Menerapkan Metode Divide and Conquer pada Kompresi Gambar Menggunakan Quadtree

Muhammad Habibi Husni - 13521169
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): habibihusni18@gmail.com

***Abstract*— Kompresi gambar menjadi penting dalam era digital yang maju, di mana ukuran dan kompleksitas file gambar menjadi tantangan. Metode kompresi *Quadtree* dengan pendekatan *Divide and Conquer* dapat digunakan dalam penyelesaian masalah tersebut. Algoritma akan memecah gambar menjadi blok-blok kecil, menghilangkan redundansi, dan mengurangi penggunaan ruang penyimpanan. Namun, kompleksitas algoritma *quadtree* tidak cukup efisien untuk gambar besar, dan belum efisien untuk *threshold* tinggi. Peningkatan diperlukan dengan mengoptimalkan kalkulasi kualitas gambar, memanfaatkan algoritma kompresi lainnya, dan mempertimbangkan penggunaan dalam aplikasi yang lebih kompleks.**

Kata kunci*— *kompresi gambar; quadtree; divide and conquer; algoritma kompresi

I. PENDAHULUAN

Dalam era digital yang semakin maju ini, penggunaan dan pertukaran gambar secara luas telah menjadi bagian tak terpisahkan dari kehidupan kita sehari-hari. Dari media sosial hingga aplikasi *messaging*, gambar digunakan untuk menyampaikan informasi, mengekspresikan emosi, dan mengabadikan momen berharga. Namun, pertumbuhan pesat dalam penggunaan gambar digital juga menyebabkan lonjakan besar dalam ukuran dan kompleksitas *file* gambar. Keterbatasan ruang penyimpanan, kecepatan transfer, dan keterbatasan *bandwidth* internet yang masih ada membuat perluasan dan pertukaran gambar menjadi tantangan yang signifikan. Inilah yang mendorong kebutuhan akan teknik kompresi gambar yang efisien.

Salah satu metode yang digunakan dalam kompresi gambar adalah metode *Divide and Conquer* (Bagi dan Kuasai). Metode ini memiliki pendekatan yang berfokus pada memecah masalah besar menjadi submasalah yang lebih kecil, yang kemudian diselesaikan secara terpisah sebelum digabungkan kembali menjadi solusi yang lengkap. Dalam konteks kompresi gambar, metode *Divide and Conquer* diterapkan menggunakan struktur data yang disebut *Quadtree*.

Quadtree adalah struktur data yang efisien untuk merepresentasikan dan mengompresi gambar. Dengan menggunakan *quadtree*, gambar dapat dipecah menjadi blok-blok yang lebih kecil dan lebih sederhana. Setiap blok tersebut kemudian dapat diwakili oleh satu simpul dalam struktur *quadtree*. Pendekatan ini memungkinkan penghilangan redundansi dan penyimpanan gambar dengan menggunakan lebih sedikit ruang penyimpanan. Selain itu, metode *divide and conquer* dengan *quadtree* juga memungkinkan untuk mengurangi waktu transfer gambar melalui jaringan dengan mengirimkan hanya bagian-bagian yang relevan dari gambar.

II. LANDASAN TEORI

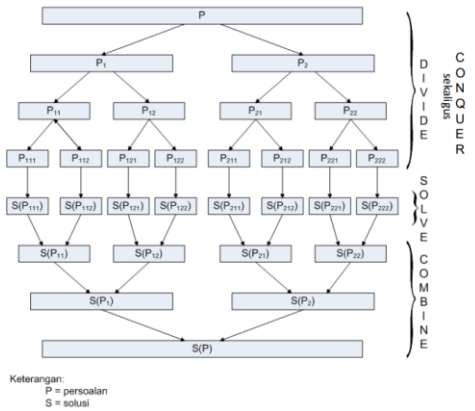
A. Divide and Conquer

Algoritma *divide and conquer* merupakan salah satu dari banyak macam algoritma penyelesaian masalah yang sering digunakan dalam dunia sehari-hari terutama dalam pemrograman. Metode *divide and conquer* melakukan pendekatan solusi dengan membagi masalah yang akan diselesaikan menjadi masalah yang lebih kecil, lebih mudah dipecahkan, dan independen. Setelah masalah-masalah kecil tersebut diselesaikan, solusinya digabungkan kembali untuk memperoleh solusi keseluruhan. Pendekatan ini memungkinkan kita untuk mengatasi masalah yang kompleks dengan lebih efisien dan terorganisir, sehingga meningkatkan kinerja dan keefektifan dalam menyelesaikan masalah tersebut. Pada dasarnya, Algoritma ini menyelesaikan sebuah masalah dalam tiga tahap:

1. *Divide*: tahap ini melibatkan pemecahan masalah awal menjadi submasalah yang lebih kecil dan dapat lebih mudah diselesaikan. Masalah awal dibagi-bagi menjadi bagian-bagian yang lebih kecil, sehingga setiap bagian tersebut menjadi lebih mudah untuk ditangani secara terpisah. Pemecahan ini dapat dilakukan dengan membagi masalah menjadi dua submasalah yang lebih kecil dengan ukuran yang setengah atau dengan membaginya menjadi beberapa bagian tergantung pada sifat dan kompleksitas masalah. Pemecahan masalah menjadi submasalah ini memungkinkan kita untuk mengatasi kompleksitas yang terkait dengan masalah awal secara terpisah dan meningkatkan efisiensi dalam menyelesaikan masalah keseluruhan.
2. *Conquer*: tahap ini melibatkan penyelesaian masing-masing submasalah yang telah didapat secara terpisah. Setelah dilakukannya tahap *divide*, masing-masing submasalah tersebut akan diselesaikan secara mandiri menggunakan metode yang sama atau berbeda. Solusi setiap submasalah tersebut dapat ditemukan dengan berbagai cara, seperti rekursi, prosedural, atau teknik

lainnya. Pada tahap *conquer*, fokus diutamakan dalam penyelesaian submasalah secara efisien dan akurat. Tahap ini merupakan hal yang penting, karena solusi yang ditemukan pada masing-masing submasalah juga akan berkontribusi pada solusi akhir secara keseluruhan.

3. *Combine*: Pada tahap ini, solusi-solusi submasalah yang telah didapatkan pada tahap *conquer* akan digabungkan Kembali menjadi sebuah solusi yang menyelesaikan permasalahan awal secara sepenuhnya. Proses penggabungan ini biasanya melibatkan manipulasi data atau penggabungan hasil dari setiap submasalah, tergantung pada jenis masalah yang sedang diselesaikan. Melalui tahap *combine*, metode *divide and conquer* memungkinkan kita untuk memperoleh solusi akhir yang terintegrasi dari masalah yang awalnya kompleks dengan memanfaatkan solusi dari submasalah yang lebih kecil.



Gambar 1 Ilustrasi Proses *Divide and Conquer* [2]

Secara umum, *divide and conquer* melibatkan proses rekursif untuk memecah masalah menjadi submasalah yang lebih kecil. Akibat unsur rekursif, maka kompleksitas sebuah algoritma *divide and conquer* akan sulit dihitung dengan metode iteratif. Oleh karena itu, teorema Master digunakan untuk menyelesaikan masalah tersebut. Misalkan T(n) adalah fungsi monoton menaik yang memenuhi relasi rekurens:

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

dengan $n = b^k; k = 1,2,\ldots; a \geq 1; b \geq 2; c, d \geq 0$, maka berlaku:

$$T(n) = \begin{cases} O(n^d), a < b^d \\ O(n^d log\ n), a = b^d \\ O(n^{log_b a}), a > b^d \end{cases}$$

B. Image Compression

Kompresi gambar merupakan sebuah proses yang dilakukan pada suatu *file* gambar dengan tujuan untuk meminimalkan ukurannya dalam *byte* tanpa membuat kualitas gambar tersebut berada di bawah batas yang diterima. Dengan menurunkan ukuran *file*, maka akan lebih banyak gambar yang dapat disimpan dalam sebuah ukuran penyimpanan. Lalu dalam konteks pengiriman data melalui jaringan nirkabel, gambar tersebut juga mengurangi *bandwidth* yang digunakan sehingga dapat mempercepat pengiriman data.

Terdapat dua jenis kompresi gambar, yaitu:

- *Lossy compression*: yaitu metode kompresi gambar yang mengurangi ukuran gambar dengan cara menghilangkan informasi yang tidak penting atau data yang *redundant* pada gambar secara permanen. Oleh karena ini, *lossy compression* bersifat *irreversible* atau tidak dapat dikembalikan ke gambar semula. Dalam proses ini, gambar akan kehilangan sebagian kualitasnya, namun tingkat kehilangan ini dikendalikan agar tetap dapat diterima secara visual. Proses kompresi ini memanfaatkan karakteristik sifat perseptual manusia terhadap gambar. Beberapa komponen atau area gambar yang tidak terlalu anggap penting atau kurang diperhatikan mata manusia dapat dihilangkan dengan toleransi tertentu. Terdapat beberapa metode yang umum digunakan dalam kompresi ini:
    - *Transform coding*: mentransformasi domain gambar dari domain spasial menjadi domain frekuensi.
    - *Chroma subsampling*: mengurangi jumlah informasi warna dalam gambar
    - *Color quantization:* mengurangi dimensi warna gambar. Warna yang dihilangkan akan direfrensikan ke warna lain.

  Salah satu format gambar yang bersifat *lossy* adalah JPEG yang umum digunakan pada internet.
- *Lossless compression:* kompresi *lossless* melibatkan metode kompresi yang tidak menghilangkan informasi penting dari gambar maupun merusak kualitasnya. Kompresi *lossless* memanfaatkan pola-pola atau redundansi yang ada pada gambar dan lalu merepresentasikan informasi tersebut menjadi pola yang lebih kecil. Dengan ini, kompresi *lossless* dapat membentuk kualitas gambar semula tanpa kehilangan informasi apa pun meski telah dikompresi. Namun, kompresi *lossless* tidak terlalu banyak mengurangi ukuran gambar layaknya *lossy compression*. *Lossless compression* umum digunakan dalam situasi yang mementingkan detail dari gambar. Salah satu format gambar yang bersifat *lossless* adalah PNG.

Dalam *image processing*, juga diperlukan perhitungan kualitatif terhadap kualitas dari sebuah gambar kompresi terhadap gambar aslinya. Dengan hal itu, terdapat beberapa metrik kualitas yang umum digunakan dalam *image compression*, yaitu sebagai berikut:

- *Mean-Squared Error* (MSE): merupakan sebuah metrik yang merepresentasikan nilai kumulatif dari

nilai eror kuadrat dari gambar terkompresi dengan gambar aslinya. Dihitung dengan rumus:

$$MSE = \frac{N}{NM}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}[X(i,j) - Y(i,j)]^2$$

- *Peak signal-to-noise ratio* (PSNR): PSNR merupakan turunan dari MSE yang menunjukkan rasio dari intensitas maksimal suatu *pixel* terhadap kuadrat dari distorsinya dengan satuan db. Semakin besar nilai PSNR, semakin dekat kualitas gambar terkompresi dengan gambar aslinya. Dihitung dengan rumus:

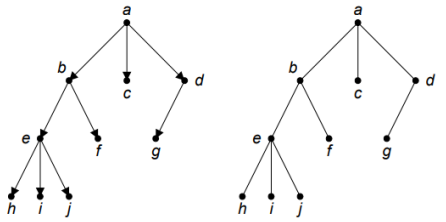
$$PSNR(x,y) = \frac{10log_{10}\left[max\left(max(x), max(y)\right)\right]^2}{(x-y)^2}$$

- *Structural similarity* (SSIM) *index*: Merupakan sebuah model persepsi yang menghitung kesamaan struktur antar gambar. Sebuah indeks SSIM didapatkan dari penggabungan struktur gambar, *luminance*, dan kontras. Dapat dihitung dengan rumus:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_1^2 + \mu_2^2 + c_1)(\sigma_1^2 + \sigma_2^2 + c_2)}$$

C. *Quadtree*

Quadtree merupakan variasi dari pohon berakar. Pohon berakar adalah graf tak-berarah terhubung yang tidak mengandung sirkuit dan salah satu simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah menuju simpul anaknya sehingga menjadi graf berarah.

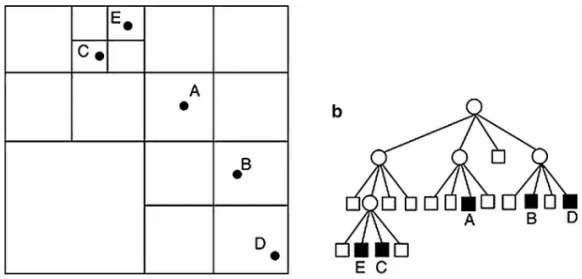


(a) Pohon berakar (b) sebagai perjanjian, tanda panah pada sisi dapat dibuang

Gambar 2 Ilustrasi Pohon Berakar [1]

Quadtree merupakan variasi dari *binary tree* yang memiliki sifat berikut:

- Merupakan pohon n-ary dengan n = 4
- Setiap simpul di dalam *quadtree* memiliki tepat 4 buah simpul anak
- Tiap anak dibedakan menjadi kuadran 1, 2, 3, dan 4.
- Karena terdapat perbedaan urutan anak, maka *quadtree* merupakan pohon terurut.
- Dalam konteks bidang dua dimensi, masing-masing anak dari simpul merepresentasikan data dari simpul *parent* yang sudah dibagi menjadi empat daerah sama besar.



Gambar 3 Ilustrasi Quadtree Beserta Data yang Dipegang Masing-Masing Simpulnya. [4]

Dikarenakan sifatnya yang membagi data menjadi empat buah data yang lebih kecil, *quadtree* banyak digunakan dalam partisi ruang dua dimensi dengan secara rekursif membagi sebuah daerah menjadi empat kuadran. Namun selain itu, *quadtree* juga dapat digunakan untuk merepresentasikan nilai titik.

Seperti halnya sebuah pohon berakar, *quadtree* memiliki beberapa terminologi, yaitu:

- Anak (*child)* dan Orangtua (*parent*): menunjukkan relasi antara sebuah simpul dengan simpul lain yang terhubung pada sebuah pohon berakar.
- Lintasan (*path*): yaitu rute yang ditempuh dari satu simpul ke simpul yang lain
- Saudara Kandung (*sibling*): yaitu relasi antara simpul yang terhubung kepada *parent* yang sama.
- Upapohon (*subtree*): yaitu pohon yang merupakan himpunan dari pohon lainnya
- Derajat (*degree*): yaitu banyak *outward edge* atau banyak anak dari sebuah simpul, dalam *quadtree*, setiap simpul berderajat empat.
- Daun (*leaf*): yaitu simpul yang tidak memiliki anak
- Simpul Dalam (*internal nodes*): yaitu simpul yang memiliki anak.
- Aras (*level*) atau Tingkat: yaitu jarak menuju sebuah simpul dari simpul *root.*
- Tinggi (*height*) atau Kedalaman (*depth*): yaitu tingkat tertinggi yang dimiliki oleh setiap simpul pada suatu pohon.

Quadtree memiliki beberapa bentuk yaitu:

- *Quadtree* daerah: yaitu *quadtree* yang simpul-simpulnya merepresentasikan partisi daerah yang diperoleh dengan membagi daerah awal menjadai empat buah daerah yang lebih kecil. Pada masing-masing simpul akan memiliki sebuah nilai yang merepresentasikan daerah yang dicangkupinya.
- *Quadtree* titik: yaitu *quadtree* yang merepresentasikan sebuah titik pada bidang dua dimensi.
- *Edge Quadtree*: yaitu *quadtree* yang masing-masing simpulnya menyimpan informasi garis. Beberapa

aplikasi yaitu merepresentasikan sebuah kurva, yang kurva tersebut akan dibagi oleh *quadtree* hingga satu simpul menyimpan satu segmen atau mencapai batas kedalamannya.

III. PEMBAHASAN

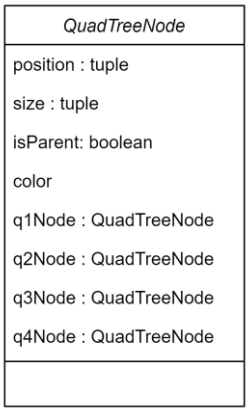
A. *Algoritma Kompresi*

Berdasarkan sifatnya, sebuah *quadtree* merepresentasikan seperempat data dari sebuah sampel *parent*. Oleh karena itu, *quadtree* dapat dimanfaatkan untuk melakukan pengkompresian sebuah gambar dengan melakukan pembagian empat buah daerah gambar dan melakukan *divide and conquer* lalu menganalisis pengkompresian dari masing-masing bagian tersebut secara rekursif.

Berikut adalah definisi tahap pembagian Langkah *divide and conquer* dalam *quadtree compression*:

1. *Divide*: yaitu membagi gambar menjadi empat kuadran yang ukurannya diusahakan sama ketika gambar saat ini masih cukup detail.
2. *Conquer*: yaitu ketika bagian gambar saat ini memiliki kualitas yang tidak terlalu penting atau berada dibawah *threshold* tertentu, maka pembagian diberhentikan. Lalu bagian gambar tersebut dapat direpresentasikan menjadi sebuah warna rata-rata dari gambar tersebut.
3. *Combine*: Setiap bagian gambar yang telah dikompresi digabungkan kembali untuk membentuk gambar secara penuh.

Untuk mengakomodasi data pembagian gambar, maka digunakan struktur data *quadtree*. Berikut adalah struktur data yang disimpan dalam sebuah simpul *quadtree* di permasalahan ini:



Gambar 4 Struktur Data Dari QuadTreeNode

Dengan memanfaatkan bentuk ini, maka algoritma kompresi gambar dengan menggunakan *quadtree* dapat dideskripsikan sebagai berikut:

```
function quadtreeCompression(quadTreeNode, image, threshold):
    if (image quality below threshold) then
        quadTreeNode.color <- meanColor(image)
        -> create image with quadTreeNode.color and size of quadTreeNode.size
    else
        q1, q2, q3, q4 <- quadTreeNode.subdivide()
        q1Image <- image[firstQuadrant]
        q2Image <- image[secondQuadrant]
        q3Image <- image[thirdQuadrant]
        q4Image <- image[fourthQuadrant]
        q1CompressedImage <- quadtreeCompression(q1, q1Image, threshold)
        q2CompressedImage <- quadtreeCompression(q2, q2Image, threshold)
        q3CompressedImage <- quadtreeCompression(q3, q3Image, threshold)
        q4CompressedImage <-quadtreeCompression(q4, q4Image, threshold)
        -> combineImage(q1CompressedImage,
                        q2CompressedImage,
                        q3CompressedImage,
                        q4CompressedImage)
```

Gambar 5 Pseudocode Dari Kompresi Quadtree

Dengan memanggil *quadTreeCompression(rootNode, image, threshold)*, maka fungsi tersebut akan mengembalikan sebuah gambar hasil kompresi berdasarkan *threshold* kualitas yang sudah diatur. Dari metode kompresi ini, dapat diperolah bahwa kompresi ini merupakan *lossy compression* karena informasi dari bagian gambar yang kualitasnya kurang dari *threshold* dibuang dan digantikan dengan rata-rata dari gambar tersebut.

Dalam menentukan kualitas sebuah bagian gambar terdapat beberapa metode yang dapat digunakan, yaitu *mean squared error* (MSE), *Peak signal-to-noise ratio* (PSNR), dan *Stuctural Similarity* (SSIM) *index*. Dalam makalah ini, penulis menggunakan perhitungan PSNR dengan acuan MSE sebagai berikut:

$$PSNR = 10 \log_{10} \frac{MaxValue^2}{MSE}$$

Dengan *MaxValue* adalah 255 berdasarkan nilai maksimal sebuah nilai gambar dalam sebuah *pixel*.

B. *Merepresentasikan Hasil Kompresi ke Dalam File*

Algoritma sebelumnya hanya menghasilkan rekonstruksi gambar yang bagian yang tidak penting sudah dihilangkan dan lalu menampilkan gambar dari hasil kompresi *quadtree* ke layar. Untuk menghasilkan kompresi data yang sebenarnya, diperlukan representasi data dari hasil kompresi *quadtree* dan membuatnya menjadi sebuah *file* yang dapat dibentuk kembali menjadi gambar semula.

Pada makalah ini, penulis menggunakan struktur data *file* sebagai berikut:

- 4-byte pertama: sebuah integer yang menunjukkan lebar dari gambar
- 4-byte kedua: sebuah integer menunjukkan tinggi dari gambar
- 4-byte ketiga: sebuah integer yang menunjukkan banyaknya simpul dari *quadtree* gambar, misalkan dengan *n.*
- *n*-bit berikutnya: merupakan kumpulan bit yang menunjukkan apakah sebuah simpul ke-*n* merupakan sebuah *parent* berurutan dimulai dari simpul *root.*

- *Byte* sisanya: menunjukkan nilai warna dari sebuah simpul anak secara berurut berdasarkan dalam konstruksi pohon. Dengan tiap anak akan memiliki warna dengan format RGB sehingga memerlukan tiga *byte* untuk tiap simpulnya.

Dengan format seperti di atas, maka banyaknya informasi bagian gambar yang perlu disimpan akan jauh lebih kecil daripada menyimpan warna dari tiap *pixel*. *File* tersebut dapat dibaca Kembali dan direkonstruksi menjadi *quadtree* hasil kompresi awal dengan menggunakan algoritma berikut:

```
function reconstructImageTree(input/output quadTreeNode,
                              input/output isParent[],
                              input/output colorList[]) -> Image
    if (isParent.front()) then
        res <- image with color of colorList.front()
               with size of quadTreeNode.size
        colorList.pop_front()
        isParent.pop_front()
        -> res
    else
        q1, q2, q3, q4 <- quadTreeNode.subdivide()
        isParent.pop_front()
        q1CompressedImage <- reconstructImageTree(q1, isParent, colorList)
        q2CompressedImage <- reconstructImageTree(q2, isParent, colorList)
        q3CompressedImage <- reconstructImageTree(q3, isParent, colorList)
        q4CompressedImage <- reconstructImageTree(q4, isParent, colorList)
        -> combineImage(q1CompressedImage,
                        q2CompressedImage,
                        q3CompressedImage,
                        q4CompressedImage)
```

Gambar 6 Pseudocode Dari Rekonstruksi Quadtree

Setelah fungsi dijalankan dengan memanggil *reconstructImageTree(rootNode, isParent, colorList)*, maka akan dikembalikan *quadtree* yang sesuai dengan hasil kompresi gambar awal serta tampilan gambar hasil kompresi. Dengan ini hasil kompresi dapat disimpan secara efisien dalam *secondary storage* dan dapat dibaca kembali.

IV. Implementasi Dan Pengujian

A. *Implementasi*

Dalam implementasi yang telah disebutkan di bab sebelumnya, penulis menggunakan Bahasa Python. Berikut adalah implementasi Bahasa Python dari struktur data *quadtree*:

```python
class QuadTreeNode():
    def __init__(self,position : tuple, size: tuple) -> None:
        self.position = position
        self.size = size
        self.isParent = False
        self.color = None
        self.q1Node = None
        self.q2Node = None
        self.q3Node = None
        self.q4Node = None

    def _createChildNode(self,position,size):
        return QuadTreeNode(position,size)

    def divideRegion(self):
        if (not self.isDivideable()):
            return None
        self.isParent = True
        w, h = self.size
        x, y = self.position
        half_width = w // 2
        half_height = h // 2
        self.q1Node = self._createChildNode((x, y), (half_width, half_height))
        self.q2Node = self._createChildNode((x + half_width, y), (w-half_width, half_height))
        self.q3Node = self._createChildNode((x,y + half_height), (half_width, h-half_height))
        self.q4Node = self._createChildNode((x + half_width, y + half_height), (w-half_width, h-half_height))
        return self.q1Node, self.q2Node, self.q3Node, self.q4Node

    def isDivideable(self):
        return not self.isParent and self.size[0] > 1 and self.size[1] > 1

    def createImage(self):
        if self.isParent:
            img1 = self.q1Node.createImage()
            img2 = self.q2Node.createImage()
            img3 = self.q3Node.createImage()
            img4 = self.q4Node.createImage()
            return np.concatenate([
                np.concatenate([img1, img2], axis=0),
                np.concatenate([img3, img4], axis=0)
            ], axis=1)
        else:
            meanImage = np.tile(self.color, self.size).reshape(self.size[0], self.size[1], -1)
            return meanImage
```

Gambar 7 Implementasi *QuadTreeNode*

Berikut adalah hasil implementasi algoritma *compression* menggunakan metode *divide and conquer* beserta algoritma rekonstruksi:

```python
class ImageCompressionNode(QuadTreeNode):
    def __init__(self, positon: tuple, imageData: np.array):
        w, h, _ = imageData.shape
        super().__init__(positon, (w, h))
        self.imageData = imageData
        self.color = np.mean(imageData, axis=(0,1)).astype(np.uint8)
        self.quality = PSNR(imageData, self.color)
```

```python
def compress(self):
    self._divideImg(self.rootNode)
    return self.rootNode.createImage()

def _divideImg(self, node: ImageCompressionNode, depth = 1):
    if node.quality >= self.threshold or (self.maxDepth is not None and depth >= self.maxDepth):
        return
    else:
        nodes = node.divideRegion()
        if (nodes == None):
            return
        else:
            q1Node, q2Node, q3Node, q4Node = nodes
            self._divideImg(q1Node, depth + 1)
            self._divideImg(q2Node, depth + 1)
            self._divideImg(q3Node, depth + 1)
            self._divideImg(q4Node, depth + 1)
```

```python
def reconstruct(self):
    self._reconstruct(self.rootNode, self.dividedFlags, self.colors)
    return self.rootNode.createImage()

def _reconstruct(self, node: QuadTreeNode, dividedFlags: list, colors: list):
    if (len(dividedFlags) == 0):
        return
    if dividedFlags.pop():
        nodes = node.divideRegion()
        if nodes is not None:
            q1Node, q2Node, q3Node, q4Node = nodes
            self._reconstruct(q1Node, dividedFlags, colors)
            self._reconstruct(q2Node, dividedFlags, colors)
            self._reconstruct(q3Node, dividedFlags, colors)
            self._reconstruct(q4Node, dividedFlags, colors)
    else:
        node.color = colors.pop()
        node.isParent = False
```

Gambar 8 Implementasi Algoritma Kompresi dan Rekonstruksi

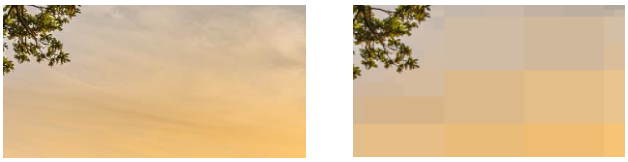
B. *Hasil Pengujian*

Berikut adalah hasil kompresi dari tiga gambar berbeda dengan menggunakan *threshold* sebesar 30 db, dan batas kedalaman maksimal sebesar 11:

(a)

(b)

Gambar 9 Perbandingan Hasil Kompresi : (a) sebelum (b) sesudah



(a) (b)

Gambar 10 *Close-up* Perbandingan Hasil Kompresi: (a) sebelum (b) sesudah

Dari hasil kompresi, dapat dilihat bahwa terjadinya pengabstrakan warna terhadap bagian gambar yang kurang memiliki detail, namun berusaha untuk tetap membiarkan bagian yang memiliki detail yang lebih rinci.

Dengan mem-variasikan besar parameter, maka besar *file* hasil konversi juga akan berubah. Berikut adalah pengambilan data kompresi dari gambar yang sama dengan parameter berbeda:

Threshold (db)	*Max Depth*	Ukuran File (MB)	*Runtime* (detik)
Gambar asli		5,34	-
50	11	3,11	37,49
40	11	2,32	30,92
30	11	2,02	27,10
20	11	1,27	18,07
15	11	0,50	8,76
50	12	11,8	135,69

C. Analisis Hasil Pengujian

Dari data tersebut, dapat dilihat bahwa hasil kompresi dengan perbandingan ukuran *file* gambar awal berada pada cakupan 60% hingga 9,3% dengan *range threshold* 50 hingga 15 db. Dari perbandingan tersebut dapat dilihat kompresi *quadtree* memiliki rasio kompresi yang cukup bagus hanya dengan memanfaatkan algoritmanya sendiri tanpa menambah algoritma kompresi tambahan. Tentunya, semakin kecil *threshold* yang diberikan akan memberikan ukuran *file* yang jauh lebih kecil, namun hal tersebut membuang banyak nilai-nilai detail penting.

Dengan melihat waktu jalan algoritma, dapat dilihat bahwa semakin besar parameter diberikan, semakin besar juga jalannya algoritma. Terutama pada kenaikan satu kedalaman menjadi 12 kedalaman maksimal, waktu *runtime* meningkat hingga tiga kali lipat dan ukuran *file* juga meningkat tiga kali lipat. Hal ini disebabkan oleh kompleksitas algoritma kompresi *quadtree*. Dengan melihat relasi rekurensnya, dan memisalkan n dengan lebar gambar, dan m dengan tinggi gambar, di dapat kompleksitas sebagai berikut:

$$T(nm) = \begin{cases} 4T\left(\frac{nm}{4}\right) + 2nm, & n \geq 2 \text{ and } m \geq 2 \\ 1, & otherwise \end{cases}$$

Penjelasan dari relasi rekurens tersebut adalah: setiap panggilan rekursif, algoritma harus menghitung rata-rata dari seluruh *pixel* gambar dan lalu melakukan penghitungan kembali untuk mendapatkan nilai PSNR gambar tersebut. Dan lalu algoritma melakukan panggilan rekursif kepada dirinya sebanyak empat kali untuk mengolah sebanyak seperempat *pixel* awal. Dengan menggunakan Teorema Master, maka kompleksitas algoritma secara keseluruhan adalah:

$$O(nm \log nm) = O(nm(log\ n + log\ m))$$

Hal ini menunjukkan bahwa pertambahan *pixel* karena pertambahan ukuran pada lebar dan panjang gambar akan membuat banyaknya satuan proses meningkat secara logaritmik linear. Pada kasus implementasi, batas kedalaman dibatasi sehingga pertumbuhan dari *log nm* dapat dibatasi menjadi sebuah konstanta. Namun, ketika batas tersebut bertambah, algoritma berjalan jauh lebih lama, sesuai dengan data yang sudah didapatkan dari hasil pengujian.

V. KESIMPULAN DAN SARAN

Dewasa ini, kita menyadari bahwa penggunaan dan pertukaran gambar telah menjadi bagian penting dalam kehidupan digital kita. Namun, pertumbuhan pesat ini juga telah menyebabkan masalah ukuran dan kompleksitas *file* gambar. Untuk mengatasi tantangan ini, diperlukan teknik kompresi gambar yang efisien. Salah satu metode yang digunakan adalah metode Divide and Conquer dengan menggunakan struktur data Quadtree. Quadtree memungkinkan gambar untuk dipecah menjadi blok-blok yang lebih kecil, menghilangkan redundansi, dan mengurangi penggunaan ruang penyimpanan. Dengan pendekatan ini, kita dapat mengirim hanya bagian relevan dari gambar, mengurangi waktu transfer melalui jaringan.

Dari hasil penelitan, dapat diperoleh bahwa kompresi *quadtree* memiliki hasil konversi yang cukup bagus, meski dengan mengorbankan beberapa detail dari gambar awal. Selain itu, kompleksitas algoritma berbentuk logaritmik linier sehingga cocok untuk memproses gambar yang kecil, namun tidak cukup efisien untuk memproses gambar yang lebih besar. Selain itu, Selain itu, kompresi *quadtree* belum efisien untuk *threshold* yang tinggi karena ukuran hasil kompresi dapat melebihi ukuran *file* asli.

Untuk penggunaan yang lebih luas dan kompleks, diperlukan peningkatan dalam beberapa aspek. Peningkatan dapat dilakukan dengan mempercepat kalkulasi perhitungan kualitas gambar untuk meningkatkan kecepatan algoritma. Selain itu, memanfaatkan algoritma kompresi lainnya bersama dengan kompresi *quadtree* dapat memperkecil ukuran hasil kompresi. Dengan langkah-langkah ini, kompresi *quadtree* dapat menjadi solusi yang lebih efisien dan sesuai dengan kebutuhan aplikasi yang beragam.

UCAPAN TERIMA KASIH

Pertama sekali penulis mengucapkan puji syukur kepada Allah Swt. atas segala nikmat yang telah Diberikan-Nya sehingga penulis dapat menyelesaikan ini yang berjudul "Menerapkan Metode Divide and Conquer pada Kompresi Gambar Menggunakan Quadtree" sebagai pemenuhan tugas makalah IF2211 Strategi Algoritma. Penulis juga ingin menyampaikan ucapan terima kasih kepada Bapak Dr. Rinaldi Munir, selaku Dosen Kelas K01 Mata Kuliah IF2211 Strategi Algoritma beserta teman dan keluarga yang telah memberikan bantuan dan dorongan dalam penyelesaian makalah ini.

REFERENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2021-2022/Pohon-2021-Bag2.pdf. Diakses pada tanggal 18 Mei 2023

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf. Diakses pada tanggal 22 Mei 2023

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf. Diakses pada tanggal 22 Mei 2023

[4] https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00. Diakses pada 19 Mei 2023

[5] http://www.gitta.info/DataCompress/en/html/rastercomp_chain.html. Diakses pada 18 Mei 2023

[6] https://www.techtarget.com/whatis/definition/lossless-and-lossy-compression. Diakses pada 18 Mei 2023.

[7] https://sumn2u.medium.com/quality-metrics-in-image-compression-f6de68df1b8. Diakses pada 22 Mei 2023

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Yogyakarta, 22 Mei 2023

Muhammad Habibi Husni - 13521169