Aplikasi Konsep Pohon K-D dalam Pengolahan Citra Digital 'Color Quantization'

Suryani Mulia Utami - 13524042
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: suryaniutami449@gmail.com , 13524042@std.stei.itb.ac.id

***Abstrak*—Citra (gambar) digital berwarna umumnya disimpan dalam sistem 24-bit yang mampu merepresentasikan hingga 16 juta warna RGB. Namun, untuk alasan efisiensi penyimpanan dan kompatibilitas sistem, seringkali digunakan representasi 8-bit yang hanya mendukung maksimal 256 warna. Untuk mengonversi gambar dari sistem 24-bit ke 8-bit, digunakan teknik *Color Quantization* yang bertujuan mengurangi jumlah warna tanpa menurunkan kualitas visual gambar secara signifikan. Makalah ini mengimplementasikan teknik *color quantization* pada gambar digital. Proses pemilihan warna representatif dilakukan dengan algoritma K-Means untuk membentuk palet warna, sementara pemetaan piksel ke palet terdekat menggunakan struktur data pohon K-D. Hasil akhir berupa gambar hasil kuantisasi dibandingkan dengan gambar aslinya untuk mengevaluasi efektivitas teknik ini dalam mempertahankan kualitas visual gambar.**

***Kata kunci*—Color Quantization (kuantisasi warna), Reduksi Warna, Citra Digital, Representasi 8-bit, Pohon K-D, K-Means clustering**

I. PENDAHULUAN

Pengolahan citra adalah suatu proses yang berkaitan dengan analisis dan manipulasi gambar, melibatkan aspek persepsi visual. Proses ini memiliki ciri di mana data masukan dan keluaran sama-sama berbentuk citra. Citra sendiri merupakan representasi visual atau tiruan dari suatu objek tertentu. Hasil citra dapat diperoleh dari berbagai sistem perekaman, seperti foto (bersifat optik), sinyal video (analog), maupun dalam bentuk digital yang dapat disimpan langsung di media penyimpanan elektronik [1].

Citra digital merupakan representasi visual yang bersifat diskrit dan dapat diolah oleh komputer. Citra ini disusun sebagai array bilangan yang merepresentasikan intensitas terang pada titik-titik tertentu (piksel). Pada citra digital berwarna, setiap piksel menampilkan warna dengan mengatur intensitas dari tiga komponen warna primer: merah (R), hijau (G), dan biru (B) [1]. Pada beberapa sistem digital, tiap-tiap warna primer dikuantisasi dengan resolusi 8-bit (24-bit total perpiksel) untuk menghasilkan warna yang mulus dan realistis (disebut *true-color*). Total warna yang dapat direpresentasikan sistem citra 24-bit ini adalah 2^{24} = 16.777.216 warna berbeda. Namun, kebutuhan memori yang besar pada representasi ini dianggap tidak efisien untuk beberapa aplikasi. Supaya hemat, sistem tersebut kemudian hanya menggunakan 8-bit perpiksel yang dipakai sebagai indeks ke tabel (palet) warna yang berisi hingga 2^8 = 256 nilai warna. Artinya, pada sistem citra 8-bit, gambar hanya dapat menggunakan warna di antara 256 warna dari total 16 juta lebih warna yang mungkin [2].

Gambar digital *true-color* umumnya mengandung puluhan ribu hingga ratusan ribu warna unik, tergantung kompleksitas dan kualitas gambar asli. Namun, untuk kebutuhan efisiensi atau kompatibilitas sistem, seringkali diperlukan representasi yang lebih sederhana. Pemilihan warna pengganti yang tidak tepat dapat menurunkan kualitas visual gambar. Salah satu teknik yang umum digunakan untuk menangani masalah ini adalah *Color Quantization*, yaitu teknik yang memungkinkan gambar direkonstruksi menggunakan jumlah warna yang jauh lebih sedikit dari gambar aslinya. Teknik ini menjadi semakin penting dalam konteks sistem citra layar 8-bit, yang hanya mampu menampilkan 256 warna. *Color quantization* memungkinkan pengurangan jumlah tanpa mengubah struktur gambar dan mengorbankan kualitas visual secara signifikan.

Pada makalah ini, akan diimplementasikan struktur pohon K-D dan algoritma K-Means dalam pembuatan program *color quantization* pada gambar digital menggunakan bahasa Python. Selanjutnya, dilakukan analisis untuk melihat sejauh mana pengurangan jumlah warna memengaruhi kualitas visual gambar, sekaligus mensimulasikan penerapan *color quantization* dalam praktik nyata, khususnya untuk mengonversi gambar 24-bit agar kompatibel dengan sistem citra digital 8-bit. Melalui pendekatan ini, akan dievaluasi apakah teknik *color quantization* benar-benar mampu mempertahankan kualitas visual gambar secara efektif meskipun jumlah warnanya dikurangi secara drastis.

II. LANDASAN TEORI

Teori utama yang digunakan dalam makalah ini adalah konsep pohon K-Dimensi, yang digunakan untuk melakukan pemetaan setiap piksel pada gambar dengan warna yang sesuai. Untuk memahami metode yang digunakan dalam makalah ini, diperlukan pemahaman mengenai konsep dasar graf dan pohon. Oleh karena itu, berikut ini dijelaskan beberapa teori dasar yang dirujuk dari [3][4][5].

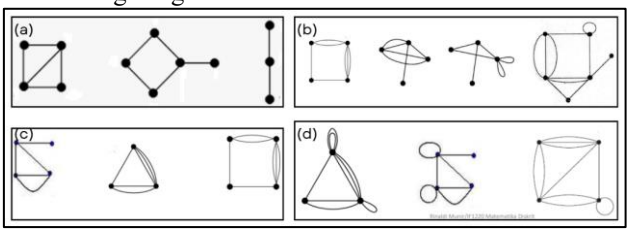
A. Graf

Graf adalah struktur yang terdiri dari titik-titik yang dihubungkan oleh garis. Titik-titik tersebut disebut simpul

(*vertices*), sedangkan garis yang menghubungkan antar simpul disebut sisi (*edges*). Graf dapat didefinisikan sebagai G = (V, E), dimana V merupakan himpunan $\{v_1, v_2, ..., v_n\}$ tidak kosong dari simpul-simpul, dan E merupakan himpunan $\{e_1, e_2, ..., e_n\}$ boleh kosong dari sisi yang menghubungkan sepasang simpul.

Berdasarkan ada-tidaknya gelang atau sisi ganda, graf dapat digolongkan menjadi dua macam:

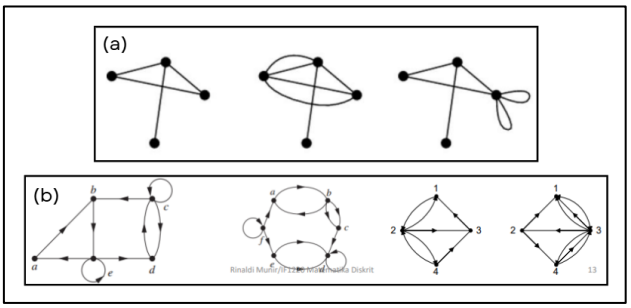
1. **Graf sederhana (*Simple graph)***
   Graf sederhana adalah graf yang tidak mengandung baik sisi ganda, maupun gelang.
2. **Graf tak sederhana (*Unsimple graph*)**
   Graf tak sederhana terbagi menjadi 2 jenis:
   - **Graf ganda**, adalah graf yang mengandung sisi ganda
   - **Graf semu**, adalah graf yang mengandung sisi gelang



Gambar 2.1 Contoh jenis graf: (a) graf sederhana, (b) graf tak sederhana, (c) graf ganda, (d) graf semu.
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/

Berdasarkan orientasi arah pada sisi, graf dapat dibedakan atas 2 jenis:

1. **Graf tak-berarah (*Directed Graph*)**
   Graf tak-berarah adalah graf yang tidak mempunyai orientasi arah.
2. **Graf berarah (*Undirected Graph*)**
   Graf berarah adalah graf yang setiap sisinya diberikan orientasi arah.



Gambar 2.2 Contoh jenis graf: (a) graf tak-berarah, (b) graf berarah.
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/

Selain itu, terdapat beberapa istilah atau terminologi di dalam graf:

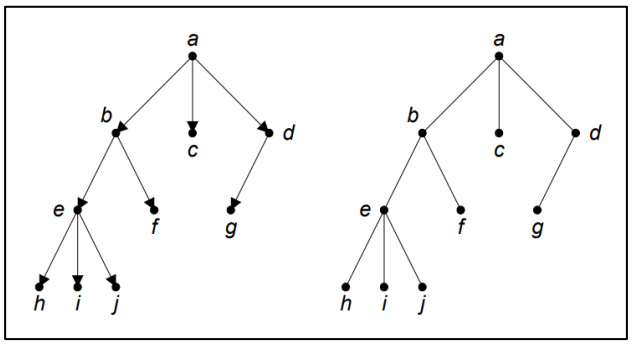
1. **Ketetanggaan (*Adjacent*)**
   Dua buah simpul dikatakan bertetangga bila terdapat sebuah sisi yang menghubungkan kedua simpul tersebut karena keduanya terhubung langsung.
2. **Bersisian (*Incidency)***
   Untuk sembarang sisi e = (v_j, v_k) dikatakan e bersisian dengan simpul v_j dan e bersisian dengan simpul v_k.
3. **Simpul terpencil (*Isolated Vertex)***
   Simpul yang tidak mempunyai sisi yang bersisian dengannya.
4. **Graf kosong (*Null Graph*)**
   Graf yang himpunan sisinya merupakan himpunan kosong.
5. **Derajat (*Degree*)**
   Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut.
6. **Lintasan (*Path)***
   Misalkan simpul awal v_0 dan simpul tujuan v_n. Maka, lintasan ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk v_0 , e_1 , v_1 , e_2 , v_2 ,... , v_n –1 , e_n , v_n sedemikian sehingga $e_1 = (v_0 , v_1)$, $e_2 = (v_1 , v_2)$, ... , $e_n = (v_n\text{-}1 , v_n)$ adalah sisi-sisi dari graf G.
7. **Sirkuit (*Cirkuit)* atau Siklus (*Cycle)***
   Lintasan yang berawal dan berakhir pada simpul yang sama.
8. **Keterhubungan (*Connected)***
   Dua buah simpul (v_1 dan v_2) dikatakan terhubung jika terdapat lintasan dari simpul v_1 ke simpul v_2. G disebut graf terhubung jika untuk setiap pasang simpul v_i dan v_j dalam himpunan V terdapat lintasan dari v_i ke v_j . Sebaliknya, graf tersebut tidak terhubung.
9. **Upagraf (*Subgraph)***
   Misalkan G = (V, E) adalah sebuah graf. $G_1 = (V_1, E_1)$ adalah upagraf dari G jika $V_1 \subseteq V$ dan $E_1 \subseteq E$.

B. Pohon

Pohon adalah graf yang memenuhi syarat berikut:

- Graf tak-berarah
- Graf terhubung
- Tidak mengandung sirkuit/siklus

Salah satu jenis pohon yang perlu diketahui dan akan digunakan pada makalah ini adalah **Pohon berakar (*Rooted tree*)**. Pohon berakar adalah pohon yang satu buah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah (tanda panah diperbolehkan tidak digambar).



Gambar 2.3 Duah buah pohon berakar yang sama
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/

Terdapat beberapa istilah atau terminologi di dalam pohon berakar:

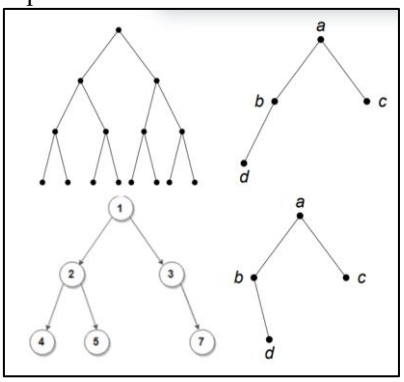
1. **Anak (*Child*) dan Orangtua (*Parent*)**

Jika simpul A terhubung ke simpul B dan A berada satu tingkat di atas B, maka A adalah orangtua dari B, dan B adalah anak dari A.

2. **Lintasan (*Path*)**
   Urutan simpul-simpul yang dilalui dari satu simpul ke simpul lain, mengikuti arah hubungan orangtua–anak.
3. **Saudara kandung (*Sibling*)**
   Simpul-simpul yang memiliki orangtua yang sama.
4. **Upapohon (*Subtree)***
   Bagian dari pohon yang terdiri dari satu simpul beserta semua keturunannya.
5. **Derajat (*Degree)***
   Derajat sebuah simpul adalah jumlah upapohon (atau jumlah anak) pada simpul tersebut.
6. **Daun (*Leaf)***
   Simpul yang berderajat nol (atau tidak mempunyai anak).
7. **Simpul dalam (*Internal nodes)***
   Simpul yang mempunyai anak.
8. **Aras (*Level)* atau tingkat**
   Jarak dari simpul ke akar, dihitung dari 0. Akar berada di aras 0, anak-anaknya di aras 1, dan seterusnya.
9. **Tinggi (*Height)* atau kedalaman**
   Aras maksimum dari suatu pohon. Bisa juga diartikan sebagai jumlah tingkat dari akar ke daun terdalam.

Berikut adalah jenis dari pohon berakar yang perlu untuk diketahui:

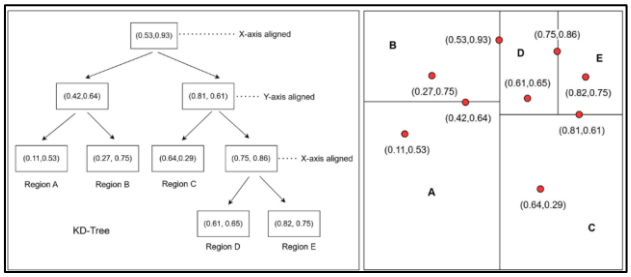
1. **Pohon terurut (*Ordered tree*)**
   Pohon di mana urutan anak-anak dari setiap simpul penting. Artinya, anak kiri dan anak kanan dianggap berbeda meskipun memiliki isi yang sama.
2. **Pohon n-ary**
   Pohon di mana setiap simpul memiliki paling banyak n anak. Misalnya, dalam pohon 3-ary, setiap simpul maksimal punya 3 anak.
3. **Pohon biner (*Binary tree*)**
   Pohon khusus dari n-ary *tree* dengan maksimal 2 anak per simpul, biasanya disebut anak kiri dan anak kanan. Pohon biner adalah pohon terurut.



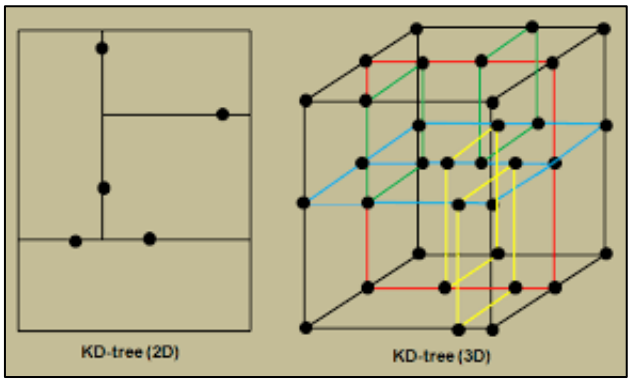
Gambar 2.4 Contoh pohon biner
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/

C. *Pohon K-D*

Pohon K-D atau *K-Dimentional Trees* adalah pohon biner yang setiap simpulnya menyimpan sebuah titik dalam bentuk *tuple*. Huruf 'K' menunjukkan jumlah dimensi dari titik tersebut. Setiap simpul non-daun membagi ruang menjadi dua bagian berdasarkan salah satu dimensi (*axis*) secara bergantian di tiap level pohon. Titik-titik yang nilainya lebih kecil akan dimasukkan ke subpohon kiri, sedangkan yang lebih besar ke subpohon kanan.



Gambar 2.5 Contoh pohon 2D dan ilustrasi partisi ruangnya
Sumber: https://pyimagesearch.com/2024/12/23/implementing-approximate-nearest-neighbor-search-with-kd-trees/



Gambar 2.6 Ilustrasi partisi ruang pada data 2D dan 3D
Sumber: https://www.researchgate.net/figure/The-example-of-the-KD-tree-Algorithms-2D-3D_fig3_263937521

D. *Pencarian Nearest Neighbor berbasis pohon K-D*

Untuk menggunakan pohon KD- dalam pencarian *nearest neighbor*, proses dimulai dari simpul akar dan menelusuri pohon dengan mengikuti jalur yang paling mungkin mengandung titik terdekat (*good side)* sampai mencapai simpul daun.

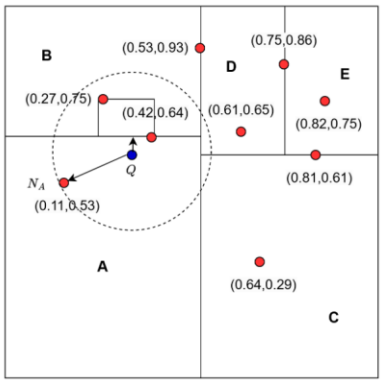
Pada setiap pencarian akan dihitung jaraknya dan disimpan jarak terbaik (terdekat). Jarak antara dua titik dalam sistem koordinat tiga dimensi dapat dihitung menggunakan rumus:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - \mathrm{z1})^2}$$

di mana:

- d adalah jarak antara dua titik,
- $x1, y1, z1, x_2, y_2, z_2$ adalah koordinat dari dua titik yang dibandingkan.

Algoritma *nearest neighbor* tidak selalu melakukan *backtracking*. Langkah ini hanya dilakukan apabila pembagian ruang oleh *axis* menyebabkan titik terdekat justru berada di sisi yang tidak dikunjungi terlebih dahulu.



Gambar 2.7 Perhitungan jarak dari titik target Q ke *axis* pembagi saat ini
Sumber: https://pyimagesearch.com/2024/12/23/implementing-approximate-nearest-neighbor-search-with-kd-trees/

Hal ini dapat diperiksa dengan cara menghitung jarak antara titik target dengan komponen koordinat pada *axis* pembagi di simpul saat ini. Jika jaraknya lebih kecil dari jarak terbaik saat ini, maka cabang pohon yang lain perlu dieksplorasi (*backtrack)*. Teknik ini memungkinkan pencarian yang efisien tanpa harus memeriksa semua titik dalam data [6].

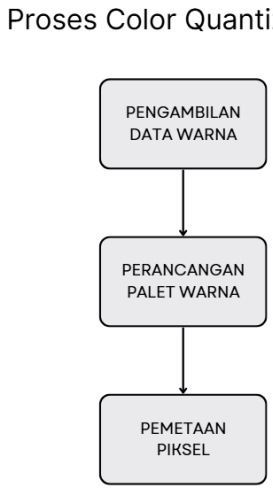
E. Kluster K-Means

K-Means merupakan salah satu algoritma *clustering* yang umum digunakan untuk mengelompokkan data berdasarkan kesamaan fitur. Algoritma ini bekerja secara iteratif mempartisi kumpulan data menjadi sejumlah kluster. *Centroid*, atau pusat kluster, adalah rata-rata atau median dari semua titik di dalam kluster, tergantung pada karakteristik data [7].

III. ANALISIS

A. Metode Eksperimen

Bagian ini menjelaskan struktur proses *color quantization* terhadap gambar digital yang dapat dibagi menjadi tiga tahapan utama, sebagai berikut:



Tahapan pertama adalah Pengambilan Data Warna dari gambar, yakni memformat setiap piksel ke dalam bentuk RGB. **Tahapan kedua** adalah Perancangan Palet Warna (*Color Palette Design*), yakni memilih sejumlah warna RGB yang paling representatif untuk mewakili keseluruhan warna dalam gambar**. Tahapan ketiga** adalah Pemetaan Piksel (*pixel mapping*), yakni mencocokkan setiap piksel pada gambar dengan warna yang paling sesuai (terdekat) dari palet sehingga menghasilkan *quantized image* yang menggunakan jumlah warna lebih sedikit dari aslinya [8].

B. Implementasi Program

Berikut ditampilkan implementasi kode menggunakan bahasa Python untuk setiap tahapan proses *Color Quantization*:

1. Pengambilan data warna dari gambar

Menginisiasi gambar yang akan digunakan sebagai *input* dengan cara memuatnya dan mengubah ukurannya menjadi 256x256 piksel. Setelahnya, gambar direpresentasikan sebagai array NumPy berdimensi *(256, 256, 3)* yang menyimpan nilai warna RGB dari masing-masing piksel dalam bentuk tiga bilangan bulat (uint8) dengan rentang 0–255.

```
def load_and_resize_image(path, size=(256, 256)):
    """
    Memuat gambar dari path, mengubah ke RGB, dan mengubah ukurannya.
    """
    image = Image.open(path).convert("RGB")
    image = image.resize(size)
    return np.array(image)
```

Seperti ini contoh *output* dari fungsi `load_and_resize_image`:

```
array([[[255, 255, 255],
        [255, 255, 255],
        [241, 241, 237],
        ...,
        [  7,   6,   2],
        [  7,   6,   4],
        [  4,   5,   8]],

       [[249, 249, 249],
        [245, 244, 245],
        [226, 226, 229],
        ...,
        [ 90,  88,  12],
        [ 88,  87,  14],
        [ 42,  44,  14]],

       [[239, 239, 239],
        [221, 221, 221],
        [199, 198, 204],
        ...,
        [107,  99,  10],
        [104,  96,   8],
        [ 48,  48,   8]],

       ...,

       [[204, 204, 204],
        [143, 143, 143],
        [115, 115, 115],
        ...,
        [143, 143, 143],
        [208, 208, 208],
        [250, 250, 250]],

       [[221, 221, 221],
        [181, 181, 181],
        [164, 164, 164],
        ...,
        [180, 180, 180],
        [223, 223, 223],
        [251, 251, 251]],

       [[238, 238, 238],
        [216, 216, 216],
        [207, 207, 207],
        ...,
        [216, 216, 216],
        [239, 239, 239],
        [252, 252, 252]]], shape=(256, 256, 3), dtype=uint8)
```

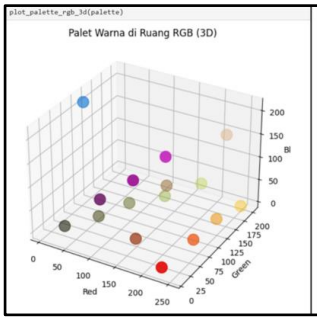
Struktur ini yang akan digunakan sebagai *input* dalam proses komputasi warna selanjutnya.

2. Perancangan palet warna

Menentukan warna-warna representatif dari gambar menggunakan K-Means. Di bawah ini adalah algoritma untuk K-Means yang menerima masukan `pixels` (*output* fungsi `load_and_resize_image`) dan `n_colors`, yakni integer jumlah warna palet yang dapat diatur sesuai keinginan.

```
def extract_palette_kmeans(pixels, n_colors=16):
    """
    Melakukan clustering warna dengan K-Means untuk mendapatkan palet warna.
    """
    pixels_reshaped = pixels.reshape(-1, 3)
    kmeans = KMeans(n_clusters=n_colors, random_state=42)
    kmeans.fit(pixels_reshaped)
    palette = np.round(kmeans.cluster_centers_).astype(np.uint8)
    return palette
```

Hasilnya, terbentuk palet warna yang akan digunakan untuk *mapping* pada tahapan selanjutnya. Bila divisualisasikan, seperti ini contoh isi dari `palette`:

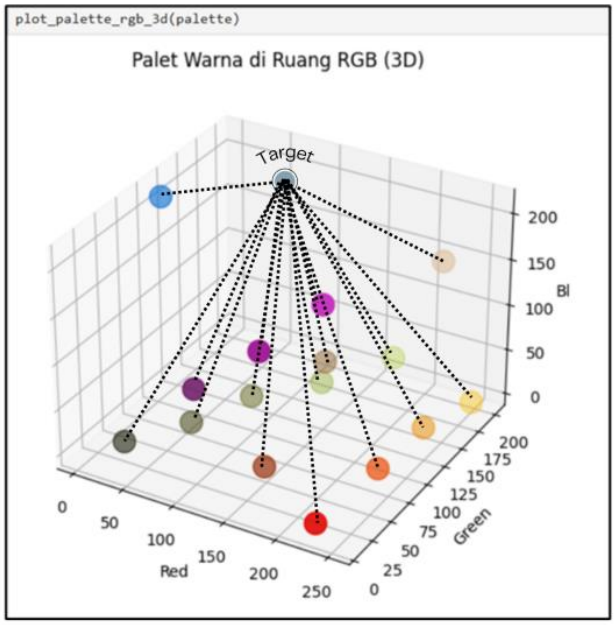



```
array([[201,  54, 193],
       [ 97, 100,  26],
       [  2, 107, 211],
       [ 64,  64,  13],
       [252, 191,   2],
       [237,  79,  10],
       [167, 201,  17],
       [ 26,  27,   7],
       [157, 118,  71],
       [223, 187, 153],
       [228,   9,   4],
       [104,  19,  98],
       [124, 159,   6],
       [151,  46,   8],
       [161,  27, 151],
       [237, 143,   5]], dtype=uint8)
```

3. Pemetaan piksel

Seperti namanya, proses yang dilakukan adalah memetakan setiap piksel pada gambar ke salah satu warna dalam palet. Misalkan himpunan warna dalam palet warna dinotasikan dengan $Y = \{yv \mid v = 1,2, \ldots, k\}$, maka setiap warna piksel xi akan disubstitusikan dengan warna yv yang paling mendekatinya [9].

Misalkan, target adalah nilai RGB dari suatu piksel.



Untuk menentukan warna substitusi dari palet, perlu dilakukan pencarian titik warna dalam palet yang memiliki jarak terdekat terhadap target dalam ruang RGB. Secara konseptual, hal ini dapat dibayangkan sebagai penarikan garis lurus dari target ke setiap titik warna di palet, lalu memilih yang paling dekat. Namun, menghitung jarak satu persatu secara manual menjadi sangat sulit, mengingat kita bekerja dalam ruang 3D dan harus memetakan 256×256 = 65.536 piksel. Itu pun baru untuk palet berisi 16 warna. Penggunaan palet yang lebih besar akan membuat proses ini nyaris mustahil dilakukan tanpa bantuan komputer.

Oleh karena itu, diperlukan suatu algoritma yang dapat menyederhanakan proses pencarian titik warna terdekat tersebut. Dalam konteks ini, digunakan algoritma *nearest neighbor* berbasis pohon pencarian K-Dimensi (K=3). Struktur data ini menyimpan koordinat dalam bentuk tuple yang terdiri dari tiga elemen, yaitu nilai R (merah), G (hijau), dan B (biru) dari setiap titik warna dalam palet.

Di bawah ini adalah algoritma untuk membangun pohon K-Dimensi dari points (adalah `palette`). Pada setiap langkah rekursif, algoritma akan mengurutkan points berdasarkan nilai koordinat pada axis saat ini, kemudian memilih titik median sebagai node utama. Selanjutnya, axis diperbarui secara siklik, dan algoritma secara rekursif membangun subpohon kiri dan subpohon kanan.

```
class kdTree:
    def __init__(self, points, depth = 0):
        self.left = None
        self.right = None

        if len(points) == 0:
            self.point = None
            return

        k = len(points[0])  # jumlah dimensi
        axis = depth % k
        points.sort(key = lambda x: x[axis])
        median_index = len(points) // 2

        self.point = points[median_index]
        self.axis = axis

        if median_index > 0:
            self.left = kdTree(points[:median_index], depth + 1)
        if median_index + 1 < len(points):
            self.right = kdTree(points[median_index + 1:], depth + 1)
```

Di bawah ini adalah algoritma untuk pencarian titik terdekat ke target. Penelusuran dimulai dari akar pohon, kemudian dilanjutkan ke cabang yang paling berpotensi mengandung titik terdekat terhadap target (disebut *'good side'*). Setelah menelusuri seluruh *good side*, algoritma akan memeriksa apakah *bad side* juga perlu ditelusuri. Keputusan ini dilakukan dengan membandingkan selisih nilai komponen koordinat pada axis saat ini dengan jarak terbaik yang sudah ditemukan. Dalam setiap langkah, program menghitung jarak dari titik yang sedang diperiksa (`current_rgb`) ke titik target, sehingga pada akhir pencarian akan dihasilkan jarak dan titik terdekat ke target yang disimpan pada variabel `best`.

```python
# Menghitung Euclidean distance antara dua titik RGB.
def distance(p1, p2):
    r1, g1, b1 = [int(v) for v in p1]
    r2, g2, b2 = [int(v) for v in p2]
    return math.sqrt((r1 - r2)**2 + (g1 - g2)**2 + (b1 - b2)**2)
```

```python
# Pencarian titik terdekat untuk satu target (pemetaan untuk satu piksel)
def nearest(node, target, depth=0, best=None):
    if node is None or node.point is None:
        return best

    axis = depth%3
    current_rgb = node.point
    current_dist = distance(current_rgb, target)

    # Inisiasi atau update best
    if best is None or current_dist < best[0]:
        best = (current_dist, current_rgb)

    # Menentukan bagian 'good side' dan 'bad side'
    if (current_rgb[axis] > target[axis]):
        good, bad = node.left, node.right
    else:
        good, bad = node.right, node.left

    # Telusuri good side terlebih dahulu
    best = nearest(good, target, depth+1, best)

    # Cek apakah perlu menelusuri bad side
    if abs(int(target[axis]) - int(current_rgb[axis])) < best[0]:
        best = nearest(bad, target, depth+1, best)

    return best
```

Fungsi `nearest` digunakan untuk melakukan pemetaan satu piksel pada gambar asli ke warna terdekat dalam palet. Untuk melakukan pemetaan terhadap seluruh piksel pada gambar, digunakan fungsi `map_pixels_to_palette` yang memanfaatkan `nearest` secara iteratif.

```python
# Pemetaan untuk seluruh piksel menggunakan fungsi nearest
def map_pixels_to_palette(pixels, palette, kdtree):
    height, width, _ = pixels.shape
    mapped_pixels = np.zeros_like(pixels)

    for i in range(height):
        for j in range(width):
            target = tuple(int(v) for v in pixels[i, j])
            _, closest_rgb = nearest(kdtree, target)
            mapped_pixels[i, j] = np.array(closest_rgb, dtype=np.uint8)

    return mapped_pixels
```

Hasil dari proses ini berupa gambar hasil kuantisasi yang direpresentasikan dalam array NumPy berdimensi *(256, 256, 3)*. Setiap elemen pada array tersebut menyimpan nilai warna RGB dari masing-masing piksel dalam bentuk tiga bilangan bulat (uint8) dengan rentang nilai 0–255. Seperti ini contoh *output* dari fungsi `map_pixels_to_palette`:

```
array([[[223, 187, 153],
        [223, 187, 153],
        [223, 187, 153],
        ...,
        [ 26,  27,   7],
        [ 26,  27,   7],
        [ 26,  27,   7]],

       [[223, 187, 153],
        [223, 187, 153],
        [223, 187, 153],
        ...,
        [ 97, 100,  26],
        [ 97, 100,  26],
        [ 26,  27,   7]],

       [[223, 187, 153],
        [223, 187, 153],
        [223, 187, 153],
        ...,
        [ 97, 100,  26],
        [ 97, 100,  26],
        [ 64,  64,  13]],

       ...,

       [[223, 187, 153],
        [157, 118,  71],
        [157, 118,  71],
        ...,
        [157, 118,  71],
        [223, 187, 153],
        [223, 187, 153]],

       [[223, 187, 153],
        [223, 187, 153],
        [223, 187, 153],
        ...,
        [223, 187, 153],
        [223, 187, 153],
        [223, 187, 153]],

       [[223, 187, 153],
        [223, 187, 153],
        [223, 187, 153],
        ...,
        [223, 187, 153],
        [223, 187, 153],
        [223, 187, 153]]], shape=(256, 256, 3), dtype=uint8)
```

C. *Hasil Program*

Berikut ditampilkan kode dari program utama *Color Quantization* yang diimplementasikan menggunakan bahasa Python:

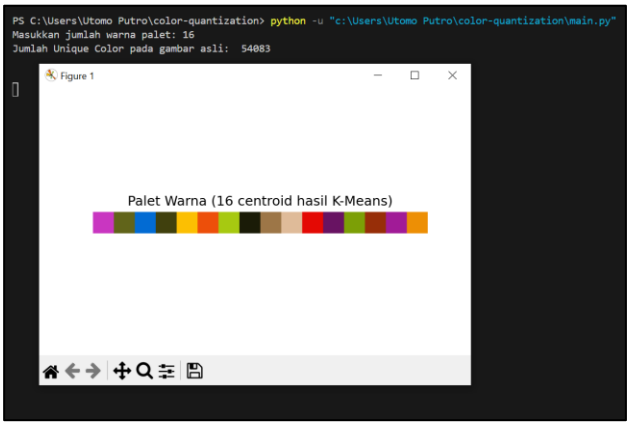
```python
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
import math
```

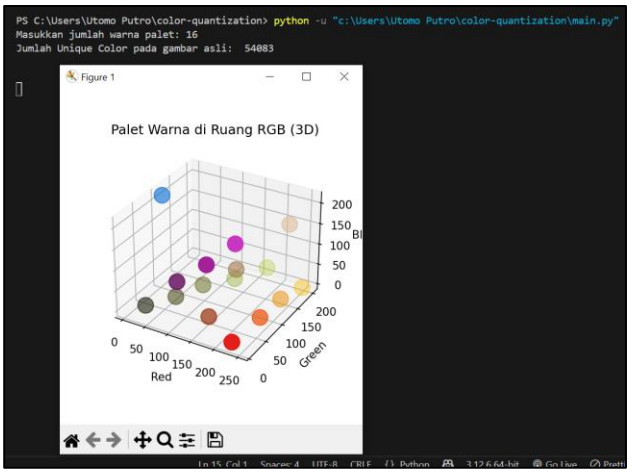
```python
from function import *

n_colors = int(input("Masukkan jumlah warna palet: "))
# load dan resize gambar
path = r"C:\Users\Utomo Putro\color-quantization\fotosampel.jpg"
original_pixels = load_and_resize_image(path, size=(256, 256))

print("Jumlah Unique Color pada gambar asli: ", count_unique_colors(original_pixels))
print("\n")

# buat palette dengan jumlah warna sesuai input user
palette = extract_palette_kmeans(original_pixels, n_colors)
show_palette(palette)
plot_palette_rgb_3d(palette)

# bangun pohon dari palette
P = [tuple(rgb) for rgb in palette]
tree = kdTree(P)

# lakukan pixel mapping
quantized_pixels = map_pixels_to_palette(original_pixels, palette, tree)

print("Jumlah Unique Color setelah kuantisasi: ", count_unique_colors(quantized_pixels))
print("\n")

print("Perbandingan gambar sebelum dan setelah kuantisasi: \n")
show_comparison(original_pixels, quantized_pixels)
```

Gambar 3.1 *Input* program
Sumber: https://webneel.com/daily/7-colorful-photography?size=_original

Setelah mengatur path gambar yang ingin dikuantisasi, program dapat dijalankan. Pertama-tama, program akan meminta user memasukkan jumlah warna palet yang diinginkan. Setelah itu, program akan menampilkan jumlah warna unik pada gambar asli, menampilkan gambar palet dalam bentuk bar, menampilkan gambar palet warna di ruang RGB, menampilkan jumlah warna unik setelah kuantisasi, dan menampilkan perbandingan gambar hasil kuantisasi.

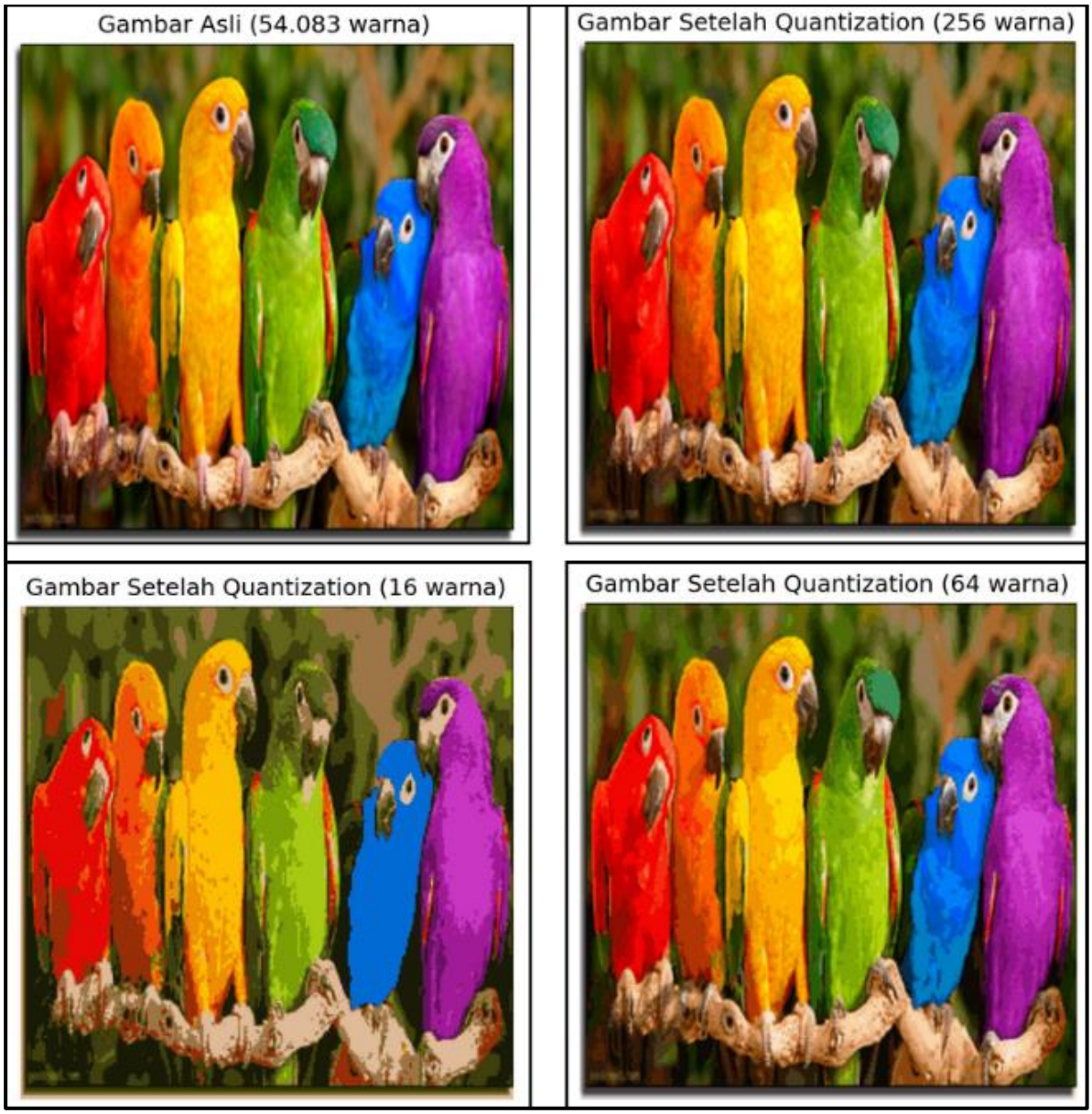
Berikut adalah contoh *output* dari program dengan input jumlah warna palet=16:





Program telah diuji dengan berbagai jumlah warna, seperti 8, 32, 64, dan 256. Dari percobaan tersebut, diperoleh hasil bahwa program dapat berjalan dengan baik untuk seluruh variasi *input* tersebut maupun kemungkinan input lainnya, selama jumlah warna yang dimasukkan tidak melebihi jumlah warna unik pada gambar asli. Dari percobaan ini, diperoleh hasil bahwa teknik *color quantization* merupakan metode yang efektif untuk mengurangi jumlah warna tanpa mengorbankan kualitas visual secara signifikan.

Sebagai contoh, pada gambar yang diuji (hasil dapat berbeda untuk *input* gambar lainnya), penggunaan palet 8 atau 64 warna menghasilkan kualitas visual yang masih terlihat kasar. Namun, pada palet 256 warna, hasil gambar hampir tidak bisa dibedakan dari gambar aslinya. Temuan ini menunjukkan bahwa teknik *color quantization* berbasis pohon K-D dan K-means terbukti efektif dalam mempertahankan kualitas visual gambar pada proses konversi gambar digital ke dalam format 8-bit yang hanya mendukung hingga 256 warna.



IV. KESIMPULAN

Pohon K-D adalah struktur data pohon biner yang digunakan untuk membagi ruang berdimensi banyak, seperti ruang warna RGB. Dalam konteks *color quantization*, pohon ini memungkinkan proses pencarian warna terdekat dalam palet dilakukan jauh lebih efisien dibandingkan pencarian linear. Sementara itu, K-Means merupakan algoritma *clustering* yang umum digunakan untuk mengelompokkan data berdasarkan kesamaan fitur, seperti pengelompokkan warna untuk menghasilkan palet.

Namun, metode yang digunakan dalam makalah ini memiliki keterbatasan dalam hal efisiensi. Kompleksitas komputasi cukup tinggi karena membandingkan seluruh piksel gambar [10][11]. Dalam praktik industri, seringkali digunakan pendekatan yang lebih efisien, seperti *octree-based clustering* dengan histogram warna. Pendekatan tersebut mampu mengurangi beban komputasi karena hanya mempertimbangkan frekuensi warna unik, bukan seluruh piksel [12]. Meskipun bukan pendekatan paling efisien, implementasi K-Means dan pohon K-D dalam makalah ini tetap merepresentasikan prinsip dasar *color quantization* pada gambar digital secara baik.

V. UCAPAN TERIMA KASIH

Penulis menyampaikan puji syukur kepada Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya sehingga makalah ini dapat disusun dan diselesaikan dengan baik sebagai bagian dari tugas akhir mata kuliah IF1220 Matematika Diskrit. Penulis juga menyampaikan rasa terima kasih yang sebesar-besarnya kepada Dr. Ir. Rinaldi Munir, M.T. dan Arrival Dwi Sentosa, S.Kom., M.T. selaku dosen pengampu mata kuliah ini, atas ilmu, arahan, serta bimbingan yang telah diberikan selama perkuliahan. Penulis juga mengucapkan terima kasih kepada Dessi Puji Lestari, S.T., M.Eng., Ph.D. selaku pengampu mata

kuliah WI2002 Literasi Data dan IA, yang melalui penjelasannya mengenai pengolahan citra digital telah menginspirasi penulis dalam menentukan topik makalah ini. Selain itu, penulis berterima kasih kepada seluruh penulis jurnal, artikel, serta sumber referensi lainnya yang telah menjadi rujukan dalam penyusunan makalah ini. Akhir kata, penulis memohon maaf apabila terdapat kekeliruan dalam penulisan maupun penyampaian isi makalah ini.

VIDEO LINK YOUTUBE

https://youtu.be/KpljkOng5mE?si=wHoufwduzYpzqngH

REFERENSI

[1] D. I. S. Saputra, M. A. Triwibowo, M. F. Noeris, dan M. Alasad, "Pengolahan citra negatif klise menjadi citra true color dengan Matlab," *Jurnal Ilmiah SISFOTENIKA*, vol. 7, no. 1, pp. 86-95, 2017.

[2] M. Orchard dan C. Bouman, "Color Quantization of Images," *IEEE Trans. on Sig. Proc*., vol. 39, no. 12, pp. 2677-2690, 1991.

[3] R. Munir, *Graf (Bag. 1)*, 2024. [Daring]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf. [Diakses: 19 Juni 2025].

[4] R. Munir, *Pohon (Bag. 1),* 2024. [Daring]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/23-Pohon-Bag1-2024.pdf. [Diakses: 19 Juni 2025].

[5] R. Munir, *Pohon (Bag. 2),* 2024. [Daring]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/24-Pohon-Bag2-2024.pdf. [Diakses: 19 Juni 2025].

[6] P. Mangla, *Implementing Approximate Nearest Neighbor Search with KD-Trees*, 2024. [Daring]. Tersedia: https://pyimagesearch.com/2024/12/23/implementing-approximate-nearest-neighbor-search-with-kd-trees/. [Diakses: 14 Juni 2025].

[7] E. Kavlakoglu dan V. Winland, *Apa itu k-means clustering?*, 2024. [Daring]. Tersedia: https://www.ibm.com/id-id/topics/k-means-clustering. [Diakses: 14 Juni 2025].

[8] P. S. Heckbert, "Color image quantization for frame buffer display," *Computer Graphics*, vol. 16, no. 3, pp. 297-307, 1982.

[9] G. H. Liu dan J. Y. Yang, "Content-based image retrieval using color difference histogram," *Pattern Recognition*, vol. 46, no. 1, pp. 188-198, 2013.

[10] H. Paulus dan M. Frackiewicz, "New approach for initialization of k-means technique applied to color quantization," dalam *Proceedings of the 2010 2nd International Conference on Information Technology, (2010 ICIT)*, Gdansk, Poland, 28–30 Juni 2010, pp. 205–209.

[11] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, dan A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 24, no. 7, pp. 881–892, 2002.

[12] L. Cris, *Color quantization, minimizing variance, and k-d trees*, 2018. [Daring]. Tersedia: https://www.crisluengo.net/archives/932/. [Diakses: 10 Juni 2025].

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.

Bandung, 1 Juni 2025

Suryani Mulia Utami
13524042

LAMPIRAN

Link program:
https://github.com/suryaniutaami/color-quantization.git