

Kompresi Gambar Menggunakan Metode Quadtree

Laporan Tugas Kecil 2

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma



oleh

Aramazaya 13523082

DAFTAR ISI

DAFTAR ISI	2
BAB I	
DESKRIPSI MASALAH	3
BAB II	
PENJELASAN ALGORITMA	4
BAB III	
SOURCE CODE PROGRAM	5
BAB IV	
PERCOBAAN	9
IV.1 Pengujian 1	9
IV.2 Pengujian 2	10
IV.3 Pengujian 3	11
IV.4 Pengujian 4	13
IV.5 Pengujian 5	14
IV.6 Pengujian 6	15
IV.7 Pengujian 7	16
IV.8 Analisis Hasil Percobaan	18
V	
KESIMPULAN	19
LAMPIRAN	19

BAB I

DESKRIPSI MASALAH

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 IF2211 Strategi Algoritma – Tugas Kecil 2 1 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.

Tugas kali ini adalah membuat aplikasi untuk mengimplementasikan QuadTree untuk kompresi gambar. Bahasa yang dipakai dibatasi untuk C/C++/C#/Java. Pengguna dapat menerapkan batasan-batasan pada parameter untuk mengonfigurasi gambar hasil yang diinginkan.

BAB II

PENJELASAN ALGORITMA

Algoritma yang digunakan adalah algoritma *Divide and Conquer*. Algoritma ini bekerja dalam tiga tahap. Tahap pertama adalah *Divide* dimana persoalan dibagi menjadi persoalan-persoalan kecil. Selanjutnya, Persoalan-persoalan tersebut di *Conquer* atau diselesaikan, dan solusinya disatukan (*Combine*).

Dalam persoalan QuadTree, gambar yang semulanya berukuran besar dibagi menjadi berbagai *block*. Tiap *block* dapat memiliki empat anak *block* lagi sehingga membentuk struktur data *Tree*.

Algoritma terus melakukan pembagian *block* (*Divide*) hingga kondisi tertentu telah tercapai. Kondisi yang dapat tercapai adalah salah satu diantara *threshold* terpenuhi, ukuran *block* sudah lebih kecil dari keinginan pengguna, atau kedalaman pohon sudah lebih besar dari keinginan pengguna. Cara untuk menghitung *Threshold* adalah dengan menggunakan empat metode yang dapat dipilih oleh pengguna. Metode-metode tersebut adalah dengan menghitung variansi nilai RGB pada *block*, deviasi mean absolut nilai RGB pada *block*, perbedaan pixel dengan RGB tertinggi dan terendah pada *block*, atau menggunakan metode entropy.

Setelah kondisi tercapai, program melakukan pemrosesan gambar (*Conquer*) per *block* dimana rata-rata data RGB yang tersimpan di *block* disimpan di *image* kosong baru. Terakhir, semua data rata-rata RGB disatukan di *image* kosong baru tersebut (*Combine*).

Pada program yang dibuat, penerapan pembagian *block* dan pemrosesan *block* dilakukan dari bagian kiri atas, kanan atas, kiri bawah, lalu kanan bawah dengan konsep *depth-first*. Tidak ada alasan khusus mengapa dilakukan *depth-first* ketimbang *breadth-first*.

BAB III

SOURCE CODE PROGRAM

TabelIII.1 Fungsi dan Prosedur

 <pre> 1 void callMAD(int width, unsigned char *outputData, unsigned char *imageData, int channels, double threshold, QuadTree* ro 2 ot, int maxDepth, int minBlockSize) { 3 double MAD = root->getMAD(imageData, channels); 4 if (MAD < threshold) {processLeafNodes(root, outputData, imageData, channels, width);} 5 else if (root->getDepth() >= maxDepth) {processLeafNodes(root, outputData, imageData, channels, width);} 6 else if (root->getInfo().getWidth()*root->getInfo().getHeight() < minBlockSize) {processLeafNodes(root, outputData, i 7 mageData, channels, width);} 8 else { 9 root->makeChild(); 10 callMAD(width, outputData, imageData, channels, threshold, root->getUpLeft(), maxDepth, minBlockSize); 11 callMAD(width, outputData, imageData, channels, threshold, root->getUpRight(), maxDepth, minBlockSize); 12 callMAD(width, outputData, imageData, channels, threshold, root->getDownLeft(), maxDepth, minBlockSize); 13 callMAD(width, outputData, imageData, channels, threshold, root->getDownRight(), maxDepth, minBlockSize); 14 } </pre>	<p>Prosedur Rekursif untuk metode error <i>Mean Absolute Deviation</i>. Prosedur akan terus rekursif hingga mencapai <i>threshold</i>, ukuran mencapai batas, atau kedalaman mencapai batas.</p>
 <pre> 1 void callMPD(int width, unsigned char *outputData, unsigned char *imageData, int channels, double threshold, QuadTree* ro 2 ot, int maxDepth, int minBlockSize) { 3 double MPD = root->getMPD(imageData, channels); 4 if (MPD < threshold) {processLeafNodes(root, outputData, imageData, channels, width);} 5 else if (root->getDepth() >= maxDepth) {processLeafNodes(root, outputData, imageData, channels, width);} 6 else if (root->getInfo().getWidth()*root->getInfo().getHeight() < minBlockSize) {processLeafNodes(root, outputData, i 7 mageData, channels, width);} 8 else { 9 root->makeChild(); 10 callMPD(width, outputData, imageData, channels, threshold, root->getUpLeft(), maxDepth, minBlockSize); 11 callMPD(width, outputData, imageData, channels, threshold, root->getUpRight(), maxDepth, minBlockSize); 12 callMPD(width, outputData, imageData, channels, threshold, root->getDownLeft(), maxDepth, minBlockSize); 13 callMPD(width, outputData, imageData, channels, threshold, root->getDownRight(), maxDepth, minBlockSize); 14 } </pre>	<p>Prosedur Rekursif untuk metode error <i>Maximum Pixel Difference</i>. Prosedur akan terus rekursif hingga mencapai <i>threshold</i>, ukuran mencapai batas, atau kedalaman mencapai batas.</p>
 <pre> 1 void callEntropy(int width, unsigned char *outputData, unsigned char *imageData, int channels, double threshold, QuadTre 2 e* root, int maxDepth, int minBlockSize) { 3 double MBD = root->getEntropy(imageData, channels); 4 if (MBD < threshold) {processLeafNodes(root, outputData, imageData, channels, width);} 5 else if (root->getDepth() >= maxDepth) {processLeafNodes(root, outputData, imageData, channels, width);} 6 else if (root->getInfo().getWidth() < minBlockSize root->getInfo().getHeight() < minBlockSize) {processLeafNodes(r 7 oot, outputData, imageData, channels, width);} 8 else { 9 root->makeChild(); 10 callEntropy(width, outputData, imageData, channels, threshold, root->getUpLeft(), maxDepth, minBlockSize); 11 callEntropy(width, outputData, imageData, channels, threshold, root->getUpRight(), maxDepth, minBlockSize); 12 callEntropy(width, outputData, imageData, channels, threshold, root->getDownLeft(), maxDepth, minBlockSize); 13 callEntropy(width, outputData, imageData, channels, threshold, root->getDownRight(), maxDepth, minBlockSize); 14 } </pre>	<p>Prosedur Rekursif untuk metode error <i>Entropy</i>. Prosedur akan terus rekursif hingga mencapai <i>threshold</i>, ukuran mencapai batas, atau kedalaman mencapai batas.</p>

Image.cpp

```
● ● ●  
1 Block(int x, int y, int width, int height, int imgwidth) : x(x), y(y), width(width), h  
eight(height), imgwidth(imgwidth) {}  
2 Block() : x(0), y(0), width(0), height(0), imgwidth() {}  
3 ~Block() {}
```

Constructor dan
Destructor Class *Block*

```
● ● ●  
1 int getX() const { return x; }  
2 int getY() const { return y; }  
3 int getWidth() const { return width; }  
4 int getHeight() const { return height; }  
5 int getImgWidth() const { return imgwidth; }
```

Getter Class *Block*

```
● ● ●  
1 RGB(double r, double g, double b, int pxc) : r(r), g(g), b  
(b), pixelCount(pxc) {}  
2 RGB() : r(0), g(0), b(0), pixelCount(0) {}
```

Constructor Struct *RGB*

```
● ● ●  
1 vector<Block> Block::divideBlock() {  
2     int halfWidth = width / 2;  
3     int halfHeight = height / 2;  
4     int rightWidth = width - halfWidth;  
5     int bottomHeight = height - halfHeight;  
6     Block UpLeft(x, y, halfWidth, halfHeight, imgwidth);  
7     Block UpRight(x + halfWidth, y, rightWidth, halfHeight, imgwidth);  
8     Block DownLeft(x, y + halfHeight, halfWidth, bottomHeight, imgwidth);  
9     Block DownRight(x + halfWidth, y + halfHeight, rightWidth, bottomHeight, imgwidth);  
10    return { UpLeft, UpRight, DownLeft, DownRight };  
11}  
12}  
13}
```

Fungsi ini membagi
Block Gambar menjadi 4
dan mengembalikan array
berisi 4 *Block* baru.

```
1 RGB Block::getRGBOAvg(const unsigned char *imageData, int channels) const {
2     double sumR = 0.0;
3     double sumG = 0.0;
4     double sumB = 0.0;
5     int pixelCount = 0;
6     for (int i = y; i < y + height; i++) {
7         for (int j = x; j < x + width; j++) {
8             int index = (i * imgwidth + j) * channels;
9             sumR += imageData[index];
10            sumG += imageData[index + 1];
11            sumB += imageData[index + 2];
12            pixelCount++;
13        }
14    }
15    return RGB(sumR / pixelCount ,sumG / pixelCount, sumB / pixelCount, pixelCount);
16 }
```

Fungsi ini mengambil nilai rata rata RGB dari sebuah *Block* dan mengembalikan nilai-nilai tersebut dalam container struct RGB.

quadtree.cpp

```
1 QuadTree(Block info, int depth) : info(info), depth(depth), UpLeft(nullptr), UpRight(nullptr), DownLeft(nullptr), DownRight(nullptr) {}
2 QuadTree() : info(Block()), depth(0), UpLeft(nullptr), UpRight(nullptr), DownLeft(nullptr), DownRight(nullptr) {}
3 ~QuadTree() {
4     delete UpLeft;
5     delete UpRight;
6     delete DownLeft;
7     delete DownRight;
8 }
```

Constructor dan Destructor untuk class QuadTree

```
1 Block getInfo() const { return info; }
2 int getDepth() const { return depth; }
3 QuadTree* getUpLeft() const { return UpLeft; }
4 QuadTree* getUpRight() const { return UpRight; }
5 QuadTree* getDownLeft() const { return DownLeft; }
6 QuadTree* getDownRight() const { return DownRight; }
```

Getter Class Quadtree

 <pre>1 void QuadTree::makeChild()</pre>	<p>Prosedur yang membuat 4 address ke QuadTree baru dengan info berupa block dari fungsi divide. Dipakai untuk perhitungan jumlah simpul dan juga kedalaman maximal.</p>
 <pre>1 double QuadTree::getRGBVariance(const unsigned char *imageData, int channels) const</pre>	<p>Fungsi yang mengembalikan nilai double berupa varians dari sebuah <i>Block</i>.</p>
 <pre>1 double QuadTree::getMPD(const unsigned char *imageData, int channels) const</pre>	<p>Fungsi yang mengembalikan nilai double berupa Max Pixel Difference dari sebuah <i>Block</i>.</p>
 <pre>1 double QuadTree::getMAD(const unsigned char *imageData, int channels) const</pre>	<p>Fungsi yang mengembalikan nilai double berupa Mean Absolute Deviation dari sebuah <i>Block</i>.</p>
 <pre>1 double QuadTree::getEntrophy(const unsigned char *imageData, int channels) const</pre>	<p>Fungsi yang mengembalikan nilai entropi dari sebuah <i>Block</i>.</p>

Untuk kode lebih lengkap berada pada lampiran

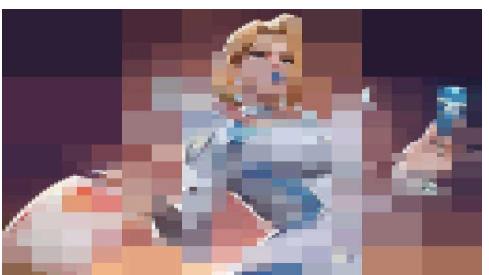
BAB IV

PERCOBAAN

IV.1 Pengujian 1

Input	Output
 <p>970px x 544px Maximum Depth : 6 Minimum Size : 10 Threshold Variance: 300 Threshold MAD : 30 Threshold MPD : 100 Threshold Entropy : 0.1</p>	<p>Metode Variansi:</p>  <p>Metode Mean Absolute Deviation:</p>  <p>Metode Maximum Pixel Difference:</p>  <p>Metode Entropy:</p> 

IV.2 Pengujian 2

Input	Output
 <p>968px x 544px Maximum Depth : 10 Minimum Size : 50 Threshold Variance: 500 Threshold MAD : 100 Threshold MPD : 180 Threshold Entropy : 1</p>	<p>Metode Variansi:</p>  <p>Metode Mean Absolute Deviation:</p>  <p>Metode Maximum Pixel Difference:</p>  <p>Metode Entropy:</p> 

IV.3 Pengujian 3

Input	Output
<p>Playtime past 2 weeks: 0h View global achievement stats 42 of 42 (100%) achievements earned:</p> <p>Personal Achievements</p> <p>Elden Ring Obtained all achievements</p> <p>Elden Lord Achieved the "Elden Lord" ending</p> <p>Age of the Stars Achieved the "Age of the Stars" ending</p> <p>Lord of Frenzied Flame Achieved the "Lord of Frenzied Flame" ending</p>	<p>Metode Variansi:</p> <p>Playtime past 2 weeks: 0h View global achievement stats 42 of 42 (100%) achievements earned:</p> <p>Personal Achievements</p> <p>Elden Ring Obtained all achievements</p> <p>Elden Lord Achieved the "Elden Lord" ending</p> <p>Age of the Stars Achieved the "Age of the Stars" ending</p> <p>Lord of Frenzied Flame Achieved the "Lord of Frenzied Flame" ending</p>
<p>1080px x 2400px Maximum Depth : 1000 Minimum Size : 1 Threshold Variance: 100 Threshold MAD : 10</p>	<p>Metode Mean Absolute Deviation:</p> <p>Playtime past 2 weeks: 0h View global achievement stats 42 of 42 (100%) achievements earned:</p> <p>Personal Achievements</p> <p>Elden Ring Obtained all achievements</p> <p>Elden Lord Achieved the "Elden Lord" ending</p> <p>Age of the Stars Achieved the "Age of the Stars" ending</p> <p>Lord of Frenzied Flame Achieved the "Lord of Frenzied Flame" ending</p>

Threshold MPD : 10
Threshold Entropy : 0.001

Metode Maximum Pixel Difference:



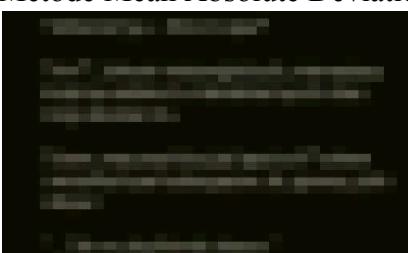
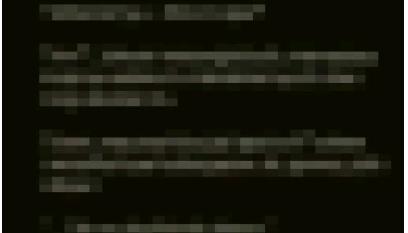
Metode Entropy:



IV.4 Pengujian 4

Input	Output
 <p>1280px x 720px Maximum Depth : 12 Minimum Size : 100 Threshold Variance: 1000 Threshold MAD : 15 Threshold MPD : 5 Threshold Entropy : 0.3</p>	<p>Metode Variansi:</p>  <p>Metode Mean Absolute Deviation:</p>  <p>Metode Maximum Pixel Difference:</p>  <p>Metode Entropy:</p> 

IV.5 Pengujian 5

Input	Output
<p>"Sebenarnya... Kita ini apa?"</p> <p>"Hm?" Jokowi menangkat alis, menyesap kopinya sebelum meletakannya di atas meja bundar itu.</p> <p>"Kamu maunya kita jadi apa toh?" aksen <i>mendhok</i> nya kedengeran uh, gemes, pikir Wowo</p> <p>" ... Be my boyfriend please."</p> <p>657px x 409px Maximum Depth : 5 Minimum Size : 10 Threshold Variance: 100 Threshold MAD : 1 Threshold MPD : 0.5 Threshold Entropy : 0.05</p>	<p>Metode Variansi:</p>  <p>Metode Mean Absolute Deviation:</p>  <p>Metode Maximum Pixel Difference:</p>  <p>Metode Entropy:</p> 

IV.6 Pengujian 6

Input	Output
 <p>1125px x 925px Maximum Depth : 7 Minimum Size : 10 Threshold Variance: 100 Threshold MAD : 0.5 Threshold MPD : 0.1 Threshold Entropy : 0.1</p>	<p>Metode Variansi:</p>  <p>Metode Mean Absolute Deviation:</p>  <p>Metode Maximum Pixel Difference:</p>  <p>Metode Entropy:</p> 

IV.7 Pengujian 7

Input	Output
 A portrait photograph of a man with dark hair, wearing a dark suit jacket, a white shirt, and a red patterned tie. The background is solid red. <p>1509px x 1957px Maximum Depth : 15 Minimum Size : 5 Threshold Variance: 100 Threshold MAD : 0.01 Threshold MPD : 0.5 Threshold Entropy : 0.1</p>	Metode Variansi:  The same portrait as the input, but with significant noise applied using the Variance method. The image appears grainy and lacks fine detail. Metode Mean Absolute Deviation:  The same portrait as the input, but with significant noise applied using the Mean Absolute Deviation method. The image appears grainy and lacks fine detail.

Metode Maximum Pixel Difference:



Metode Entropy:



IV.8 Analisis Hasil Percobaan

Didapatkan analisa algoritma sebagai berikut:

- Kompresi paling efektif ada pada percobaan 7 dimana hasil kompresi berada di kisaran 35% dan gambar masih dapat terlihat bagus.
- Lama proses berada di kisaran 30-100ms untuk semua kecuali entropi dimana entropi dapat naik hingga 1000ms.
- Kompleksitas Waktu untuk metode variansi, MAD, dan MPD mengikuti rumus yang mirip dimana terdapat 2 pass pada variansi yang berarti algoritma tersebut memiliki kompleksitas $O(n)$ untuk tiap block dan $O(n \log n)$ untuk semua block. $O(n \log n)$ didapatkan karena jika maksimal setiap bagian membagi 4 hingga n kali berarti terdapat $\log_4 n = \log n$ level, sehingga hasil Big-O adalah $O(n) \times O(n \log n) = O(n \log n)$.
- Hal diatas berlaku untuk MPD dan MAD karena mereka juga menggunakan satu passing n .
- Untuk entropi didapatkan $O(n+k)$ per dimana k adalah 256 untuk tiap warna pada RGB. Hasil akhir Big-O tetap $O(n \log n)$.
- Semua metode memiliki kompleksitas ruang $O(n)$ dimana n adalah jumlah pixel di gambar. Metode entropi mungkin memerlukan sedikit lebih banyak ruang namun perbedaannya tidak berpengaruh besar.
- Penggunaan kompresi QuadTree lebih bagus untuk gambar dengan banyak noise dalam segi kualitas, bagus untuk gambar dengan sedikit noise untuk segi efisiensi, dan buruk untuk tulisan.

BAB V

KESIMPULAN

Didapatkan bahwa kompresi gambar dengan metode Quadtree memiliki keunggulan ketika gambar memiliki banyak *noise* jika hanya berfokus pada kualitas. Jika hanya bercermin pada efisiensi, didapatkan bahwa gambar dengan sedikit *noise* dapat terproses dengan lebih cepat. Dari segi efisiensi, Metode ini cukup baik karena dapat dianggap cukup cepat dengan kompleksitas ruang yang kecil. Metode ini dapat menunjukan dengan jelas implementasi strategi *Divide and Conquer* pada sebuah algoritma yang dapat dipakai sehari-hari.

LAMPIRAN

1. Repozitori Github
https://github.com/Aramazaya/Tucil2_13523082
2. Source Code Images
https://drive.google.com/file/d/1_sAKeEybQJ3k9KxFGpPKuHqEVGeOEbDH/view?usp=sharing

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	✓	