(const std::string &imagePath);
11   Image(const Image &other);
12
13   Image &operator=(const Image &other);
14
15   ~Image();
16
```

```
17    std::string getImagePath() const { return mImagePath; }
18    std::string getFileExt() const { return mFileExt; }
19
20    bool load();
21    bool save(const std::string &outputPath);
22    long long estimateFileSize() const;
23
24    int getWidth() const { return mImageWidth; }
25    int getHeight() const { return mImageHeight; }
26    int getChannels() const { return mChannels; }
27    long long getFileSize() const { return mFileSize; }
28    unsigned char *getImageData() const { return mImageData; }
29
30    std::array<unsigned char, 3> getColorAt(int x, int y) const;
31    unsigned char getAlphaAt(int x, int y) const;
32
33    int getIdxAt(int x, int y, int channel) const;
34
35    long long getChannelBlockSum(int x, int y, int width, int height,
36                                 int channel) const;
37    long long getChannelSquareBlockSum(int x, int y, int width, int
       height,
38                                       int channel) const;
39
40    void setColorAt(int x, int y, unsigned char r, unsigned char g,
41                    unsigned char b);
42    void setAlphaAt(int x, int y, unsigned char a);
43
44    void setBlockColorAt(int x, int y, int width, int height, unsigned
       char r,
45                         unsigned char g, unsigned char b);
46    void computeSummedSquareTable();
47    void computeSummedAreaTable();
48
49  private:
50    std::string mImagePath;
51    std::string mFileExt;
52    int mImageWidth;
53    int mImageHeight;
54    int mChannels;
55    unsigned char *mImageData;
56
57    long long mFileSize;
58
59    long long *mSummedAreaTable;
60    long long *mSummedSquareTable;
61  };
62
63  #endif
```

Program 3.11. image.h

3.3.2 image.cpp

```cpp
#include "image.h"
#include <vector>
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_writer.h"
// #include "utils/debug.h"
#include <algorithm>
#include <filesystem>
#include <iostream>

namespace fs = std::filesystem;

Image::Image(const std::string &imagePath)
    : mImagePath(fs::absolute(imagePath).string()), mFileExt(""),
      mImageWidth(0), mImageHeight(0), mChannels(0),
    mImageData(nullptr),
      mFileSize(0), mSummedAreaTable(nullptr),
    mSummedSquareTable(nullptr) {}

Image::Image(const Image &other)
    : mImagePath(other.mImagePath), mFileExt(other.mFileExt),
      mImageWidth(other.mImageWidth), mImageHeight(other.mImageHeight),
      mChannels(other.mChannels), mImageData(nullptr),
      mFileSize(other.mFileSize), mSummedAreaTable(nullptr),
      mSummedSquareTable(nullptr) {

  if (other.mImageData) {
    int dataSize = mImageWidth * mImageHeight * mChannels;
    mImageData = new unsigned char[dataSize];
    std::memcpy(mImageData, other.mImageData, dataSize);
  }

  if (other.mSummedAreaTable) {
    int tableSize = mImageWidth * mImageHeight * mChannels;
    mSummedAreaTable = new long long[tableSize];
    std::memcpy(mSummedAreaTable, other.mSummedAreaTable,
                tableSize * sizeof(long long));
  }
  if (other.mSummedSquareTable) {
    int tableSize = mImageWidth * mImageHeight * mChannels;
    mSummedSquareTable = new long long[tableSize];
    std::memcpy(mSummedSquareTable, other.mSummedSquareTable,
                tableSize * sizeof(long long));
  }
}

Image &Image::operator=(const Image &other) {
  if (this != &other) {
```

```
48      delete[] mImageData;
49      delete[] mSummedAreaTable;
50      delete[] mSummedSquareTable;
51
52      mImagePath = other.mImagePath;
53      mImageWidth = other.mImageWidth;
54      mImageHeight = other.mImageHeight;
55      mFileExt = other.mFileExt;
56      mChannels = other.mChannels;
57
58      if (other.mImageData) {
59        int dataSize = mImageWidth * mImageHeight * mChannels;
60        mImageData = new unsigned char[dataSize];
61        std::memcpy(mImageData, other.mImageData, dataSize);
62      } else {
63        mImageData = nullptr;
64      }
65
66      if (other.mSummedAreaTable) {
67        int tableSize = mImageWidth * mImageHeight * mChannels;
68        mSummedAreaTable = new long long[tableSize];
69        std::memcpy(mSummedAreaTable, other.mSummedAreaTable,
70                    tableSize * sizeof(long long));
71      } else {
72        mSummedAreaTable = nullptr;
73      }
74      if (other.mSummedSquareTable) {
75        int tableSize = mImageWidth * mImageHeight * mChannels;
76        mSummedSquareTable = new long long[tableSize];
77        std::memcpy(mSummedSquareTable, other.mSummedSquareTable,
78                    tableSize * sizeof(long long));
79      } else {
80        mSummedSquareTable = nullptr;
81      }
82    }
83    return *this;
84  }
85
86  Image::~Image() {
87    if (mImageData) {
88      stbi_image_free(mImageData);
89    }
90    delete[] mSummedAreaTable;
91    delete[] mSummedSquareTable;
92  }
93
94  bool Image::load() {
95
96    fs::path filepath(mImagePath);
97
98    mFileExt = filepath.extension().string();
```

```
 99    std::transform(mFileExt.begin(), mFileExt.end(), mFileExt.begin(),
100                   [](unsigned char c) { return std::tolower(c); });
101
102    try {
103      mFileSize = fs::file_size(filepath);
104    } catch (const fs::filesystem_error &e) {
105      std::cerr << "Error getting file size: " << e.what() << std::endl;
106    }
107
108    mImageData =
109        stbi_load(mImagePath.c_str(), &mImageWidth, &mImageHeight,
       &mChannels, 0);
110    if (!mImageData) {
111      std::cerr << "Error: " << stbi_failure_reason() << std::endl;
112      return true;
113    }
114    if (mChannels < 3) {
115      std::cerr << "Non-RGB image not supported" << std::endl;
116      return true;
117    }
118    return false;
119  }
120
121  bool Image::save(const std::string &outputPath = "") {
122    std::string savePath = outputPath;
123    bool isSuccess = false;
124
125    // idk
126    if (savePath.empty()) {
127      fs::path originalPath(mImagePath);
128      savePath = originalPath.replace_extension(mFileExt).string();
129    }
130
131    if (mFileExt == ".png") {
132      isSuccess =
133          stbi_write_png(savePath.c_str(), mImageWidth, mImageHeight,
       mChannels,
134                         mImageData, mImageWidth * mChannels) != 0;
135    } else if (mFileExt == ".jpg" || mFileExt == ".jpeg") {
136
137      isSuccess = stbi_write_jpg(savePath.c_str(), mImageWidth,
       mImageHeight,
138                                 mChannels, mImageData,
139                                 68 // quality
140                                 ) != 0;
141    } else if (mFileExt == ".bmp") {
142      isSuccess = stbi_write_bmp(savePath.c_str(), mImageWidth,
       mImageHeight,
143                                 mChannels, mImageData) != 0;
144    } else if (mFileExt == ".tga") {
```

```
145       isSuccess = stbi_write_tga(savePath.c_str(), mImageWidth,
      mImageHeight,
146                                  mChannels, mImageData);
147     } else {
148
149       savePath = outputPath.empty()
150                      ?
      fs::path(mImagePath).replace_extension("png").string()
151                      : outputPath;
152
153       isSuccess =
154           stbi_write_png(savePath.c_str(), mImageWidth, mImageHeight,
      mChannels,
155                          mImageData, mImageWidth * mChannels) != 0;
156     }
157     if (isSuccess) {
158       try {
159         mFileSize = fs::file_size(savePath);
160       } catch (const fs::filesystem_error &e) {
161         std::cerr << "Error getting file size: " << e.what() << std::endl;
162       }
163     }
164     return isSuccess;
165   }
166
167   static void count_bytes(void *context, void *data, int size) {
168     (void)data;
169     size_t *total = reinterpret_cast<size_t *>(context);
170     *total += static_cast<size_t>(size);
171   }
172
173   long long Image::estimateFileSize() const {
174     size_t totalBytes = 0;
175
176     if (mFileExt == ".png") {
177       stbi_write_png_to_func(count_bytes, &totalBytes, mImageWidth,
      mImageHeight,
178                              mChannels, mImageData, mImageWidth *
      mChannels);
179     } else if (mFileExt == ".jpg" || mFileExt == ".jpeg") {
180       stbi_write_jpg_to_func(count_bytes, &totalBytes, mImageWidth,
      mImageHeight,
181                              mChannels, mImageData, 68);
182     } else if (mFileExt == ".bmp") {
183       stbi_write_bmp_to_func(count_bytes, &totalBytes, mImageWidth,
      mImageHeight,
184                              mChannels, mImageData);
185     } else if (mFileExt == ".tga") {
186       stbi_write_tga_to_func(count_bytes, &totalBytes, mImageWidth,
      mImageHeight,
187                              mChannels, mImageData);
```

```
188   } else {
189     stbi_write_png_to_func(count_bytes, &totalBytes, mImageWidth,
      mImageHeight,
190                            mChannels, mImageData, mImageWidth *
      mChannels);
191   }
192   return totalBytes;
193 }
194
195 void Image::computeSummedAreaTable() {
196   // DEBUG_TIMER("Compute summed area table");
197   int totalPixels = mImageWidth * mImageHeight;
198   int tableSize = totalPixels * mChannels;
199   mSummedAreaTable = new long long[tableSize];
200   std::fill(mSummedAreaTable, mSummedAreaTable + tableSize, 0);
201
202   for (int y = 0; y < mImageHeight; ++y) {
203     for (int x = 0; x < mImageWidth; ++x) {
204       int idx = y * (mImageWidth * mChannels) + x * mChannels;
205       for (int c = 0; c < mChannels; ++c) {
206         long long currentVal = static_cast<long long>(mImageData[idx +
      c]);
207         long long left = (x > 0) ? mSummedAreaTable[idx - mChannels +
      c] : 0;
208         long long above =
209             (y > 0) ? mSummedAreaTable[idx - (mImageWidth * mChannels)
      + c] : 0;
210         long long aboveLeft =
211             (x > 0 && y > 0)
212                 ? mSummedAreaTable[idx - (mImageWidth * mChannels) -
      mChannels +
213                                    c]
214                 : 0;
215         mSummedAreaTable[idx + c] = currentVal + left + above -
      aboveLeft;
216       }
217     }
218   }
219 }
220
221 void Image::computeSummedSquareTable() {
222   // DEBUG_TIMER("Compute square area table");
223   int totalPixels = mImageWidth * mImageHeight;
224   int tableSize = totalPixels * mChannels;
225   mSummedSquareTable = new long long[tableSize];
226   std::fill(mSummedSquareTable, mSummedSquareTable + tableSize, 0);
227
228   for (int y = 0; y < mImageHeight; ++y) {
229     for (int x = 0; x < mImageWidth; ++x) {
230       int idx = y * (mImageWidth * mChannels) + x * mChannels;
231       for (int c = 0; c < mChannels; ++c) {
```

```
232           long long currentVal = static_cast<long long>(mImageData[idx +
      c]);
233           long long squareVal = currentVal * currentVal;
234           long long left = (x > 0) ? mSummedSquareTable[idx - mChannels +
      c] : 0;
235           long long above =
236               (y > 0) ? mSummedSquareTable[idx - (mImageWidth *
      mChannels) + c]
237                       : 0;
238           long long aboveLeft =
239               (x > 0 && y > 0)
240                   ? mSummedSquareTable[idx - (mImageWidth * mChannels) -
241                                        mChannels + c]
242                   : 0;
243           mSummedSquareTable[idx + c] = squareVal + left + above -
      aboveLeft;
244         }
245       }
246     }
247   }
248
249   int Image::getIdxAt(int x, int y, int channel) const {
250     return y * (mImageWidth * mChannels) + x * mChannels + channel;
251   }
252
253   std::array<unsigned char, 3> Image::getColorAt(int x, int y) const {
254     std::array<unsigned char, 3> color = {0, 0, 0}; // RGB
255     if (x < 0 || x >= mImageWidth || y < 0 || y >= mImageHeight)
256       return color;
257
258     color[0] = mImageData[getIdxAt(x, y, 0)];
259     color[1] = mImageData[getIdxAt(x, y, 1)];
260     color[2] = mImageData[getIdxAt(x, y, 2)];
261     return color;
262   }
263
264   unsigned char Image::getAlphaAt(int x, int y) const {
265     if (mChannels == 4) {
266       return mImageData[getIdxAt(x, y, 3)];
267     }
268     return 255;
269   }
270
271   long long Image::getChannelBlockSum(int x, int y, int width, int height,
272                                       int channel) const {
273
274     long long A =
275         (x > 0 && y > 0) ? mSummedAreaTable[getIdxAt(x - 1, y - 1,
      channel)] : 0;
276     long long B =
```

```
277       (y > 0) ? mSummedAreaTable[getIdxAt(x + width - 1, y - 1,
      channel)] : 0;
278   long long C =
279       (x > 0) ? mSummedAreaTable[getIdxAt(x - 1, y + height - 1,
      channel)] : 0;
280   long long D =
281       mSummedAreaTable[getIdxAt(x + width - 1, y + height - 1,
      channel)];
282
283   return D - B - C + A;
284 }
285
286 long long Image::getChannelSquareBlockSum(int x, int y, int width, int
      height,
287                                           int channel) const {
288   long long A = (x > 0 && y > 0)
289                     ? mSummedSquareTable[getIdxAt(x - 1, y - 1,
      channel)]
290                     : 0;
291   long long B =
292       (y > 0) ? mSummedSquareTable[getIdxAt(x + width - 1, y - 1,
      channel)] : 0;
293   long long C =
294       (x > 0) ? mSummedSquareTable[getIdxAt(x - 1, y + height - 1,
      channel)]
295               : 0;
296   long long D =
297       mSummedSquareTable[getIdxAt(x + width - 1, y + height - 1,
      channel)];
298   return D - B - C + A;
299 }
300
301 void Image::setColorAt(int x, int y, unsigned char r, unsigned char g,
302                        unsigned char b) {
303   mImageData[getIdxAt(x, y, 0)] = r;
304   mImageData[getIdxAt(x, y, 1)] = g;
305   mImageData[getIdxAt(x, y, 2)] = b;
306 }
307
308 void Image::setAlphaAt(int x, int y, unsigned char a) {
309   mImageData[getIdxAt(x, y, 3)] = a;
310 }
311
312 void Image::setBlockColorAt(int startX, int startY, int blockWidth,
313                             int blockHeight, unsigned char r, unsigned
      char g,
314                             unsigned char b) {
315   if (!mImageData) {
316     return;
317   }
318
```

```
319   int endX = std::min(startX + blockWidth, mImageWidth);
320   int endY = std::min(startY + blockHeight, mImageHeight);
321   if (startX < 0 || startY < 0 || startX >= mImageWidth ||
322       startY >= mImageHeight)
323     return;
324
325   int effectiveBlockWidth = endX - startX;
326   int rowSize = effectiveBlockWidth * mChannels;
327   std::vector<unsigned char> rowBuffer(rowSize);
328
329   for (int i = 0; i < effectiveBlockWidth; ++i) {
330     rowBuffer[i * mChannels + 0] = r;
331     rowBuffer[i * mChannels + 1] = g;
332     rowBuffer[i * mChannels + 2] = b;
333     if (mChannels == 4) {
334       rowBuffer[i * mChannels + 3] = 255;
335     }
336   }
337
338   for (int y = startY; y < endY; ++y) {
339     unsigned char *rowPtr = mImageData + (y * mImageWidth + startX) *
      mChannels;
340     std::memcpy(rowPtr, rowBuffer.data(), rowSize);
341   }
342 }
```

Program 3.12. image.cpp

3.3.3 image_sequence.h

```
1  #ifndef IMAGE_SEQUENCE_H
2  #define IMAGE_SEQUENCE_H
3
4  #include "image/image.h"
5  #include <string>
6  #include <utility>
7  #include <vector>
8
9  class ImageSequence {
10 public:
11   ImageSequence() = default;
12
13   ImageSequence(const ImageSequence &) = default;
14   ImageSequence &operator=(const ImageSequence &) = default;
15   ~ImageSequence() = default;
16
17   void addImage(Image image, int delay) {
18     mFrames.emplace_back(std::move(image), delay);
19   }
20
```

```
21    bool save(const std::string &output_path) const;
22
23    int size() const noexcept { return mFrames.size(); }
24    bool empty() const noexcept { return mFrames.empty(); }
25
26  private:
27    std::vector<std::pair<Image, int>> mFrames;
28  };
29
30  #endif
```

Program 3.13. image_sequence.h

3.3.4 image_sequence.cpp

```
1   #include "image_sequence.h"
2   #include "gif.h"
3   // #include "utils/debug.h"
4   #include <stdexcept>
5   #include <vector>
6
7   namespace {
8   std::vector<uint8_t> convert_to_rgba(const Image &img) {
9     const int width = img.getWidth();
10    const int height = img.getHeight();
11    const int src_channels = img.getChannels();
12    const uint8_t *src = img.getImageData();
13
14    std::vector<uint8_t> dst(width * height * 4);
15
16    for (int y = 0; y < height; ++y) {
17      for (int x = 0; x < width; ++x) {
18        const size_t src_idx = (y * width + x) * src_channels;
19        const size_t dst_idx = (y * width + x) * 4;
20
21        for (int c = 0; c < 3; ++c) {
22          dst[dst_idx + c] = src[src_idx + c];
23        }
24
25        dst[dst_idx + 3] = (src_channels == 4) ? src[src_idx + 3] : 0xFF;
26      }
27    }
28    return dst;
29  }
30  } // namespace
31
32  bool ImageSequence::save(const std::string &output_path) const {
33    // DEBUG_TIMER("Saving Gif");
34    if (mFrames.empty())
35      return false;
```

```
36
37    const auto &[first_image, first_delay] = mFrames.front();
38    const int width = first_image.getWidth();
39    const int height = first_image.getHeight();
40
41    GifWriter writer;
42    if (!GifBegin(&writer, output_path.c_str(), width, height,
       first_delay)) {
43      return false;
44    }
45
46    try {
47      for (const auto &[img, delay] : mFrames) {
48        if (img.getWidth() != width || img.getHeight() != height) {
49          throw std::runtime_error("All images must have identical
       dimensions");
50        }
51
52        if (img.getChannels() != 4) {
53          const auto pixel_data = convert_to_rgba(img);
54          if (!GifWriteFrame(&writer, pixel_data.data(), width, height,
       delay)) {
55            throw std::runtime_error("GIF frame write failed");
56          }
57        } else {
58          const auto pixel_data = img.getImageData();
59          if (!GifWriteFrame(&writer, pixel_data, width, height, delay)) {
60            throw std::runtime_error("GIF frame write failed");
61          }
62        }
63      }
64    } catch (...) {
65      GifEnd(&writer);
66      return false;
67    }
68
69    return GifEnd(&writer);
70  }
```

Program 3.14. image_sequence.cpp

3.4 Controller

3.4.1 compression_controller.h

```
1  #ifndef COMPRESSION_CONTROLLER_H
2  #define COMPRESSION_CONTROLLER_H
3
4  #include "error_measurement/error_method.h"
5  #include <functional>
```

```cpp
#include <string>

enum class ProgressStage {
  Loading,
  Precompute,
  FindingTarget,
  BuildingTree,
  TransformingImage,
  SavingImage,
  CreatingGif,
  Finished
};

struct CompressionResult {
  double computationTime;
  size_t originalFileSize;
  size_t compressedFileSize;
  double compressionPercentage;
  int quadtreeDepth;
  int quadtreeNodeCount;
  std::string outputFilePath;
  std::string gifOutputPath;
};

class CompressionController {
private:
  std::string mInputPath;
  ErrorMethod *mErrorMethod;
  double mThreshold;
  int mMinBlockSize;
  double mTargetCompression;
  std::string mOutputPath;
  std::string mGifOutputPath;

  std::string mFileExt;

  CompressionResult result;

  void findTargetCompression(Image &, long long);

public:
  CompressionController() : mErrorMethod(nullptr){};

  std::string getInputPath() const { return mInputPath; }
  ErrorMethod *getErrorMethod() const { return mErrorMethod; }
  double getThreshold() const { return mThreshold; }
  int getMinBlockSize() const { return mMinBlockSize; }
  double getTargetCompression() const { return mTargetCompression; }
  std::string getOutputPath() const { return mOutputPath; }
  std::string getGifOutputPath() const { return mGifOutputPath; }
  std::string getFileExt() const { return mFileExt; }
```

```
57    CompressionResult getResult() const { return result; }
58
59    bool setInputPath(std::string);
60    bool setErrorMethod(ErrorMethod *);
61    bool setThreshold(double);
62    bool setMinBlockSize(int);
63    bool setTargetCompression(double);
64    bool setOutputPath(std::string);
65    bool setGifOutputPath(std::string);
66
67    bool run(std::function<void(const ProgressStage &)>);
68  };
69
70  #endif
```

Program 3.15. compression_controller.h

3.4.2 compression_controller.cpp

```
1   #include "controller/compression_controller.h"
2   #include "quadtree/quadtreeimage.h"
3   #include <algorithm>
4   #include <cassert>
5   #include <cstdlib>
6   #include <filesystem>
7   #include <string>
8   #include <utility>
9
10  namespace fs = std::filesystem;
11
12  bool CompressionController::setInputPath(std::string path) {
13    fs::path filePath(path);
14    filePath = fs::absolute(filePath);
15    std::string ext = filePath.extension().string();
16
17    std::transform(ext.begin(), ext.end(), ext.begin(),
18                   [](unsigned char c) { return std::tolower(c); });
19
20    if (fs::exists(filePath)) {
21      if (ext == ".png" || ext == ".jpg" || ext == ".jpeg" || ext ==
       ".tga" ||
22          ext == ".hdr" || ext == ".bmp") {
23        mFileExt = ext;
24        mInputPath = filePath.string();
25        return true;
26      }
27    }
28    return false;
29  }
30
```

```
31  bool CompressionController::setErrorMethod(ErrorMethod *method) {
32    if (mErrorMethod != nullptr) {
33      delete mErrorMethod;
34      mErrorMethod = nullptr;
35    }
36    mErrorMethod = method;
37    return true;
38  }
39
40  bool CompressionController::setThreshold(double threshold) {
41    assert(mErrorMethod);
42    if (mErrorMethod->isInErrorBound(threshold)) {
43      mThreshold = threshold;
44      return true;
45    }
46    return false;
47  }
48  bool CompressionController::setMinBlockSize(int blockSize) {
49    mMinBlockSize = blockSize;
50    return true;
51  }
52  bool CompressionController::setTargetCompression(double target) {
53    if (target >= 0.0 && target <= 1.0) {
54      mTargetCompression = target;
55      return true;
56    }
57    return false;
58  }
59  bool CompressionController::setOutputPath(std::string path) {
60    fs::path filePath(path);
61    std::string ext = filePath.extension().string();
62
63    std::transform(ext.begin(), ext.end(), ext.begin(),
64                   [](unsigned char c) { return std::tolower(c); });
65
66    if (ext == mFileExt) {
67      mOutputPath = filePath.string();
68      return true;
69    }
70    return false;
71  }
72  bool CompressionController::setGifOutputPath(std::string path) {
73    fs::path filePath(path);
74    std::string ext = filePath.extension().string();
75
76    std::transform(ext.begin(), ext.end(), ext.begin(),
77                   [](unsigned char c) { return std::tolower(c); });
78
79    if (ext == ".gif") {
80      mGifOutputPath = filePath.string();
81      return true;
```

```
82    }
83    return false;
84  }
85  bool CompressionController::run(
86      std::function<void(const ProgressStage &)> progressCallback) {
87
88    progressCallback(ProgressStage::Loading);
89    Image image(mInputPath);
90    image.load();
91
92    progressCallback(ProgressStage::Precompute);
93    image.computeSummedAreaTable();
94    std::string methodId = mErrorMethod->getIdentifier();
95    if (methodId == "SIM" || methodId == "VAR") {
96      image.computeSummedSquareTable();
97    }
98
99    if (mTargetCompression) {
100     progressCallback(ProgressStage::FindingTarget);
101     long long targetSize = (1.0 - mTargetCompression) *
       image.getFileSize();
102     findTargetCompression(image, targetSize);
103   }
104
105   progressCallback(ProgressStage::BuildingTree);
106   QuadtreeImage quadtree(image, mThreshold, mMinBlockSize,
       mErrorMethod);
107   if (!quadtree.build()) {
108     return false;
109   }
110
111   progressCallback(ProgressStage::TransformingImage);
112   Image resultImage = quadtree.apply();
113
114   progressCallback(ProgressStage::SavingImage);
115   resultImage.save(mOutputPath);
116
117   result.originalFileSize = image.getFileSize();
118   result.compressedFileSize = resultImage.getFileSize();
119   result.compressionPercentage =
120       (1.0 -
121        static_cast<double>(resultImage.getFileSize()) /
       image.getFileSize()) *
122       100;
123   result.quadtreeDepth = quadtree.getDepth();
124   result.quadtreeNodeCount = quadtree.getNodeCount();
125   result.outputFilePath = mOutputPath;
126   result.gifOutputPath = mGifOutputPath;
127
128   if (!mGifOutputPath.empty()) {
129     progressCallback(ProgressStage::CreatingGif);
```

```
130       ImageSequence resultSequence = quadtree.applyAnimation();
131       resultSequence.save(mGifOutputPath);
132     }
133
134     progressCallback(ProgressStage::Finished);
135     return true;
136   }
137
138   void CompressionController::findTargetCompression(Image &image,
139                                                     long long targetSize)
        {
140     long long leftSize, rightSize, middleSize;
141     Image res(mInputPath);
142     double rightThreshold = mErrorMethod->getUpperBound();
143     double leftThreshold = mErrorMethod->getLowerBound();
144     if (mErrorMethod->getIdentifier() == "SIM") {
145       std::swap(rightThreshold, leftThreshold);
146     }
147     double middleThreshold;
148     QuadtreeImage quadtreeLeft(image, leftThreshold, mMinBlockSize,
        mErrorMethod);
149     quadtreeLeft.build();
150     res = quadtreeLeft.apply();
151     leftSize = res.estimateFileSize();
152     if (targetSize > leftSize) {
153       mThreshold = leftThreshold;
154       return;
155     }
156
157     QuadtreeImage quadtreeRight(image, rightThreshold, mMinBlockSize,
158                                 mErrorMethod);
159     quadtreeRight.build();
160     res = quadtreeRight.apply();
161     rightSize = res.estimateFileSize();
162
163     if (targetSize < rightSize) {
164       mThreshold = rightThreshold;
165       return;
166     }
167
168     for (int i = 1; i < 32; i++) {
169       middleThreshold = (leftThreshold + rightThreshold) / 2.0;
170       if (std::abs(middleThreshold - rightThreshold) < 0.00001 &&
171           std::abs(middleThreshold - leftThreshold) < 0.00001) {
172         break;
173       }
174       QuadtreeImage quadtreeMiddle(image, middleThreshold, mMinBlockSize,
175                                    mErrorMethod);
176       quadtreeMiddle.build();
177       res = quadtreeMiddle.apply();
178       middleSize = res.estimateFileSize();
```

```
179
180     if (std::abs(middleSize - targetSize) < 10) {
181       mThreshold = middleThreshold;
182       return;
183     }
184     if (targetSize > middleSize) {
185       rightThreshold = middleThreshold;
186     } else {
187       leftThreshold = middleThreshold;
188     }
189   }
190   mThreshold = middleThreshold;
191   return;
192 }
```

Program 3.16. compression_controller.cpp

3.5 Utils

3.5.1 debug.h

```
1  #ifndef DEBUG_H
2  #define DEBUG_H
3
4  #include <chrono>
5  #include <iostream>
6  #include <string>
7
8  class DebugTimer {
9  public:
10   DebugTimer(const std::string &name = "Block")
11       : mName(name), mStart(std::chrono::high_resolution_clock::now())
     {}
12
13   ~DebugTimer() {
14     auto end = std::chrono::high_resolution_clock::now();
15     auto duration =
16         std::chrono::duration_cast<std::chrono::milliseconds>(end -
     mStart)
17             .count();
18     std::cout << "[DEBUG] " << mName << " took " << duration << " ms."
19               << std::endl;
20   }
21
22 private:
23   std::string mName;
24   std::chrono::high_resolution_clock::time_point mStart;
25 };
26
27 #define DEBUG_TIMER(name) DebugTimer debugTimer_##__LINE__(name)
```

```
28
29 #endif
```

Program 3.17. debug.h

3.5.2 style.h

```
1  #ifndef STYLE_H
2  #define STYLE_H
3
4  #include <string>
5
6  /**
7   * Kid: "Mom, can we have Tailwind?"
8   * Mom: "No, we have Tailwind at home."
9   * Tailwind at home: style.h
10  *
11  * @warning Requires ANSI-compatible terminal support
12  *
13  * @author y4nked/kiwz
14  * @notes This code doesn't include the full capabilities of ANSI
      escapes,
15  * please feel free to add it by your self from this source:
16  * https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797 or any
      other
17  * sources.
18  *
19  * Example usage:
20  * #include "style.h"   // Adjust your path
21  * // Basic text coloring
22  * std::cout << style::BLUE << "This text is blue\n" << style::RESET;
23  *
24  * // Combining styles
25  * std::cout << style::BOLD << style::RED << "Bold red text"
26  *           << style::BLUE << " this is bold blue\n" << style::RESET;
27  *
28  * // Background and text color
29  * std::cout << style::BG_GREEN << style::BLACK << "Green background
      with black
30  * text"
31  *           << style::RESET << "\n";
32  *
33  * // Cursor manipulation
34  * std::cout << style::HIDE_CURSOR;  // Hide the cursor
35  * // ... do something ...
36  * std::cout << style::SHOW_CURSOR;  // Show the cursor again
37  *
38  * // Terminal effects
39  * std::cout << style::CLEAR_SCREEN;  // Clear the screen
40  * std::cout << style::SOUND;         // Trigger a beep
```

```
41  */
42
43 // Terminal effects
44 inline const std::string CLEAR_SCREEN =
45     "\033c"; // Only clear the shown screen (not really)
46 inline const std::string FULL_CLEAR_SCREEN = "\033[2J\033[3J\033[H";
47 inline const std::string SOUND = "\a";
48
49 // Cursor manipulation
50 inline const std::string HIDE_CURSOR = "\033[?25l";
51 inline const std::string SHOW_CURSOR = "\033[?25h";
52 inline const std::string BLOCK_CURSOR = "\033[2 q";        // Blinking
       block
53 inline const std::string STEADY_BLOCK_CURSOR = "\033[0 q"; // Steady
       block
54 inline const std::string UNDERLINE_CURSOR = "\033[4 q";    // Blinking
       underline
55 inline const std::string STEADY_UNDERLINE_CURSOR =
56     "\033[3 q";                                            // Steady
       underline
57 inline const std::string BAR_CURSOR = "\033[6 q";         // Blinking bar
58 inline const std::string STEADY_BAR_CURSOR = "\033[5 q"; // Steady bar
59
60 // Text formatting
61 inline const std::string RESET = "\033[0m";          // Reset all styles
62 inline const std::string BOLD = "\033[1m";           // Bold text
63 inline const std::string DIM = "\033[2m";            // Dim text
64 inline const std::string ITALIC = "\033[3m";         // Italic text
65 inline const std::string UNDERLINE = "\033[4m";      // Underline text
66 inline const std::string BLINK = "\033[5m";          // Blink text (COOL)
67 inline const std::string REVERSE = "\033[7m";        // Reverse colors
68 inline const std::string HIDDEN = "\033[8m";         // Hidden text
69 inline const std::string STRIKETHROUGH = "\033[9m"; // Strikethrough
       text
70
71 // Text colors
72 inline const std::string BLACK = "\033[30m";
73 inline const std::string RED = "\033[31m";
74 inline const std::string GREEN = "\033[32m";
75 inline const std::string YELLOW = "\033[33m";
76 inline const std::string BLUE = "\033[34m";
77 inline const std::string MAGENTA = "\033[35m";
78 inline const std::string CYAN = "\033[36m";
79 inline const std::string WHITE = "\033[37m";
80
81 // Bright text colors
82 inline const std::string BRIGHT_BLACK = "\033[90m";
83 inline const std::string BRIGHT_RED = "\033[91m";
84 inline const std::string BRIGHT_GREEN = "\033[92m";
85 inline const std::string BRIGHT_YELLOW = "\033[93m";
86 inline const std::string BRIGHT_BLUE = "\033[94m";
```

```
87  inline const std::string BRIGHT_MAGENTA = "\033[95m";
88  inline const std::string BRIGHT_CYAN = "\033[96m";
89  inline const std::string BRIGHT_WHITE = "\033[97m";
90
91  // Background colors
92  inline const std::string BG_BLACK = "\033[40m";
93  inline const std::string BG_RED = "\033[41m";
94  inline const std::string BG_GREEN = "\033[42m";
95  inline const std::string BG_YELLOW = "\033[43m";
96  inline const std::string BG_BLUE = "\033[44m";
97  inline const std::string BG_MAGENTA = "\033[45m";
98  inline const std::string BG_CYAN = "\033[46m";
99  inline const std::string BG_WHITE = "\033[47m";
100
101 // Bright background colors
102 inline const std::string BG_BRIGHT_BLACK = "\033[100m";
103 inline const std::string BG_BRIGHT_RED = "\033[101m";
104 inline const std::string BG_BRIGHT_GREEN = "\033[102m";
105 inline const std::string BG_BRIGHT_YELLOW = "\033[103m";
106 inline const std::string BG_BRIGHT_BLUE = "\033[104m";
107 inline const std::string BG_BRIGHT_MAGENTA = "\033[105m";
108 inline const std::string BG_BRIGHT_CYAN = "\033[106m";
109 inline const std::string BG_BRIGHT_WHITE = "\033[107m";
110
111 #endif // STYLE_H
```

Program 3.18. style.h

3.6 CLI

3.6.1 cli.h

```
1  #pragma once
2
3  #include "controller/compression_controller.h"
4  #include <atomic>
5  #include <chrono>
6  #include <cmath>
7  #include <cstdint>
8  #include <functional>
9  #include <iomanip>
10 #include <iostream>
11 #include <memory>
12 #include <mutex>
13 #include <optional>
14 #include <string>
15 #include <thread>
16 #include <utility>
17 #include <vector>
18
```

```cpp
#include "utils/style.h"

#ifdef _WIN32
#include <conio.h>
#include <windows.h>
#else
#include <sys/ioctl.h>
#include <termios.h>
#include <unistd.h>
#endif

class CLI {
public:
  CLI();
  ~CLI();

  // Main method to run the CLI interface
  CompressionResult run();

private:
  CompressionController compression;

  std::chrono::steady_clock::time_point spinnerStartTime =
      std::chrono::steady_clock::now();
  double totalElapsedTime = 0;

  // Terminal handling and utilities
  void setupTerminal();
  void resetTerminal();
  void clearScreen();
  void clearVisibleScreen();
  void clearLine();
  std::string doubleToString(double value, int precision);
  void moveCursorUp(int lines);
  void moveCursorToColumn(int col);
  std::pair<int, int> getTerminalSize();

  // Input handling
  char getChar();
  int getCharWithTimeout(int);
  std::string readInput(const std::string &placeholder = "");
  int handleArrowKeys();

  // Text formatting and display
  void printTitle(const std::string &title);
  void printHeader(const std::string &header);
  void printPrompt(const std::string &prompt, bool error = false);
  void printHint(const std::string &hint);
  void printKeybind(const std::vector<std::pair<std::string,
   std::string>>);
  void printHelp();
```

```
69    void printErrorMessage(const std::string &message);
70    void printPreviousInputs();
71
72    // Selection handling
73    template <typename T>
74    T selectOption(const std::string &prompt,
75                   const std::vector<std::pair<std::string, T>> &options);
76
77    // Input gathering methods
78    std::string getFilePath(const std::string &prompt,
79                            const std::string &description,
80                            const std::string &placeholder, bool
       mustExist,
81                            bool optional,
82                            std::vector<std::string> fileExtensions);
83    double getNumberInput(const std::string &prompt,
84                          const std::string &description, double min,
       double max,
85                          const std::string &placeholder);
86    int getIntInput(const std::string &prompt, const std::string
       &description,
87                    int min, int max, const std::string &placeholder);
88
89    // Progress indicators
90    void startSpinner(const std::string &message);
91    void updateSpinnerMessage(const std::string &message);
92    void updateSpinnerProgress(double progress);
93    void stopSpinner(bool success, const std::string &message);
94
95    // Process execution
96    CompressionResult processImage();
97
98    // Navigation
99    bool goToPreviousInput();
100   void showHelpMenu();
101
102   // State variables
103   struct InputState {
104     std::string inputFilePath;
105     int errorMethod;
106     double threshold;
107     int minBlockSize;
108     double compressionTarget;
109     std::string outputFilePath;
110     std::string gifOutputPath;
111     int currentInputStep;
112   };
113
114   InputState state;
115   std::vector<std::string> inputPromptHistory;
116
```

```
117   // Spinner related
118   std::atomic<bool> spinnerActive;
119   std::unique_ptr<std::thread> spinnerThread;
120   std::mutex spinnerMutex;
121   std::string spinnerMessage;
122
123   // Terminal state
124 #ifdef _WIN32
125   HANDLE hStdin;
126   HANDLE hStdout;
127   DWORD oldConsoleMode;
128 #else
129   struct termios oldTermios;
130 #endif
131
132   // Braille spinner frames
133   static const std::vector<std::string> SPINNER_FRAMES;
134 };
```

Program 3.19. cli.h

3.6.2 cli.cpp

```
1  #include "cli.h"
2  #include "controller/compression_controller.h"
3  #include "error_measurement/emm_entropy.h"
4  #include "error_measurement/emm_mad.h"
5  #include "error_measurement/emm_mpd.h"
6  #include "error_measurement/emm_ssim.h"
7  #include "error_measurement/emm_variance.h"
8  #include "utils/style.h"
9  #include <algorithm>
10 #include <climits>
11 #include <cstddef>
12 #include <filesystem>
13 #include <iomanip>
14 #include <sstream>
15
16 #include <condition_variable>
17 #include <csignal>
18 #include <cstdlib>
19 #include <string>
20
21 // handle SIGINT
22 void sigintHandler(int signum) {
23   std::cout << CLEAR_SCREEN << "Interrupt received. Exiting...\n";
24   std::exit(signum);
25 }
26
27 std::condition_variable spinnerCv;
```

```
28 std::mutex spinnerCvMutex;
29 namespace fs = std::filesystem;
30
31 const std::vector<std::string> CLI::SPINNER_FRAMES = {"    ", "    ",
       "    ", "    ", "    ",
32                                                      "    ", "    ",
       "    ", "    ", "    "};
33
34 CLI::CLI() : spinnerActive(false) {
35   setupTerminal();
36   state.currentInputStep = 0;
37 }
38
39 CLI::~CLI() {
40   if (spinnerActive) {
41     spinnerActive = false;
42     if (spinnerThread && spinnerThread->joinable()) {
43       spinnerThread->join();
44     }
45   }
46   resetTerminal();
47 }
48
49 void CLI::setupTerminal() {
50 #ifdef _WIN32
51   // windows terminal setup
52   hStdin = GetStdHandle(STD_INPUT_HANDLE);
53   hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
54   GetConsoleMode(hStdin, &oldConsoleMode);
55   SetConsoleOutputCP(CP_UTF8);
56
57   // for the output handle, enable virtual terminal processing
58   DWORD outMode = 0;
59   GetConsoleMode(hStdout, &outMode);
60   SetConsoleMode(hStdout, outMode | ENABLE_VIRTUAL_TERMINAL_PROCESSING);
61
62   // for input, retrieve the current mode
63   DWORD inMode = 0;
64   GetConsoleMode(hStdin, &inMode);
65   // Make sure ENABLE_PROCESSED_INPUT is set to process CTRL+C
66   // and ENABLE_QUICK_EDIT_MODE is disabled
67   SetConsoleMode(hStdin,
68                  (inMode | ENABLE_PROCESSED_INPUT) &
       ~ENABLE_QUICK_EDIT_MODE);
69 #else
70   // unix terminal setup
71   struct termios newTermios;
72   tcgetattr(STDIN_FILENO, &oldTermios);
73   newTermios = oldTermios;
74   newTermios.c_lflag &= ~(ICANON | ECHO);
75   tcsetattr(STDIN_FILENO, TCSANOW, &newTermios);
```

```
76  #endif
77    // clear screen and show cursor
78    std::cout << "\033[?25h" << std::flush;
79  }
80
81  void CLI::resetTerminal() {
82  #ifdef _WIN32
83    SetConsoleMode(hStdin, oldConsoleMode);
84  #else
85    tcsetattr(STDIN_FILENO, TCSANOW, &oldTermios);
86  #endif
87    // show cursor
88    std::cout << "\033[?25h" << std::flush;
89  }
90
91  void CLI::clearScreen() { std::cout << FULL_CLEAR_SCREEN << std::flush;
       }
92  void CLI::clearVisibleScreen() { std::cout << CLEAR_SCREEN <<
       std::flush; }
93
94  void CLI::clearLine() { std::cout << "\r\033[K" << std::flush; }
95
96  std::string CLI::doubleToString(double value, int precision = 2) {
97    std::ostringstream oss;
98    oss << std::fixed << std::setprecision(precision) << value;
99    return oss.str();
100 }
101
102 void CLI::moveCursorUp(int lines) {
103   std::cout << "\033[" << lines << "A" << std::flush;
104 }
105
106 void CLI::moveCursorToColumn(int col) {
107   std::cout << "\033[" << col << "G" << std::flush;
108 }
109
110 std::pair<int, int> CLI::getTerminalSize() {
111 #ifdef _WIN32
112   CONSOLE_SCREEN_BUFFER_INFO csbi;
113   GetConsoleScreenBufferInfo(hStdout, &csbi);
114   return {csbi.srWindow.Right - csbi.srWindow.Left + 1,
115           csbi.srWindow.Bottom - csbi.srWindow.Top + 1};
116 #else
117   struct winsize w;
118   ioctl(STDOUT_FILENO, TIOCGWINSZ, &w);
119   return {w.ws_col, w.ws_row};
120 #endif
121 }
122
123 char CLI::getChar() {
124 #ifdef _WIN32
```

```
125   return _getch();
126 #else
127   char buf = 0;
128   if (read(STDIN_FILENO, &buf, 1) < 0) {
129     return -1;
130   }
131   return buf;
132 #endif
133 }
134
135 int CLI::getCharWithTimeout(int timeout_ms) {
136 #ifdef _WIN32
137   // on Windows, we can use _kbhit() in a loop with a sleep to check
       for input
138   auto start = std::chrono::steady_clock::now();
139   while (!_kbhit()) {
140     auto now = std::chrono::steady_clock::now();
141     if (std::chrono::duration_cast<std::chrono::milliseconds>(now -
       start)
142             .count() >= timeout_ms) {
143       return -1; // timeout reached, no input available
144     }
145     Sleep(1); // sleep for 1 ms to reduce CPU usage
146   }
147   return _getch(); // input is available, return the character
148 #else
149   fd_set set;
150   FD_ZERO(&set);
151   FD_SET(STDIN_FILENO, &set);
152
153   struct timeval timeout;
154   timeout.tv_sec = timeout_ms / 1000;
155   timeout.tv_usec = (timeout_ms % 1000) * 1000;
156
157   int rv = select(STDIN_FILENO + 1, &set, nullptr, nullptr, &timeout);
158   if (rv <= 0) {
159     // timeout or error: no data available
160     return -1;
161   }
162   // data is available, so use your getChar() function
163   return getChar();
164 #endif
165 }
166
167 int CLI::handleArrowKeys() {
168 #ifdef _WIN32
169   char c = getChar();
170   if (c == 3) {
171     return -2;
172   }
173   switch (c) {
```

```
174     case 72:
175       return 1; // up
176     case 80:
177       return 2; // down
178     case 75:
179       return 3; // left
180     case 77:
181       return 4; // right
182     default:
183       return c;
184     }
185   #else
186     char c = getChar();
187     if (c == 27) { // ESC key detected
188       int nextChar = getCharWithTimeout(50);
189       if (nextChar == -1) {
190         return 27;
191       }
192       if (nextChar == '[') {
193         // Handle escape sequences for arrow keys or other keys
194         switch (getChar()) {
195         case 'A':
196           return 1;
197         case 'B':
198           return 2;
199         case 'D':
200           return 3;
201         case 'C':
202           return 4;
203         default:
204           return 0;
205         }
206       }
207       return 0;
208     }
209     return c;
210   #endif
211   }
212
213   std::string CLI::readInput(const std::string &placeholder) {
214     std::string input;
215     bool firstDraw = true;
216     bool inputActive = true;
217     std::cout << SHOW_CURSOR << BLOCK_CURSOR;
218     std::cout << RESET << BRIGHT_BLACK << placeholder << RESET;
219     moveCursorToColumn(5); // move back to the start position after '>'
220
221     while (inputActive) {
222       int c = handleArrowKeys();
223
224       if (c == -2) {
```

```
225         // call the SIGINT handler directly
226         sigintHandler(SIGINT);
227         return "";          // this line won't be reached if
        sigintHandler exits
228       } else if (c == 27) { // ESC
229         goToPreviousInput();
230         return "BACK";
231       } else if (c == 13 || c == 10) { // enter
232         std::cout << std::endl;
233         inputActive = false;
234       } else if (c == 127 || c == 8) { // backspace
235         if (!input.empty()) {
236           input.pop_back();
237           clearLine();
238
239           std::cout << BRIGHT_MAGENTA << "  > " << RESET << input;
240           if (input.empty() && !placeholder.empty()) {
241             std::cout << BRIGHT_BLACK << placeholder << RESET;
242             firstDraw = true;
243             moveCursorToColumn(5);
244           }
245           std::cout << std::flush;
246         }
247       } else if (c >= 32 && c <= 126) { // printable ASCII
248         // clear placeholder if this is the first character
249         if (input.empty() && firstDraw && !placeholder.empty()) {
250           clearLine();
251           std::cout << BRIGHT_MAGENTA << "  > " << RESET;
252           firstDraw = false;
253         }
254         input += static_cast<char>(c);
255         std::cout << static_cast<char>(c) << std::flush;
256       }
257     }
258
259     return input;
260   }
261
262   void CLI::printTitle(const std::string &title) {
263     std::string paddedTitle = "  " + title + " ";
264
265     std::cout << BOLD << paddedTitle << std::endl << std::endl;
266   }
267
268   void CLI::printHeader(const std::string &header) {
269     std::cout << BOLD << header << RESET << std::endl << std::endl;
270   }
271
272   void CLI::printPrompt(const std::string &prompt, bool error) {
273     std::cout << "  ";
274     if (error) {
```

```
275       std::cout << RED << ITALIC << "* " << RESET;
276     }
277     std::cout << prompt << std::endl;
278   }
279
280   void CLI::printHint(const std::string &hint) {
281     clearLine();
282     std::cout << BRIGHT_BLACK << "  " << hint << RESET << std::endl;
283   }
284
285   void CLI::printKeybind(std::vector<std::pair<std::string, std::string>>
          keys) {
286     std::cout << "  ";
287     for (int i = 0; (size_t)i < keys.size(); i++) {
288       std::cout << BRIGHT_BLACK;
289       if (i) {
290         std::cout << "    ";
291       }
292       std::cout << RESET << DIM << keys[i].first << RESET << BRIGHT_BLACK
          << " "
293                 << keys[i].second << RESET;
294     }
295     std::cout << std::endl;
296   }
297
298   void CLI::printErrorMessage(const std::string &message) {
299     clearLine();
300     std::cout << std::endl;
301     clearLine();
302     std::cout << RED << ITALIC << "  " << message << RESET << std::endl;
303   }
304
305   void CLI::printPreviousInputs() {
306     for (size_t i = 0; i < inputPromptHistory.size(); i++) {
307       std::cout << RESET << "  " << inputPromptHistory[i] << RESET <<
          std::endl;
308     }
309     std::cout << std::endl;
310   }
311
312   bool CLI::goToPreviousInput() {
313     if (state.currentInputStep > 0) {
314       state.currentInputStep--;
315       if (!inputPromptHistory.empty()) {
316         inputPromptHistory.pop_back();
317       }
318       return true;
319     }
320     return false;
321   }
322
```

```
323 std::string CLI::getFilePath(const std::string &prompt,
324                                const std::string &description,
325                                const std::string &placeholder, bool
       mustExist,
326                                bool optional,
327                                std::vector<std::string> fileExtensions) {
328   bool isError = false;
329   while (true) {
330     clearLine();
331     printPrompt(prompt, isError);
332     std::vector<std::pair<std::string, std::string>> keybinds = {
333         {"ENTER", "confirm"}};
334 
335     // im just lazy to differentiate file input and output
336     if (!mustExist) {
337       keybinds.push_back({"ESCAPE", "back"});
338     }
339     std::cout << "\n";
340     clearLine();
341     std::cout << "\n";
342     clearLine();
343     std::cout << "\n";
344     clearLine();
345     printKeybind(keybinds);
346     moveCursorUp(4);
347 
348     printHint(description);
349 
350     clearLine();
351     std::cout << BRIGHT_MAGENTA << "  > " << RESET;
352     std::string path = readInput(placeholder);
353 
354     if (path.empty()) {
355       if (optional) {
356         return path;
357       }
358       moveCursorUp(4 + isError);
359       isError = true;
360       printErrorMessage("Path cannot be empty");
361       continue;
362     }
363 
364     if (path == "BACK") {
365       std::cout << "\n";
366       clearLine();
367       std::cout << "\n";
368       clearLine();
369 
370       moveCursorUp(4 + isError);
371       if (mustExist)
372         continue;
```

```
373         return "BACK";
374       }
375
376       // check existence or directory validity
377       if (mustExist) {
378         if (!fs::exists(path)) {
379           moveCursorUp(4 + isError);
380           isError = true;
381           printErrorMessage("File doesn't exist");
382           continue;
383         }
384       } else {
385         // for non-required files, check that the parent directory exists
386         fs::path dirPath = fs::path(path).parent_path();
387         if (!dirPath.empty() && !fs::exists(dirPath)) {
388           moveCursorUp(4 + isError);
389           isError = true;
390           printErrorMessage("Output directory does not exist");
391           continue;
392         }
393       }
394
395       // check file extension if a list is provided
396       if (!fileExtensions.empty()) {
397         std::string ext = fs::path(path).extension().string();
398         std::transform(ext.begin(), ext.end(), ext.begin(),
399                        [](unsigned char c) { return std::tolower(c); });
400         bool validExt = false;
401         for (const auto &allowedExt : fileExtensions) {
402           std::string lowerAllowed = allowedExt;
403           std::transform(lowerAllowed.begin(), lowerAllowed.end(),
404                          lowerAllowed.begin(),
405                          [](unsigned char c) { return std::tolower(c); });
406           if (ext == lowerAllowed) {
407             validExt = true;
408             break;
409           }
410         }
411         if (!validExt) {
412           std::string errMsg;
413           if (fileExtensions.size() == 1) {
414             errMsg = "File extension must be " + fileExtensions[0];
415           } else {
416             errMsg = "File extension must be ";
417             for (size_t i = 0; i < fileExtensions.size(); i++) {
418               errMsg += fileExtensions[i];
419               if (i < fileExtensions.size() - 2) {
420                 errMsg += ", ";
421               } else if (i == fileExtensions.size() - 2) {
422                 errMsg += " or ";
423               }
```

```
424             }
425           }
426           moveCursorUp(4 + isError);
427           isError = true;
428           printErrorMessage(errMsg);
429           continue;
430         }
431       }
432
433       return path;
434     }
435   }
436
437   double CLI::getNumberInput(const std::string &prompt,
438                              const std::string &description, double min,
439                              double max, const std::string &placeholder) {
440     bool isError = false;
441     while (true) {
442       clearLine();
443       printPrompt(prompt, isError);
444       std::vector<std::pair<std::string, std::string>> keybinds = {
445           {"ENTER", "confirm"}, {"ESCAPE", "back"}};
446       clearLine();
447       std::cout << "\n";
448       clearLine();
449       std::cout << "\n";
450       clearLine();
451       std::cout << "\n";
452       clearLine();
453       printKeybind(keybinds);
454       moveCursorUp(4);
455
456       printHint(description);
457       std::cout << BRIGHT_MAGENTA << "  > " << RESET;
458       std::string input = readInput(placeholder);
459
460       if (input == "BACK") {
461         clearScreen();
462         printTitle("QuadTree Image Compression");
463         printPreviousInputs();
464         return -1;
465       }
466
467       if (input.empty()) {
468         moveCursorUp(4 + isError);
469         isError = true;
470         printErrorMessage("Input cannot be empty");
471         continue;
472       }
473
474       try {
```

```
475         double value = std::stod(input);
476         if (value < min || value > max) {
477           moveCursorUp(4 + isError);
478           isError = true;
479           std::ostringstream errorMsg;
480           errorMsg << "Value must be between " << min << " and " << max;
481           printErrorMessage(errorMsg.str());
482           continue;
483         }
484         return value;
485       } catch (const std::exception &) {
486         moveCursorUp(4 + isError);
487         isError = true;
488         printErrorMessage("Invalid number format");
489       }
490     }
491   }
492
493   int CLI::getIntInput(const std::string &prompt, const std::string
        &description,
494                        int min, int max, const std::string &placeholder) {
495     bool isError = false;
496     while (true) {
497       clearLine();
498       printPrompt(prompt, isError);
499       std::vector<std::pair<std::string, std::string>> keybinds = {
500           {"ENTER", "confirm"}, {"ESCAPE", "back"}};
501       clearLine();
502       std::cout << "\n";
503       clearLine();
504       std::cout << "\n";
505       clearLine();
506       std::cout << "\n";
507       clearLine();
508       printKeybind(keybinds);
509       moveCursorUp(4);
510
511       printHint(description);
512       std::cout << BRIGHT_MAGENTA << "  > " << RESET;
513       std::string input = readInput(placeholder);
514
515       if (input == "BACK") {
516         clearScreen();
517         printTitle("QuadTree Image Compression");
518         printPreviousInputs();
519         return -1;
520       }
521
522       if (input.empty()) {
523         moveCursorUp(4 + isError);
524         isError = true;
```

```
525       printErrorMessage("Input cannot be empty");
526       continue;
527     }
528
529     try {
530       int value = std::stoi(input);
531       if (value < min || value > max) {
532         moveCursorUp(4 + isError);
533         isError = true;
534         std::ostringstream errorMsg;
535         if (max != INT_MAX) {
536           errorMsg << "Value must be between " << min << " and " << max;
537         } else {
538           errorMsg << "Value must be at least 1";
539         }
540         printErrorMessage(errorMsg.str());
541         continue;
542       }
543       return value;
544     } catch (const std::exception &) {
545       moveCursorUp(4 + isError);
546       isError = true;
547       printErrorMessage("Invalid number format");
548     }
549   }
550 }
551
552 template <typename T>
553 T CLI::selectOption(const std::string &prompt,
554                     const std::vector<std::pair<std::string, T>>
      &options) {
555   printPrompt(prompt, false);
556
557   size_t selected = 0;
558   bool selecting = true;
559
560   while (selecting) {
561     // print all options
562     clearLine();
563     for (size_t i = 0; i < options.size(); i++) {
564       clearLine();
565       if (i == selected) {
566         std::cout << "  " << BRIGHT_MAGENTA << "> " << RESET << BOLD
567                   << BRIGHT_GREEN << options[i].second << ". "
568                   << options[i].first << RESET << std::endl;
569       } else {
570         std::cout << "    " << options[i].second << ". " <<
      options[i].first
571                   << std::endl;
572       }
573     }
```

```
574
575     std::vector<std::pair<std::string, std::string>> keybinds = {
576         {"   ", "up"},
577         {"   ", "down"},
578         {"1-" + std::to_string((int)options.size()), "jump to"},
579         {"ENTER", "confirm"},
580         {"ESCAPE", "back"}};
581
582     std::cout << "\n" << HIDE_CURSOR;
583     printKeybind(keybinds);
584
585     int key = handleArrowKeys();
586     if (key == -2) {
587       // call the SIGINT handler directly
588       sigintHandler(SIGINT);
589       return 0;          // this line won't be reached if sigintHandler
      exits
590     } else if (key == 1) {
591       if (selected > 0)
592         selected--;
593       moveCursorUp(options.size() + 2);
594     } else if (key == 2) {
595       if (selected < options.size() - 1)
596         selected++;
597       moveCursorUp(options.size() + 2);
598     } else if (key == 13 || key == 10) {
599       selecting = false;
600     } else if (key >= '1' && key <= '9') {
601       int index = key - '1';
602       if (index >= 0 && index < static_cast<int>(options.size())) {
603         selected = index;
604       }
605       moveCursorUp(options.size() + 2);
606     } else if (key == 27) {
607       if (goToPreviousInput()) {
608         clearScreen();
609         printTitle("QuadTree Image Compression");
610         printPreviousInputs();
611         return 0;
612       }
613       moveCursorUp(options.size() + 2);
614     } else {
615       moveCursorUp(options.size() + 2);
616     }
617   }
618
619   std::cout << RESET << std::endl;
620   return options[selected].second;
621 }
622
623 void CLI::startSpinner(const std::string &message) {
```

```
624   if (spinnerActive) {
625     stopSpinner(true, "Previous operation completed");
626   }
627
628   spinnerActive = true;
629   spinnerMessage = message;
630   spinnerStartTime = std::chrono::steady_clock::now();
631
632   spinnerThread = std::make_unique<std::thread>([this]() {
633     int frame = 0;
634     std::unique_lock<std::mutex> cvLock(spinnerCvMutex,
      std::defer_lock);
635     while (spinnerActive) {
636       std::string currentMessage;
637       {
638         std::lock_guard<std::mutex> lock(spinnerMutex);
639         currentMessage = spinnerMessage;
640       }
641
642       auto now = std::chrono::steady_clock::now();
643       double elapsedMs =
644           std::chrono::duration<double, std::milli>(now -
      spinnerStartTime)
645               .count();
646       std::string timeStr;
647       if (elapsedMs < 1000) {
648         timeStr = doubleToString(elapsedMs) + " ms";
649       } else {
650         timeStr = doubleToString(elapsedMs / 1000.0).substr(0, 4) + "
      s";
651       }
652
653       clearLine();
654       std::cout << "  " << CYAN << SPINNER_FRAMES[frame] << RESET << " "
655                 << currentMessage << " " << BRIGHT_BLACK << "(" <<
      timeStr
656                 << ")" << RESET << std::flush;
657
658       frame = (frame + 1) % SPINNER_FRAMES.size();
659
660       // wait for 80ms or until notified
661       cvLock.lock();
662       spinnerCv.wait_for(cvLock, std::chrono::milliseconds(80),
663                          [this]() { return !spinnerActive; });
664       cvLock.unlock();
665     }
666   });
667 }
668
669 void CLI::stopSpinner(bool success, const std::string &message) {
670   if (!spinnerActive)
```

```
671       return;
672
673     spinnerActive = false;
674     // notify the spinner thread to wake up immediately
675     spinnerCv.notify_all();
676     if (spinnerThread && spinnerThread->joinable()) {
677       spinnerThread->join();
678     }
679
680     // reasure elapsed time with high precision
681     auto now = std::chrono::steady_clock::now();
682     double elapsedMs =
683         std::chrono::duration<double, std::milli>(now -
       spinnerStartTime).count();
684     totalElapsedTime += elapsedMs;
685
686     std::string timeStr;
687     if (elapsedMs < 1000) {
688       timeStr = doubleToString(elapsedMs) + " ms";
689     } else {
690       timeStr = doubleToString(elapsedMs / 1000.0).substr(0, 4) + " s";
691     }
692
693     clearLine();
694     if (success) {
695       std::cout << "  " << GREEN << "   " << RESET << " " << message;
696     } else {
697       std::cout << "  " << RED << "   " << RESET << " " << message;
698     }
699     std::cout << " " << BRIGHT_BLACK << "(" << timeStr << ")" << RESET
700               << std::endl;
701   }
702
703   void CLI::updateSpinnerMessage(const std::string &message) {
704     std::lock_guard<std::mutex> lock(spinnerMutex);
705     spinnerMessage = message;
706   }
707   /*
708   void CLI::updateSpinnerProgress(double progress) {
709     // if i even need, i could add progress bar functionality here
710   }
711   */
712   CompressionResult CLI::processImage() {
713
714     CompressionResult result;
715     auto stageInfo =
716         [](ProgressStage stage) -> std::pair<std::string, std::string> {
717       switch (stage) {
718       case ProgressStage::Loading:
719         return {"Loading source image", "Source image loaded"};
720       case ProgressStage::Precompute:
```

```
721         return {"Precomputing summed area table", "Summed area table
        computed"};
722       case ProgressStage::FindingTarget:
723         return {"Finding target compression",
724                 "Target compression achieved with "};
725       case ProgressStage::BuildingTree:
726         return {"Building quadtree", "Building quadtree finished"};
727       case ProgressStage::TransformingImage:
728         return {"Compressing image", "Image compressed"};
729       case ProgressStage::SavingImage:
730         return {"Saving compressed image", "Compressed image saved"};
731       case ProgressStage::CreatingGif:
732         return {"Creating GIF", "GIF saved"};
733       case ProgressStage::Finished:
734         return {"Finishing", "Finished"};
735       }
736       return {"Processing", "Processing finished"};
737     };
738
739     ProgressStage lastStage = ProgressStage::Loading;
740     bool firstCall = true;
741
742     bool runResult = compression.run([&](const ProgressStage &stage) {
743       auto [startMsg, stopMsg] = stageInfo(stage);
744
745       if (!firstCall) {
746         if (lastStage == ProgressStage::FindingTarget) {
747           stopSpinner(true, stageInfo(lastStage).second +
748
        doubleToString(compression.getThreshold()) +
749                                  " threshold");
750         } else {
751
752           stopSpinner(true, stageInfo(lastStage).second);
753         }
754       } else {
755         firstCall = false;
756       }
757
758       if (stage != ProgressStage::Finished) {
759         startSpinner(startMsg);
760       }
761
762       lastStage = stage;
763     });
764
765     if (runResult) {
766       return compression.getResult();
767     }
768     return result;
769 }
```

```
770
771 CompressionResult CLI::run() {
772   std::signal(SIGINT, sigintHandler);
773   clearVisibleScreen();
774   printTitle("QuadTree Image Compression");
775
776   if (state.currentInputStep == 0) {
777     std::string filePath = getFilePath(
778         "Enter input image file path", "Path could be absolute or
      relative",
779         "path/to/image.jpg", true, false,
780         {".jpeg", ".png", ".jpg", ".bmp", ".hdr", ".tga"});
781     state.inputFilePath = filePath;
782     compression.setInputPath(filePath);
783     inputPromptHistory.push_back("Input image: " +
     compression.getInputPath());
784     state.currentInputStep++;
785   }
786
787   if (state.currentInputStep == 1) {
788     std::vector<std::pair<std::string, int>> errorMethods = {
789         {"Variance", 1},
790         {"Mean Absolute Deviation (MAD)", 2},
791         {"Max Pixel Differece (MPD)", 3},
792         {"Entropy", 4},
793         {"Structural Similarity Index Measure (SSIM)", 5},
794     };
795
796     clearScreen();
797     printTitle("QuadTree Image Compression");
798     printPreviousInputs();
799
800     int method =
801         selectOption<int>("Select error measurement method",
      errorMethods);
802     if (method == 0)
803       return run();
804
805     state.errorMethod = method;
806     switch (method) {
807     case 1:
808       compression.setErrorMethod(new EMM::Variance());
809       break;
810     case 2:
811       compression.setErrorMethod(new EMM::MeanAbsoluteDeviation());
812       break;
813     case 3:
814       compression.setErrorMethod(new EMM::MaximumPixelDifference());
815       break;
816     case 4:
817       compression.setErrorMethod(new EMM::Entropy());
```

```
818         break;
819       case 5:
820         compression.setErrorMethod(new
        EMM::StructuralSimilarityIndexMeasure());
821         break;
822       }
823       inputPromptHistory.push_back("Error method: " +
824                                    errorMethods[method - 1].first);
825       state.currentInputStep++;
826     }
827
828     if (state.currentInputStep == 2) {
829       clearScreen();
830       printTitle("QuadTree Image Compression");
831       printPreviousInputs();
832
833       double lower = compression.getErrorMethod()->getLowerBound();
834       double upper = compression.getErrorMethod()->getUpperBound();
835       double threshold =
836           getNumberInput("Enter error threshold",
837                          "Threshold is in range " + doubleToString(lower)
        + "-" +
838                              doubleToString(upper),
839                          lower, upper, "");
840       if (threshold < 0)
841         return run();
842
843       state.threshold = threshold;
844       compression.setThreshold(threshold);
845       inputPromptHistory.push_back("Threshold: " +
        doubleToString(threshold));
846       state.currentInputStep++;
847     }
848
849     if (state.currentInputStep == 3) {
850       clearScreen();
851       printTitle("QuadTree Image Compression");
852       printPreviousInputs();
853
854       int blockSize =
855           getIntInput("Enter minimum block size (pixels )",
856                       "Block size is in range 1-INT_MAX (image not loaded
        yet)",
857                       1, INT_MAX, "");
858       if (blockSize < 0)
859         return run();
860
861       state.minBlockSize = blockSize;
862       compression.setMinBlockSize(blockSize);
863       inputPromptHistory.push_back(
864           "Min block size: " + std::to_string(blockSize) + " pixels ");
```

```
865       state.currentInputStep++;
866     }
867
868     if (state.currentInputStep == 4) {
869       clearScreen();
870       printTitle("QuadTree Image Compression");
871       printPreviousInputs();
872
873       double target =
874           getNumberInput("Enter compression target",
875                          "0 for no target and 1.0 for 100%", 0.0, 1.0,
       "");
876       if (target < 0)
877         return run();
878
879       state.compressionTarget = target;
880       compression.setTargetCompression(target);
881       if (target == 0) {
882         inputPromptHistory.push_back("Compression target: None");
883       } else {
884         inputPromptHistory.push_back(
885             "Compression target: " + doubleToString(target * 100) + "%");
886       }
887       state.currentInputStep++;
888     }
889
890     if (state.currentInputStep == 5) {
891       clearScreen();
892       printTitle("QuadTree Image Compression");
893       printPreviousInputs();
894
895       std::string outputPath = getFilePath(
896           "Enter output file path", "Path could be absolute or relative",
897           "path/to/output.png", false, false, {compression.getFileExt()});
898       if (outputPath == "BACK")
899         return run();
900
901       state.outputFilePath = outputPath;
902       if (!compression.setOutputPath(outputPath)) {
903         std::cerr << "OUTPUTPATH WRONG " << std::endl;
904       }
905       inputPromptHistory.push_back("Output file: " +
      compression.getOutputPath());
906       state.currentInputStep++;
907     }
908
909     if (state.currentInputStep == 6) {
910       clearScreen();
911       printTitle("QuadTree Image Compression");
912       printPreviousInputs();
913
```

```
914       std::string gifPath = getFilePath(
915           "Enter GIF visualization path (or leave empty to skip)",
916           "Path could be absolute or relative",
917           "path/to/visualization.gif (optional)", false, true, {".gif"});
918       if (gifPath == "BACK")
919         return run();
920
921       state.gifOutputPath = gifPath;
922       compression.setGifOutputPath(gifPath);
923       if (gifPath.empty()) {
924         inputPromptHistory.push_back("GIF visualization: None");
925       } else {
926         inputPromptHistory.push_back("GIF visualization: " +
927                                      compression.getGifOutputPath());
928       }
929       state.currentInputStep++;
930     }
931
932     clearScreen();
933     printTitle("QuadTree Image Compression");
934     printPreviousInputs();
935
936     std::cout << BOLD << "  Processing image..." << RESET << std::endl
937               << std::endl;
938     CompressionResult result = processImage();
939
940     std::cout << std::endl;
941     // clearScreen();
942     printTitle("Compression Results");
943
944     const int labelWidth = 25;
945     auto formatSize = [](size_t size) -> std::string {
946       std::ostringstream oss;
947       if (size < 1024) {
948         oss << size << " B";
949       } else if (size < 1024 * 1024) {
950         oss << std::fixed << std::setprecision(2) << (size / 1024.0) << "
       KB";
951       } else {
952         oss << std::fixed << std::setprecision(2) << (size / (1024.0 *
       1024.0))
953             << " MB";
954       }
955       return oss.str();
956     };
957
958     std::string timeStr;
959     if (totalElapsedTime < 1000) {
960       timeStr = doubleToString(totalElapsedTime) + " ms";
961     } else {
```

```
962       timeStr = doubleToString(totalElapsedTime / 1000.0).substr(0, 4) +
      " s";
963     }
964
965     std::cout << BOLD << "  Performance Metrics" << RESET << std::endl;
966     std::cout << "  " << std::left << std::setw(labelWidth)
967               << "Computation Time:" << timeStr << std::endl;
968
969     std::cout << std::endl << BOLD << "  File Information" << RESET <<
      std::endl;
970     std::cout << "  " << std::left << std::setw(labelWidth)
971               << "Original Size:" << formatSize(result.originalFileSize)
972               << std::endl;
973     std::cout << "  " << std::left << std::setw(labelWidth)
974               << "Compressed Size:" <<
      formatSize(result.compressedFileSize)
975               << std::endl;
976     std::cout << "  " << std::left << std::setw(labelWidth)
977               << "Compression Ratio:" << std::fixed <<
      std::setprecision(2)
978               << result.compressionPercentage << "%" << std::endl;
979
980     std::cout << std::endl
981               << BOLD << "  Quadtree Statistics" << RESET << std::endl;
982     std::cout << "  " << std::left << std::setw(labelWidth)
983               << "Tree Depth:" << result.quadtreeDepth << std::endl;
984     std::cout << "  " << std::left << std::setw(labelWidth)
985               << "Node Count:" << result.quadtreeNodeCount << std::endl;
986
987     if (compression.getTargetCompression()) {
988
989       std::cout << "  " << std::left << std::setw(labelWidth)
990                 << "Target Threshold:"
991                 << std::to_string(compression.getThreshold()) <<
      std::endl;
992     }
993
994     std::cout << std::endl << BOLD << "  Output Files" << RESET <<
      std::endl;
995     std::cout << "  " << std::left << std::setw(labelWidth)
996               << "Compressed File:" << result.outputFilePath << std::endl;
997
998     if (!result.gifOutputPath.empty()) {
999       std::cout << "  " << std::left << std::setw(labelWidth)
1000                << "Visualization GIF:" << result.gifOutputPath <<
      std::endl;
1001    }
1002
1003    std::cout << std::endl;
1004    std::cout << GREEN << "  Compression completed successfully!" << RESET
1005              << std::endl;
```

```
1006    std::cout << std::endl;
1007    return result;
1008  }
```

Program 3.20. cli.cpp

3.7 Main

3.7.1 `main.cpp`

```
1  #include "cli/cli.h"
2
3  int main() {
4    CLI cli;
5    cli.run();
6    return 0;
7  }
```

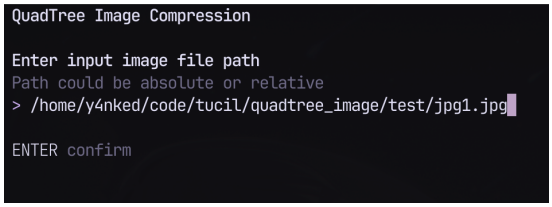
Program 3.21. main.cpp

Bab 4

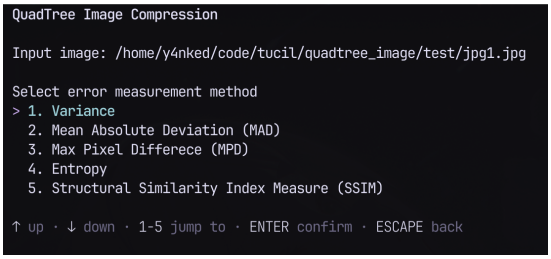
Uji Kasus

4.1 Pengujian Input

4.1.1 Pengujian Path Absolute Linux



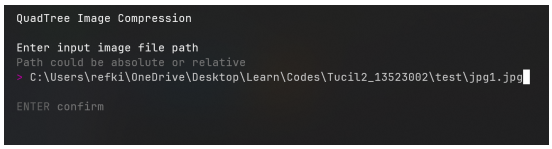
(a) Gambar Input



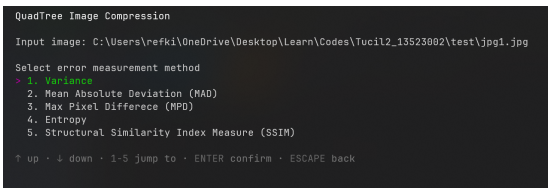
(b) Gambar Output

Gambar 4.1. Perbandingan Gambar Input dan Output untuk **Pengujian Path Absolute Linux**.

4.1.2 Pengujian Path Absolute Windows



(a) Gambar Input

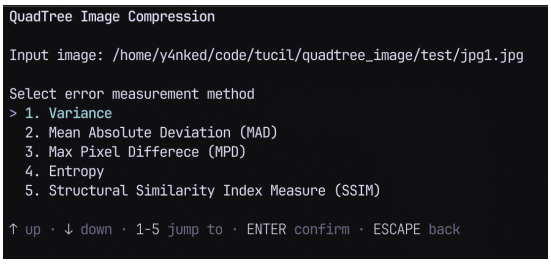


(b) Gambar Output

Gambar 4.2. Perbandingan Gambar Input dan Output untuk **Pengujian Path Absolute Windows**.

4.1.3 Pengujian Path Relative

(a) Gambar Input

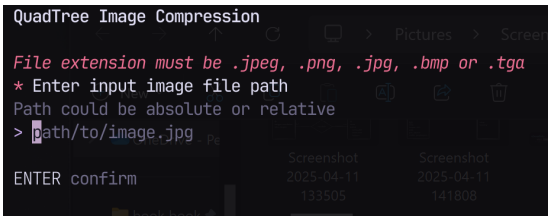


(b) Gambar Output

Gambar 4.3. Perbandingan Gambar Input dan Output untuk **Pengujian Path Relative**.

4.1.4 Pengujian Path Tidak Ada atau Salah

(a) Gambar Input



(b) Gambar Output

Gambar 4.4. Perbandingan Gambar Input dan Output untuk **Pengujian Path Tidak Ada atau Salah**.

4.1.5 Pengujian Threshold di Luar Batas

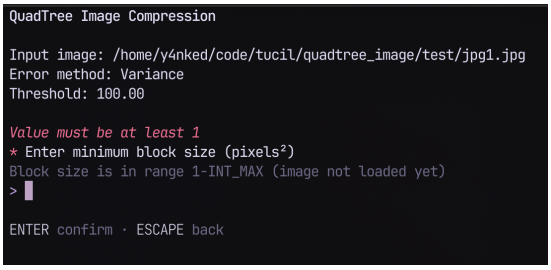
(a) Gambar Input

(b) Gambar Output

Gambar 4.5. Perbandingan Gambar Input dan Output untuk **Pengujian Threshold di Luar Batas**.

4.1.6 Pengujian Minimum Block Size di Luar Batas

(a) Gambar Input

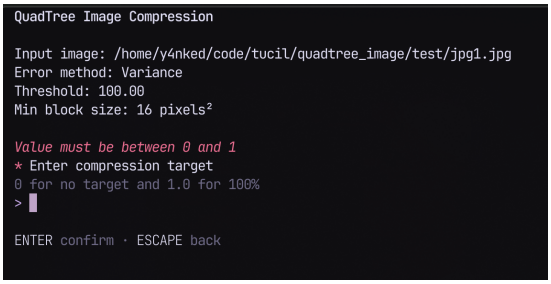


(b) Gambar Output

Gambar 4.6. Perbandingan Gambar Input dan Output untuk **Pengujian Minimum Block Size di Luar Batas**.

4.1.7 Pengujian Compression Target di Luar Batas

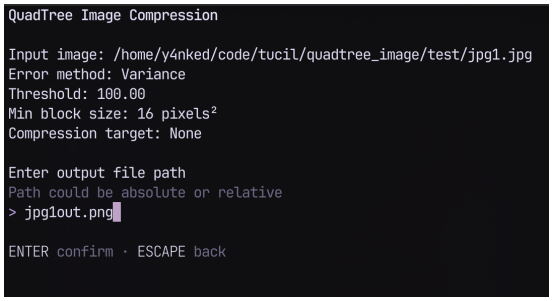
(a) Gambar Input



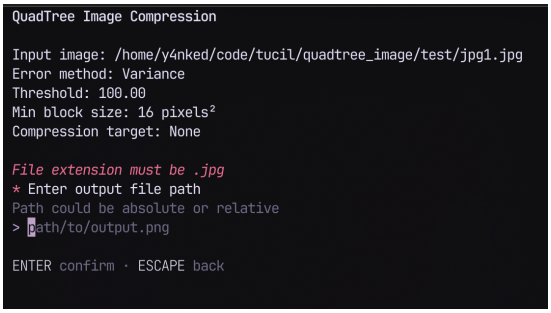
(b) Gambar Output

Gambar 4.7. Perbandingan Gambar Input dan Output untuk **Pengujian Compression Target di Luar Batas**.

4.1.8 Pengujian Output File Berbeda Ekstensi



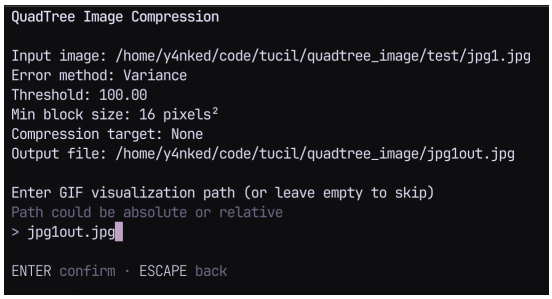
(a) Gambar Input



(b) Gambar Output

Gambar 4.8. Perbandingan Gambar Input dan Output untuk **Pengujian Output File Berbeda Ekstensi**.

4.1.9 Pengujian Output File GIF Salah Ekstensi



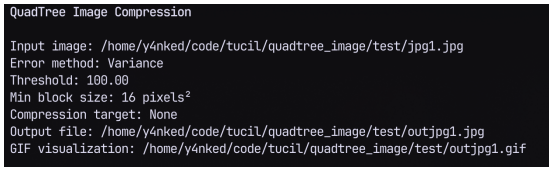
(a) Gambar Input

(b) Gambar Output

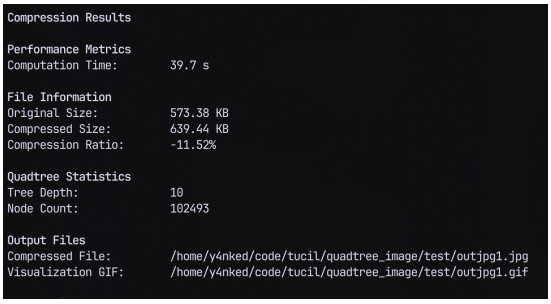
Gambar 4.9. Perbandingan Gambar Input dan Output untuk **Pengujian Output File GIF Salah Ekstensi**.

4.2 Pengujian Metode Eror

4.2.1 Metode Variansi



(a) Gambar Input



(b) Gambar Output

Gambar 4.10. Perbandingan Gambar Input dan Output untuk **Metode Variansi**.

(a) Gambar Awal

(b) Gambar Akhir

Gambar 4.11. Perbandingan Gambar Awal dan Akhir untuk **Metode Variansi**.

4.2.2 Metode Mean Absolute Deviation

```
QuadTree Image Compression

Input image: /home/y4nked/code/tucil/quadtree_image/test/jpg2.jpg
Error method: Mean Absolute Deviation (MAD)
Threshold: 10.00
Min block size: 16 pixels²
Compression target: None
Output file: /home/y4nked/code/tucil/quadtree_image/test/outjpg2.jpg
GIF visualization: /home/y4nked/code/tucil/quadtree_image/test/outjpg2.gif
```

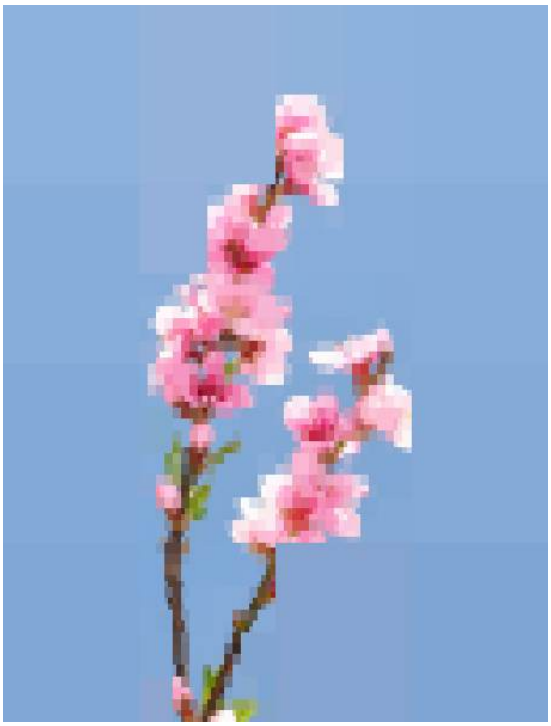
(a) Gambar Input

```
Compression Results

Performance Metrics
Computation Time:       203.04 ms

File Information
Original Size:          78.10 KB
Compressed Size:        15.62 KB
Compression Ratio:      80.00%

Quadtree Statistics
Tree Depth:             8
Node Count:             2165

Output Files
Compressed File:        /home/y4nked/code/tucil/quadtree_image/test/outjpg2.jpg
Visualization GIF:      /home/y4nked/code/tucil/quadtree_image/test/outjpg2.gif
```

(b) Gambar Output

Gambar 4.12. Perbandingan Gambar Input dan Output untuk **Metode Mean Absolute Deviation**.

(a) Gambar Awal



(b) Gambar Akhir

Gambar 4.13. Perbandingan Gambar Awal dan Akhir untuk **Metode Mean Absolute Deviation**.

4.2.3 Metode Max Pixel Difference

```
QuadTree Image Compression

Input image: /home/y4nked/code/tucil/quadtree_image/test/png1.png
Error method: Max Pixel Differece (MPD)
Threshold: 30.00
Min block size: 16 pixels²
Compression target: None
Output file: /home/y4nked/code/tucil/quadtree_image/test/outpng1.png
GIF visualization: /home/y4nked/code/tucil/quadtree_image/test/outpng1.gif
```

(a) Gambar Input

```
Compression Results

Performance Metrics
Computation Time:       1.28 s

File Information
Original Size:          993.13 KB
Compressed Size:        112.15 KB
Compression Ratio:      88.71%

Quadtree Statistics
Tree Depth:             9
Node Count:             27609

Output Files
Compressed File:        /home/y4nked/code/tucil/quadtree_image/test/outpng1.png
Visualization GIF:      /home/y4nked/code/tucil/quadtree_image/test/outpng1.gif
```

(b) Gambar Output

Gambar 4.14. Perbandingan Gambar Input dan Output untuk **Metode Max Pixel Difference**.

(a) Gambar Awal

(b) Gambar Akhir

Gambar 4.15. Perbandingan Gambar Awal dan Akhir untuk **Metode Max Pixel Difference**.

4.2.4 Metode Entropy

```
QuadTree Image Compression

Input image: /home/y4nked/code/tucil/quadtree_image/test/png2.png
Error method: Entropy
Threshold: 0.40
Min block size: 16 pixels²
Compression target: None
Output file: /home/y4nked/code/tucil/quadtree_image/test/outpng2.png
GIF visualization: /home/y4nked/code/tucil/quadtree_image/test/outpng2.gif
```

(a) Gambar Input

```
Compression Results

Performance Metrics
Computation Time:       4.61 s

File Information
Original Size:          2.08 MB
Compressed Size:        522.40 KB
Compression Ratio:      75.45%

Quadtree Statistics
Tree Depth:             10
Node Count:             113925

Output Files
Compressed File:        /home/y4nked/code/tucil/quadtree_image/test/outpng2.png
Visualization GIF:      /home/y4nked/code/tucil/quadtree_image/test/outpng2.gif
```

(b) Gambar Output

Gambar 4.16. Perbandingan Gambar Input dan Output untuk **Metode Entropy**.

(a) Gambar Awal

(b) Gambar Akhir

Gambar 4.17. Perbandingan Gambar Awal dan Akhir untuk **Metode Entropy**.

4.2.5 Metode SSIM

```
QuadTree Image Compression

Input image: /home/y4nked/code/tucil/quadtree_image/test/jpg3.jpg
Error method: Structural Similarity Index Measure (SSIM)
Threshold: 0.80
Min block size: 16 pixels²
Compression target: None
Output file: /home/y4nked/code/tucil/quadtree_image/test/outjpg3.jpg
GIF visualization: /home/y4nked/code/tucil/quadtree_image/test/outjpg3.gif
```

(a) Gambar Input

```
Compression Results

Performance Metrics
Computation Time:       826.51 ms

File Information
Original Size:          33.64 KB
Compressed Size:        28.05 KB
Compression Ratio:      16.62%

Quadtree Statistics
Tree Depth:             9
Node Count:             2985

Output Files
Compressed File:        /home/y4nked/code/tucil/quadtree_image/test/outjpg3.jpg
Visualization GIF:      /home/y4nked/code/tucil/quadtree_image/test/outjpg3.gif
```

(b) Gambar Output

Gambar 4.18. Perbandingan Gambar Input dan Output untuk **Metode SSIM**.

(a) Gambar Awal

(b) Gambar Akhir

Gambar 4.19. Perbandingan Gambar Awal dan Akhir untuk **Metode SSIM**.

4.3 Pengujian Target Kompresi

(a) Gambar Awal

(b) Gambar Akhir

Gambar 4.20. Perbandingan Gambar Awal dan Akhir untuk **Pengujian Target Kompresi**.

4.3.1 Pengujian Target Kompresi

```
QuadTree Image Compression

Input image: /home/y4nked/code/tucil/quadtree_image/test/jpg1.jpg
Error method: Variance
Threshold: 0.00
Min block size: 1 pixels²
Compression target: 34.50%
Output file: /home/y4nked/code/tucil/quadtree_image/test/outjpg1target.jpg
GIF visualization: /home/y4nked/code/tucil/quadtree_image/test/outjpg1target.gif
```

(a) Gambar Input

```
Compression Results

Performance Metrics
Computation Time:       48.7 s

File Information
Original Size:          573.38 KB
Compressed Size:        376.77 KB
Compression Ratio:      34.29%

Quadtree Statistics
Tree Depth:             13
Node Count:             78049
Target Threshold:       817.395719

Output Files
Compressed File:        /home/y4nked/code/tucil/quadtree_image/test/outjpg1target.jpg
Visualization GIF:      /home/y4nked/code/tucil/quadtree_image/test/outjpg1target.gif
```

(b) Gambar Output

Gambar 4.21. Perbandingan Gambar Input dan Output untuk **Pengujian Target Kompresi**.

Bab 5

Analisis Hasil

5.1 Analisis Kompleksitas

5.1.1 Analisis Kompleksitas Waktu

1. **Kompleksitas Per Node**
   Fungsi `calculateError` memiliki kompleksitas $\mathcal{O}(w \times h)$ untuk sebuah node dengan ukuran $w \times h$. Tetapi untuk metode eror Variance dan SSIM fungsi `calculateError` dapat dijalankan dalam $\mathcal{O}(1)$ (karena SAT).

2. **Total Jumlah Node**
   Dalam kasus terburuk, setiap node dibagi terus-menerus hingga mencapai ukuran minimum $M \times M$. Dengan demikian, jumlah node yang dibentuk dalam quadtree adalah:
   $$\mathcal{O}\Big(\frac{W \times H}{M^2}\Big),$$
   di mana W dan H adalah lebar dan tinggi gambar.

3. **Kompleksitas Total `calculateError`**
   Pada level ke-k dari quadtree:
   - Setiap node memiliki ukuran $\frac{W}{2^k} \times \frac{H}{2^k}$,
   - Jumlah node di level ke-k adalah 4^k.

   Oleh karena itu, total area yang diproses di level ke-k adalah:
   $$4^k \times \left(\frac{W}{2^k} \times \frac{H}{2^k}\right) = W \times H,$$
   yang merupakan area total gambar dan konstan per level.

4. **Jumlah Level Maksimum (Kedalaman Quadtree)**
   Jumlah level maksimum atau kedalaman tree diperkirakan sekitar
   $$\mathcal{O}\Big(\log \frac{\max\{W, H\}}{M}\Big),$$
   karena pada setiap level, ukuran node berkurang menjadi seperempatnya (jika tidak memenuhi *threshold*) hingga mencapai ukuran minimum M^2.

5. **Kompleksitas Total Waktu**
   Karena setiap level memproses total area $W \times H$ dengan operasi $\mathcal{O}(W \times H)$ dan terdapat $\mathcal{O}\left(\log \frac{\max\{W,H\}}{M}\right)$ level, maka total kompleksitas waktu algoritma pembuatan quadtree adalah:
   $$\mathcal{O}\left(W \times H \times \log \frac{\max\{W,H\}}{M}\right).$$

Kesimpulan:
Kompleksitas waktu algoritma pembuatan quadtree (worst-case) adalah

$$\mathcal{O}\Big(W \times H \times \log \frac{\max\{W,H\}}{M}\Big),$$

di mana:

- W dan H adalah dimensi gambar, dan
- M adalah ukuran minimum blok (yaitu, $M \times M$).

5.1.2 Kompleksitas Ruang

Untuk kompleksitas ruang, kita mempertimbangkan dua hal utama: struktur quadtree dan kebutuhan memori untuk *image sequence* (apabila kita ingin menampilkan transformasi per level).

- **Struktur Quadtree:** Dalam worst-case, quadtree dapat membelah hingga setiap piksel menjadi blok terkecil, sehingga jumlah node mencapai orde $\mathcal{O}(W \times H)$. Masing-masing node menyimpan informasi posisi, ukuran, dan pointer ke empat anak, sehingga total ruang untuk quadtree adalah $\mathcal{O}(W \times H)$.
- **Summed Area Table (SAT) & Summed Squared Area Table (SSAT):** Keduanya berukuran sama dengan gambar, yakni $\mathcal{O}(W \times H)$ untuk menyimpan kumulatif penjumlahan piksel maupun penjumlahan kuadrat piksel.
- **Penyimpanan Image Sequence:** Jika kita menyimpan hasil pewarnaan *setiap* level pembagian, maka banyaknya level mendekati $\log_4(\max\{W,H\}/M)$. Untuk setiap level, kita bisa menyimpan satu citra lengkap berukuran $W \times H$. Oleh sebab itu, total ruang yang dibutuhkan adalah

  $$\mathcal{O}\Big((W \times H) \times \log(\tfrac{\max\{W,H\}}{M})\Big).$$

  Nilai ini bisa menjadi signifikan lebih besar dibandingkan sekadar $\mathcal{O}(W \times H)$ apabila kita ingin mempertahankan animasi transformasi per level.

Kesimpulan Ruang: *Struktur quadtree* dan *summed area table* masing-masing memerlukan $\mathcal{O}(W \times H)$ memori. Akan tetapi, *bila* kita juga menyimpan citra *setiap* level untuk keperluan animasi, kebutuhan ruang bisa meningkat menjadi

$$\mathcal{O}\Big(W \times H \times \log(\tfrac{\max\{W,H\}}{M})\Big).$$

di mana:

- W dan H adalah dimensi gambar, dan
- M adalah ukuran minimum blok (yaitu, $M \times M$).

Daftar Pustaka

[1] Tanner York. ?Quadtrees for Image Compression? 2020. URL: https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00.

[2] Rinaldi Munir. ?Algoritma Divide and Conquer (Bagian 1)? 2025. URL: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf.

[3] Rinaldi Munir. ?Algoritma Divide and Conquer (Bagian 2)? 2025. URL: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-(2025)-Bagian2.pdf.

[4] Rinaldi Munir. ?Algoritma Divide and Conquer (Bagian 3)? 2025. URL: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/09-Algoritma-Divide-and-Conquer-(2025)-Bagian3.pdf.

[5] Wikipedia. ?Summed-area table? 2024. URL: https://en.wikipedia.org/wiki/Summed-area_table.

[6] Zhou Wang. ?Image quality assessment: from error visibility to structural similarity? 2019. URL: https://ieeexplore.ieee.org/document/1284395.

[7] Shu-Kai Fan. ?An Entropy-based Method for Color Image Registration? 2013. URL: https://www.scitepress.org/papers/2013/42105/42105.pdf.

Lampiran

Pranala Repository GitHub

Repository kode program dapat diakses pada pranala berikut:
https://github.com/l0stplains/Tucil2_13523002

Tabel Pengerjaan

	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	✓	

Tabel 1. Checklist Pengerjaan Tugas