Aplikasi Algoritma Flood Fill dalam Software Pemrosesan Grafis

Matthew Kevin Amadeus 13518035
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesa 10 Bandung 40132, Indonesia
mkamadeus.mka@gmail.com 13518035@std.stei.itb.ac.id

***Abstrak*—Makalah ini menjelaskan mengenai bagaimana penggunaan algoritma flood fill dalam software yang berbasiskan pada pemrosesan grafis. Walaupun mungkin tidak terlalu terlihat, kegunaan dari algoritma flood fill ini sudah diterapkan dalam berbagai kakas yang disediakan oleh software pemrosesan grafis tersebut, seperti untuk membuat suatu seleksi atau mengisi suatu bagian dengan warna tertentu.**

Kata kunci—flood fill; BFS; DFS; image processing; selection; paint bucket; magic wand; Photoshop

I. Pengenalan

Penggunaan software untuk memanipulasi foto atau konten digital lainnya saat ini sudah menjadi hal yang sangat lumrah. Baik bagi para profesional yang membuat seni fotografi, animasi, dan lainnya; bahkan sampai ke masyarakat awam saja sudah dengan mudah menggunakan software pemrosesan grafis ini untuk keperluan pribadi. Dengan mudah, di masa kini software semacam itu sudah dengan mudah diakses oleh kalangan manapun.

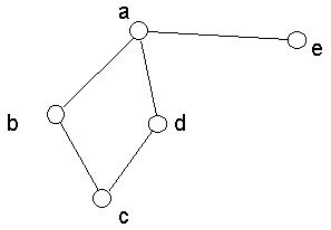
Di balik penggunaannya yang luas di masyarakat, dalam software pemrosesan grafis ini terdapat hal menarik yang bisa diperhatikan dari sisi algoritmanya. Contohnya adalah seperti software-software yang telah diberikan kemampuan untuk mendeteksi suatu subjek dan memisahkannya dari latar belakang suatu gambar. Contoh yang lebih sederhananya adalah software suatu kakas yang disediakan dalam software untuk mengisi suatu daerah dengan sebuah warna ataupun diberikan kemampuan untuk memilih suatu daerah tersebut.

Dari contoh-contoh yang diberikan itu, beberapa implementasi algoritma itu dapat direalisasikan dengan suatu jenis yang disebut sebagai algoritma flood fill. Dalam makalah ini, akan dibahas aplikasi algoritma tersebut dalam contoh-contoh yang dapat merepresentasikan ide yang disebut sebelumnya secara sederhana.

II. Dasar Teori

A. *Graf*

Untuk memahami flood fill, pertama harus memahami konsep dari teori graf terlebih dahulu, terutama untuk traversal graf. Graf adalah sekumpulan titik yang bisa disebut sebagai *vertices/nodes* yang terhubung melalui sekumpulan sisi yang bisa disebut sebagai *edges*. Graf bisa memiliki atau tidak memiliki nilai berat yang bisa disebut sebagai *weight* di sisi-sisinya, tergantung jenis graf dan representasi suatu permasalahannya.



Gambar 2.1 : Ilustrasi awal mula penelusuran dengan algoritma *flood fill*. (Sumber: http://www.analytictech.com/mb021/graphtheory.htm)

Dalam memecahkan suatu masalah, masalah tersebut terkadang relatif lebih mudah dan jelas dikatakan memiliki representasi permasalahan dalam bentuk graf, contoh seperti merepresentasikan kota-kota sebagai titik-titik yang terdefinisi dalam graf dan jalan-jalan yang menghubungkan kota-kota dengan merepresentasikan jalan-jalan tersebut sebagai sisi yang menghubungkan titik-titik yang ada. Terkadang, mungkin penggunaan graf kurang terlihat di bidang-bidang seperti contohnya dalam penggunaannya dalam software-software pemrosesan grafis.

B. *Penelusuran Graf*

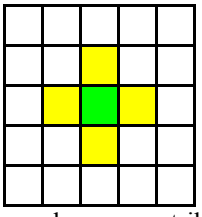
Terdapat beberapa algoritma yang digunakan untuk melakukan penelusuran graf. Pada makalah ini, akan dibahas dua penelusuran graf yang paling umum, yaitu penelusuran BFS (*Breadth First Search*) dan DFS (*Depth First Search*).

Algoritma BFS, sesuai namanya akan melakukan penelusuran secara “melebar”. Kata melebar di sini memiliki arti bahwa penelusuran akan dilakukan untuk setiap titik yang terhubung dengan titik tersebut. Setelah semua titik yang terhubung telah ditelusuri, maka penelusuran yang serupa akan dilakukan untuk titik-titik berikutnya. Proses akan diulang sampai semua titik telah dikunjungi. Algoritma BFS memiliki kemiripan dengan struktur data *queue* dalam melakukan urutan penelusurannya.

Algoritma DFS memiliki prinsip yang bertolak belakang dengan algoritma BFS. Jika BFS memiliki prinsip untuk mencari seluruh titik yang terhubung dengan titik yang sekarang sedang diproses, DFS memiliki prinsip untuk memilih salah satu titik yang terhubung dan belum dikunjungi secara terus menerus hingga tidak dapat melakukan penelusuran lebih lanjut. Bila tidak bisa melakukan penelusuran lanjut dari suatu titik, akan dilakukan *backtracking* untuk mencari titik lain yang mungkin ditelusuri dari titik-titik sebelumnya, dan akan mengulang proses yang sama. Algoritma DFS biasa dimanfaatkan dalam prinsip *backtracking* secara umum. Algoritma DFS ini memiliki kemiripan dengan struktur data *stack* dalam melakukan urutan penelusurannya.

C. Representasi Matriks Sebagai Sebuah Graf

Dalam penyelesaian masalah yang berhubungan dengan matriks, kadang bisa ditemui suatu masalah yang berhubungan dengan graf. Persoalan matriks yang mengandung unsur penelusuran bisa dianggap sebagai suatu graf implisit. Untuk lebih jelasnya, perhatikan ilustrasi berikut.



Gambar 2.2 : Ilustrasi penelusuran matriks melalui pendekatan teori graf. (Sumber: Penulis)

Kotak hijau tersebut menyimbolkan titik awal penelusuran suatu matriks, sedangkan kotak kuning adalah kemungkinan jalur yang mungkin dilalui dari kotak hijau tersebut. Dengan sudut pandang seperti ini, kotak-kotak tersebut mudah disimbolkan sebagai suatu graf. Untuk setiap kotak di dalam matriks itu dapat direpresentasikan sebagai titik dalam graf, dan jalur-jalur yang mungkin disimbolkan sebagai sisi-sisi dari graf tersebut. Dalam kasus ini, sisi yang mungkin adalah kotak yang bersebelahan tepat bersebelahan dengan kotak yang sedang diproses.

D. Algoritma Flood Fill

Dengan representasi matriks dalam bentuk graf seperti bagian sebelumnya, secara sederhana algoritma flood fill adalah algoritma yang memanfaatkan penelusuran graf baik BFS maupun DFS dalam sebuah matriks dengan aturan penelusuran valid tertentu. Persoalan yang sering diselesaikan menggunakan algoritma flood fill ini salah satunya adalah melakukan simulasi pengisian suatu daerah tertentu yang dibatasi dengan suatu penanda. Secara lebih formal, *flood fill* adalah suatu algoritma yang digunakan untuk mencari simpul yang terhubung dengan suatu simpul lainnya dalam suatu array multidimensi [1], sehingga matriks seperti gambar dapat diaplikasikan algoritma *flood fill* ini. Untuk jelasnya bisa ditunjukan dalam contoh berikut ini.

```
############
#***#***#**#
#***#******#
#***#******#
############
```

Gambar 3.2 : Ilustrasi awal mula penelusuran dengan algoritma *flood fill*. (Sumber: Penulis)

Persoalan tersebut memisalkan simbol * dengan daerah kosong dan simbol # dengan tembok. Misalkan daerah kosong tersebut akan diisi dengan air, algoritma flood fill dapat menentukan daerah mana saja yang akan terisi dengan air dan daerah mana yang tidak terisi dengan air. Bila pengisian air dilakukan pada simbol yang ditandai tersebut, maka hasilnya akan sebagai berikut:

```
############
#***#***#**#
#***#******#
#***#******#
############
```

Gambar 2.3 : Ilustrasi hasil akhir dari penggunaan algoritma *flood fill*. (Sumber: Penulis)

Dalam sudut pandang penelusuran graf, algoritma flood fill bisa dilihat sebagai berikut:

1) Kotak untuk melakukan pengisian air disimbolkan sebagai titik untuk memulai penelusuran graf.
2) Kotak-kotak yang berada tepat di sebelah kotak yang sedang diproses merepresentasikan titik dalam graf yang terhubung dengan titik yang sedang diproses.

Dengan memahami sudut pandang tadi, baik penelusuran secara BFS ataupun DFS bisa diaplikasikan untuk algoritma flood fill.

E. Program Adobe Photoshop

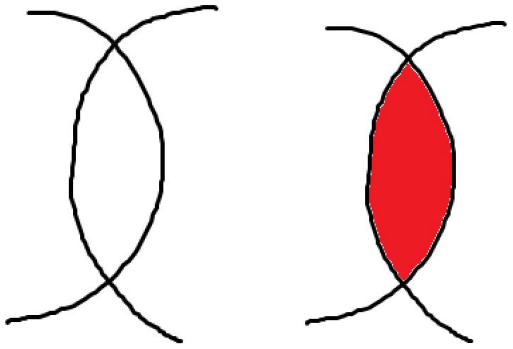
Dalam makalah ini, algoritma yang dijelaskan akan bersinggungan dengan program pengolahan grafis yang sangat populer di dunia seni digital, yaitu Adobe Photoshop. Adobe Photoshop ini adalah bagian dari kumpulan program Adobe Creative Cloud yang dibuat oleh Adobe. Photoshop memiliki kemampuan untuk mengolah grafis yang berbasis pixel (*raster*); berbeda dengan Adobe Illustrator yang mengolah secara *vector*.

Gambar 2.4 : Logo dari aplikasi Adobe Photoshop. (Sumber: https://iconscout.com/icon/adobe-photoshop-4)

Adobe Photoshop ini memiliki beberapa fitur penting yang akan dibahas agar dapat memahami isi dari makalah ini. Fitur pertama yaitu adanya kemampuan untuk mengubah informasi yang terkandung dalam suatu pixel. Sederhananya, Photoshop dapat mengubah warna dan transparansi suatu pixel. Selain itu, fitur yang tidak kalah penting adalah fitur yang memungkinkan pengguna untuk membuat seleksi. Seleksi di Photoshop ini memiliki kemampuan untuk membatasi suatu operasi pada daerah seleksi tersebut. Misalnya, pengguna hendak memberi warna pada gambar di bagian yang spesifik. Hal tersebut dalam Photoshop mudah dicapai menggunakan fitur seleksi tersebut.

III. Paint Bucket

Paint Bucket adalah salah satu kakas yang populer dalam aplikasi pemrosesan grafis, bahkan di program yang sederhana seperti Microsoft Paint sudah disediakan dari dulu. Jika diperhatikan, cara kerja kakas Paint Bucket yang disediakan dalam Microsoft Paint tersebut bekerja seperti algoritma flood fill, yaitu dengan mengisi bagian yang memiliki warna yang sama dengan suatu warna yang telah ditentukan.



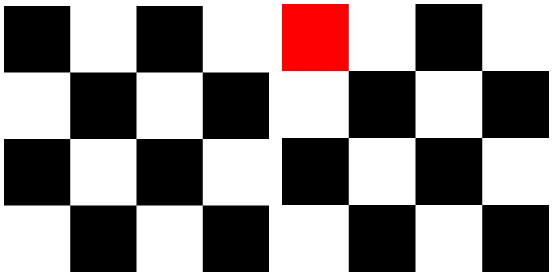
Gambar 3.1 : Ilustrasi penggunaan *paint bucket* menggunakan program Microsoft Paint(Sumber: Penulis)

Dalam program Adobe Photoshop, kakas *paint bucket* ini juga disediakan. Cara kerjanya pun sama, yaitu mengisi pada bagian dengan warna yang serupa pada satu daerah [2]. Bisa dikatakan, *paint bucket* ini adalah kakas yang populer dan standar di program-program pemrosesan grafis. Dalam Photoshop, terdapat fitur untuk mengatur parameter *tolerance*. Terkait fitur ini akan dibahas selanjutnya.

Untuk membuat algoritma yang serupa sebenarnya tidaklah sulit bila mengerti konsep dari algoritma flood fill. Penulis telah membuat implementasinya menggunakan bahasa Python dengan bantuan library OpenCV, sehingga memungkinkan penulis untuk dengan mudah melakukan pengubahan pada gambar melalui Python. Implementasi untuk melakukan BFS kurang lebih sebagai berikut .

```
# BFS Algorithm
visited = [[False for j in range(cols)] for i in
range(rows)]

starting_pixel = (0, 127)

visited[starting_pixel[0]][starting_pixel[1]] = True
queue = []
queue.append(starting_pixel)

directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]

while(len(queue) != 0):
        found = False
        current_pixel = queue.pop(0)
        for d in directions:
        new_pixel = tuple(map(operator.add,
current_pixel, d))
        if(is_pixel_valid(new_pixel[0], new_pixel[1])
and not visited[new_pixel[0]][new_pixel[1]] and
np.all(image[current_pixel] == image[new_pixel])):
        found = True
        queue.append(new_pixel)
        visited[new_pixel[0]][new_pixel[1]] = True

        image[current_pixel] = [0, 0, 255]


cv2.imshow('lol', image)
cv2.waitKey(0)
```

Jika program tersebut dijalankan untuk suatu gambar uji, hasilnya akan sebagai berikut:

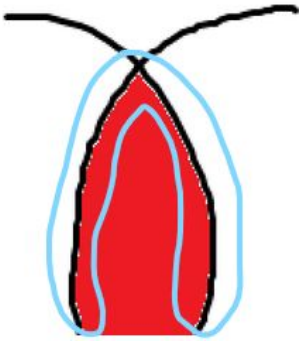


Gambar 3.2 : Ilustrasi sebelum dan sesudah gambar pola catur diterapkan *flood fill* berdasarkan warna. (Sumber: Penulis)

Cara kerja program ini adalah dengan memanfaatkan prinsip algoritma flood fill menggunakan penelusuran secara BFS. Program ini akan membaca dari pixel mana penelusuran gambar (atau pengisian warna) akan dimulai. Dari situ, akan dilakukan penelusuran ke pixel-pixel yang berada di sebelah pixel awal tersebut. Dalam kasus ini, penelusuran hanya boleh dilakukan ke pixel yang berada di tepat sebelah pixel mulainya penelusuran dan hanya boleh ke pixel dengan warna yang sama.

Namun, bila diperhatikan lagi pada contoh Paint Bucket pada program Microsoft Paint tadi, terdapat bagian yang tidak diisi secara sempurna; masih terdapat sedikit bagian yang terlihat putih. Hal ini terjadi karena dalam implementasinya, algoritma flood fill-nya hanya melakukan pengecekan untuk warna yang sama (*strictly equal*), seperti dalam program yang dibuat oleh penulis sebelumnya. Oleh karena itu, beberapa

program lain membuat sebuah parameter tambahan agar penggunaan alat paint bucket ini menjadi lebih fleksibel. Parameter tambahan tersebut disebut dengan nilai toleransi (*tolerance value*).



Gambar 3.3 : Ilustrasi pada Gambar 3.1 yang memiliki ketidaksempurnaan dalam pewarnaan daerah. (Sumber: Penulis)

Secara penelusuran graf, nilai toleransi ini memungkinkan untuk melakukan penelusuran kepada pixel yang memiliki warna yang tidak sama, tetapi masih mirip atau mendekati warna awal penelusuran dimulai. Nilai RGB dari pixel awal mulai penelusuran akan di rata-rata, sehingga nanti akan digunakan untuk menentukan apakah akan dilakukan penelusuran kepada pixel tersebut nantinya, dengan memperhitungkan seberapa jauh rata-rata nilai RGB pixel yang dituju tersebut dengan pixel awal. Jika masih dalam rentang toleransi yang ditentukan, maka penelusurannya masih bisa dilanjutkan. Rentang tersebut bisa didefinisikan sebagai berikut:

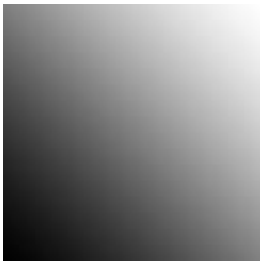
$$|avg(start) - avg(target)| \leq tolerance$$

Persamaan tersebut memiliki arti yang bisa didefinisikan sebagai berikut: avg(start) adalah rata-rata nilai RGB dari pixel awal, avg(target) adalah rata-rata nilai RGB dari pixel yang dituju, sedangkan tolerance adalah nilai toleransi dari penelusuran.

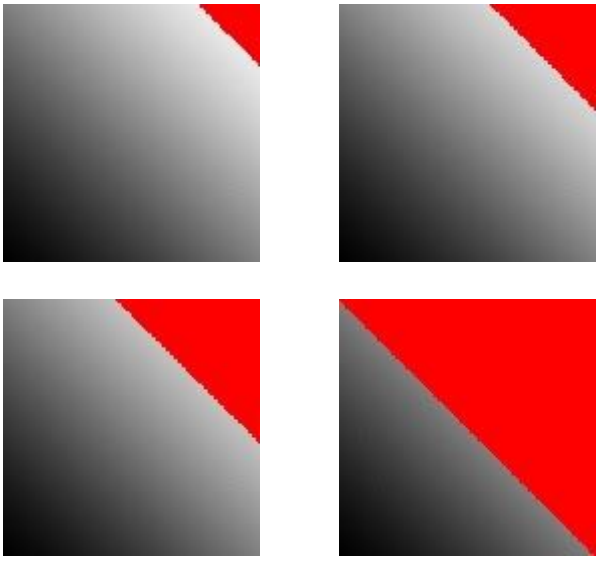
Dalam implementasinya, tidak perlu dilakukan banyak perubahan. Karena nilai toleransi ini terkait dengan informasi yang terdapat dalam suatu pixel dalam gambar tersebut, maka hanya perlu dilakukan perubahan terhadap bagian memasukkan cabang tersebut ke dalam queue. Berikut adalah sedikit dari bagian implementasi yang dimodifikasi:

```
while(len(queue) != 0):
        found = False
        current_pixel = queue.pop(0)
        for d in directions:
        new_pixel = tuple(map(operator.add,
current_pixel, d))
        if(is_pixel_valid(new_pixel[0],
new_pixel[1])):
        new_pixel_mean = np.mean(image[new_pixel])
        if(not visited[new_pixel[0]][new_pixel[1]] and
starting_pixel_mean - tolerance <= new_pixel_mean <=
starting_pixel_mean + tolerance):
                found = True
                queue.append(new_pixel)
                visited[new_pixel[0]][new_pixel[1]] =
True

        image[current_pixel] = [0, 0, 255]
```

Perbedaan yang paling terlihat adalah ada pengecekan terhadap toleransi pada bagian yang digarisbawahi. Untuk menunjukan maksud dari implementasi tersebut, penulis sudah melakukan beberapa percobaan terhadap nilai toleransi yang memengaruhi kerja dari algoritma *flood fill* yang dibuat.



Gambar 3.4 : Kondisi awal gambar yang diterapkan algoritma *floodfill* (Sumber : Penulis)



Gambar 3.5 : Contoh hasil penerapan algoritma *flood fill* berdasarkan warna dengan nilai toleransi yang berbeda-beda: 20 (kiri atas), 40 (kanan atas), 60 (kiri bawah), dan 128(kanan bawah). (Sumber : Penulis)

Dari ilustrasi tersebut, bisa terlihat bagaimana nilai toleransi dapat memengaruhi penelusuran dari algoritma *flood fill* tersebut. Dari algoritma ini, terdapat banyak hal yang dapat diterapkan lebih lanjut. Penerapan lebih lanjutnya akan dibahas pada bagian berikutnya.

IV. Magic Wand

Dalam perangkat lunak Adobe Photoshop, terdapat salah satu kakas yang bernama magic wand. Magic wand ini merupakan kakas yang berfungsi untuk melakukan seleksi pada gambar. Terdapat beberapa kakas yang serupa, seperti *Quick Selection Tool*, *Lasso Tool, Pen Tool*, dan sebagainya yang digunakan untuk membuat seleksi dalam *Photoshop*, akan tetapi *Magic Wand* ini adalah kakas yang prinsip kerjanya serupa dengan paint bucket tool tadi, yaitu dengan melakukan seleksi terhadap daerah dengan warna yang serupa [3].

Cara kerja dari *Magic Wand* tidak jauh berbeda dengan *Paint Bucket*; hanya perlu dilakukan klik pada suatu daerah yang hendak diseleksi. Setelah klik, akan muncul suatu seleksi yang ditunjukan dengan garis putus-putus. Tandanya, daerah tersebut merupakan daerah yang terseleksi. Ciri khas dari kakas *Magic Wand* ini adalah kakas ini melakukan seleksi berdasarkan warna awal yang dipilih.

Gambar 4.1 :Contoh gambar yang diseleksi menggunakan *Magic Wand* dalam Photoshop (Sumber : Penulis)

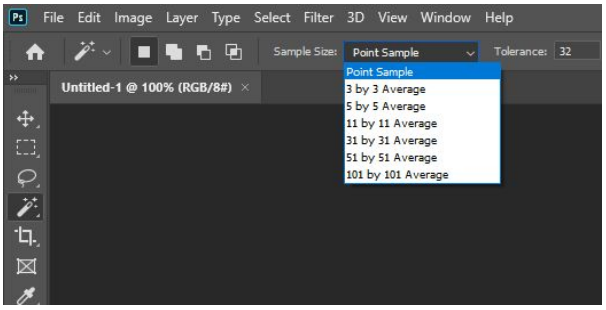
Perbedaan yang jelas berbeda dengan *Paint Bucket* adalah hasil keluaran dari kakas itu sendiri. *Magic Wand* menghasilkan sebuah seleksi, sedangkan *Paint Bucket* langsung mengisi suatu warna yang dipilih di satu daerah. Kedua kakas ini penting karena memberikan opsi bagi para pengguna aplikasi. Dengan adanya sistem seleksi ini, pengguna dapat melakukan aksi yang lebih fleksibel. Aksi -aksi yang dilakukan hanya terbatas pada daerah seleksi tersebut saja.

Secara algoritma, tentu cara kerjanya tidak berbeda jauh, hanya perlu dilakukan pengubahan keluarannya, seperti yang sudah disebutkan tadi. Mungkin hasil keluarannya dapat berupa sekumpulan pixel yang menyatakan daerah seleksinya, namun hal seperti ini akan memakan banyak memori. Sehingga, untuk mendefinisikan suatu seleksi dengan metode ini diperlukan suatu struktur data yang berbeda, contohnya seperti *quadtree*. Quadtree merupakan struktur data yang tergolong dalam jenis *tree* yang biasa digunakan untuk menyimpan representasi data dari suatu daerah.

V. Jenis-Jenis Sampling

Dalam *Adobe Photoshop*, terdapat parameter lain yang bisa diatur dalam penggunaan *Magic Wand*, yaitu pengaturan untuk mode samplingnya. Dalam beberapa kakas lain juga terdapat parameter tersebut, namun pada makalah ini Penulis akan membahas khusus untuk *Magic Wand,* karena berhubungan dengan algoritma *flood fill*.

Mode sampling yang dimaksud oleh *Photoshop* adalah pemilihan warna pada mulanya. Pada *Magic Wand* di *Photoshop* terdapat beberapa mode sampling, mulai dari *point sample* (mengambil satu pixel untuk menjadi acuan), *3 by 3 average* (mengambil 3x3 pixel dari seleksi awal, dirata-ratakan dan dijadikan acuan), bahkan hingga *101 by 101 average*. Semua penggunaannya tergantung dari kasusnya.



Gambar 5.1 :Tampilan pemilihan ukuran sampel dalam Photoshop (Sumber : Penulis)

Secara implementasinya, dalam algoritma *flood fill*-nya mengenai sampling ini akan mengubah cara kerja algoritmanya dari titik awalnya. Untuk posisi awalnya tidak ada perubahan, namun yang perlu diubah adalah warna acuannya. Warna acuannya harus dilakukan perhitungan terlebih dahulu sebelum melakukan algoritma *flood fill*. Untuk lebih jelasnya bisa diperhatikan implementasi di bawah ini.

Pada mulanya, hanya dilakukan rata-rata nilai RGB dari titik awal mulai penelusuran dengan potongan kodenya sebagai berikut:

```
starting_pixel = (0, 127)
starting_pixel_mean = np.mean(image[starting_pixel])
```

Untuk melakukan sampling untuk suatu area tertentu, perlu dilakukan pengulangan untuk menelusuri pixel di sekeliling titik sampel dan merata-ratakan hasil rata-rata tersebut. Untuk memudahkan implementasi, dalam potongan kode ini Penulis menggunakan bantuan NumPy. Implementasinya adalah seperti berikut:

```
starting_pixel = (0, 127)
sample_average_list = np.asarray([])
for i in range(-sampling_radius, sampling_radius+1):
        for j in range(-sampling_radius,
sampling_radius+1):
        r, c = starting_pixel
        if(is_pixel_valid(r + i, c + j)):
        sample_average_list = np.append(
```

```
                sample_average_list, np.mean(image[(r
+ i, c + j)])
        )

starting_pixel_mean = np.mean(sample_average_list)
```

Kunci dari sampling ini ada pada bagian kode yang digarisbawahi. Potongan kode tersebut bertujuan untuk mengambil rata-rata dari pixel di sekeliling pixel awal *sampling* dan menyimpannya ke dalam sebuah array. Kemudian, array itu akan dirata-ratakan lagi untuk mendapat rata-rata dari nilai RGB-nya. Selain itu, berbeda dengan Photoshop, Penulis memilih untuk mengimplementasikan ukuran sampel dengan radius pengambilan sampel untuk fleksibilitas dari program.

Sampling ini berguna ketika range warna yang hendak didapatkan tidak serupa. Parameter sampling ini bila dikombinasikan dengan menggunakan parameter toleransi, akan menghasilkan suatu seleksi yang baik.

Link Video YouTube

Untuk menjadi pendukung makalah yang dibuat, Penulis juga telah menyiapkan sebuah video yang telah dipost di YouTube. Berikut ini adalah link untuk video yang disebutkan: https://youtu.be/xVTS6NxGv_4

Link Repository Github

Selain dari itu, program yang dibuat penulis untuk mendukung pembuatan makalah ini tersedia untuk publik. Program ini dibuat menggunakan bahasa Python dengan bantuan Library OpenCV untuk mengakses warna-warna pada pixel di suatu gambar. Repository Github tersebut dapat dicari pada akun Github Penulis : https://github.com/mkamadeus/

Ucapan Terima Kasih

Pertama, Penulis hendak mengucapkan terima kasih dan syukur saya kepada Tuhan YME atas rahmat dan berkat-Nya yang diberikan dalam bentuk ilmu pengetahuan; tanpa berkat-Nya makalah ini tidak dapat selesai tepat pada waktunya. Selain itu, Penulis ingin mengucapkan terima kasih kepada dosen-dosen pengampu mata kuliah IF2211 Strategi Algoritma, Dr. Masayu Leylia Khodra, ST., MT., Dr. Nur Ulfa Maulidevi, ST., M.Sc., dan Dr. Ir. Rinaldi Munir, MT. atas ilmu pengetahuan dan tugas-tugas yang menarik selama mengambil mata kuliah ini. Saya berharap dari hasil dari kuliah ini dapat berguna bagi orang lain.

References

[1] (n.d.). Flood Fill Algorithm - Techie Delight. Retrieved May 1, 2020, from https://www.techiedelight.com/flood-fill-algorithm/

[2] "Photoshop Paint Bucket Tool - Media College." https://www.mediacollege.com/adobe/photoshop/tool/paint-bucket.html. Accessed 1 May. 2020.

[3] "The Magic Wand Tool - Photoshop Selections." https://www.photoshopessentials.com/basics/selections/magic-wand-tool/. Accessed 1 May. 2020.

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 Mei 2020

Matthew Kevin Amadeus
13518035