IMPLEMENTASI SEGMENTASI CITRA MENGGUNAKAN METODE GRAPH YANG EFISIEN

SKRIPSI

MEILINDA SIAHAAN
050803009

DEPARTEMEN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS SUMATERA UTARA
MEDAN
2009

IMPLEMENTASI SEGMENTASI CITRA MENGGUNAKAN METODE GRAPH YANG EFISIEN

SKRIPSI

Diajukan untuk melengkapi tugas dan memenuhi syarat mencapai gelar Sarjana Sains.

MEILINDA SIAHAAN
050803009

DEPARTEMEN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS SUMATERA UTARA
MEDAN
2009

PERSETUJUAN

Judul	:	IMPLEMENTASI SEGMENTASI CITRA MENGGUNAKAN METODE GRAPH YANG EFISIEN
Kategori	:	SKRIPSI
Nama	:	MEILINDA SIAHAAN
Nomor Induk Mahasiswa	:	050803009
Program Studi	:	SARJANA (S1) MATEMATIKA
Departemen	:	MATEMATIKA
Fakultas	:	MATEMATIKA DAN ILMU PENGETAHUAN ALAM (FMIPA) UNIVERSITAS SUMATERA UTARA

Diluluskan di
Medan, Desember 2009

Komisi Pembimbing :

Pembimbing 2	Pembimbing 1
Syahril Efendi, S.Si, M.IT	Drs. James P. Marbun, M.Kom
NIP 196711101996021001	NIP 195806111986031002

Diketahui/Disetujui oleh
Departemen Matematika FMIPA USU
Ketua,

Dr. Saib Suwilo, M.Sc.
NIP 196401091988031004

PERNYATAAN

IMPLEMENTASI SEGMENTASI CITRA MENGGUNAKAN METODE GRAPH YANG EFISIEN

SKRIPSI

Saya mengakui bahwa skripsi ini adalah hasil kerja saya sendiri, kecuali beberapa kutipan dan ringkasan yang masing-masing disebutkan sumbernya.

Medan, Desember 2009

MEILINDA SIAHAAN
050803009

PENGHARGAAN

Puji dan syukur kehadirat Tuhan Yang Maha Esa atas anugerah, rahmat serta bimbingan-Nya kepada penulis, sehingga penulis dapat menyelesaikan skripsi ini.

Penulis menyampaikan terima kasih yang tulus kepada orangtua tercinta, serta seluruh keluarga yang telah memberi dukungan baik moril maupun materil selama ini.

Penulis juga menyampaikan terima kasih kepada:

1. Bapak Drs. James P. Marbun, M.Kom dan Bapak Syahril Efendi, S.Si, M.IT selaku dosen pembimbing dalam menyelesaikan skripsi ini, yang telah banyak memberikan bimbingan dengan penuh kesabaran untuk menyempurnakan skripsi ini sehingga penulis dapat menyelesaikan skripsi ini dengan baik.
2. Bapak DR. Saib Suwilo, M.Sc dan Bapak Drs. Henry Rani Sitepu, M.Si, selaku Ketua dan Sekretaris Departemen Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Sumatera Utara.
3. Bapak Prof. DR. Herman Mawengkang dan Bapak Drs. Marihat Situmorang, M.Kom, selaku dosen penguji.
4. Semua dosen dan staf administrasi di Departemen Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Sumatera Utara.
5. Semua teman *Golden Generation* (stambuk 2005) Departemen Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Sumatera Utara atas dukungan serta doanya, terutama kepada yang terkasih: Johannes Sinaga yang telah memberikan perhatian, dukungan, dan motivasi kepada penulis dan juga ucapan trima kasih kepada sahabat-sahabat terbaik, Christine R. Pangaribuan, Ester C. Situmeang, Imelda Y.Y.F., Juli Loisiana Butarbutar, Ruth Endaria Ginting, Trisnawati Sitompul, Alice dan Josephine serta kepada Reynold Lumban Tobing yang telah banyak memberi masukan kepada penulis.

Penulis menyadari masih banyak kekurangan dalam penulisan ini, untuk itu penulis meminta saran dan kritik yang membangun dari pembaca sekalian.

Akhir kata penulis berharap semoga skripsi ini dapat bermanfaat bagi kita semua dan juga semoga Tuhan Yang Maha Esa membalas semua jasa bapak-bapak dan teman-teman sekalian.

Medan, Desember 2009

Penulis,

Meilinda Siahaan

ABSTRAK

Saat ini, pengolahan citra dapat lebih mudah dilakukan dengan menggunakan pengolahan citra digital. Salah satu operasi dari pengolahan citra digital adalah proses segmentasi citra. Segmentasi citra merupakan proses memisahkan objek-objek dari citra ke dalam bagian-bagian, sehingga informasi yang ada pada citra dapat dengan mudah dibaca. Graph yang efisien adalah salah satu metode dalam segmentasi citra. Metode graph yang efisien ini mendefinisikan predikat dari sebuah batas antara dua region yang bersinggungan pada sebuah citra dengan menggunakan representasi berbasis graph. Kemudian mengembangkan algoritma segmentasi berbasis graph dengan menggunakan dua model ketetanggaan yang berbeda. *Preprocessing* sebaiknya dilakukan sebelum melakukan proses segmentasi citra. Ada banyak macam *preprocessing* yang dapat dilakukan dan objek yang dihasilkan oleh metode graph yang efisien untuk sebuah gambar berbeda-beda berdasarkan *preprocessing* yang dilakukan.

IMPLEMENTATION OF IMAGE SEGMENTATION USING EFFICIENT GRAPH METHOD

ABSTRACT

Nowadays, image processing could be easier to do by using digital image processing. One of that is segmentation image processing. Segmentation is a process to separate objects of image into regions, so that the information of image could be easy to read. Efficient graph is one of method in segmentation. Efficient graph method deffine a predicate for measuring the evidence for a boundary between two regions using a graph based representation. Then apply the algorithm using two different kinds of local neighborhoods in constructing the graph. Preprocessing as well to do before do segmentation image segmentation. There are many kinds of preprocessing which can be done and the object resulted by efficient graph method for an image are different each other based on the preprocessing.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

Halaman

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Pada dasarnya ada tiga bidang yang menangani pengolahan data berbentuk citra, yaitu: grafika komputer, pengolahan citra, dan visi komputer. Pada bidang grafika komputer banyak dilakukan proses yang bersifat sintesis yang mempunyai ciri data masukan berbentuk deskriptif dengan keluaran hasil proses yang berbentuk citra. Sedangkan proses di dalam bidang visi komputer merupakan kebalikan dari proses grafika komputer. Terakhir, bidang pengolahan citra merupakan proses pengolahan dan analisis citra yang banyak melibatkan persepsi visual, yakni data masukan maupun data keluarannya berbentuk citra.

Pada pengolahan citra terdapat enam jenis operasi pengolahan, yaitu peningkatan kualitas citra, restorasi citra, kompresi citra, segmentasi citra, analisis citra, dan rekonstruksi citra. Pada umumnya informasi yang ada dalam suatu citra terletak pada strukturnya. Agar mudah memahami suatu citra dapat dilakukan dengan menyederhanakan struktur citra tersebut. Salah satu metode untuk menyederhanakan struktur citra adalah dengan melakukan proses segmentasi citra (*image segmentation*).

Segmentasi citra (*image segmentation*) adalah suatu tahap pada proses analisis citra yang bertujuan untuk memperoleh informasi yang ada dalam citra tersebut dengan membagi citra ke dalam daerah-daerah terpisah dimana setiap daerah adalah homogen dan mengacu pada sebuah kriteria keseragaman yang jelas. Proses segmentasi citra merupakan proses dasar dan penting di dalam komputer visi. Segmentasi yang dilakukan pada citra harus tepat agar informasi yang terkandung di dalamnya dapat diterjemahkan dengan baik. Terdapat banyak metode dalam melakukan segmentasi pada citra. Salah satu metode yang dapat digunakan dalam proses segmentasi citra adalah metode graph yang efisien.

Metode graph yang efisien ini mendefinisikan predikat dari sebuah batas antara dua *region* yang bersinggungan pada sebuah citra dengan menggunakan representasi berbasis graph. Kemudian mengembangkan algoritma segmentasi berbasis graph dengan menggunakan dua model ketetanggaan yang berbeda.

Yang menjadi bagian penting dalam penelitian ini adalah bagaimana metode graph yang efisen ini dapat melakukan proses segmentasi citra sehingga informasi yang terkandung di dalamnya dapat diterjemahkan dengan baik.

1.2 Perumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan sebelumnya didapat rumusan masalah, yaitu bagaimana menerapkan metode graph yang efisien dalam mengenali *detail* dari suatu citra.

1.3 Batasan Masalah

Agar pembahasan penelitian ini lebih terarah, maka masalah yang dibahas dibatasi pada proses pengolahan citra dalam segmentasi citra dengan menggunakan metode graph yang efisien.

1.4 Tujuan Penelitian

Tujuan pembuatan tugas akhir ini adalah untuk menerapkan metode graph yang efisien untuk mengenal pola suatu citra menggunakan Matlab.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah:

1. Dapat digunakan untuk menerjemahkan informasi yang terdapat pada suatu citra digital.
2. Dapat dimanfaatkan untuk proses tingkat tinggi lebih lanjut yang dilakukan terhadap citra seperti proses klasifikasi citra dan identifikasi objek.

1.6 Metodologi Penelitian

Metodologi penelitian yang digunakan pada studi ini adalah:

1) Studi Literatur dan Pemahaman

   Penulisan ini dimulai dengan studi kepustakaan, yaitu mengumpulkan bahan-bahan referensi yang membahas tentang pengolahan citra, segmentasi citra, algoritma berbasis graph, analisis dan perancangan sistem segmentasi citra dengan metode graph.

2) Analisis

   Pada tahap ini dilakukan pengumpulan fakta-fakta yang mendukung perancangan sistem pengenalan pola dengan mengadakan konsultasi dengan dosen pembimbing maupun dosen yang berkemampuan dalam bidang ini.

3) Perancangan dan Implementasi

   Perancangan dan implementasi segmentasi citra didasarkan pada metode graph yang efisien menggunakan alat bantu aplikasi Matlab 7.0.1.

4) Pengujian

   Pada tahap ini sistem yang sudah dirancang diuji oleh perancang dan membandingkan solusi pada sistem dengan pemikiran dosen pembimbing maupun dosen yang berkemampuan dalam bidang ini.

5) Penyusunan laporan dan kesimpulan akhir

Pada tahap ini dilakukan penyusun laporan hasil analisis ke dalam format penulisan tugas akhir dengan disertai kesimpulan akhir.

1.7 Tinjauan Pustaka

Aniati Murni (1992) menyatakan secara umum tujuan dari proses klasifikasi citra adalah untuk mendapatkan gambar atau peta tematik. Proses segmentasi mempunyai tujuan yang hampir serupa dengan proses klasifikasi tidak terawasi. Proses segmentasi citra adalah membagi suatu citra menjadi wilayah-wilayah yang homogen berdasarkan kriteria keserupaan yang tertentu antara tingkat keabuan suatu piksel dengan tingkat keabuan piksel-piksel tetangganya. Proses segmentasi citra ini lebih ke proses prapengolahan pada sistem pengenalan objek dalam citra.

Marvin Ch. Wijaya (2007) menyatakan dalam pengenalan maupun pengolahan citra, masalah persepsi visual mempunyai peranan penting. Penentuan apa yang dapat dilihat tidak dapat hanya ditentukan oleh manusia itu sendiri. Mata merupakan bagian dari sistem visual manusia. Sistem visual itu sangat sulit dipelajari, terlebih jika ingin menyingkap proses yang melatarbelakangi timbulnya suatu persepsi, seperti pada peristiwa pengenalan (*recoqnition*). Sehingga untuk menghindari persepsi yang salah, dilakukanlah pengolahan citra digital dengan menggunakan komputer, baik berupa mikrokomputer sederhana maupun komputer besar tergantung jumlah data dan jenis pengolahannya. Matlab merupakan salah satu alat (*tools*) pemrograman untuk membantu bidang pendidikan dan penelitian khususnya untuk pengolahan citra pada segmentasi citra.

R. Sigit (2005) membahas tentang langkah-langkah pemrograman dalam proses pengolahan citra. Disini dipelajari lebih kepada pengolahan program dalam proses segmentasi dan deteksi tepi.

Suparman (2007) menjelaskan tentang proses komputer vision terdiri atas empat langkah dasar, yaitu akusisi citra, pengolahan citra, analisis citra, dan pemahaman citra. Proses akusisi citra yaitu menterjemahkan transformasi visual ke

dalam sebuah format yang bisa dimanipulasi lebih lanjut oleh otak. Kemudian proses pengolahan citra membantu memperbaiki kualitas citra agar komputer dapat menganalisa dan memahami lebih efisien. Selanjutnya proses analisis citra bertujuan untuk mengidentifikasi ciri-ciri khas dan karakteristik dari citra tersebut serta mencari sisi dan batas-batasnya. Langkah yang terakhir yaitu pemahaman citra yakni mengidentifikasi objek-objek spesifik dan hubungannya.

Rinaldi Munir (2003) menjelaskan bahwa graph merupakan kumpulan simpul (*nodes*) yang dihubungkan satu sama lain melalui sisi/busur (*edges*). Suatu graph G terdiri dari dua himpunan yaitu himpunan simpul (V) dan himpunan sisi/busur (E). Simpul-simpul pada graph dapat merupakan obyek sembarang seperti kota, atom-atom suatu zat, nama anak, jenis buah, komponen alat elektronik dan sebagainya. Sedangkan sisi/busur dapat menunjukkan hubungan (relasi) sembarang seperti rute penerbangan, jalan raya, sambungan telepon, ikatan kimia, dan lain-lain.

BAB 2

LANDASAN TEORI

2.1 Pengertian Citra Digital

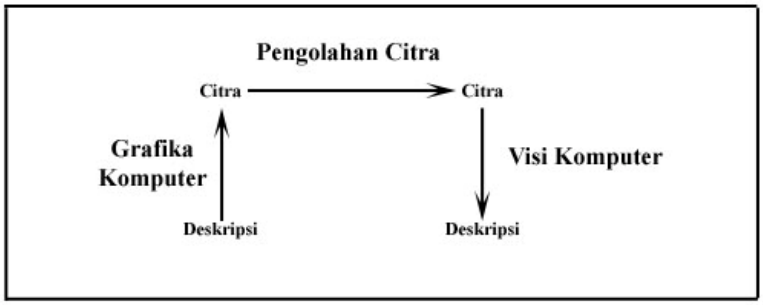
Citra adalah gambar dua dimensi yang dihasilkan dari gambar analog dua dimensi yang kontinu menjadi gambar diskrit melalui proses digitalisasi. Citra yang terlihat merupakan cahaya yang direfleksikan dari sebuah objek. Sumber cahaya menerangi objek lalu objek memantulkan kembali sebagian dari berkas cahaya tersebut dan pantulan cahaya ditangkap oleh alat-alat optik, misal mata manusia, kamera, *scanner*, sensor satelit, dan sebagainya, kemudian direkam. Citra sebagai keluaran suatu sistem perekaman data dapat bersifat optik berupa foto, bersifat analog berupa sinyal-sinyal video seperti gambar pada monitor televisi, atau bersifat digital yang dapat langsung disimpan pada suatu media penyimpanan magnetik. Citra juga dapat dikelompokkan menjadi dua yaitu: citra tampak seperti foto/gambar, lukisan, dan apa saja yang tampak di layar monitor/televisi, hologram, dan sebagainya serta citra tidak tampak seperti data foto/gambar dalam file, dan citra yang direpresentasikan dalam fungsi matematis. (Gonzalez, 1987). Ada beberapa macam tipe suatu citra didasarkan dari format penyimpanan warnanya, yaitu:

1. Citra biner adalah citra yang setiap pikselnya hanya bernilai 0 (warna hitam) dan 1 (warna putih).
2. Citra skala keabuan adalah citra yang setiap pikselnya mempunyai kemungkinan warna antara hitam (minimal) dan putih (maksimal).
3. Citra warna (*true color*) adalah citra yang setiap pikselnya memiliki warna yang merupakan kombinasi dari tiga warna dasar, yaitu merah, kuning dan hijau (RGB).
4. Citra warna berindeks adalah citra yang setiap pikselnya mewakili indeks dari suatu tabel warna yang tersedia (biasanya disebut palet warna).

2.2 Pengertian Pengolahan Citra

Meskipun sebuah citra kaya informasi, namun seringkali citra yang dimiliki mengalami penurunan mutu (degradasi), misalnya mengandung cacat atau derau (*noise*), warnanya terlalu kontras, kurang tajam, kabur (*blurring*), dan sebagainya. Tentu saja citra semacam ini menjadi lebih sulit diinterpretasi karena informasi yang disampaikan oleh citra tersebut menjadi berkurang. Agar citra yang mengalami gangguan mudah diinterpretasi (baik oleh manusia maupun mesin), maka citra tersebut perlu dimanipulasi menjadi citra lain yang kualitasnya lebih baik. Bidang yang menyangkut hal ini adalah pengolahan citra (*image processing*).

Pada dasarnya ada tiga bidang yang menangani pengolahan data berbentuk citra, yaitu: grafika komputer, pengolahan citra, dan visi komputer. Pada bidang grafika komputer banyak dilakukan proses yang bersifat sintesis yang mempunyai ciri data masukan berbentuk deskriptif dengan keluaran hasil proses yang berbentuk citra. Sedangkan proses di dalam bidang visi komputer merupakan kebalikan dari proses grafika komputer. Terakhir, bidang pengolahan citra merupakan proses pengolahan dan analisis citra dengan data masukan maupun data keluarannya berbentuk citra. Pengolahan citra merupakan proses pengolahan dan analisis citra yang banyak melibatkan persepsi visual. Pengolahan Citra bertujuan memperbaiki kualitas citra agar mudah diinterpretasi oleh manusia atau mesin (dalam hal ini komputer). Teknik-teknik pengolahan citra mentransformasikan citra menjadi citra lain. Jadi, masukannya adalah citra dan keluarannya juga citra, namun citra keluaran mempunyai kualitas lebih baik daripada citra masukan. (Murni, 1992). Hubungan antara ketiga bidang tersebut ditunjukkan pada Gambar 2.1.



Gambar 2.1 Tiga bidang yang berkaitan dengan citra

2.3 Operasi Pengolahan Citra

Operasi-operasi yang dilakukan dalam pengolahan citra banyak ragamnya. Namun, secara umum, pada pengolahan citra terdapat enam jenis operasi pengolahan, yaitu:

1. Peningkatan kualitas citra (*image enhancement*)

   Jenis operasi ini bertujuan untuk memperbaiki kualitas citra dengan cara memanipulasi parameter-parameter citra. Dengan operasi ini, ciri-ciri khusus yang terdapat di dalam citra lebih ditonjolkan. Contoh-contoh operasi peningkatan kualitas citra:

   a. Perbaikan kontras gelap/terang
   b. Perbaikan tepian objek (*edge enhancement*)
   c. Penajaman (*sharpening*)
   d. Pemberian warna semu (*pseudocoloring*)
   e. Penapisan derau (*noise filtering*)

2. Restorasi citra (*image restoration*)

   Operasi ini bertujuan menghilangkan/meminimumkan cacat pada citra. Tujuan restorasi citra hampir sama dengan operasi peningkatan kualitas citra. Bedanya, pada restorasi citra penyebab degradasi gambar diketahui. Contoh-contoh operasi restorasi citra:

   1. penghilangan kesamaran (*deblurring*).
   2. penghilangan derau (*noise*)

3. Kompresi citra (*image compression*)

   Jenis operasi ini dilakukan agar citra dapat direpresentasikan dalam bentuk yang lebih kompak sehingga memerlukan memori yang lebih sedikit. Hal penting yang harus diperhatikan dalam kompresi citra adalah citra yang telah dikompresikan harus tetap mempunyai kualitas gambar yang bagus. Contoh metode kompresi citra adalah metode JPEG. Misalkan ada citra kapal yang berukuran 258 KB. Hasil kompresi citra dengan metode JPEG dapat mereduksi ukuran citra semula sehingga menjadi 49 KB saja.

4. Segmentasi citra (*image segmentation*)

   Operasi ini adalah suatu tahap pada proses analisis citra yang bertujuan untuk memperoleh informasi yang ada dalam citra tersebut dengan membagi citra ke dalam daerah-daerah terpisah dimana setiap daerah adalah homogen dan mengacu pada sebuah kriteria keseragaman yang jelas. Segmentasi yang dilakukan pada citra harus tepat agar informasi yang terkandung di dalamnya dapat diterjemahkan dengan baik.

5. Analisis citra (*image analysis*)

   Jenis operasi ini bertujuan menghitung besaran kuantitif dari citra untuk menghasilkan deskripsinya. Teknik analisis citra mengekstraksi ciri-ciri tertentu yang membantu dalam identifikasi objek. Proses segmentasi kadangkala diperlukan untuk melokalisasi objek yang diinginkan dari sekelilingnya. Contoh-contoh operasi analisis citra:

   1. Pendeteksian tepi objek (*edge detection*)
   2. Ekstraksi batas (*boundary*)
   3. Representasi daerah (*region*)

6. Rekonstruksi citra (*image reconstruction*)

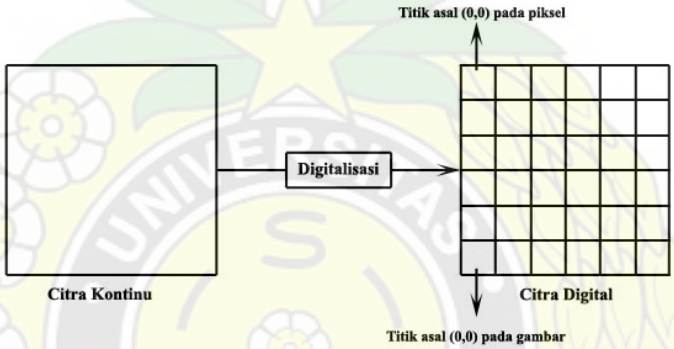
   Jenis operasi ini bertujuan untuk membentuk ulang objek dari beberapa citra hasil proyeksi. Operasi rekonstruksi citra banyak digunakan dalam bidang medis. Misalnya beberapa foto *rontgen* dengan sinar *X* digunakan untuk membentuk ulang gambar organ tubuh.

2.4 Proses Digitalisasi Citra

Agar suatu citra dapat diolah dengan komputer digital, maka citra harus direpresentasikan secara numerik dengan nilai-nilai diskrit melalui proses digitalisasi citra. Digitalisasi citra merupakan representasi citra dari fungsi kontinu menjadi nilai-nilai diskrit. Dimana citra kontinu dihasilkan dari sumber cahaya yang menyinari objek, kemudian objek memantulkan kembali sebagian dari berkas cahaya tersebut

dan pantulan cahaya ini ditangkap oleh sensor visual pada sistem optik, misalnya mata manusia dan kamera analog. Alat yang biasanya digunakan untuk mengkonversi citra kontinu menjadi citra digital adalah *scanner*. Untuk mengubah citra yang bersifat kontinu menjadi citra digital diperlukan proses pembuatan kisi-kisi arah horizontal dan vertikal, sehingga diperoleh gambar dalam bentuk *array* dua dimensi. Adapun proses digitalisasi citra terdiri atas dua tahap, yaitu:

1. Digitalisasi spasial (x, y), sering disebut sebagai *sampling*.
2. Digitalisasi intensitas $f(x, y)$, sering disebut sebagai kuantisasi.



Gambar 2.2 Proses digitalisasi citra

Ada perbedaan antara koordinat gambar yang di *scanner* dengan koordinat matriks (hasil digitalisasi). Titik asal (0,0) pada gambar dan elemen (0,0) pada matriks tidaklah sama. Koordinat *x* dan *y* pada gambar dimulai dari sudut kiri bawah, sedangkan penomoran piksel pada matriks dimulai dari sudut kiri atas.

2.4.1 Digitalisasi Spasial

Dari proses digitalisasi spasial diperoleh citra digital berbentuk empat persegi panjang,dan dimensi ukurannya dinyatakan dengan matriks *N* baris dan *M* kolom $(N \times M)$, ukuran tersebut sering disebut resolusi. Semakin tinggi resolusinya, maka semakin kecil ukuran pikselnya atau semakin banyak jumlah piksel, maka semakin halus gambar yang diperoleh karena informasi yang hilang dari pengelompokan skala keabuan pada spasial semakin kecil.

2.4.2 Digitalisasi Intensitas

Proses digitalisasi intensitas yaitu membagi skala keabuan $(0, L)$ menjadi *G* buah level yang dinyatakan dengan suatu harga bilangan bulat (*integer*), dimana *G* diambil dari perpangkatan dua dan nilai skala keabuan tersebut biasa disebut dengan piksel (*picture element*). Sehingga persamaan digitalisasi intensitas ditulis sebagai berikut:

$$G = 2^m$$

Nilai maksimumnya (dalam indeks): $G = 2^m - 1$

dimana:

G = derajat keabuan dan

m = bilangan bulat positif

Dari skala keabuan $(0, L)$, nilai intensitas 0 menyatakan hitam, nilai intensitas *L* menyatakan putih, dan nilai intensitas antara 0 sampai *L* bergeser dari hitam sampai putih. Hitam dinyatakan dengan nilai skala keabuan terendah, sedangkan putih dinyatakan dengan nilai skala keabuan tertinggi. Jumlah bit yang dibutuhkan untuk merepresentasikan nilai skala keabuan piksel disebut kedalaman piksel (*pixel depth*). Citra sering diasosiasikan dengan kedalaman pikselnya. Jadi, citra dengan kedalaman 8 bit disebut juga citra 256 warna skala keabuan. Besarnya daerah skala keabuan yang digunakan menentukan resolusi kecerahan dari gambar yang diperoleh. Semakin banyak jumlah skala keabuan, berarti jumlah bit intensitasnya makin makin banyak dan gambar yang diperoleh semakin bagus karena skala keabuannya akan semakin tinggi sehingga mendekati citra asli.

Tabel 2.1 Nilai skala keabuan dan kedalaman pikselnya

Skala Keabuan	Rentang Nilai Keabuan	*Pixel Depth*
2^1	0 , 1	1 bit
2^2	0 sampai 7	2 bit
2^3	0 sampai 15	3 bit

2^8	0 sampai 255	8 bit

2.5 Graph yang Didasarkan Segmentasi Citra

2.5.1 Dasar-Dasar Graph

Graph adalah salah satu pokok bahasan matematika diskrit yang telah lama dikenal dan banyak diaplikasikan pada berbagai bidang. Secara umum, graph *G* didefinisikan sebagai pasangan himpunan (V, E), ditulis dengan notasi $G = (V, E)$ yang dalam hal ini V adalah himpunan tidak kosong dari simpul-simpul (*nodes*) dan E adalah himpunan sisi/busur (*edges*) yang menghubungkan sepasang simpul. (Munir, 2003).

Kegunaan graph sangat banyak. Umumnya graph digunakan untuk memodelkan suatu masalah sehingga menjadi lebih mudah, yaitu dengan cara merepresentasikan objek-objek tersebut. Contoh pemodelan suatu masalah dengan menggunakan graph dapat dilihat pada penggambaran rangkaian listrik, senyawa kimia, jaringan komunikasi, jaringan *network* komputer, analisis algoritma, peta, struktur hierarki sosial, dan lain-lain.

Graph dapat dikelompokkan menjadi beberapa jenis tergantung dari sudut pandang pengelompokannya. Pengelompokan graph dapat dipandang berdasarkan dari ada tidaknya sisi ganda, berdasarkan jumlah simpul, atau berdasarkan orientasi arah pada sisi. Berdasarkan ada tidaknya *loop* atau sisi ganda pada suatu graph, maka secara umum graph dapat digolongkan menjadi dua jenis, yaitu:

1. Graph sederhana (*simple graph*) adalah graph yang tidak mengandung *loop* maupun sisi ganda. Contohnya graph yang mempresentasikan jaringan komputer. Simpul menyatakan komputer, sedangkan sisi menyatakan saluran telepon untuk berkomunikasi. Saluran telepon dapat berkomunikasi dua arah.
2. Graph tak sederhana (*unsimple graph*) adalah graph yang mengandung sisi ganda ataupun *loop*. Ada dua macam graph tak sederhana yaitu graph ganda (*multigraph*) yaitu graph yang mengandung sisi ganda dan graph semu (*pseudograph*) yaitu graph yang mengandung *loop*.

Sisi pada graph dapat mempunyai orientasi arah. Berdasarkan orientasi arah pada sisi, maka secara umum graph dibedakan atas dua jenis, yaitu:

1. Graph tak berarah (*undirected graph*) adalah graph yang sisinya tidak mengandung orientasi arah.
2. Graph berarah (*directed graph* atau *digraph*) adalah graph yang setiap sisinya mempunyai orientasi arah.

Ada beberapa sifat yang berkaitan dengan graph. Berikut adalah sifat-sifat yang sering digunakan, yaitu:

1. Bertetangga (*adjacent*)

   Dua buah simpul pada graph tak berarah *G* dikatakan bertetangga bila keduanya terhubung langsung dengan sebuah sisi.

2. Bersisian (*incident*)

   Untuk sembarang sisi $e = (u, v)$, sisi *e* dikatakan bersisian dengan simpul *u* dan simpul *v*.

3. Derajat (*degree*)

   Derajat suatu simpul pada graph tak berarah adalah jumlah sisi yang bersisian dengan simpul tersebut.

4. Lintasan (*path*)

   Lintasan yang panjangnya *n* dari simpul awal v_0 ke simpul tujuan v_n di dalam graph *G* ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk $v_0, e_1, v_1, e_2, v_2, \ldots, v_{n-1}, e_n, v_n$ sedemikian sehingga $e_1 = (v_0, v_1), e_2 = (v_1, v_2), \ldots, e_n = (v_{n-1}, v_n)$

5. Sirkuit (*circuit*) atau siklus (*cycle*)

   Lintasan yang berawal dan berakhir pada simpul yang sama disebut sirkuit atau siklus.

6. Terhubung (*connected*)

   Graph tak berarah *G* disebut graph terhubung (*connected graph*) jika untuk setiap pasang simpul *u* dan *v* di dalam himpunan *V* terdapat lintasan dari *u* ke *v* (yang berarti ada lintasan dari *u* ke *v*).

7. Graph berbobot (*weighted graph*)

   Graph berbobot adalah graph yang setiap sisinya diberikan suatu bobot atau nilai.

Graph terhubung yang tidak mengandung sirkuit disebut pohon (*tree*). Diantara sekian banyak konsep dalam graph, konsep pohon (*tree*) merupakan konsep yang paling penting, karena terapannya yang luas dalam berbagai bidang ilmu, baik dalam bidang komputer maupun di luar bidang komputer.

Yang dimaksud dengan pohon merentang (*spanning tree*) yaitu misalkan $G = (V, E)$ merupakan graph tak berarah terhubung yang bukan pohon, yang berarti di *G* terdapat beberapa sirkuit. G dapat diubah menjadi pohon $T = (V_1, E_1)$ dengan cara memutuskan sirkuit-sirkuit yang ada, yakni memilih sebuah sirkuit, kemudian menghapus sebuah sisi dari sirkuit tersebut.

Jika *G* adalah graph berbobot, maka bobot pohon merentang *T* dari *G* didefinisikan sebagai jumlah bobot semua sisi di *T*. Pohon merentang yang berbeda mempunyai bobot yang berbeda pula. Diantara semua pohon merentang di *G*, pohon merentang yang berbobot minimum dinamakan *Minimum spanning tree* (MST). MST memiliki terapan yang cukup luas dalam prakteknya. Misalkan pemerintah akan membangun jalur rel kereta api yang menghubungkan sejumlah kota. Membangun jalur rel kereta api memerlukan banyak biaya. Karena itu pembangunan jalur itu tidak perlu menghubungkan langsung dua buah kota, tetapi cukup membangun jalur kereta seperti pohon merentang. Karena di dalam sebuah graph mungkin saja terdapat lebih

dari satu pohon merentang, maka harus dicari pohon merentang dengan jumlah jarak terpendek, dengan kata lain harus dicari *Minimum spanning tree*-nya.

2.5.2 Representasi Graph dalam Proses Segmentasi Citra

Teknik graph yang didasarkan segmentasi pada umumnya merepresentasikan masalah graph $G = (V, E)$ dimana setiap simpul $v_i \in V$, $i = 1,2,3,\ldots,n$ dianggap piksel-piksel dari citra dan sisi E merupakan pasangan-pasangan dari piksel-piksel yang bertetanggaan. Setiap sisi yang menghubungkan piksel-piksel pada citra memiliki bobot (*weight*) yaitu nilai intensitas keabuan pada citra itu sendiri. Edge yang menghubungkan piksel (simpul) p dan q pada graph memiliki bobot (*weighted graph*) yaitu ditunjukkan pada rumus berikut:

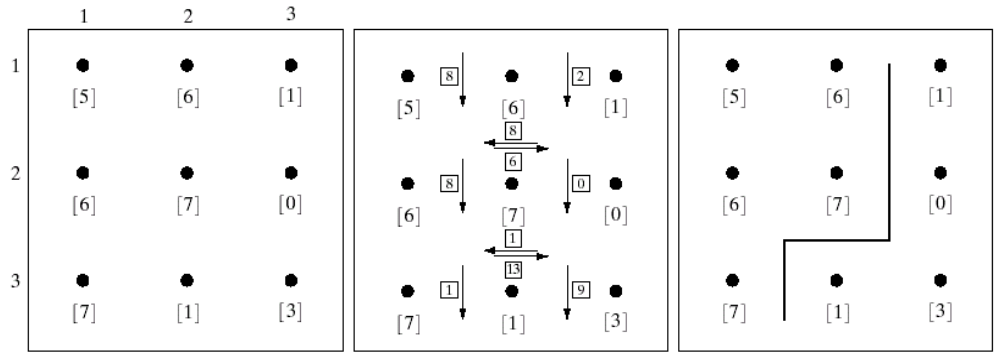
$$w((p,q)) = H - [f(p) - f(q)]$$

dimana:

H : nilai intensitas keabuan tertinggi pada citra

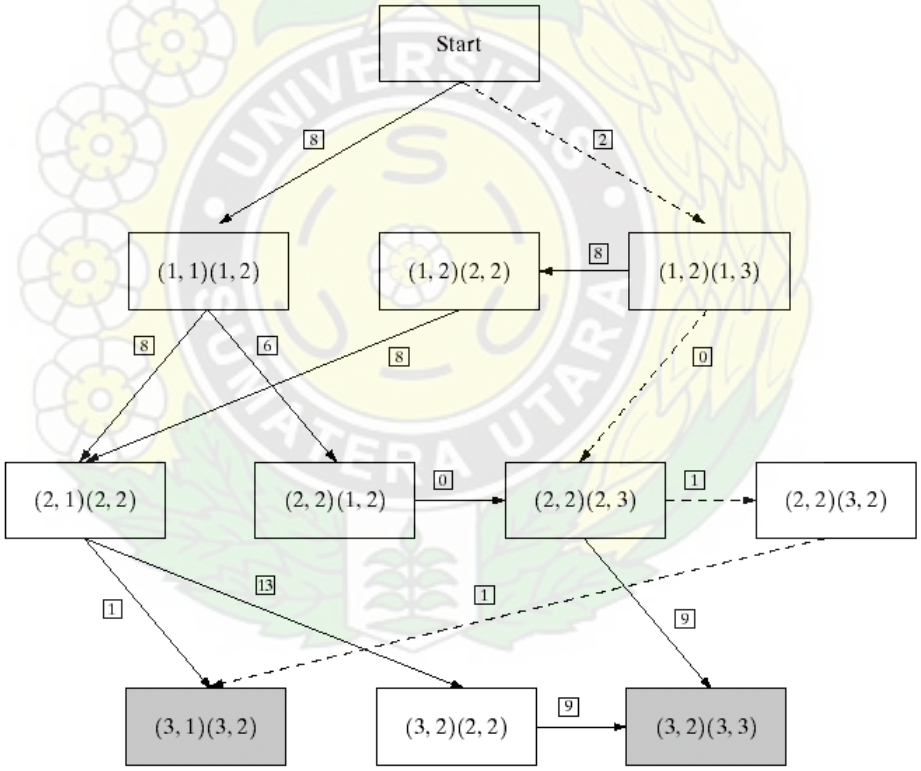
$f(p)$dan $f(q)$: nilai keabuan piksel *p* dan *q*

Pada contoh berikut, akan dilakukan proses segmentasi dasar dengan menggunakan teori graph pada citra berukuran 3×3. Untuk memudahkan, diasumsikan bahwa sisi di mulai dari baris teratas dan berakhir di baris terakhir. Dengan kata lain, sisi hanya bisa berawal dari (1,1)(1,2) atau (1,2)(1,3), dan berakhir di (3,1)(3,2) atau (3,2)(3,3) (Proses ini ditunjukkan pada Gambar 2.3 dan Gambar 2.4).



a b c

Gambar 2.3 (a) citra berukuran 3×3. (b) segmentasi citra dan nilainya. (c) penyesuaian sisi pada lintasan sisi dengan nilai terendah pada graph



Gambar 2.4 Representasi graph pada gambar 2.3(c)

Dalam hal proses segmentasi citra, graph $G = (V, E)$ merupakan graph tidak berarah (*undirected graph*), dengan simpul-simpul $v_i \in V$ merupakan himpunan elemen-elemen yang akan disegmentasi dan sisi $(v_i, v_j) \in E$ merupakan pasangan-pasangan dari piksel-piksel ketetanggaannya. Setiap sisi $(v_i, v_j) \in E$ memiliki bobot

$w\left((v_i, v_j)\right)$ atau $w(e)$, dimana bobot-bobotnya tidak negatif dan tidak sama diantara simpul v_i dan v_j yang bertetanggaan.

Proses segmentasi dalam pendekatan pada graph dilakukan melalui proses partisi graph yaitu dilakukan pengelompokan dengan cara mempartisi himpunan dari simpul-simpulnya. Segmentasi S merupakan partisi dari V ke dalam komponen-komponen C sehingga setiap komponen $C \in S$ bersesuaian dengan komponen yang terhubung dalam sebuah graph $G' = (V, E')$ dimana $E' \geq E$. Atau dengan kata lain, segmentasi yang dilakukan diinduksi oleh sebuah subset dari sisi E.

Ada berbeda cara dalam menentukan kualitas dari segmentasi, tetapi pada umumnya elemen yang diharapkan dalam sebuah komponen adalah sama dan elemen pada komponen yang berbeda adalah tidak sama. Ini berarti bahwa sisi diantara dua simpul di komponen yang sama harus mempunyai bobot yang relatif rendah dan sisi diantara simpul-simpul di berbeda komponen harus mempunyai bobot yang tinggi. Tujuan mempartisi citra ke dalam himpunan yang berbeda daerah adalah untuk memperoleh struktur dari citra tersebut, untuk merepresentasi citra sehingga menjadi lebih kompak, serta untuk mengetahui perbedaan struktur citra yang memiliki level tinggi maupun level rendah.

2.5.2.1 Daerah Pasangan Berurutan dari Perbandingan Predikatnya

Didefinisikan D sebagai predikat untuk mengevaluasi apakah ada atau tidak bukti untuk sebuah batasan antara dua komponen pada segmentasi (dua daerah dari citra). Predikat ini didasarkan pada ukuran ketidaksamaan antar elemen sepanjang batas dua relatif komponen ke sebuah ukuran ketidaksamaan antara elemen ketetanggaan dalam setiap dua komponen. Hasil predikat membandingkan perbedaan inter-komponen dengan perbedaan antar-komponen dan ini sesuai dengan data lokal karakteristik yang berlaku. (Felzenszwalb, 2004).

Didefinisikan *internal difference* (Int) pada komponen $C \geq V$ adalah bobot terbesar dalam komponen *minimum spanning tree* ($MST(C, E)$) yaitu,

$$Int(C) = \max_{e \in MST(C,E)} w(e)$$

Salah satu intuisi yang mendasari ukuran ini adalah komponen C yang diberikan hanya terhubung ketika setidaknya bobot dari sisi $Int(C)$ diperhitungkan.

Didefinisikan *difference between* (Dif) antara dua komponen $C_1, C_2 \subseteq V$ adalah bobot minimum dari sisi yang menghubungkan dua komponen yaitu,

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w\left((v_i, v_j)\right)$$

Jika tidak ada sisi yang terhubung antara C_1 dan C_2, maka anggap $Dif(C_1, C_2) = \infty$. Pada prinsipnya perbedaan ukuran akan menjadi masalah, karena hanya mencerminkan bobot terkecil dari sisi antar dua komponen. Dalam prakteknya ditemukan bahwa ukuran ini bekerja dengan baik meskipun ada keterbatasannya. Selain itu, definisi berubah menjadi menggunakan bobot rata-rata, atau *quantile* lain, dalam rangka membuat bentuk yang lebih kuat.

Perbandingan daerah predikat mengevaluasi apakah ada bukti untuk batas antara pasangan atau komponen dengan memeriksa apakah perbedaan antara komponen-komponen $Dif(C_1, C_2)$, adalah relatif terbesar tehadap perbedaan internal setidaknya satu dari komponen $Int(C_1)$ dan $Int(C_2)$. Sebuah fungsi batas digunakan untuk mengontrol derajat (*degree*) dimana perbedaan antara komponen-komponen harus lebih besar daripada minimum *internal difference*. Didefinisikan pasangan berurutan perbandingan predikat sebagai berikut:

$$D(C_1, C_2) = \begin{cases} \text{benar} & \text{jika } Dif(C_1, C_2) > M\ Int(C_1, C_2) \\ \text{salah} & \text{untuk yang lainnya} \end{cases}$$

Dimana minimum *internal difference* ($M\ Int$) didefinisikan sebagai berikut:

$$M\ Int(C_1, C_2) = \min\left(int(C_1) + \pi(C_1), int(C_2) + \pi(C_2)\right)$$

Fungsi ambang (*threshold*) π berguna untuk mengontrol derajat, dimana perbedaan antar dua komponen harus lebih besar dari *internal difference*-nya. Hal ini menunjukkan adanya bukti batasan antara dua komponen. Untuk komponen yang

kecil, $Int(C)$ bukanlah ukuran yang baik dari untuk karakteristik lokal data. Dalam kasus yang lebih besar, ketika $|C| = 1$, $Int(C) = 0$. Oleh karena itu, digunakan fungsi ambang berdasarkan ukuran komponennya, yaitu,

$$\pi(C) = \frac{k}{|C|}$$

dimana $|C|$ dinotasikan sebagai ukuran C dan k sebagai parameter konstanta. Jelasnya, untuk komponen yang kecil dibutuhkan bukti yang kuat sebagai batasan. Dalam prakteknya k menetapkan skala pengamatan, dalam k yang lebih besar menyebabkan k menjadi lebih besar. Bagaimanapun k bukan ukuran komponen minimum. Komponen yang lebih kecil diperbolehkan bila ada perbedaan yang cukup besar antar komponen bertetangga.

2.6 Matlab (*Matrix Laboratory*)

Matlab (*matrix laboratory*) merupakan bahasa pemrograman dengan kemampuan tinggi dalam bidang komputasi. Matlab adalah sebuah bahasa (pemrograman) dengan cara kerja tinggi (*high performance*) untuk teknik komputasi, yang mengintegrasikan komputasi, visualisasi, dan pemrograman di dalam lingkungan yang mudah penggunaannya dalam memecahkan persoalan dengan solusinya yang dinyatakan dengan notasi matematika. Matlab dikembangkan oleh MathWorks, yang pada awalnya dibuat untuk memberikan kemudahan mengakses data matriks pada proyek LINPACK dan EINSPACK. Selanjutnya menjadi sebuah aplikasi untuk komputasi matriks. Dari sejak awal dipergunakan, matlab memperoleh masukan ribuan pemakai. Penggunaan matlab antara lain: matematika dan komputasi, pengembangan algortima, pemodelan, simulasi dan prototipe, analisis data, eksplorasi, dan visualisasi, grafik untuk sains dan teknik, serta pengembangan aplikasi, termasuk pembuatan *graphical user interface*. (Wijaya, 2007).

Sistem matlab terdiri dari lima bagian utama, yaitu:

1. Bahasa (pemrograman) matlab

Bagian ini adalah bahasa (pemrograman) tingkat tinggi yang menggunakan matriks/*array* dengan pernyataan aliran kendali program, struktur data, masukan/keluaran, dan fitur-fitur pemrograman berorientasi objek.

2. Lingkungan kerja matlab

   Bagian ini adalah sekumpulan peralatan dan fasilitas matlab yang digunakan oleh pengguna. Fasilitas yang dimaksud misalnya untuk mengelola variabel di dalam ruangan kerja (*workspace*) dan melakukan impor dan ekspor data data. Sedangkan peralatan disediakan untuk pengembangan, pengelolaan, proses *debugging* dan pembuatan M-*files* untuk aplikasi matlab.

3. Penanganan grafik

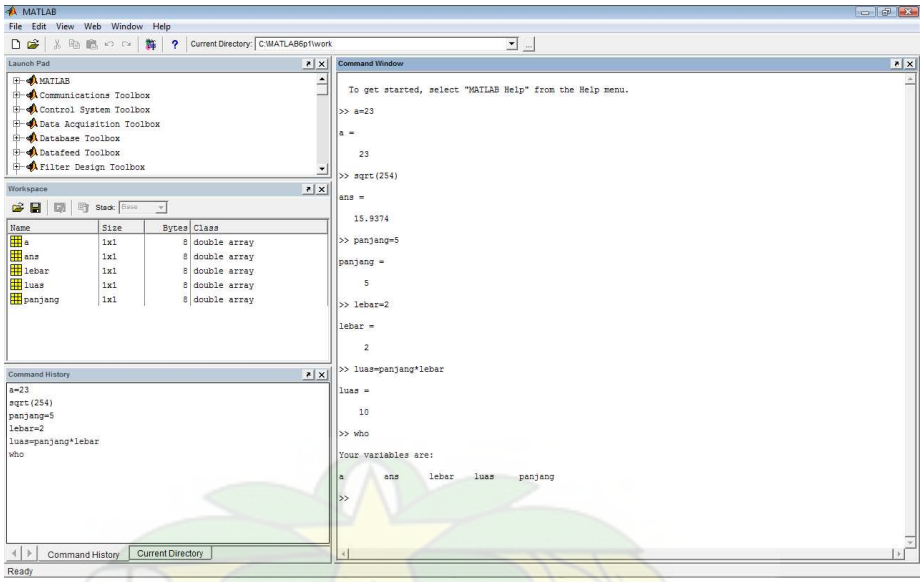
   Bagian ini adalah sistem grafik matlab, termasuk perintah-perintah (program) tingkat tinggi untuk visualisasi data dimensi dua dan dimensi tiga, pengolahan citra, animasi, dan presentasi grafik. Selain itu juga, bagian ini juga termasuk perintah-perintah (program) tingkat rendah untuk menetapkan sendiri tampilan grafik seperti halnya membuat *graphical user interface* untuk aplikasi matlab.

4. Pustaka (*library*) fungsi matematis matlab

   Bagian ini adalah koleksi algoritma komputasi mulai dari fungsi dasar seperti menjumlahkan (*sum*), menentukan nilai sinus (*sine*), kosinus (*cosine*), dan aritmatika bilangan kompleks, serta fungsi-fungsi seperti invers matriks, nilai eigen matriks, fungsi Bessel, dan FFT (*fast fourier transform*).

5. API (*application program interface*) matlab

   Bagian ini adalah pustaka untuk menuliskan program dalam bahasa C dan Fortran yang berinteraksi dengan matlab, termasuk fasilitas untuk memanggil rutin program dari matlab, memanggil matlab sebagai mesin komputasi dan untuk pembacaan serta penulisan MAT-*files*.



Gambar 2.5 Jendela utama matlab

Beberapa bagian penting dalam matlab antara lain:

1. Jendela perintah (*command window*)

   Pada jendela perintah, semua perintah matlab dituliskan dan dieksekusi. Pada dasarnya jendela inilah inti dari pemrograman matlab yang menjadi media utama satu-satunya untuk berinteraksi dengan matlab.

2. Jendela ruang kerja (*workspace window*)

   Jendela ini berfungsi sebagai navigator bagi pemakai dalam menyediakan informasi mengenai variabel yang sedang aktif dalam *workspace* pada saat pemakaian. *Workspace* merupakan suatu lingkungan abstrak yang menyimpan seluruh variabel dan perintah yang pernah digunakan selama penggunaan matlab berlangsung.

3. Jendela *command history*

Jendela ini berfungsi sebagai penyimpanan perintah-perintah yang pernah dikerjakan sebelumnya pada suatu *workspace*.

4. *Launch pad*

   Matlab menyediakan *launch pad* untuk memudahkan mengakses produk-produk matlab seperti demo dan dokumentasi.

5. *Preference*

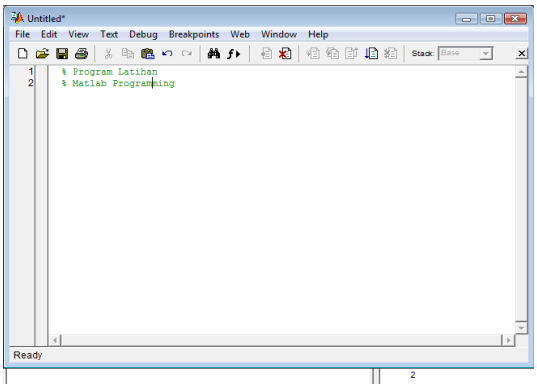
   Preference merupakan fasilitas yang digunakan untuk mengatur segala sesuatu tentang matlab, misalnya pengaturan jenis, ukuran maupun warna *font* untuk *keyword*, komentar, *string*, *error*, dan sebagainya. *Preference* dapat dibuka dari menu *file*.

6. *Current directory*

   Current directory digunakan untuk menentukan *directory* aktif yang digunakan matlab. Jika akan menjalankan sebuah fungsi, maka harus memastikan bahwa fungsi berada di dalam *directory* aktif atau mengubah *directory* aktifnya ke *directory* tempat fungsinya berada. Jika tidak, matlab akan memberikan pesan kesalahan.

7. Matlab *editor*

   Jendela ini berfungsi untuk membuat *script* program matlab. Walaupun *script* dapat dibuat dengan menggunakan berbagai program *editor* seperti *notepad*, *wordpad*, *word* dan sebagainya, namun sangat dianjurkan untuk menggunakan matlab *editor* karena kemampuannya dalam mendeteksi kesalahan pengetikan sintaks oleh *programmer*. (Sugiharto, 2006).



Gambar 2.6 Matlab editor

Ada banyak tipe data dasar pada matlab. Masing-masing tipe data memiliki bentuk yang sama yaitu *array*. *Array* minimal berukuran 0×0 dan dapat bertambah menjadi *array n* dimensi dengan sembarang ukuran. Tipe-tipe datanya yaitu:

a. Logika yaitu tipe data yang memiliki nilai *true* atau *false* dan masing-masing direpresentasikan dengan angka 1 atau 0. Matlab menghasilkan nilai *logic* dari operasi relasi (==, <=, ...) maupun operasi *logic* (&&, xor, ...).
b. Char yaitu tipe data yang melibatkan karakter. Sebuah string dari karakter merupakan *array* dari karakter yang berukuran $1 \times n$.
c. Numerik yaitu tipe data yang berupa bilangan, baik bilangan bulat bertanda maupun yang tidak, serta bilangan pecahan.
d. *Cell* yaitu tipe data yang dapat diibaratkan sebagai sebuah *array* atau kotak-kotak ataupun kontainer yang dapat memuat data yang berbeda atau tipe data matlab lainnya. Setipa sel dalam sel *array* dapat memuat semua jenis tipe data matlab yang meliputi *array*, teks, objek simbol, sel *array*, maupun struktur.
e. Struktur yaitu tipe data yang berorientasi *array* dengan *field-field* yang memiliki nama dan dapat memuat segala jenis data, termasuk sel *array* atau struktur lainnya.
f. *Java classes* yaitu tipe data yang digunakan matlab sebagai *interface*-nya dengan java.
g. *Function handles* yaitu tipe data yang digunakan untuk menampilkan informasi yang digunakan dalam referensi sebuah fungsi. Ketika tipe data dibuat, matlab menangkap semua informasi tentang fungsi.

BAB 3

PEMBAHASAN

3.1 Masalah yang Akan Dipecahkan

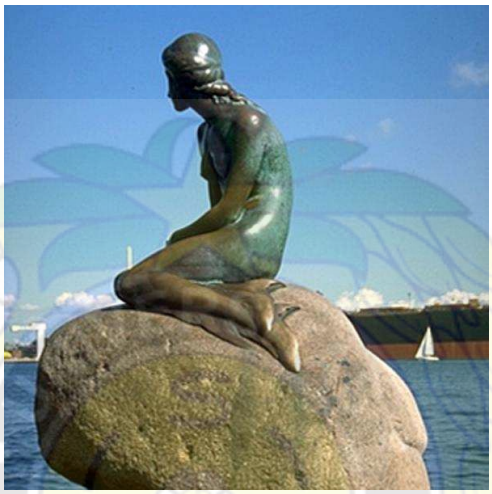
Segmentasi citra (*image segmentation*) merupakan salah satu proses penting yang banyak aplikasinya dalam pengolahan citra, namun tekniknya cukup rumit. Segmentasi citra diperlukan karena proses ini merupakan proses identifikasi objek yang dinyatakan dalam bentuk dasar, dimana objek dalam suatu citra akan lebih mudah dikenal apabila strukturnya jelas. Melalui proses segmentasi citra diharapkan objek yang terkandung pada gambar dapat dideskripsikan dan dikenali. Proses-proses yang termasuk dalam segmentasi citra, yaitu:

1. *Preprocessing,* merupakan kumpulan dari proses yang digunakan untuk dapat menghasilkan segmentasi yang baik.
   a. Konversi yaitu proses mengubah citra menjadi citra lain. Dalam hal ini konversi yang dilakukan adalah konversi citra RGB menjadi citra *grayscale*.

   b. Konvolusi yaitu proses menghilangkan komponen *noise* yang variasinya mempunyai pola acak dan bersifat jangka pendek serta tidak mempunyai korelasi dengan *noise* yang terjadi di titik-titik sekitarnya dan mempunyai karakter pemerataan atau pengaburan.
   c. Filterisasi yaitu proses meningkatkan mutu citra yang mengalami gangguan pada saat citra awal diproses oleh perekam citra digital. Dalam hal ini digunakan jenis filter tipe gaussian, karena filter ini sangat baik untuk menghilangkan *noise* yang bersifat sebaran normal, yang banyak dijumpai pada citra hasil proses digitalisasi menggunakan kamera.
2. Segmentasi citra dengan metode graph yang efisien yaitu melakukan proses pengelompokan komponen-komponen yang terdapat didalam citra.

3.2 Objek Citra yang Diteliti

Objek citra yang akan diteliti pada proses segmentasi citra dengan menggunakan metode graph yang efisien adalah file citra yang berformat JPG/JPEG (*Joint Photographic Experts Group*), berukuran 600×600 piksel, dengan skala keabuan 256 (8 bit). Gambar dibawah ini adalah objek yang akan diteliti.

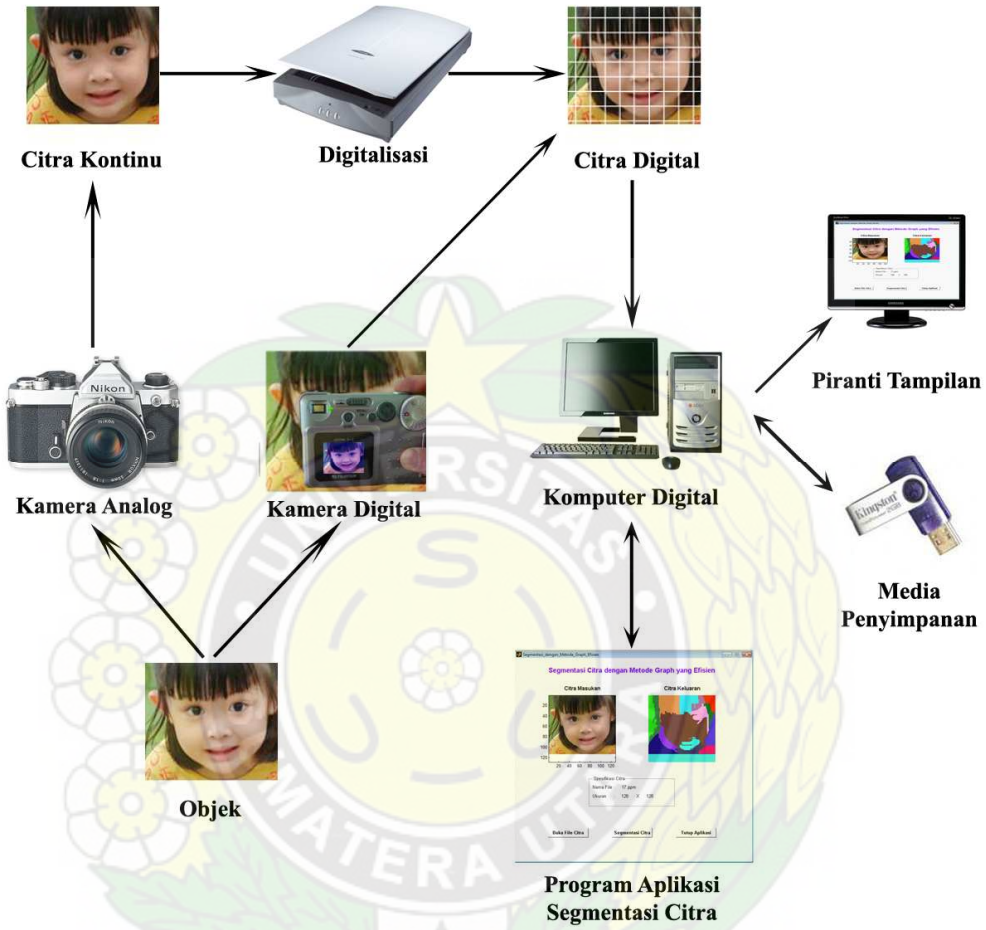


Gambar 3.1 Objek citra yang akan diteliti

3.3 Pengolahan Citra Digital

Citra digital direpresentasikan dengan dua dimensi yang disebut larik (*array*) atau matriks. Operasi pada citra digital pada dasarnya adalah memanipulasi atau memodifikasi elemen-elemen matriks. Elemen yang dimanipulasi dapat berupa elemen tunggal (sebuah piksel), sekumpulan elemen yang berdekatan, atau keseluruhan elemen matriks. Citra digital dapat didefinisikan sebagai dua variabel $f(x, y)$, dimana x dan y adalah koordinat spasial dan nilai $f(x, y)$ adalah intensitas citra pada koordinat tersebut. Sebuah citra diubah ke bentuk digital agar dapat disimpan dalam memori komputer atau media lain. Proses mengubah citra ke bentuk digital bisa dilakukan dengan beberapa perangkat, misalnya *scanner*, kamera digital, dan *handycam*. Ketika sebuah citra sudah diubah ke dalam bentuk digital (selanjutnya

disebut citra digital), bermacam-macam proses pengolahan citra dapat dilakukan terhadap citra tersebut. (Ahmad, 2005).



Gambar 3.2 Proses pengolahan citra digital

Pada awal permulaan melalui proses pengolahan citra digital dari objek sampai pada merekam objek menggunakan kamera analog dihasilkan citra kontinu, dimana citra kontinu ini berasal dari sumber cahaya menyinari objek, objek memantulkan kembali sebagian dari berkas cahaya tersebut, pantulan ini ditangkap oleh sensor visual pada sistem optik pada kamera analog.

Digitalisasi merupakan sistem penangkap citra digital yang melakukan penjelajahan citra dan menkonversinya ke representasi numerik sebagai masukan bagi komputer digital. Hasil dari digitalisasi adalah matriks yang elemen-elemennya

menyatakan nilai intensitas cahaya pada suatu titik. Alat yang digunakan untuk digitalisasi adalah *scanner*. Digitalisasi terdiri dari tiga komponen dasar yaitu sensor citra yang bekerja sebagai pengukur intensitas cahaya, perangkat penjelajah yang berfungsi merekam hasil pengukuran intensitas pada seluruh bagian citra, dan pengubah analog ke digital yang berfungsi melakukan spasial dan kuantitas.

Komputer digital yang digunakan pada sistem pemroses citra dapat bervariasi dari komputer mikro sampai komputer besar yang mampu melakukan bermacam-macam fungsi pada citra digital resolusi tinggi. Program aplikasi pengolahan citra digital berfungsi untuk mengolah citra masukan yang akan diperbaiki kualitas atau mutu citra masukan menjadi citra keluaran yang menghasilkan citra yang lebih mudah dideskripsikan.

Piranti tampilan peraga berfungsi untuk mengkonversi matriks intensitas yang merepresentasikan citra ke tampilan yang dapat diinterpretasikan oleh mata manusia. Contoh piranti tampilan adalah monitor. Media penyimpanan adalah piranti yang mempunyai kapasitas memori besar sehingga gambar dapat disimpan secara permanen agar dapat diproses lagi pada waktu mendatang. Alat yang digunakan untuk media penyimpanan adalah *hard disk*, disket atau media penyimpanan lainnya.

3.4 Algoritma Segmentasi Citra menggunakan Metode Graph yang Efisien

Untuk sampai pada perancangan program, akan ditentukan dahulu bentuk algoritma dalam mendukung proses segmentasi citra menggunakan metode graph yang efisien, yaitu:

Inputnya adalah sebuah graph $G = (V, E)$ dengan n simpul dan m sisi. Outputnya adalah sebuah segmentasi dari V ke komponen-komponen $S = (C_1, \ldots, C_r)$.

1. Urutkan sisi E ke dalam $(e_1, e_2, \ldots, e_m)$ dengan bobot sisi tidak menurun.
2. Mulai dengan segmentasi S^0, dimana setiap simpul v_i ada didalam komponen itu sendiri.
3. Ulangi langkah 4 untuk setiap e_q dari $(e_1, e_2, \ldots, e_m)$.

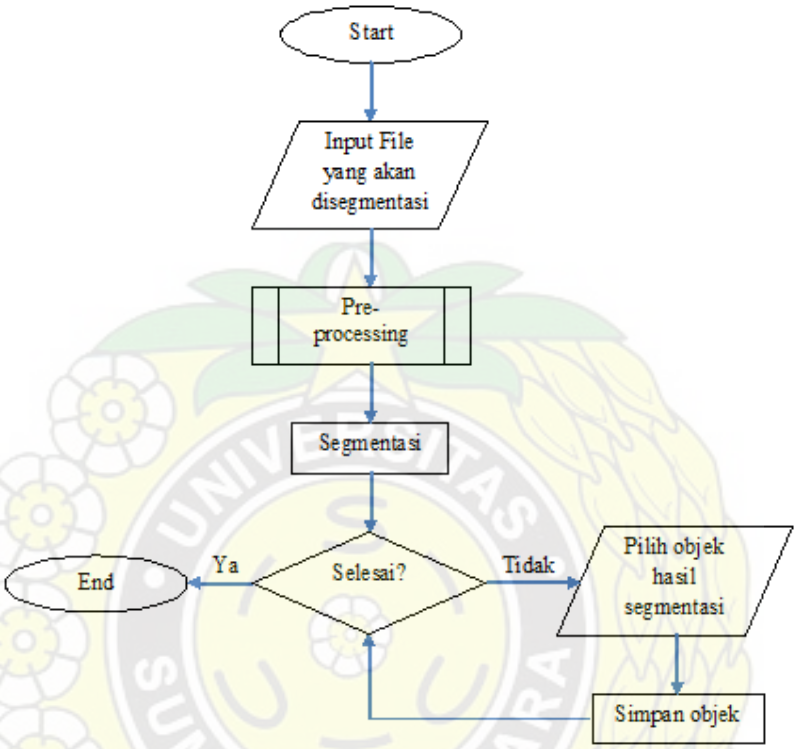
4. jika bobot e_q relatif lebih kecil dari *internal difference* komponen yang terhubung, maka gabungkan komponen, selain itu tidak tidak berlaku apapun. Atau lebih formalnya, jika $w(e_q) \leq MInt(C_i, C_j)$ dimana $C_i, C_j \in S$ adalah komponen yang jelas terhubung dengan e_q, maka perbaharui S dengan menggabungkan C_i dan C_j.
5. Ulangi $S = S^m$. (Felzenswalb *et al*, 2004).

3.5 *Flowchart* Segmentasi Citra Menggunakan Metode Graph yang Efisien

Suatu program yang baik haruslah bebas dari kesalahan dan kemungkinan kesalahan serta memiliki urutan sistematis dan terarah agar prosesnya dapat terlaksana secara efisien. Untuk itu sebelum menyusun suatu program, perlu disusun urutan-urutan proses yang logis dan sistematis. Maka akan ditentukan dahulu bentuk *flowchart* dalam mendukung proses segmentasi citra, diantaranya:

3.5.1 *Flowchart* Secara Garis Besar

Secara garis besar algoritma segmentasi citra ditunjukkan sebagai berikut.



Gambar 3.3 *Flowchart* secara garis besar

Keterangan:

(i) *Input file* yang nantinya akan disegmentasi, setelah nama *file* di-*input*, gambar di-*load* dan dipersiapkan untuk segmentasi.

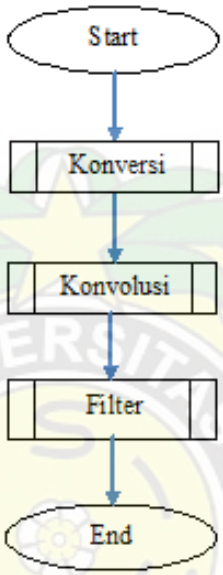
(ii) *Preprocessing* di sini masih terdiri dari beberapa proses dan akan dibahas lebih lanjut setelah ini.

(iii) Gambar yang sudah disiapkan melalui proses *preprocessing* disegmentasi dengan menggunakan metode graph yang efisien.

(iv) Proses dapat berhenti di sini atau dapat juga dilakukan proses segmentasi ulang untuk parameter berbeda.

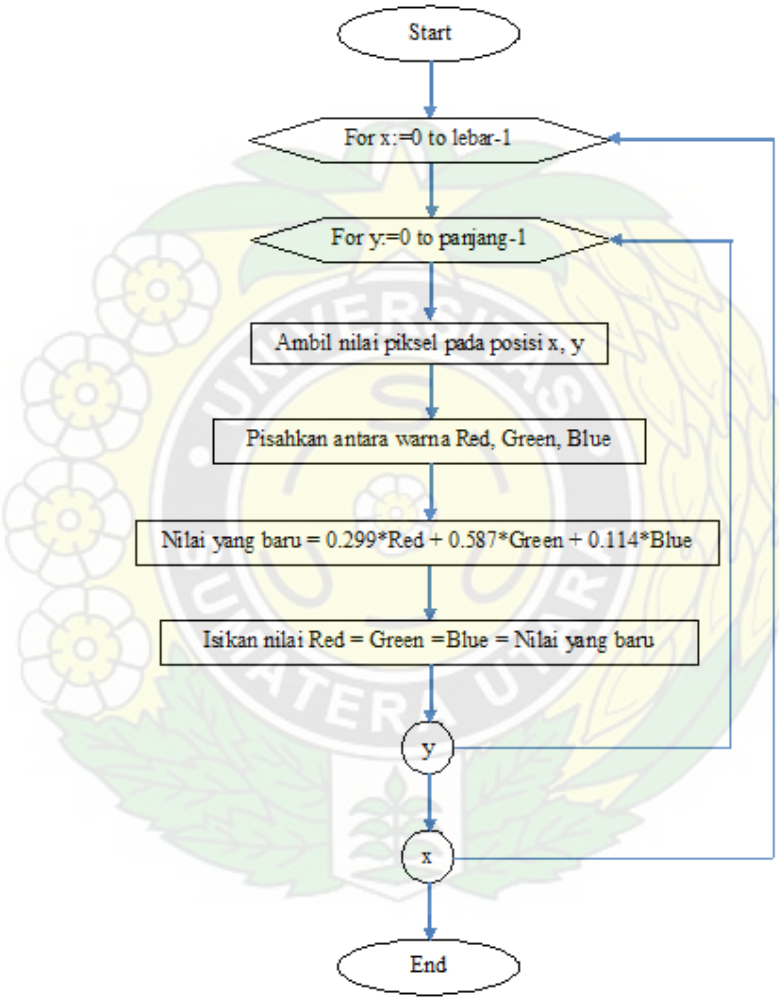
3.5.2 *Flowchart Preprocessing*

Preprocessing merupakan kumpulan dari proses yang digunakan untuk dapat menghasilkan segmentasi yang terbaik. *Preprocessing* yang digunakan adalah konversi, konvolusi, dan filterisasi. *Flowchart* ditunjukkan sebagai berikut.



Gambar 3.4 *Flowchart preprocessing*

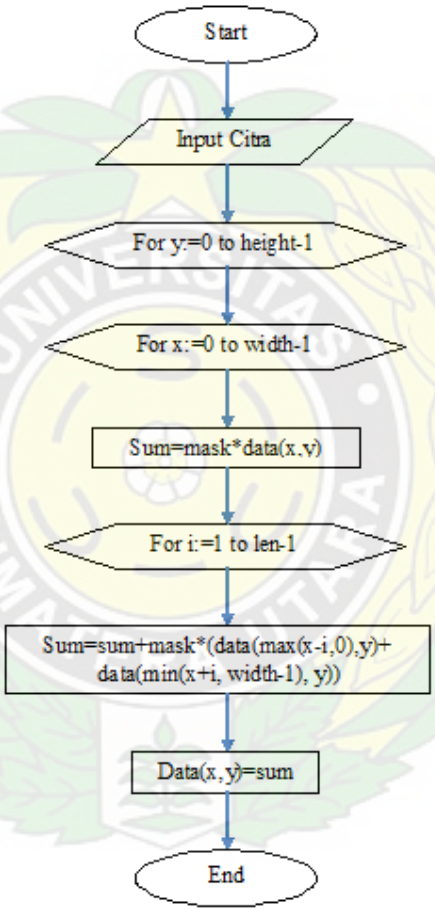
Dari proses *preprocessing* yang terlihat pada *flowchart* sebelumnya, dikatakan bahwa salah satu proses awal dari proses *preprocessing* ini adalah proses konversi. Dalam hal ini, proses konversi yang dilakukan adalah proses konversi citra RGB menjadi citra *grayscale* (gambar yang memiliki tingkat warna abu-abu). Pada gambar 3.5 dijelaskan bagaimana cara kerja proses ini dalam bentuk *flowchart*.



Gambar 3.5 *Flowchart* Konversi

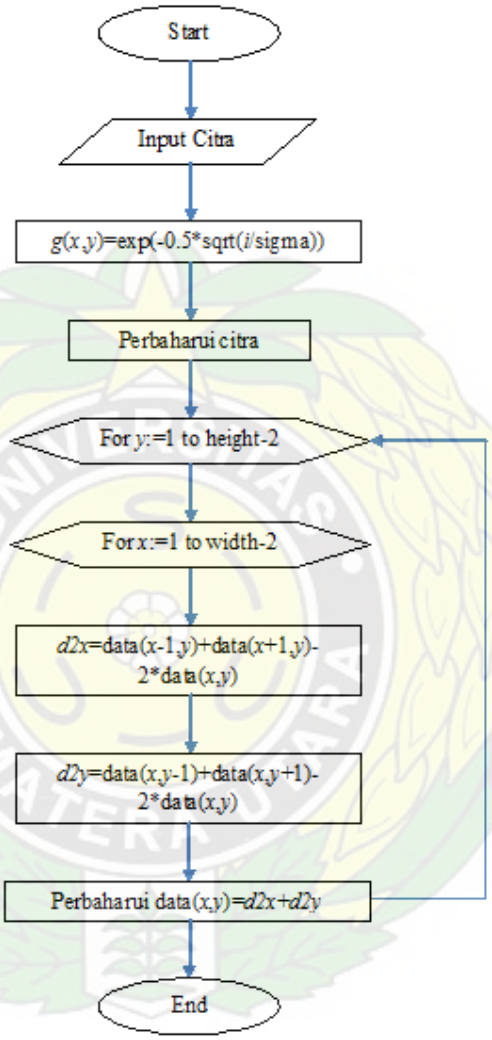
Nilai tiap titik citra yang akan di konversi akan disamakan nilai *red*, *green*, dan *blue*-nya sehingga untuk tiap titik hanya memiliki satu nilai saja yang disebut nilai *gray level*. Proses ini mengambil persentasi tertentu dari masing-masing warna kemudian dijumlahkan untuk mendapatkan nilai yang baru.

Proses kedua setelah proses konversi dari proses *preprocessing* adalah proses konvolusi. Proses konvolusi sangat berguna pada proses pengolahan citra seperti perbaikan kualitas citra (*image enhancement*), penghilangan derau, penghalusan/ pelembutan citra, deteksi tepi, dan penajaman tepi. Dalam hal ini, proses konvolusi digunakan untuk menghilangkan derau dan menajamkan tepi. Pada Gambar 3.6 dijelaskan bagaimana cara kerja proses ini dalam bentuk *flowchart* secara sederhana.



Gambar 3.6 *Flowchart* konvolusi

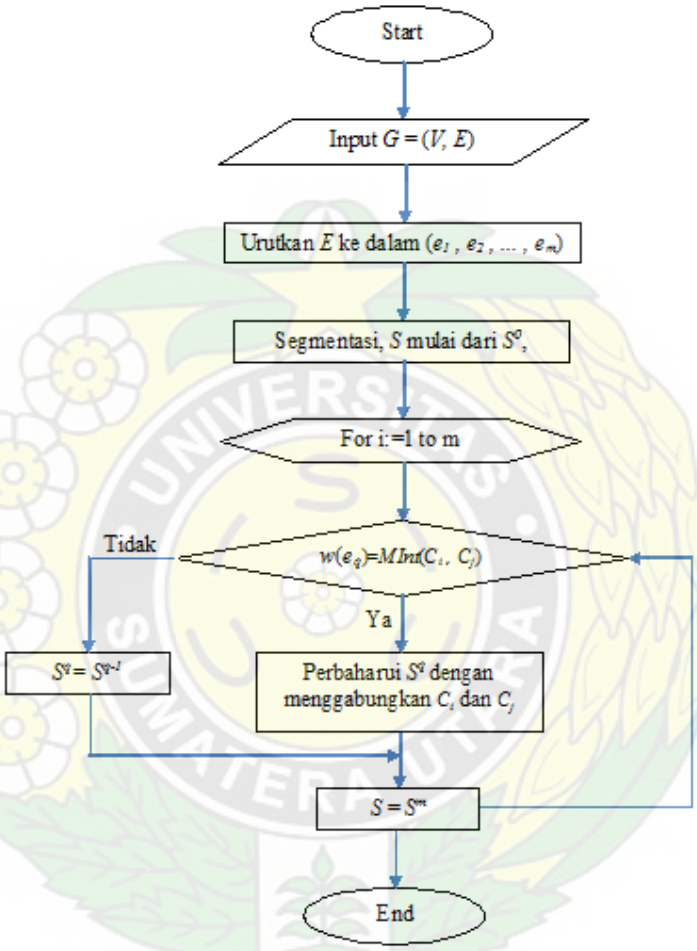
Proses terakhir dari proses *preprocessing* adalah proses filterisasi, dimana proses yang dilakukan disini adalah untuk menghilangkan *noise* dan menghaluskan citra kemudian mendeteksi tepi dari citra. Proses ditunjukkan pada *flowchart* berikut.



Gambar 3.7 *Flowchart* Filterisasi

3.5.3 *Flowchart* Segmentasi Citra

Proses segmentasi citra merupakan proses utama dari pengolahan citra ini. Berikut ditampilkan *flowchart* dari proses tersebut.



Gambar 3.8 *Flowchart* segmentasi citra

BAB 4

RANCANGAN PROGRAM

4.1 Komponen Rancangan Program

Untuk membangun suatu program aplikasi segmentasi citra dengan metode graph yang efisien menggunakan matlab, komponen yang diperlukan pada *form* utama yang akan dirancang adalah sebagai berikut:

Tabel 4.1 Komponen *form* utama

Komponen	Properti	Keterangan
Form	Tag	Figure1
	String	Segmentasi_Citra_dengan_Metode_Graph_yang_Efisien
Gambar1	Tag	Axe1
	Height	Auto
	Width	Auto
Gambar2	Tag	Axe2
	Height	Auto
	Width	Auto
Nama File	Tag	kosong1
	String	-
Ukuran	Tag	kosong2
	String	-
	Tag	kosong3
	String	-
Tombol1	Tag	tombol_buka
	String	Buka File Citra
Tombol2	Tag	segmentasi
	String	Segmentasi Citra
Tombol3	Tag	tombol_tutup
	String	Tutup Aplikasi

Pada *form* utama sebagai *form* induk, *form* ini memiliki fungsi sebagai berikut:

1. Membuka file citra masukan.
2. Menampilkan citra masukan.
3. Menampilkan spesifikasi citra.
4. Menampilkan proses perubahan citra keluaran.
5. Menyimpan file citra keluaran.
6. Menutup program.

Adapun batasan dari program ini adalah file citra yang digunakan harus berukuran sama dan dengan 256 tingkat skala keabuan (8 bit).

Gambar 4.1 Tampilan *form* utama

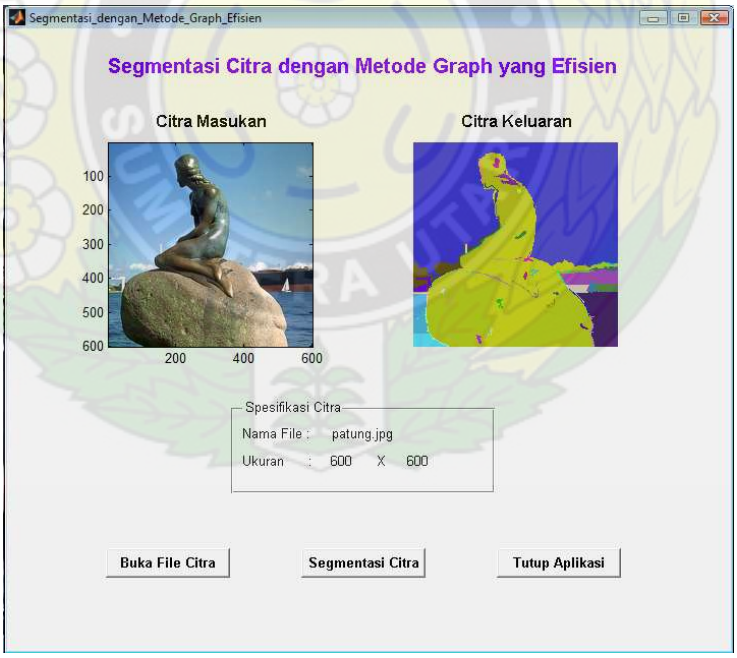
4.2 Uji Program

Untuk mengimplementasikan proses program segmentasi citra dengan metode graph yang efisien menggunakan matlab, program dibuat dengan memiliki kemampuan untuk:

1. Membuka file yang berisi citra digital.
2. Menampilkan citra di layar.
3. Melakukan proses segmentasi citra dengan metode graph yang efisien.
4. Menampilkan citra digital hasil proses segmentasi.
5. Menyimpan file hasil proses.

5.2.1 Proses Segmentasi Citra dengan Metode Graph yang Efisien

Sebelum melakukan proses segmentasi citra dengan metode graph yang efisien, terlebih dahulu file citra dipilih dan dimasukkan ke dalam program aplikasi citra yang telah dibuat. Setelah file citra dimasukkan dan ditampilkan pada *form* utama pada posisi citra masukan, dapat dilakukan proses segmentasi citra dengan metode graph yang efisien.



Gambar 4.2 Proses segmentasi citra menggunakan metode graph yang efisien

Dalam proses segmentasi citra dengan metode graph yang efisien, dapat dilihat pada citra masukan terdapat citra dengan batas-batas yang belum jelas hanya dengan melihat menggunakan penglihatan mata saja. Setelah melakukan pengaturan

menggunakan metode graph yang efisien dengan nilai $\mathbf{sigma = 0.5}$, $\mathbf{\mathit{threshold} = 1000}$, dan $\mathbf{komponen = 100}$, maka terlihatlah hasil proses tersebut pada citra keluaran yakni diperoleh batas-batas atau daerah-daerah terpisah dimana setiap daerah adalah homogen dan mengacu pada sebuah kriteria keseragaman yang jelas sehingga dengan mudah dapat diambil informasi yang terkandung didalam citra tersebut.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari pembahasan yang telah dilakukan, maka dapat disimpulkan bahwa graph yang mendasari segmentasi citra merupakan cara yang cukup efisien dan cukup efektif dalam menampilkan segmentasi citra. Nilai ambang/*threshold* dan jumlah minimum simpul harus ditunjukkan sehingga sebuah komponen dapat dianggap sebagai suatu bagian citra yang berperan penting dalam menentukan segmentasi. Algoritma yang digunakan sebenarnya sangat efisien, tetapi penerapannya tidak akan efektif untuk melihat keefektifan dari graph yang mendasari segmentasi citra.

Dari hasil pengujian dengan beberapa nilai parameter yang berbeda, diketahui bahwa semakin tinggi nilai sigma untuk proses *smoothing* citra, maka akan semakin rendah nilai komponen yang dihasilkan, sehingga semakin berkurang pengenalan objek pada citra. Sedangkan semakin tinggi nilai *threshold* yang ditentukan, maka semakin rendah pula nilai komponen yang dihasilkan.

5.2 Saran

Algoritma yang dirancang masih dapat dikembangkan lebih jauh lagi untuk memperoleh hasil yang lebih baik. Beberapa hal yang perlu dikembangkan untuk penelitian lebih lanjut antara lain:

1. Mencoba banyak *preprocessing* lainnya, sehingga dapat mengetahui apakah ada *preprocessing* yang lebih baik lagi.
2. Pengujian algoritma sebaiknya dikembangkan dengan menggunakan lebih banyak data serta piranti pengujian, baik perangkat lunak maupun perangkat keras yang lebih baik.

DAFTAR PUSTAKA

Ahmad, Usman. 2005. *Pengolahan Citra Digital dan Teknik Pemrogramannya*. Yogyakarta: Graha Ilmu.

Ballard, D. H. Dan Brown, C. M. 1982. *Computer Vision*. New Jersey: Prentice-Hall, Inc.

Chalasani, Sandeep. 2007. "Graph Based Image Segmentation". *Journal of Department of Electrical and Computer Engineering, Clemson University.*

Felzenszwalb, P. F. dan Huttenlocher, D. P. 2004. "Efficient Graph-Based Image Segmentation". *International Journal of Computer Vision* 59(2): hal. 167-181.

Gonzalez, C. R dan Wintz, Paul. 1987. *Digital Image Processing.* Canada: Addison-Wesley Publishing Company.

Hariadi, Victor. 2008. "Pemanfaatan Graph dalam Penyelesaian Permasalahan Segmentasi Citra". *Prosiding Seminar Nasional Manajemen Teknologi VII*.

Hermawan, Arief. 2006. *Jaringan Saraf Tiruan (Teori dan Aplikasi)*. Yogyakarta: Andi.

Kusumadewi, Sri. 2003. *Artificial Intelligence (Teknik dan Aplikasinya)*. Yogyakarta: Graha Ilmu.

Munir, Rinaldi. 2003. *Matematika Diskrit*. Bandung: Informatika Bandung.

Murni, Aniati dan Setiawan, Suryana. 1992. *Pengantar Pengolahan Citra*. Jakarta: PT. Elex Media Komputindo.

Sigit, R., Basuki, A., Ramadijanti, N., dan Pramadihanto, D. 2005. *Step by Step Pengolahan Citra Digital*. Yogyakarta: Andi.

Sugiharto, Aris. 2006. *Pemrograman GUI dengan Matlab.* Yogyakarta: Andi.

Suparman dan Marlan. 2007. *Komputer Masa Depan (Pengenalan Artificial Intelligence)*. Yogyakarta: Andi.

Wijaya, Marvin Ch. dan Prijono, Agus. 2007. *Pengolahan Citra Digital Menggunakan Matlab*. Bandung: Informatika Bandung.

LAMPIRAN A. *LISTING* PROGRAM *FORM* UTAMA

```
% --Form Utama Segmentasi Citra--

function varargout = Segmentasi_dengan_Metode_Graph_Efisien(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@Segmentasi_dengan_Metode_Graph_Efisien_OpeningFcn, ...
                   'gui_OutputFcn',
@Segmentasi_dengan_Metode_Graph_Efisien_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before Segmentasi_dengan_Metode_Graph_Efisien is
made visible.
function Segmentasi_dengan_Metode_Graph_Efisien_OpeningFcn(hObject,
eventdata, handles, varargin)

% Choose default command line output for
Segmentasi_dengan_Metode_Graph_Efisien
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);



% --- Outputs from this function are returned to the command line.
function varargout =
Segmentasi_dengan_Metode_Graph_Efisien_OutputFcn(hObject, eventdata,
handles)

% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in tombol_buka.
function tombol_buka_Callback(hObject, eventdata, handles)
```

```
% Memilih Citra
[FileName,PathName] = uigetfile({'*.jpg'}, 'Pilih File Citra Asli');
if isequal(FileName, 0)
    errordlg('Error...!!!','No File Selected');
    return;
else
    handles.data1=imread(fullfile(PathName,FileName));
    guidata(hObject,handles);
    handles.current_data1=handles.data1;

    % Menampilkan Citra Pada Citra Masukan
    axes(handles.axes1);
    image(handles.current_data1);
end

% Membaca Spesifikasi Citra
set(handles.kosong1,'String',FileName);
set(handles.kosong2,'String',size(handles.data1,1));
set(handles.kosong3,'String',size(handles.data1,2));



% --- Executes on button press in Tombol_tutup.
function Tombol_tutup_Callback(hObject, eventdata, handles)

% Menyimpan Citra
result=handles.result;
[FileName_simpan,PathName_simpan] = uiputfile('*.jpg','Simpan File
Citra Hasil Segmentasi');
imwrite(result, fullfile(PathName_simpan, FileName_simpan), 'jpg');

% Menutup Program
selection = questdlg(['Keluar ' get(handles.figure1,'Name') '?'],...
    ['Keluar ' get(handles.figure1,'Name') '...'],...
    'Ya','Tidak','Ya');
if strcmp(selection,'Tidak')
    return;
end
delete(handles.figure1)



% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)

image=handles.data1;
imwrite(image,'temp.ppm','ppm');

% Proses Segmentasi Citra
[status, result]=system(['segment.exe 0.5 1000 100 temp.ppm
segmentasi.ppm']);
result=imread('segmentasi.ppm');

if status~=0
  errordlg('segmentasi : Sistem','Eksekusi sistem error  - check
fungsi ''Segment''!')
```

```
end
handles.result=result;
guidata(hObject,handles);
handles.current_result=handles.result;

% Menampilkan Citra pada Citra Keluaran
axes(handles.axes2);
imshow(handles.current_result);
```

LAMPIRAN B. *LISTING* PROGRAM KONVOLUSI

----------(convolve.h)----------

```
/* konvolusi */

#ifndef CONVOLVE_H
#define CONVOLVE_H

#include <vector>
#include <algorithm>
#include <cmath>
#include <math.h>
#include "image.h"

/* konvolusi src dengan mask dan dst mengembalikan */
static void convolve_even(image<float> *src, image<float> *dst,
             std::vector<float> &mask) {
  int width = src->width();
  int height = src->height();
  int len = mask.size();

  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      float sum = mask[0] * imRef(src, x, y);
      for (int i = 1; i < len; i++) {
    sum += mask[i] *
        (imRef(src, __max(x-i,0), y) +
       imRef(src, __min(x+i, width-1), y));
      }
      imRef(dst, y, x) = sum;
    }
  }
}

/* konvolusi src dengan mask dan dst mengembalikan */
static void convolve_odd(image<float> *src, image<float> *dst,
            std::vector<float> &mask) {
  int width = src->width();
  int height = src->height();
  int len = mask.size();

  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      float sum = mask[0] * imRef(src, x, y);
      for (int i = 1; i < len; i++) {
    sum += mask[i] *
      (imRef(src, __max(x-i,0), y) -
       imRef(src, __min(x+i, width-1), y));
      }
      imRef(dst, y, x) = sum;
    }
  }
}

#endif
```

LAMPIRAN C. *LISTING* PROGRAM FILTERISASI

----------(filter.h)----------

```
/* filter sederhana */

#ifndef FILTER_H
#define FILTER_H

#include <vector>
#include <cmath>
#include "image.h"
#include "misc.h"
#include "convolve.h"
#include "imconv.h"

#define WIDTH 4.0

/* normalkan mask sehingga akan menggabungkannya jadi satu */
static void normalize(std::vector<float> &mask) {
  int len = mask.size();
  float sum = 0;
  for (int i = 1; i < len; i++) {
    sum += fabs(mask[i]);
  }
  sum = 2*sum + fabs(mask[0]);
  for (i = 0; i < len; i++) {
    mask[i] /= sum;
  }
}

/* buat filter */
#define MAKE_FILTER(name, fun)
static std::vector<float> make_ ## name (float sigma) {
  sigma = __max(sigma, 0.01F);
  int len = (int)ceil(sigma * WIDTH) + 1;
  std::vector<float> mask(len);
  for (int i = 0; i < len; i++) {
    mask[i] = fun;
  }
  return mask;
}

MAKE_FILTER(fgauss, exp(-0.5*square(i/sigma)));

/* convolve image with gaussian filter */
static image<float> *smooth(image<float> *src, float sigma) {
  std::vector<float> mask = make_fgauss(sigma);
  normalize(mask);

  image<float> *tmp = new image<float>(src->height(), src->width(),
false);
  image<float> *dst = new image<float>(src->width(), src->height(),
false);
  convolve_even(src, tmp, mask);
  convolve_even(tmp, dst, mask);
```

```
  delete tmp;
  return dst;
}

/* konvolusi citra dengan filter gaussian */
image<float> *smooth(image<uchar> *src, float sigma) {
  image<float> *tmp = imageUCHARtoFLOAT(src);
  image<float> *dst = smooth(tmp, sigma);
  delete tmp;
  return dst;
}

/* hitung laplacian */
static image<float> *laplacian(image<float> *src) {
  int width = src->width();
  int height = src->height();
  image<float> *dst = new image<float>(width, height);

  for (int y = 1; y < height-1; y++) {
    for (int x = 1; x < width-1; x++) {
      float d2x = imRef(src, x-1, y) + imRef(src, x+1, y) -
    2*imRef(src, x, y);
      float d2y = imRef(src, x, y-1) + imRef(src, x, y+1) -
    2*imRef(src, x, y);
      imRef(dst, x, y) = d2x + d2y;
    }
  }
  return dst;
}

#endif
```

LAMPIRAN D. *LISTING* PROGRAM KONVERSI

----------(imconv.h)----------

```
/* konversi citra */

#ifndef CONV_H
#define CONV_H

#include <climits>
#include "image.h"
#include "imutil.h"
#include "misc.h"

#define RED_WEIGHT  0.299
#define GREEN_WEIGHT    0.587
#define BLUE_WEIGHT 0.114

static image<uchar> *imageRGBtoGRAY(image<rgb> *input) {
  int width = input->width();
  int height = input->height();
  image<uchar> *output = new image<uchar>(width, height, false);

  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      imRef(output, x, y) = (uchar)
    (imRef(input, x, y).r * RED_WEIGHT +
     imRef(input, x, y).g * GREEN_WEIGHT +
     imRef(input, x, y).b * BLUE_WEIGHT);
    }
  }
  return output;
}

static image<rgb> *imageGRAYtoRGB(image<uchar> *input) {
  int width = input->width();
  int height = input->height();
  image<rgb> *output = new image<rgb>(width, height, false);

  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      imRef(output, x, y).r = imRef(input, x, y);
      imRef(output, x, y).g = imRef(input, x, y);
      imRef(output, x, y).b = imRef(input, x, y);
    }
  }
  return output;
}

static image<float> *imageUCHARtoFLOAT(image<uchar> *input) {
  int width = input->width();
  int height = input->height();
  image<float> *output = new image<float>(width, height, false);

  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
```

```
      imRef(output, x, y) = imRef(input, x, y);
    }
  }
  return output;
}

static image<float> *imageINTtoFLOAT(image<int> *input) {
  int width = input->width();
  int height = input->height();
  image<float> *output = new image<float>(width, height, false);

  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      imRef(output, x, y) = imRef(input, x, y);
    }
  }
  return output;
}

static image<uchar> *imageFLOATtoUCHAR(image<float> *input,
                 float min, float max) {
  int width = input->width();
  int height = input->height();
  image<uchar> *output = new image<uchar>(width, height, false);

  if (max == min)
    return output;

  float scale = UCHAR_MAX / (max - min);
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      uchar val = (uchar)((imRef(input, x, y) - min) * scale);
      imRef(output, x, y) = bound(val, (uchar)0, (uchar)UCHAR_MAX);
    }
  }
  return output;
}

static image<uchar> *imageFLOATtoUCHAR(image<float> *input) {
  float min, max;
  min_max(input, &min, &max);
  return imageFLOATtoUCHAR(input, min, max);
}

static image<long> *imageUCHARtoLONG(image<uchar> *input) {
  int width = input->width();
  int height = input->height();
  image<long> *output = new image<long>(width, height, false);

  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      imRef(output, x, y) = imRef(input, x, y);
    }
  }
  return output;
}
```

```
static image<uchar> *imageLONGtoUCHAR(image<long> *input, long min,
long max) {
  int width = input->width();
  int height = input->height();
  image<uchar> *output = new image<uchar>(width, height, false);

  if (max == min)
    return output;

  float scale = UCHAR_MAX / (float)(max - min);
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      uchar val = (uchar)((imRef(input, x, y) - min) * scale);
      imRef(output, x, y) = bound(val, (uchar)0, (uchar)UCHAR_MAX);
    }
  }
  return output;
}

static image<uchar> *imageLONGtoUCHAR(image<long> *input) {
  long min, max;
  min_max(input, &min, &max);
  return imageLONGtoUCHAR(input, min, max);
}

static image<uchar> *imageSHORTtoUCHAR(image<short> *input,
                    short min, short max) {
  int width = input->width();
  int height = input->height();
  image<uchar> *output = new image<uchar>(width, height, false);

  if (max == min)
    return output;

  float scale = UCHAR_MAX / (float)(max - min);
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      uchar val = (uchar)((imRef(input, x, y) - min) * scale);
      imRef(output, x, y) = bound(val, (uchar)0, (uchar)UCHAR_MAX);
    }
  }
  return output;
}

static image<uchar> *imageSHORTtoUCHAR(image<short> *input) {
  short min, max;
  min_max(input, &min, &max);
  return imageSHORTtoUCHAR(input, min, max);
}

#endif
```

LAMPIRAN E. *LISTING* PROGRAM SEGMENTASI CITRA

----------(segment.cpp)----------

```
#include <cstdio>
#include <cstdlib>
#include "image.h"
#include "misc.h"
#include "pnmfile.h"
#include "segment-image.h"

int main(int argc, char **argv) {
  if (argc != 6) {
    fprintf(stderr, "usage: %s sigma k min input(ppm) output(ppm)\n",
argv[0]);
    return 1;
  }

  float sigma = atof(argv[1]);
  float k = atof(argv[2]);
  int min_size = atoi(argv[3]);

  printf("loading input image.\n");
  image<rgb> *input = loadPPM(argv[4]);

  printf("processing\n");
  int num_ccs;
  image<rgb> *seg = segment_image(input, sigma, k, min_size,
&num_ccs);
  savePPM(seg, argv[5]);

  printf("got %d components\n", num_ccs);
  printf("done! uff...thats hard work.\n");

  return 0;
}
```

----------(segment-image.h)----------

```
#ifndef SEGMENT_IMAGE
#define SEGMENT_IMAGE

#include <cstdlib>
#include "image.h"
#include "misc.h"
#include "filter.h"
#include "segment-graph.h"

// pewarnaan acak
rgb random_rgb(){
  rgb c;
  double r;

  c.r = (uchar)rand();
  c.g = (uchar)rand();
  c.b = (uchar)rand();
```

```
  return c;
}

// Ukuran ketidaksamaan antara piksel
static inline float diff(image<float> *r, image<float> *g,
image<float> *b,
             int x1, int y1, int x2, int y2) {
  return sqrt(square(imRef(r, x1, y1)-imRef(r, x2, y2)) +
          square(imRef(g, x1, y1)-imRef(g, x2, y2)) +
          square(imRef(b, x1, y1)-imRef(b, x2, y2)));
}

/*
 * menggolongkan citra
 *
 * mengembalikan gambaran segmentasi dari warna citra.
 *
 * im: citra yang akan digolongkan.
 * sigma: untuk memperhalus citra.
 * c: konstanta untuk fungsi threshold.
 * min_size: ukuran minimum komponen.
 * num_ccs: banyaknya komponen terhubung dalam segmentasi.
 */

image<rgb> *segment_image(image<rgb> *im, float sigma, float c, int
min_size,
             int *num_ccs) {
  int width = im->width();
  int height = im->height();

  image<float> *r = new image<float>(width, height);
  image<float> *g = new image<float>(width, height);
  image<float> *b = new image<float>(width, height);

  // smooth each color channel
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      imRef(r, x, y) = imRef(im, x, y).r;
      imRef(g, x, y) = imRef(im, x, y).g;
      imRef(b, x, y) = imRef(im, x, y).b;
    }
  }
  image<float> *smooth_r = smooth(r, sigma);
  image<float> *smooth_g = smooth(g, sigma);
  image<float> *smooth_b = smooth(b, sigma);
  delete r;
  delete g;
  delete b;

  // bangun graph
  edge *edges = new edge[width*height*4];
  int num = 0;
  for (y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      if (x < width-1) {
    edges[num].a = y * width + x;
    edges[num].b = y * width + (x+1);
```

```
      edges[num].w = diff(smooth_r, smooth_g, smooth_b, x, y, x+1, y);
      num++;
        }

        if (y < height-1) {
      edges[num].a = y * width + x;
      edges[num].b = (y+1) * width + x;
      edges[num].w = diff(smooth_r, smooth_g, smooth_b, x, y, x, y+1);
      num++;
        }

        if ((x < width-1) && (y < height-1)) {
      edges[num].a = y * width + x;
      edges[num].b = (y+1) * width + (x+1);
      edges[num].w = diff(smooth_r, smooth_g, smooth_b, x, y, x+1,
y+1);
      num++;
        }

        if ((x < width-1) && (y > 0)) {
      edges[num].a = y * width + x;
      edges[num].b = (y-1) * width + (x+1);
      edges[num].w = diff(smooth_r, smooth_g, smooth_b, x, y, x+1, y-
1);
      num++;
        }
      }
    }
    delete smooth_r;
    delete smooth_g;
    delete smooth_b;

    // segment
    universe *u = segment_graph(width*height, num, edges, c);
    // pemrosesan komponen yang kecil
    for (int i = 0; i < num; i++) {
      int a = u->find(edges[i].a);
      int b = u->find(edges[i].b);
      if ((a != b) && ((u->size(a) < min_size) || (u->size(b) <
min_size)))
        u->join(a, b);
    }
    delete [] edges;
    *num_ccs = u->num_sets();

    image<rgb> *output = new image<rgb>(width, height);

    // mengambil warna acak untk setiap komponen
    rgb *colors = new rgb[width*height];
    for (i = 0; i < width*height; i++)
      colors[i] = random_rgb();

    for (y = 0; y < height; y++) {
      for (int x = 0; x < width; x++) {
        int comp = u->find(y * width + x);
        imRef(output, x, y) = colors[comp];
      }
    }
```

```
  delete [] colors;
  delete u;

  return output;
}

#endif
```

----------(segment-graph.h)----------

```
#ifndef SEGMENT_GRAPH
#define SEGMENT_GRAPH

#include <algorithm>
#include <cmath>
#include "disjoint-set.h"

// fungsi threshold
#define THRESHOLD(size, c) (c/size)

typedef struct {
  float w;
  int a, b;
} edge;

bool operator<(const edge &a, const edge &b) {
  return a.w < b.w;
}

/*
 * menggolongkan graph
 *
 * mengembalikan gambaran segmentasi dari himpunan disjoint dari
 * forest
 *
 * num_vertices: banyaknya simpul di graph.
 * num_edges: banyaknya sisi di graph
 * edges: array dari sisi.
 * c: konstanta untuk fungsi threshold.
 */

universe *segment_graph(int num_vertices, int num_edges, edge *edges,
            float c) {
  // susun sisi berdasarkan bobot
  std::sort(edges, edges + num_edges);

  // buat himpunan disjoint dari forest
  universe *u = new universe(num_vertices);

  // init threshold
  float *threshold = new float[num_vertices];
  for (int i = 0; i < num_vertices; i++)
    threshold[i] = THRESHOLD(1,c);

  // Untuk setiap sisi dengan bobot tidak menurun, lakukan ini...
  for (i = 0; i < num_edges; i++) {
```

```
    edge *pedge = &edges[i];

    // Komponen yang terhubung dari sisi
    int a = u->find(pedge->a);
    int b = u->find(pedge->b);
    if (a != b) {
      if ((pedge->w <= threshold[a]) &&
      (pedge->w <= threshold[b])) {
    u->join(a, b);
    a = u->find(a);
    threshold[a] = pedge->w + THRESHOLD(u->size(a), c);
      }
    }
  }

  delete threshold;
  return u;
}

#endif
```

LAMPIRAN F. *LISTING* PROGRAM PENDUKUNG

----------(image.h)----------

```
/* pengkelasan citra */

#ifndef IMAGE_H
#define IMAGE_H

#include <cstring>

template <class T>
class image {
 public:
  /* membangun citra */
  image(const int width, const int height, const bool init = true);

  /* menghapus citra */
  ~image();

  /* init citra */
  void init(const T &val);

  /* mengcopy citra */
  image<T> *copy() const;

  /* memperoleh lebar citra */
  int width() const { return w; }

  /* memperoleh tinggi citra */
  int height() const { return h; }

  /* data citra */
  T *data;

  /* pointer */
  T **access;

 private:
  int w, h;
};

/* gunakan imRef untuk mengakses data citra */
#define imRef(im, x, y) (im->access[y][x])

/* gunakan imPtr untuk memperoleh pointer ke data citra. */
#define imPtr(im, x, y) &(im->access[y][x])

template <class T>
image<T>::image(const int width, const int height, const bool init) {
  w = width;
  h = height;
  data = new T[w * h];  // pengalokasian tempat untuk data citra
  access = new T*[h];   // pengalokasian tempat untuk pointer
```

```
  // penginisialan pointer
  for (int i = 0; i < h; i++)
    access[i] = data + (i * w);

  if (init)
    memset(data, 0, w * h * sizeof(T));
}

template <class T>
image<T>::~image() {
  delete [] data;
  delete [] access;
}

template <class T>
void image<T>::init(const T &val) {
  T *ptr = imPtr(this, 0, 0);
  T *end = imPtr(this, w-1, h-1);
  while (ptr <= end)
    *ptr++ = val;
}

template <class T>
image<T> *image<T>::copy() const {
  image<T> *im = new image<T>(w, h, false);
  memcpy(im->data, data, w * h * sizeof(T));
  return im;
}

#endif
```

----------(misc.h)----------

```
/* perangkat acak */

#ifndef MISC_H
#define MISC_H

#include <cmath>

#ifndef M_PI
#define M_PI 3.141592653589793
#endif

typedef unsigned char uchar;

typedef struct { uchar r, g, b; } rgb;

inline bool operator==(const rgb &a, const rgb &b) {
  return ((a.r == b.r) && (a.g == b.g) && (a.b == b.b));
}

template <class T>
inline T abs(const T &x) { return (x > 0 ? x : -x); };

template <class T>
```

```
inline int sign(const T &x) { return (x >= 0 ? 1 : -1); };

template <class T>
inline T square(const T &x) { return x*x; };

template <class T>
inline T bound(const T &x, const T &min, const T &max) {
  return (x < min ? min : (x > max ? max : x));
}

template <class T>
inline bool check_bound(const T &x, const T&min, const T &max) {
  return ((x < min) || (x > max));
}

inline int vlib_round(float x) { return (int)(x + 0.5F); }

inline int vlib_round(double x) { return (int)(x + 0.5); }

inline double gaussian(double val, double sigma) {
  return exp(-square(val/sigma)/2)/(sqrt(2*M_PI)*sigma);
}

#endif
```

----------(pnmfile.h)----------

```
/* dasar citra Input Output */

#ifndef PNM_FILE_H
#define PNM_FILE_H

#include <cstdlib>
#include <climits>
#include <cstring>
#include <fstream>
#include "image.h"
#include "misc.h"

#define BUF_SIZE 256

class pnm_error { };

static void read_packed(unsigned char *data, int size, std::ifstream
&f) {
  unsigned char c = 0;

  int bitshift = -1;
  for (int pos = 0; pos < size; pos++) {
    if (bitshift == -1) {
      c = f.get();
      bitshift = 7;
    }
    data[pos] = (c >> bitshift) & 1;
    bitshift--;
    }
```

```
}

static void write_packed(unsigned char *data, int size, std::ofstream
&f) {
  unsigned char c = 0;

  int bitshift = 7;
  for (int pos = 0; pos < size; pos++) {
      c = c | (data[pos] << bitshift);
      bitshift--;
      if ((bitshift == -1) || (pos == size-1)) {
    f.put(c);
    bitshift = 7;
    c = 0;
      }
  }
}

/* membaca file pnm, mengabaikan komentar */
static void pnm_read(std::ifstream &file, char *buf) {
  char doc[BUF_SIZE];
  char c;

  file >> c;
  while (c == '#') {
    file.getline(doc, BUF_SIZE);
    file >> c;
  }
  file.putback(c);

  file.width(BUF_SIZE);
  file >> buf;
  file.ignore();
}

static image<uchar> *loadPBM(const char *name) {
  char buf[BUF_SIZE];

  /* membaca header */
  std::ifstream file(name, std::ios::in | std::ios::binary);
  pnm_read(file, buf);
  if (strncmp(buf, "P4", 2))
    throw pnm_error();

  pnm_read(file, buf);
  int width = atoi(buf);
  pnm_read(file, buf);
  int height = atoi(buf);

  /* membaca data */
  image<uchar> *im = new image<uchar>(width, height);
  for (int i = 0; i < height; i++)
    read_packed(imPtr(im, 0, i), width, file);

  return im;
}

static void savePBM(image<uchar> *im, const char *name) {
```

```
  int width = im->width();
  int height = im->height();
  std::ofstream file(name, std::ios::out | std::ios::binary);

  file << "P4\n" << width << " " << height << "\n";
  for (int i = 0; i < height; i++)
    write_packed(imPtr(im, 0, i), width, file);
}

static image<uchar> *loadPGM(const char *name) {
  char buf[BUF_SIZE];

  /* membaca header */
  std::ifstream file(name, std::ios::in | std::ios::binary);
  pnm_read(file, buf);
  if (strncmp(buf, "P5", 2))
    throw pnm_error();

  pnm_read(file, buf);
  int width = atoi(buf);
  pnm_read(file, buf);
  int height = atoi(buf);

  pnm_read(file, buf);
  if (atoi(buf) > UCHAR_MAX)
    throw pnm_error();

  /* membaca data */
  image<uchar> *im = new image<uchar>(width, height);
  file.read((char *)imPtr(im, 0, 0), width * height * sizeof(uchar));

  return im;
}

static void savePGM(image<uchar> *im, const char *name) {
  int width = im->width();
  int height = im->height();
  std::ofstream file(name, std::ios::out | std::ios::binary);

  file << "P5\n" << width << " " << height << "\n" << UCHAR_MAX <<
"\n";
  file.write((char *)imPtr(im, 0, 0), width * height *
sizeof(uchar));
}

static image<rgb> *loadPPM(const char *name) {
  char buf[BUF_SIZE], doc[BUF_SIZE];

  /* read header */
  std::ifstream file(name, std::ios::in | std::ios::binary);
  pnm_read(file, buf);
  if (strncmp(buf, "P6", 2))
    throw pnm_error();

  pnm_read(file, buf);
  int width = atoi(buf);
  pnm_read(file, buf);
```

```
  int height = atoi(buf);
  pnm_read(file, buf);
  if (atoi(buf) > UCHAR_MAX)
    throw pnm_error();

  /* membaca data */
  image<rgb> *im = new image<rgb>(width, height);
  file.read((char *)imPtr(im, 0, 0), width * height * sizeof(rgb));

  return im;
}

static void savePPM(image<rgb> *im, const char *name) {
  int width = im->width();
  int height = im->height();
  std::ofstream file(name, std::ios::out | std::ios::binary);

  file << "P6\n" << width << " " << height << "\n" << UCHAR_MAX <<
"\n";
  file.write((char *)imPtr(im, 0, 0), width * height * sizeof(rgb));
}

template <class T>
void load_image(image<T> **im, const char *name) {
  char buf[BUF_SIZE];

  /* membaca header */
  std::ifstream file(name, std::ios::in | std::ios::binary);
  pnm_read(file, buf);
  if (strncmp(buf, "VLIB", 9))
    throw pnm_error();

  pnm_read(file, buf);
  int width = atoi(buf);
  pnm_read(file, buf);
  int height = atoi(buf);

  /* membaca data */
  *im = new image<T>(width, height);
  file.read((char *)imPtr((*im), 0, 0), width * height * sizeof(T));
}

template <class T>
void save_image(image<T> *im, const char *name) {
  int width = im->width();
  int height = im->height();
  std::ofstream file(name, std::ios::out | std::ios::binary);

  file << "VLIB\n" << width << " " << height << "\n";
  file.write((char *)imPtr(im, 0, 0), width * height * sizeof(T));
}

#endif
```

----------(Imutil.h)----------

```
/* perlengkapan citra */

#ifndef IMUTIL_H
#define IMUTIL_H

#include "image.h"
#include "misc.h"

/* compute Hitung Nilai Minimum dan Maksimum Citra */
template <class T>
void min_max(image<T> *im, T *ret_min, T *ret_max) {
  int width = im->width();
  int height = im->height();

  T min = imRef(im, 0, 0);
  T max = imRef(im, 0, 0);
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      T val = imRef(im, x, y);
      if (min > val)
    min = val;
      if (max < val)
    max = val;
    }
  }

  *ret_min = min;
  *ret_max = max;
}

/* threshold citra */
template <class T>
image<uchar> *threshold(image<T> *src, int t) {
  int width = src->width();
  int height = src->height();
  image<uchar> *dst = new image<uchar>(width, height);

  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      imRef(dst, x, y) = (imRef(src, x, y) >= t);
    }
  }

  return dst;
}

#endif
```

----------(disjoint-set.h)----------

```
#ifndef DISJOINT_SET
#define DISJOINT_SET
```

```
typedef struct {
  int rank;
  int p;
  int size;
} uni_elt;

class universe {
public:
  universe(int elements);
  ~universe();
  int find(int x);
  void join(int x, int y);
  int size(int x) const { return elts[x].size; }
  int num_sets() const { return num; }

private:
  uni_elt *elts;
  int num;
};

universe::universe(int elements) {
  elts = new uni_elt[elements];
  num = elements;
  for (int i = 0; i < elements; i++) {
    elts[i].rank = 0;
    elts[i].size = 1;
    elts[i].p = i;
  }
}

universe::~universe() {
  delete [] elts;
}

int universe::find(int x) {
  int y = x;
  while (y != elts[y].p)
    y = elts[y].p;
  elts[x].p = y;
  return y;
}

void universe::join(int x, int y) {
  if (elts[x].rank > elts[y].rank) {
    elts[y].p = x;
    elts[x].size += elts[y].size;
  } else {
    elts[x].p = y;
    elts[y].size += elts[x].size;
    if (elts[x].rank == elts[y].rank)
      elts[y].rank++;
  }
  num--;
}

#endif
```