PERANCANGAN DAN PEMBUATAN FUZZY EXPERT SYSTEM UNTUK ANALISIS PENYAKIT KULIT PADA ANJING

Oleh:

Alexander Setiawan

Rolly Intan

Debora Indriati

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS KRISTEN PETRA

SURABAYA

2012

LAPORAN PENELITIAN
NO: 122/Pen/Informatika/I/2012

PERANCANGAN DAN PEMBUATAN FUZZY EXPERT SYSTEM UNTUK ANALISIS PENYAKIT KULIT PADA ANJING

Oleh:

Alexander Setiawan

Rolly Intan

Debora Indriati

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS KRISTEN PETRA

SURABAYA

2012

LEMBAR IDENTITAS DAN PENGESAHAN
LAPORAN HASIL PENELITIAN

1.	a. Judul Penelitian	:	**PERANCANGAN DAN PEMBUATAN FUZZY EXPERT SYSTEM UNTUK ANALISIS PENYAKIT KULIT PADA ANJING**
	b. Nomor Penelitian	:	122/Pen/Informatika/I/2012
	c. Jalur Penelitian	:	I / ~~II / III / IV~~
2.	Ketua Peneliti		
	a. Nama lengkap dan Gelar	:	Alexander Setiawan, M.T.
	b. Jenis Kelamin	:	Laki-laki
	c. Pangkat/Golongan/NIP	:	Pembina / IV-A / 04021
	d. Bidang Ilmu yang diteliti	:	Sistem Informasi
	e. Jabatan Akademik	:	Lektor
	f. Fakultas/Jurusan	:	Fakultas Teknologi Industri / Teknik Informatika
	g. Universitas	:	Universitas Kristen Petra
3.	Anggota Tim Peneliti (I)	:	
	a. Nama lengkap dan Gelar	:	Prof. Rolly Intan, Dr. Eng.
	b. Jenis Kelamin	:	Laki-laki
	c. Pangkat/Golongan/NIP	:	Pembina Utama / IV-E / 92008
	d. Bidang Ilmu yang diteliti	:	Soft Computing
	e. Jabatan Akademik	:	Profesor
	f. Fakultas/Jurusan	:	Fakultas Teknologi Industri / Teknik Informatika
	g. Universitas	:	Universitas Kristen Petra
	Anggota Tim Peneliti (II)	:	
	a. Nama lengkap dan Gelar	:	Debora Indriati, S.Kom
	b. Jenis Kelamin	:	Perempuan
	c. Pangkat/Golongan/NIP	:	-
	d. Bidang Ilmu yang diteliti	:	Sistem Informasi
	e. Jabatan Akademik	:	-
	f. Fakultas/Jurusan	:	Fakultas Teknologi Industri / Teknik Informatika
	g. Universitas	:	Universitas Kristen Petra
4.	Lokasi Penelitian	:	Surabaya
5.	Kerjasama dengan Instansi lain		
	Nama Instansi	:	-
	Alamat	:	-
6.	Tanggal Penelitian	:	Maret 2012 s/d September 2012
7.	Biaya	:	Rp. 5.000.000,-

Surabaya, 4 Oktober 2012

Mengetahui,
Ketua Jurusan

Yulia, M.Kom
NIP. 99-036

Ketua Peneliti

Alexander Setiawan, M.T.
NIP. 04-021

Menyetujui,
Dekan Fakultas Teknologi Industri

Djoni Haryadi Setiabudi, M.Eng
NIP. 85-009

ABSTRAK

Penyakit kulit pada anjing memiliki gejala yang dapat terlihat langsung pada tubuh. Namun demikian, terdapat tingkat kesulitan yang cukup untuk mencari informasi penyakit kulit pada anjing. Bahkan beberapa dari penyakit tersebut dapat menyebar pada manusia dan hewan lain, terutama kucing dan kelinci.

Pembuatan *fuzzy expert system* dimulai dengan analisa dan pengumpulan data tentang penyakit-penyakit kulit pada anjing. Hasil analisa ini kemudian melalui proses *filter* menjadi informasi yang siap digunakan dalam sistem pakar. Sistem pakar ini dibuat dengan menggunakan metode *forward chaining* dan analisa *history*, termasuk membangun suatu relasi antar penyakit menggunakan metode perhitungan *fuzzy*. Aplikasi sistem pakar ini menggunakan Microsoft Access 2003 untuk penggunaan basis data, dan Borland Delphi 7.0 untuk pembuatan program.

Dari hasil implementasi dan pengujian aplikasi sistem pakar ini, menunjukkan dapat memproses gejala-gejala yang berhubungan dengan penyakit, dan penyakit yang berhubungan dengan penyakit lainnya secara otomatis. Sehingga dapat menghasilkan informasi kemungkinan anjing tersebut terserang suatu penyakit, dengan nilai perbandingan yang tertinggi.

Kata Kunci :

Forward Chaining, *Fuzzy, History,* Sistem Pakar.

ABSTRACT

Skin diseases at dog has symptons that are visible at the body. However, there are difficulties to look for informations about skin diseases at dog. The disease can even contagious to human or another animals, especially cat and rabbit.

The fuzzy expert system development start with analysis and data gathering about skin diseases at dog. The result of analysis using filter process become information ready to be used for expert system. This expert system developed using forward chaining method and history analysis, including building relation between diseases using fuzzy calculation method. This expert system aplication is using Microsoft Access 2003 for database, and Borland Delphi 7.0 to develop it.

From implementation result and testing of this expert system aplication, shows that it can process symptons which are related with disease, and disease that has relation with other disease automatically. Resulting information of predictions about a dog infected with a disease, with high probabilities number.

Keywords :

Expert System, Forward Chaining, Fuzzy, History.

KATA PENGANTAR

Penulis mengucapkan syukur kepada Tuhan Yang Maha Esa atas terselesaikannya penelitian ini. Penulis sadar bahwa hasil penelitian ini masih jauh dari sempurna, karena itu penulis mengharapkan saran dan kritik yang membangun dari berbagai pihak demi perbaikan dari penelitian ini.

Penulis berharap semoga penelitian ini dapat memberikan kontribusi bagi perkembangan bidang sistem pakar pada umumnya.

Akhir kata, penulis mengucapkan terima kasih kepada semua pihak yang tidak dapat disebutkan satu persatu yang telah membantu terselesaikannya penelitian ini

Surabaya, Oktober 2012

Penyusun

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1. PENDAHULUAN

1.1 Latar Belakang

Sebuah *image* dapat memproyeksikan sebuah kenangan atau kreasi dari seseorang. Namun terkadang hasil dari *image* itu sendiri kurang memuaskan hingga menyebabkan seseorang untuk memanipulasinya. Untuk memanipulasi atau membuat sebuah *image* dari komputer dibutuhkan waktu yang cukup lama terutama apabila seseorang ingin membuat *image* ber-*texture* dengan menggunakan potongan *texture* yang berukuran lebih kecil dari ukuran *image*.

Dalam kehidupan sehari-hari sering kali ditemui gambar-gambar yang mempunyai sebuah pola yang biasa disebut dengan *texture*. *Texture-texture* tersebut dapat berupa seperti tanah, tanaman, batu-batuan, bulu, dan kulit. Untuk membuat sebuah *image* yang dipenuhi dengan *texture*, dapat dilakukan dengan cara menata *texture-texture* tersebut secara berurutan hingga memenuhi seluruh *image*. Hasil yang diperoleh dengan cara tersebut dapat memuaskan apabila *texture* yang digunakan mempunyai pola yang berulang. Namun apabila *texture* yang dimiliki tidak mempunyai pola yang berulang, maka akan terlihat potongan-potongan antar *texture*. Untuk menyelesaikan masalah diatas dapat digunakan suatu cara yang dinamakan *texture synthesis*. Texture synthesis dapat juga diaplikasikan untuk berbagai kegunaan, seperti mengembalikan *image texture* yang telah rusak atau hilang. Aplikasi pengembalian *image texture* yang telah rusak atau hilang biasa disebut sabagai *image inpainting*.

Texture synthesis adalah suatu cara untuk menghasilkan *image* ber-*texture* dengan pengolahan tepi antar *texture* hingga tidak terlihat bahwa *image* tersebut berasal dari penataan *texture*.

1.2 Permasalahan

Permasalahan yang dihadapi dan diharapkan dapat diselesaikan melalui penelitian ini adalah bagaimana menghasilkan *image* ber-*texture* dengan menggunakan metode *image quilting*. Permasalahan kedua yang akan diselesaikan

pada penelitian ini adalah bagaimana melakukan *image impainting* pada suatu citra.

1.3 Tujuan Penelitian

Tujuan dari penelitian ini adalah membuat aplikasi yang mengimplementasikan metode *image quilting* untuk membantu seseorang yang ingin membuat sebuah *image* dari potongan *texture* yang berukuran kecil.

1.4 Manfaat Penelitian

Hasil penelitian ini diharapkan dapat bermanfaat bagi perkembangan bidang pengolahan citra digital khususnya untuk menentukan melakukan *texture synthesis*.

1.5 Ruang Lingkup Pembahasan

Dalam penelitian ini terdapat batasan, yaitu:

- Pembuatan aplikasi yang dapat melakukan *texture synthesis* dan *image inpainting* pada sebuah *image*.
- Format untuk *image input* adalah format BMP (8-bit, 16-bit, 24-bit, atau 32-bit) atau JPEG.
- Metode *texture synthesis* yang digunakan adalah *image quilting*.
- Penyimpanan hasil *texture synthesis* sebagai *file image* dengan format BMP (8-bit, 16-bit, 24-bit, atau 32-bit) atau JPEG.
- Pembuatan aplikasi program dengan Borland Delphi 7.0.

1.6 Sistematika Penyusunan Laporan

Laporan penelitian ini secara keseluruhan terdiri dari lima bab dimana secara garis besar masing-masing bab membahas hal-hal sebagai berikut:

BAB 1 **Pendahuluan**: berisi latar belakang, permasalahan, tujuan penelitian, manfaat penelitian, ruang lingkup permasalahan, dan sistematika penyusunan laporan.

BAB 2 **Tinjauan Pustaka**: membahas tentang teori-teori dasar yang

relevan dan metode yang digunakan untuk memecahkan persoalan yang dibahas pada penelitian ini.

BAB 3 **Metode Penelitian**: membahas tentang metode penelitian yang dilakukan serta perancangan perangkat lunak.

BAB 4 **Hasil Penelitian dan Pembahasan**: berisi tentang hasil dari penelitian, berupa perangkat lunak yang telah dikembangkan beserta dengan pengujian perangkat lunak tersebut.

BAB 5 **Kesimpulan dan Saran**: berisi kesimpulan yang mencakup beberapa hal penting pada hasil yang didapat dari penelitian dan saran-saran yang diajukan bagi penyempurnaannya.

BAB 2. TINJAUAN PUSTAKA

Image processing adalah suatu metode yang digunakan untuk memproses atau memanipulasi gambar dalam bentuk 2 dimensi. *Image processing* dapat juga dikatakan segala operasi untuk memperbaiki, menganalisa, atau mengubah suatu gambar. Konsep dasar pengolahan suatu objek pada gambar menggunakan *image processing* diambil dari kemampuan indera penglihatan manusia yang selanjutnya dihubungkan dengan kemampuan otak manusia.

2.1 *Thresholding*

Thresholding digunakan untuk mengatur *Gray-level* yang ada pada gambar. Misalkan pada sebuah *gambar*, *f(x,y)* tersusun dari objek yang terang pada sebuah *background* yang gelap. *Gray-level* milik objek dan milik *background* terkumpul menjadi 2 grup yang dominan. Salah satu cara untuk mengambil objek dari backgroundnya adalah dengan memilih sebuah nilai *threshold T* yang memisahkan grup yang satu dengan grup yang lain. Maka semua *pixel* yang memiliki nilai > *T* disebut titik objek, yang lain disebut titik *background*. Proses ini disebut *thresholding*. Sebuah gambar yang telah di *threshold g*(*x,y*) dapat didefinisikan :

$$g(x,y) = \begin{cases} 1 & jika\ f(x,y) > T \\ 0 & jika\ f(x,y) \le T \end{cases} \quad \text{.....(1)}$$

Nilai *T* dapat ditentukan melalui perhitungan rata –rata dari keseluruhan nilai warna yang ada pada gambar. Pada perhitungan ini, nilai *T* yang didapat tetap disimpan dalam bilangan *real*. nilai *T* yang didapat untuk gambar yang memiliki *histogram* yang telah ter-*equalize* adalah berkisar antara 127 dan 128. Nilai maksimum dari *T* adalah nilai tertinggi dari sistem warna yang digunakan dan nilai minimum dari *T* adalah nilai terendah dari sistem warna yang digunakan. Untuk *256-graylevel* maka nilai tertinggi *T* adalah 255 dan nilai terendahnya adalah 0 [6].

2.2 *Grayscaling*

Proses awal yang banyak dilakukan dalam *image processing* adalah mengubah *image* berwarna menjadi gambar *gray-scale*, hal ini digunakan untuk menyederhanakan model *image*.

Gray-level adalah tingkat warna abu-abu dari sebuah *pixel.* Dengan kata lain nilai yang terkandung dalam *pixel* yang menunjukan tingkat keabu-abuan *pixel* tersebut dari hitam ke putih. Biasanya nilainya ditetapkan antara 0 hingga 255 (untuk 256-*graylevel*), dengan 0 adalah hitam dan 255 adalah putih. Karena hanya terbatas 1 *byte* saja maka untuk mempresentasikan nilai *pixel* cukup 8 bit saja. *Grayscale* adalah *image* yang dari tiap *pixel*-nya memiliki *gray-level* sebagai nilai.

Grayscaling adalah proses perubahan nilai *pixel* dari warna (RGB) menjadi *gray-level*. Pada dasarnya proses ini dilakukan dengan meratakan nilai *pixel* dari 3 nilai RGB menjadi 1 nilai. Untuk memperoleh hasil yang lebih baik, nilai *pixel* tidak langsung dibagi menjadi 3 melainkan terdapat persentasi dari masing-masing nilai. Salah satu persentasi yang sering digunakan adalah 29,9% dari warna merah (*Red*), 58,7% dari warna hijau (*Green*), dan 11,4% dari warna biru (*Blue*). Nilai *pixel* didapat dari jumlah persentasi 3 nilai tersebut [6].

2.3 *Histogram*

Histogram adalah grafik yang menunjukkan distribusi dari intensitas sebuah *image* (Sigit,2001). *Histogram* dari sebuah *image digital* berupa sebuah fungsi $h(r_k) = n_k$ dimana r_k adalah nilai warna ke-k dan n_k adalah jumlah *pixel* dalam gambar yang memiliki nilai tersebut. Pada *gray-level*, r_k adalah tingkat *gray-level* ke-k. k=0, 1, 2, …, L-1. L adalah batas maksimum nilai.

Misalkan diketahui data sebagai berikut:

X = 1 3 2 5 3 0 2 1 2 4 2 3

Maka histogramnya adalah munculnya setiap nilai, yaitu: nilai 0 muncul 1 kali, nilai 1 muncul 2 kali, nilai 2 muncul 4 kali, nilai 3 muncul 3 kali, nilai 4 muncul 1 kali dan nilai 5 muncul 1 kali. Karena *image* mempunyai *gray-level* 256 yaitu (0-255) maka *histogram* menyatakan jumlah kemunculan setiap nilai 0-255.

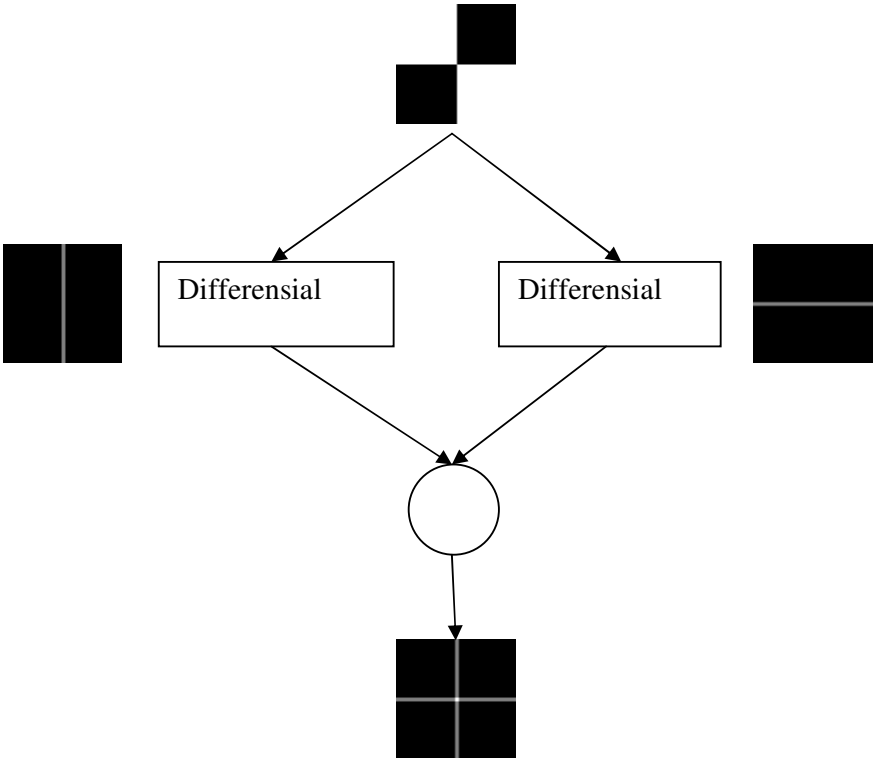
2.4 *Edge Detection*

Deteksi tepi (*Edge Detection*) pada suatu *image* adalah suatu proses yang menghasilkan tepi-tepi dari obyek *image*, tujuannya adalah :

- Untuk menandai bagian yang menjadi detail *image*
- Untuk memperbaiki detail dari citra yang kabur, yang terjadi karena error atau adanya efek dari proses akuisisi citra

Suatu titik (x,y) dikatakan sebagai tepi (*edge*) dari suatu citra bila titik tersebut mempunyai perbedaan yang tinggi dengan tetangganya [6].

Gambar 2.1 berikut ini menjelaskan bagaimana tepi suatu gambar diperoleh.



Gambar 2.1 Alur *Edge Detection*

Berdasarkan prinsip-prinsip filter pada citra maka tepi suatu gambar dapat diperoleh menggunakan *High Pass Filter* (HPF), yang mempunyai karakteristik:

$$\sum\sum H(x, y) = 0 \quad \text{.....(2)}$$

Contoh:

Diketahui fungsi citra f(x,y) sebagai berikut:

1	1	1	1	1
1	1	1	1	0
1	1	1	0	0
1	1	0	0	0
1	0	0	0	0

Dengan menggunakan filter : $H(x, y) = [-1,1]$

Maka hasil filter adalah :

0	0	0	0	1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	0

Beberapa metode untuk proses deteksi tepi ini antara lain :

1. Metode Sobel
2. Metode Prewitt

2.4.1 *Sobel Edge Detection*

Metode Sobel merupakan pengembangan metode Robert dengan menggunakan filter HPF yang diberi satu angka nol penyangga. Metode ini mengambil prinsip dari fungsi laplacian dan gaussian yang dikenal sebagai fungsi untuk mendapatkan nilai HPF. Kernel filter yang digunakan dalam metode Sobel ini adalah:

$$H = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad V = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

2.4.2 *Prewitt Edge Detection*

Metode Prewitt merupakan pengembangan metode Robert dengan menggunakan filter HPF yang diberi satu angka nol penyangga. Metode ini mengambil prinsip dari fungsi laplacian yang dikenal sebagai fungsi untuk mendapatkan nilai HPF.

Kernel filter yang digunakan dalam metode Prewitt ini adalah:

$$H = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad V = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

2.5 *Feather Blending*

Feather blending adalah suatu metode *blending* dengan memberikan bobot pada tiap *pixel image* yang akan di *blending* [1]. Besar bobot *blending* proposional dengan jarak dari tepi. Semakin jauh dari tepi maka bobot blending akan semakin mengecil. Algoritma dari *feather blending* ini dapat diuraikan sebagai berikut :

1. Dapatkan bobot untuk setiap *pixel* dari bagian *image* yang akan di-*blending*
2. Kemudian untuk setiap bobot yang ada bagikan dengan jumlah bobot yang ada sehingga bobot berada antara range 0 – 1.

   $$w_i' = w_i / (\sum_i w_i) \quad \text{.....(3)}$$

3. Kemudian lakukan *blending* dengan bobot yang telah didapatkan

   $$p_i' = (p_i \text{ x } w_i) + (p_j \text{ x } (1\text{-}w_i)) \quad \text{.....(4)}$$

   dimana :

   p_i' : *pixel* hasil

   p_i : *pixel* asal

   p_j : *pixel background*

   w_i : bobot *blending*

Contoh dari *feather blending* dapat dilihat pada Gambar 2.2.

Gambar 2.2 Contoh Hasil *Feather Blending*

2.6 *Sum of Square Differences*

Jika kita mempunyai *square* dengan ukuran m x m maka kita dapat mendefinisikan sebuah fungsi sebagai berikut [3]:

$$W_m(x,y)=\left\{u,v \middle| x-\frac{m}{2}\le u\le x+\frac{m}{2}, y-\frac{m}{2}\le v\le y+\frac{m}{2}\right\} \quad \text{.....(5)}$$

x, y diatas adalah koordinat tengah yang diinputkan dari *square*.

Dari fungsi diatas kita dapat mendapatkan perhitungan SSD sebagai berikut :

$$SSD=\sum_{(u,v)\in W_m(x,y)}\left|I_1(u,v)-I_2(u,v)\right| \quad \text{.....(6)}$$

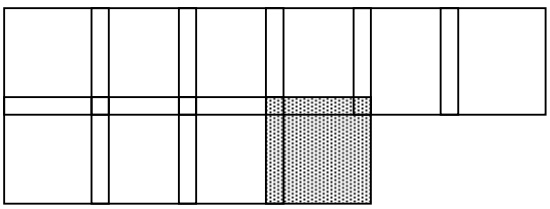
Maksud dari $\left|I_1(u,v)-I_2(u,v)\right|$ adalah untuk mendapatkan *error surface* dari Image I_1 dan Image I_2. Untuk sebuah *image*, *error surface* dapat didapatkan dengan menghitung kemiripan informasi warnanya. Penghitungan selisih warna *pixel* dapat dihitung dengan menggunakan rumus :

$$d=\sqrt{(r_2-r_1)^2+(g_2-g_1)^2+(b_2-b_1)^2} \quad \text{.....(7)}$$

2.7 *Image Quilting*

Image Quilting adalah salah satu metode dalam melakukan *texture synthesis*. Metode *Image Quilting* diperkenalkan pertama kali oleh Efros and Freeman [5]. Secara garis besar proses *image quilting* dapat dikatakan sebagai proses untuk menata potongan – potongan *image* yang disebut sebagai *patch* sehingga dapat terlihat bahwa image tidak berasal dari penataan *patch.*

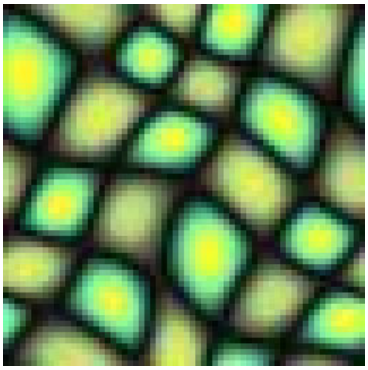
Image Quilting menggunakan *patch based texture synthesis*. Dengan menggunakan *patch based* maka waktu untuk proses dapat berjalan lebih cepat dengan melakukan *synthesis* dengan memindahkan *patch* demi *patch*. Dalam *image quilting patch* di *synthesis* dengan *raster order*.



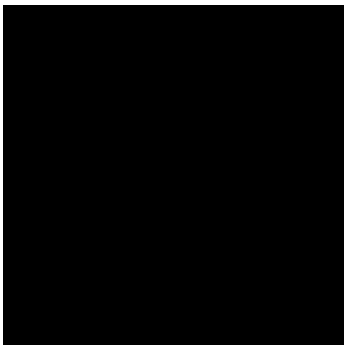
Gambar 2.3 *Synthesis Patch* dengan *Raster Order*

Hasil dari *image quilting* amat ditentukan oleh *pixel – pixel neighborhood.* Besar *neighborhood* yang akan ditelusuri pada *image quilting* dinamakan sebagai *overlap size*. *Image Quilting* menggunakan *Sum of Squared Difference* (SSD) untuk mencari *patch* yang sesuai dari *input texture* dengan *neighborhood* di sekitar *patch* yang akan di *synthesis*.

Dengan menggunakan metode *image quilting*, langkah yang harus dilakukan adalah mengambil contoh *texture* sebagai *image input*. Kemudian akan dilakukan *texture synthesis* pada *image input* sehingga menjadi *image output*. Contoh *image input* dapat dilihat pada Gambar 2.4. dan contoh *image output* awal dapat dilihat pada Gambar 2.5.

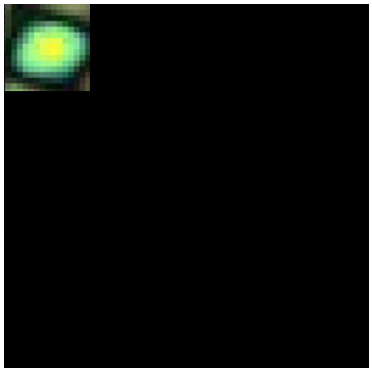


Gambar 2.4 Contoh *Image Input*



Gambar 2.5 Contoh *Image Output* Awal

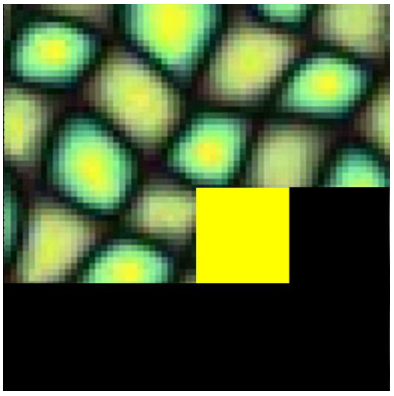
Dilakukan pengambilan suatu potongan dari *image input* secara random dengan ukuran sesuai dengan konstanta *neighborhood region*. Kemudian potongan tersebut diletakkan pada kiri atas dari *image output*. Proses ini dapat dilihat pada Gambar 2.6.



Gambar 2.6 Penempatan Potongan Pertama

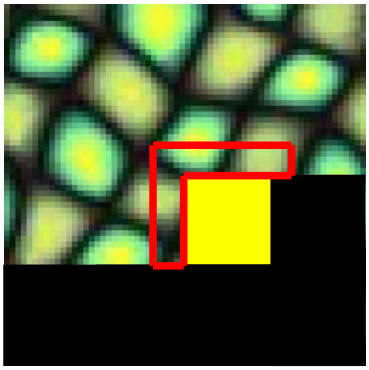
Untuk *step* selanjutnya hingga *image output* terpenuhi, dilakukan proses dengan menggunakan metode *image quilting*. Berikut adalah metode *image quilting* pada *step* ke-n

- Mencari *patch* yang akan di *synthesis*, *step* ini dapat dilihat pada Gambar 2.7.



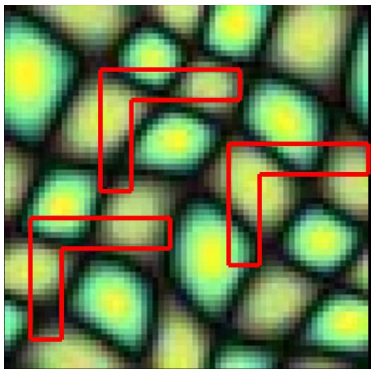
Gambar 2.7 Mencari *Patch* yang Akan di-*synthesis*

- Membangun *neighborhood region*, *step* ini dapat dilihat pada Gambar 2.8.



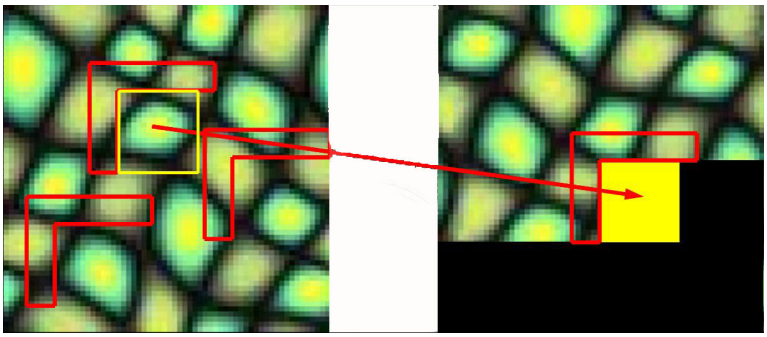
Gambar 2.8 Membangun *Neighborhood Region*

- Menemukan *neighborhood-neighborhood* yang sesuai dari *image input* dengan membandingkan statistik grafik *histogram* antara *neighborhood* dari *image input* dengan *neighborhood* dari *image output*, *step* ini dapat dilihat pada Gambar 2.9.



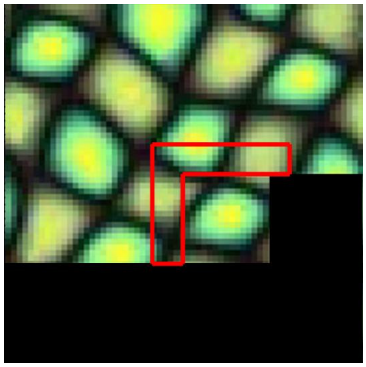
Gambar 2.9 Menemukan *Neighborhood* Sesuai dengan *Image Input*

- Mengambil *neighborhood* terbaik dari semua kandidat *neighborhood* yang ada pada *image input*, *step* ini dapat dilihat pada Gambar 2.10.



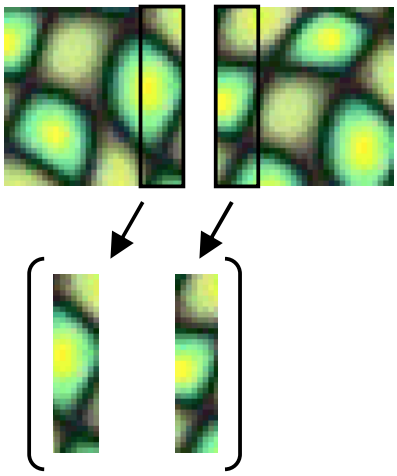
Gambar 2.10 Mengambil *Neighborhood* Terbaik

- Meletakkan *patch* dari *image input* ke *image output*, *step* ini dapat dilihat pada Gambar 2.11.

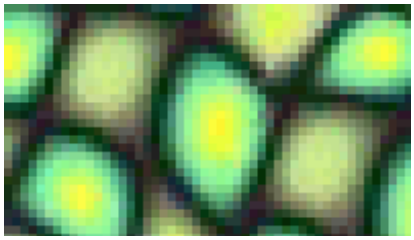


Gambar 2.11 Meletakkan *Patch* dari *Image Input* ke *Image Output*

- Melakukan perbaikan pada *overlap* dengan menggunakan *feather blending*. *Feather blendin*g dilakukan mulai dari tepi *patch* hingga besar konstanta dari *overlap*. Proses dari *feather blending* dimulai dengan meng-*generate amount feather blending* untuk tiap *pixel* [1].
  Contoh proses dari *feather blending* dapat dilihat pada Gambar 2.12. dan hasil dari proses *feather blending* dapat dilihat pada Gambar 2.13.

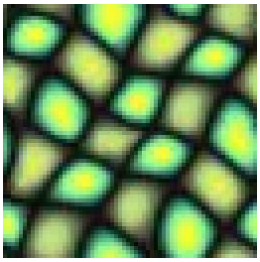


Gambar 2.12 Pengambilan *Overlap* yang Akan di *Feather Blending*



Gambar 2.13 Hasil Proses *Feather Blending*

- Proses untuk *patch* ini selesai, dan diulang hingga *image output* terpenuhi dengan *texture* dari *image input*. Hasil *image output* dapat dilihat pada Gambar 2.14.



Gambar 2.14 Hasil Akhir *Image Output*

BAB 3. METODE PENELITIAN

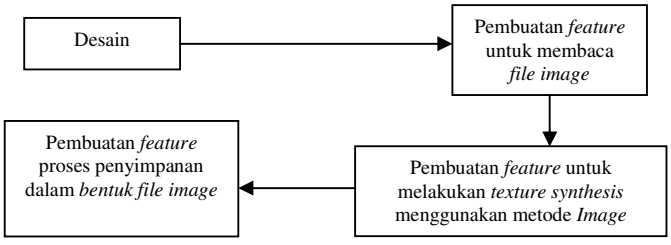
3.1 Metodologi Penelitian

Metodologi dan langkah-langkah yang digunakan untuk pengembangan penelitian ini adalah sebagai berikut:

1. Studi literatur:
   - *Texture synthesis* dengan metode *image quilting*.
   - Pencarian *patch* terbaik dengan metode SSD (*Sum of Square Differences*).
   - Pengolahan *overlap* dengan metode *feather blending*.
   - Pengaplikasian *image quilting* untuk proses *image inpainting*.
   - Pengimplementasiannya dalam Delphi7.
2. Desain *interface* beserta alur program.
3. Pengimplementasian dari studi literatur dalam *Delphi 7* menjadi sebuah perangkat lunak.
4. Pengujian dan analisis perangkat lunak:
   - Pengujian program yang telah dibuat.
   - Analisis hasil dari proses *texture synthesis*.
5. Penarikan kesimpulan.
6. Pembuatan laporan.

3.2 Perencanaan Sistem

Perangkat lunak ini adalah perangkat lunak yang dibuat khusus untuk melakukan *texture synthesis* dari sebuah *file image* yang kemudian hasil proses *texture synthesis* dapat disimpan kembali dalam bentuk *file image*. Untuk itu diperlukan rancangan dalam pembuatan perangkat lunak. Langkah-langkah pembuatan perangkat lunak digambarkan seperti pada Gambar 3.1.



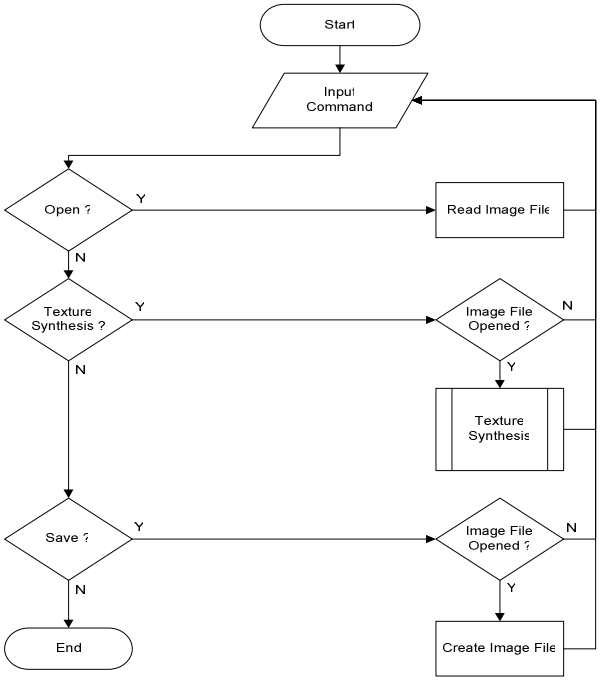
Gambar 3.1 Diagram Langkah-Langkah Pembuatan Perangkat Lunak

Keterangan :

- Desain *interface* merupakan langkah yang pertama kali dilakukan dalam pembuatan perangkat lunak ini. Desain *interface* dari perangkat lunak, dibuat semenarik dan semudah mungkin untuk digunakan oleh *user*.
- Pembuatan *feature* untuk membaca *file image* merupakan langkah untuk pembuatan proses pembacaan *file image* baik dalam *format* bmp (8-bit, 16-bit, 24-bit, 32-bit) dan jpg.
- Pembuatan *feature* untuk melakukan proses *texture synthesis* menggunakan metode *Image Quilting* merupakan langkah untuk melakukan *texture synthesis* dari sebuah *image file* yang telah dibuka dengan menggunakan metode *Image Quilting* yaitu dengan menentukan potongan-potongan yang terbaik dari *image file* yang telah dibuka yang kemudian di *generate* menjadi suatu *image file* dimana besar *image* yang dihasilkan sesuai dengan *input*-an dari *user*.
- Pembuatan *feature* proses penyimpanan dalam bentuk *file image* merupakan langkah untuk melakukan proses pembuatan *image file* baik dalam *format* bmp (8-bit, 16-bit, 24-bit, 32-bit) dan jpg dari *image* yang berada dalam kondisi *actived window* pada perangkat lunak untuk kemudian disimpan pada alamat yang dikehendaki.

3.3 Perancangan Program

Secara garis besar alur program digambarkan seperti pada Gambar 3.2.



Gambar 3.2 Diagram Alir Aplikasi Secara Garis Besar

Keterangan :

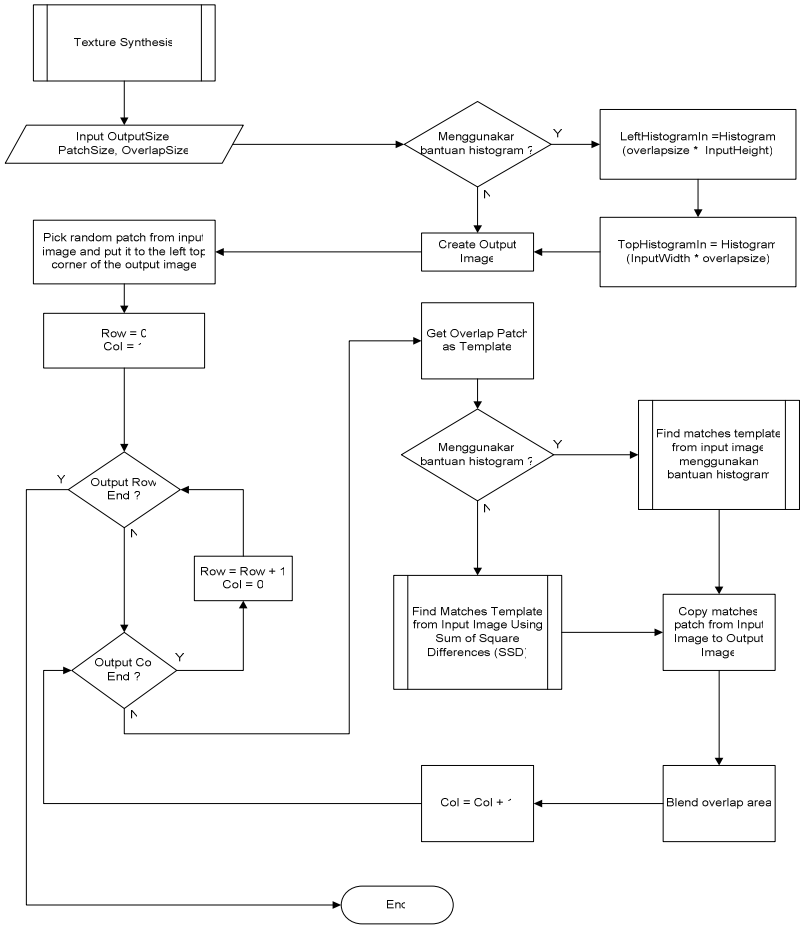
- *Input command* adalah *input* perintah dari *user* kepada aplikasi atau dengan kata lain pemilihan menu-menu yang terdapat pada perangkat lunak.
- Setelah command di-*input*-kan oleh *user*, maka dilakukan pengecekan apakah *command* merupakan proses *open*. Apabila ya, maka dilakukan proses membaca *image* dari sebuah *image file* di mana *user* dapat mencari file gambar yang hendak diproses.
- Apabila *command* bukan merupakan proses *open*, maka dilakukan pengecekkan apakah *command* merupakan proses *texture synthesis*. Apabila ya, maka dilakukan pengecekkan apakah sudah ada *image* yang telah terbuka. Apabila ya, maka dilakukan proses *texture synthesis* dari *image* dengan kondisi *actived window*. Apabila tidak, maka tidak ada proses yang dilakukan dan perangkat lunak kembali pada *state* menunggu *input*-an *command* dari *user.*
- Apabila *command* bukan merupakan proses *texture synthesis*, maka dilakukan pengecekkan apakah *command* merupakan proses *save*. Apabila ya, maka dilakukan pengecekkan apakah ada dalam kondisi *actived*

window. Apabila ya, maka dilakukan proses pembuatan *image file* dari *image* dengan kondisi *actived window*. Apabila tidak, maka tidak ada proses yang dilakukan dan perangkat lunak kembali pada *state* menunggu *input*-an *command* dari *user.*

- Apabila *command* bukan merupakan proses *save*, maka perangkat lunak dihentikan dan kembali ke *operating system.*

3.3.1 Algoritma *Feature Texture Synthesis*

Proses *texture synthesis* ini merupakan kumpulan dari beberapa proses yang digunakan untuk dapat menghasilkan *file image* ber-t*exture* dengan pengolahan tepi antar *texture* sehingga tidak terlihat bahwa *image* tersebut berasal dari penataan *texture*. Algoritma dari proses *texture synthesis* akan dijelaskan pada Gambar 3.3.



Gambar 3.3 Diagram Alir Proses *Texture Synthesis*

Keterangan :

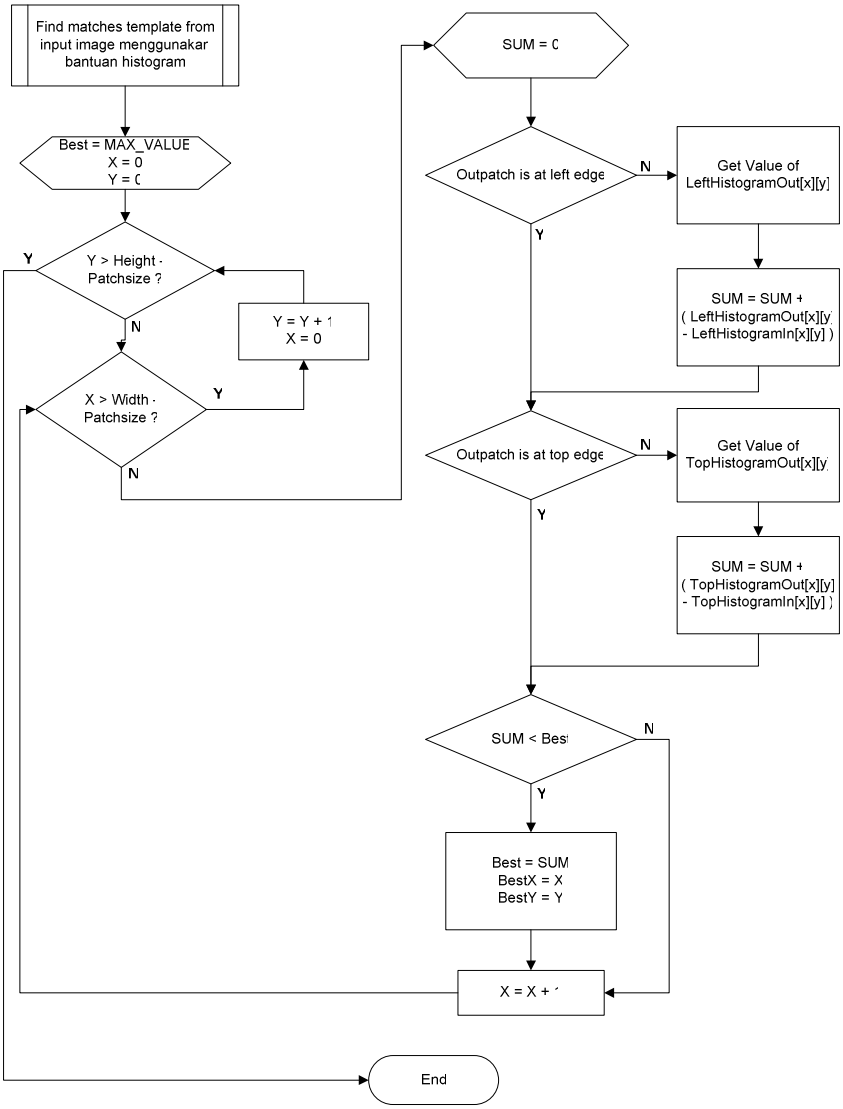
- Proses menunggu *input*-an dari *user* berupa *output size* yang merupakan besar dari *image* hasil, *overlap size* yang merupakan besar area yang akan digunakan untuk mencari *patch* yang terbaik dimana nantinya akan dilakukan proses *blending* sebesar *overlap size* tersebut. dan *patch size* yang merupakan besar *image template* yang nantinya akan dimasukkan pada *image* hasil.
- Proses pembuatan *output image* kosong di mana *image* tersebut nantinya akan diisi dengan *image* hasil proses *texture synthesis.*
- Dilakukan pengecekkan apakah menggunakan *histogram equlization* apabila ya maka dilakukan proses untuk mendapatkan nilai *lefthistogram* dan *tophistogram.* Kemudian dilanjutkan dengan proses pembuatan *image* hasil. Apabila tidak, maka proses langsung pada pembuatan *image* hasil.
- Ambil sebuah *random patch* dengan ukuran yang di-*input* kan oleh user dari *input image* kemudian diletakkan di sebelah kiri atas dari *output image.*
- Inisialisasi *Row* =0 dan *Col*=1. Lalu dilakukan pengecekkan apakah kondisi *Row* sekarang ini sudah melebihi ukuran *output image.* Apabila ya, maka proses akan berakhir. Apabila tidak, maka akan dilanjutkkan dengan pengecekkan apakah kondisi *Col* sekarang ini sudah melebihi ukuran *output image.* Jika ya, maka dilakukan perubahan nilai pada variabel *Row* yaitu *Row*=*Row*+1 dan pada variabel *Col* yaitu *Col*=0. Lalu kembali pada proses pengecekkan kondisi *Row.* Jika tidak, maka proses berlanjut pada proses *Get overlap patch as template* (pengambilan *overlap patch* sesuai dengan ukuran *patch* yang sudah di-*input* kan dari bagian sebelah kanan *patch* yang sudah dimasukkan ke dalam output image).
- Dilakukan pengecekkan apakah menggunakan bantuan *histogram.* Apabila ya, maka dilakukan proses *Find matches template from input image* (pencarian *patch* yang paling mendekati *overlap patch* dengan membandingkan bagian sebelah kiri *patch-patch* pada *input image* dengan *overlap patch*) dengan menggunakan metode bantuan *histogram.* Apabila tidak, maka dilakukan proses *Find matches template from input image* (pencarian *patch* yang paling mendekati *overlap patch* dengan membandingkan bagian sebelah kiri *patch-*

patch pada *input image* dengan *overlap patch*) dengan menggunakan metode *Sum of Square Differences* (SSD).

- Kemudian hasil *patch* yang didapatkan dari proses *find match* dimasukkan ke dalam *output image* sebesar *patch size* tersebut untuk kemudian dilakukan proses *Blend overlap area* (mencampur overlap patch dengan bagian sebelah kiri patch yang baru dimasukkan sehingga perpotongan gambar tidak terlihat). Setelah proses ini selesai maka dilakukan perubahan nilai *Col* yaitu *Col*=*Col*+1 lalu kembali ke pengecekkan kondisi *Col* saat ini.
- Proses berakhir jika semua area *output image* sudah dipenuhi oleh *image* hasil proses *texture synthesis* atau dengan kata lain kondisi *output Row*=e*nd* dan kondisi *Col*=e*nd*.

3.3.1.1 Algoritma *Feature Find Matches Template From Input Image* Menggunakan Bantuan *Histogram*

Proses *find matches template from input image* menggunakan bantuan *histogram* ini merupakan kumpulan dari beberapa proses yang digunakan mencari *patch* yang paling mendekati *overlap patch* dengan menggunakan *histogam equalization*. Algoritma dari proses *find matches template from input image* menggunakan bantuan *histogram* akan dijelaskan pada Gambar 3.4.



Gambar 3.4 Diagram Alir Proses *Find Matches Template From Input Image* Menggunakan Bantuan *Histogram*

Keterangan :

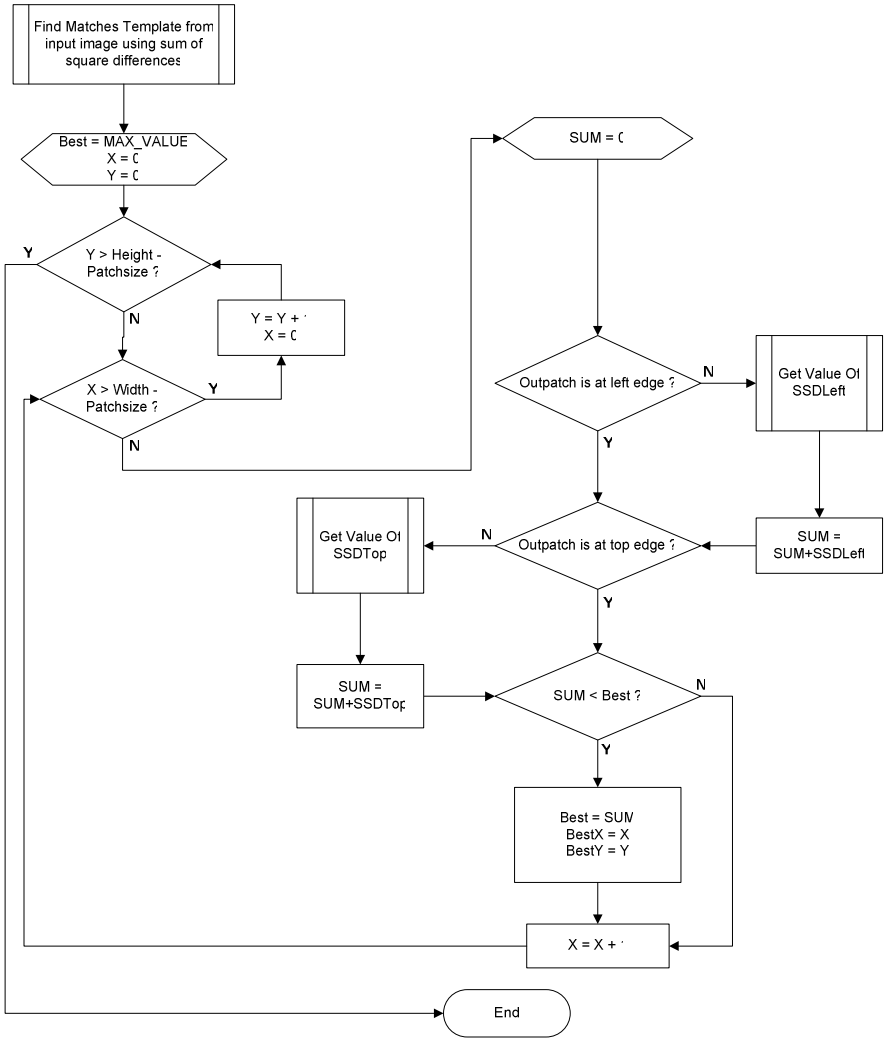
- Inisialisasi variabel Best=MAX_VALUE, X=0, dan Y=0. Di mana MAX_VALUE adalah nilai konstanta terbesar yang dapat ditampung oleh variabel Best.
- Kemudian dilakukan pengecekkan apakah variabel Y sekarang ini sudah melebihi batas ukuran tinggi dari *input image*. Jika ya, maka proses berakhir. Jika tidak, maka dilakukan pengecekkan apakah variabel X sekarang ini sudah melebihi batas ukuran lebar dari *input image*. Jika ya, maka dilakukan perubahan nilai variabel Y yaitu Y=Y+1 dan nilai variabel X=0 (menurunkan

posisi *pixel* ke bawah sejauh 1*pixel* dan menggesernya ke batas kiri *input image*). Jika tidak, maka dilakukan inisialisasi nilai variabel SUM=0.

- Setelah itu, dilakukan pengecekan apakah *outpatch* (*patch* yang akan dicari) berada pada batas kiri dari *output image*. Jika ya, maka dilakukan pengecekan apakah *outpatch* berada pada batas atas dari *output image*. Jika tidak, maka dilakukan perubahan nilai variabel SUM yaitu SUM=SUM+ (LeftHistogramOut [x] [y] – LeftHistogramIn [x] [y]). Jika *outpatch* berada pada batas atas dari *output image* maka dilakukan pengecekan apakah nilai variabel SUM lebih kecil dari nilai variabel Best. Jika *outpatch* tidak berada pada batas atas dari *output image* maka dilakukan perubahan nilai variabel SUM yaitu SUM=SUM+ (TopHistogramOut [x] [y] – TopHistogramIn [x] [y]). Jika nilai variabel SUM lebih kecil dari nilai variabel Best maka dilakukan perubahan nilai variabel Best yaitu Best=SUM, nilai variabel BestX=X, dan nilai variabel BestY=Y. Setelah perubahan nilai tersebut atau jika nilai variabel SUM lebih besar dari nilai variabel Best maka dilakukan perubahan nilai cariabel X yaitu X=X+1 (menggeser posisi *pixel* pada input image sebesar 1 *pixel* ke kanan).
- Kemudian proses dikembalikan kepada pengecekan apakah variabel X sudah melebihi batas ukuran lebar dari *input image.*
- Proses *find matches template from input image* ini berakhir jika posisi variabel Y sudah melampaui batas ukuran tinggi pada *input image* atau dengan kata lain sudah dilakukan perhitungan bantuan *histogram* terhadap seluruh bagian gambar dari *input image*.

3.3.1.2 Algoritma *Feature Find Matches Template From Input Image* Menggunakan *Sum of Square Differences* (SSD)

Proses *find matches template from input image using sum of square differences* (SSD) ini merupakan kumpulan dari beberapa proses yang digunakan mencari *patch* yang paling mendekati *overlap patch using sum of square differences* (SSD). Algoritma dari proses *find matches template from input image patch using sum of square differences* (SSD) ini akan dijelaskan pada Gambar 3.5.



Gambar 3.5 Diagram alir Proses *Find Matches Template From Input Image Using Sum of Square Differences* (SSD)

Keterangan :

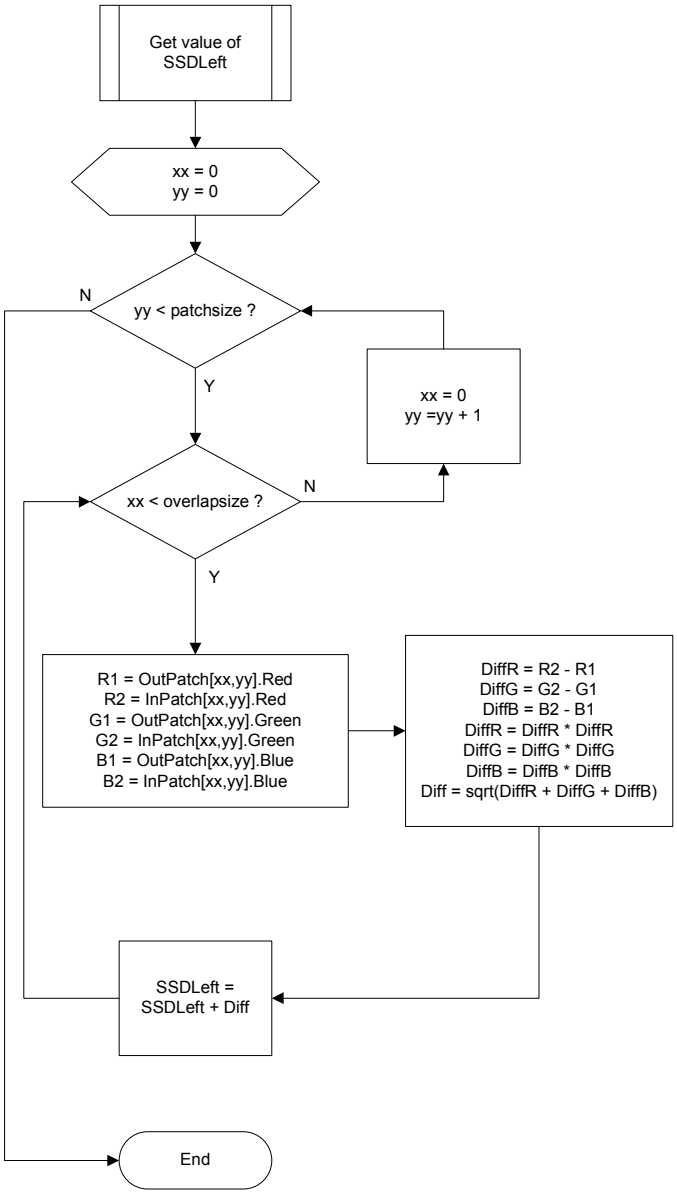
- Inisialisasi variabel Best=MAX_VALUE, X=0, dan Y=0. Di mana MAX_VALUE adalah nilai konstanta terbesar yang dapat ditampung oleh variabel Best.
- Kemudian dilakukan pengecekkan apakah variabel Y sekarang ini sudah melebihi batas ukuran tinggi dari *input image*. Jika ya, maka proses berakhir. Jika tidak, maka dilakukan pengecekkan apakah variabel X sekarang ini sudah melebihi batas ukuran lebar dari *input image*. Jika ya, maka dilakukan perubahan nilai variabel Y yaitu Y=Y+1 dan nilai variabel X=0 (menurunkan

posisi *pixel* ke bawah sejauh 1*pixel* dan menggesernya ke batas kiri *input image*). Jika tidak, maka dilakukan inisialisasi nilai variabel SUM=0.

- Setelah itu, dilakukan pengecekan apakah *outpatch* (*patch* yang akan dicari) berada pada batas kiri dari *output image*. Jika ya, maka dilakukan pengecekan apakah *outpatch* berada pada batas atas dari *output image*. Jika tidak, maka dilakukan proses untuk mendapatkan nilai SSDLeft kemudian dilanjutkan dengan proses perubahan nilai variabel SUM yaitu SUM = SUM + SSDLeft. Jika *outpatch* berada pada batas atas dari *output image* maka dilakukan pengecekan apakah nilai variabel SUM lebih kecil dari nilai variabel Best. Jika *outpatch* tidak berada pada batas atas dari *output image* , maka dilakukan proses untuk mendapatkan nilai SSDTop kemudian dilanjutkan dengan proses perubahan nilai variabel SUM yaitu SUM = SUM + SSDTop. Jika nilai variabel SUM lebih kecil dari nilai variabel Best maka dilakukan perubahan nilai variabel Best yaitu Best=SUM, nilai variabel BestX=X, dan nilai variabel BestY=Y. Setelah perubahan nilai tersebut atau jika nilai variabel SUM lebih besar dari nilai variabel Best maka dilakukan perubahan nilai cariabel X yaitu X=X+1 (menggeser posisi *pixel* pada input image sebesar 1 *pixel* ke kanan).
- Kemudian proses dikembalikan kepada pengecekan apakah variabel X sudah melebihi batas ukuran lebar dari *input image.*
- Proses *find matches template from input image* ini berakhir jika posisi variabel Y sudah melampaui batas ukuran tinggi pada *input image* atau dengan kata lain sudah dilakukan perhitungan SSD terhadap seluruh bagian gambar dari *input image*.

3.3.1.3 Algoritma Proses *Get Value of SSDLeft*

Proses *Get Value of SSDLeft* merupakan proses yang digunakan untuk mendapatkan nilai jumlah SSD sebelah kiri pada *overlap*. Keseluruhan dari proses ini dapat dilihat pada Gambar 3.6.



Gambar 3.6 Diagram Alir Proses *Get Value of SSDLeft*

Keterangan :

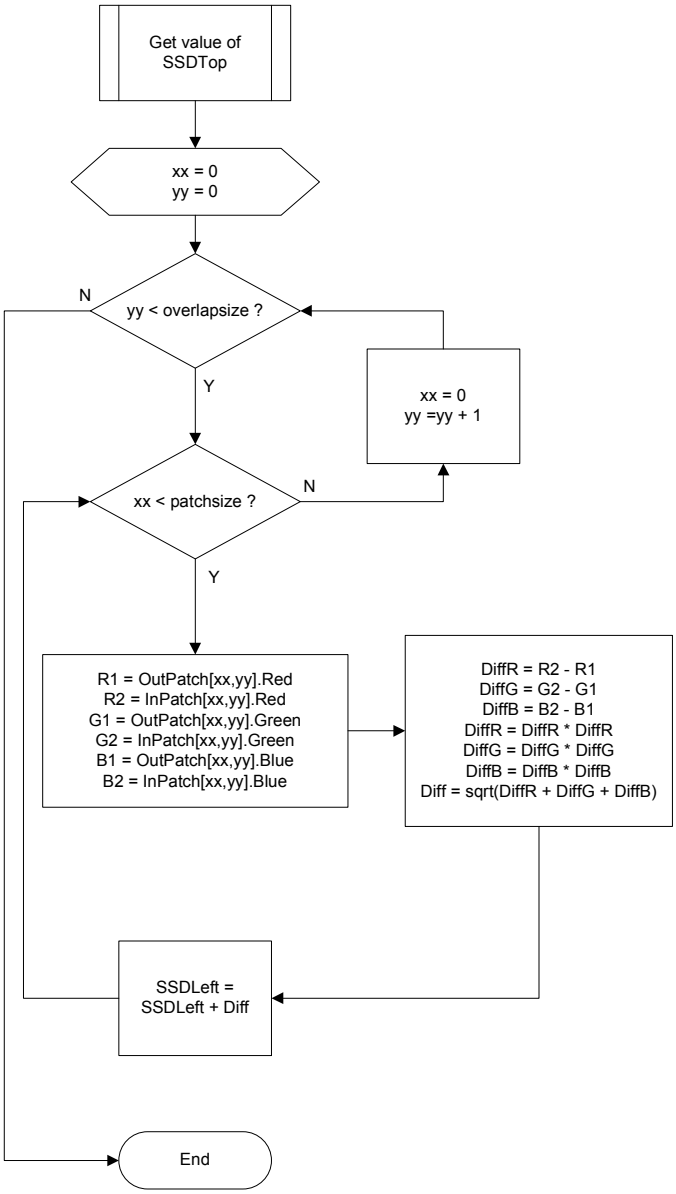
- Proses dimulai dengan melakukan inisialisasi nilai xx dan yy.
- Dilanjutkan dengan pengecekkan apakah nilai yy lebih kecil dari *patchsize.* Jika ya, maka dilakukan pengecekkan apakah xx lebih kecil dari *overlapsize* apabila tidak maka proses kembali pada pengecekkan apakah yy lebih kecil dari *patchsize*. Apabila ya, maka dilakukan proses untuk memasukkan nilai R1 dengan outpatch[xx,yy].red, R2 dengan inpatch[xx,yy].red, G1 dengan

outpatch[xx,yy].green, G2 dengan inpatch[xx,yy].green, B1 dengan outpatch[xx,yy].blue, B2 dengan inpatch[xx,yy].blue.

- Proses dilanjutkan dengan memasukkan nilai DiffR dengan R1-R2, DiffG dengan G1-G2, DiffB dengan B1-B2 dilanjutkan dengan mengubah nilai DiffR dengan DiffR*DiffR, DiffG dengan DiffG*DiffG, DiffB dengan DiffB*DiffB. Kemudian memasukkan nilai Diff dengan melakukan menjumlah DiffR, DiffG, DiffB yang kemudian di akar kuadratkan.
- Proses dilanjutkan dengan memasukkan nilai SSDLeft dengan menjumlah nilai SSDLeft dengan nilai Diff. Kemudian proses kembali pada pengecekkan apakah xx lebih kecil dari *overlapsize*.
- Pada pengecekkan apakah nilai yy lebih kecil dari *patchsize* bernilai tidak, maka proses *Get Value of SSDLeft* selesai.

3.3.1.4 Algoritma Proses *Get Value of SSDLeft*

Proses *Get Value of SSDLeft* merupakan proses yang digunakan untuk mendapatkan nilai jumlah SSD sebelah kiri pada *overlap*. Keseluruhan dari proses ini dapat dilihat pada Gambar 3.7.



Gambar 3.7 Diagram Alir Proses *Get Value of SSDTop*

Keterangan :

- Proses dimulai dengan melakukan inisialisasi nilai xx dan yy.
- Dilanjutkan dengan pengecekkan apakah nilai yy lebih kecil dari *overlapsize.* Jika ya, maka dilakukan pengecekkan apakah xx lebih kecil dari *patchsize* apabila tidak maka proses kembali pada pengecekkan apakah yy lebih kecil dari *overlapsize*. Apabila ya, maka dilakukan proses untuk memasukkan nilai R1 dengan outpatch[xx,yy].red, R2 dengan inpatch[xx,yy].red, G1 dengan

outpatch[xx,yy].green, G2 dengan inpatch[xx,yy].green, B1 dengan outpatch[xx,yy].blue, B2 dengan inpatch[xx,yy].blue.

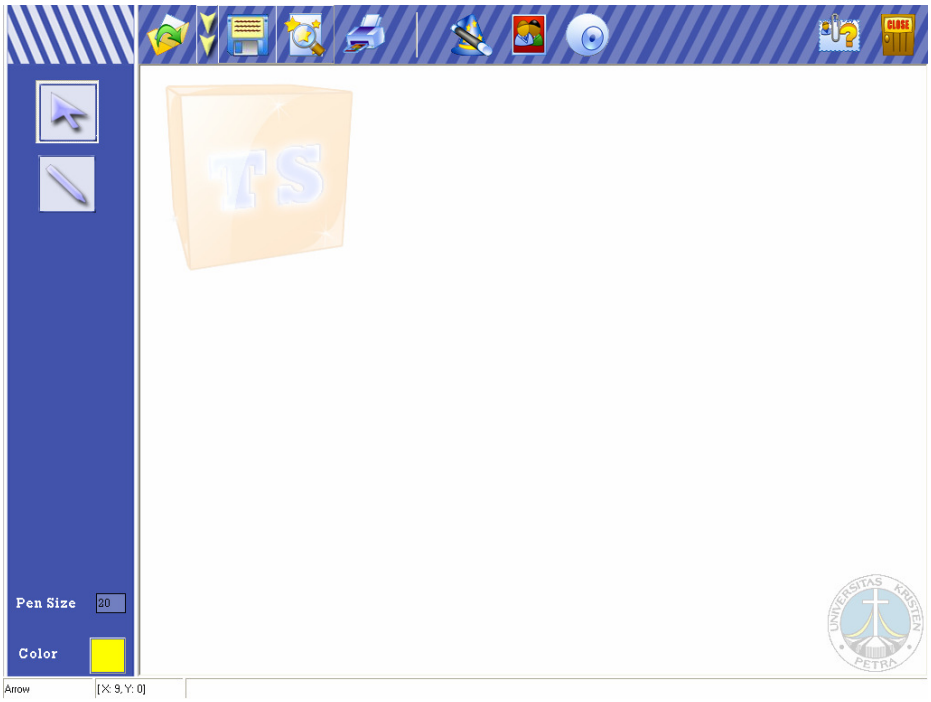
- Proses dilanjutkan dengan memasukkan nilai DiffR dengan R1-R2, DiffG dengan G1-G2, DiffB dengan B1-B2 dilanjutkan dengan mengubah nilai DiffR dengan DiffR*DiffR, DiffG dengan DiffG*DiffG, DiffB dengan DiffB*DiffB. Kemudian memasukkan nilai Diff dengan melakukan menjumlah DiffR, DiffG, DiffB yang kemudian di akar kuadratkan.
- Proses dilanjutkan dengan memasukkan nilai SSDLeft dengan menjumlah nilai SSDLeft dengan nilai Diff. Kemudian proses kembali pada pengecekkan apakah xx lebih kecil dari *patchsize*.
- Pada pengecekkan apakah nilai yy lebih kecil dari *overlapsize* bernilai tidak, maka proses *Get Value of SSDLeft* selesai.

BAB 4. HASIL PENELITIAN DAN PEMBAHASAN

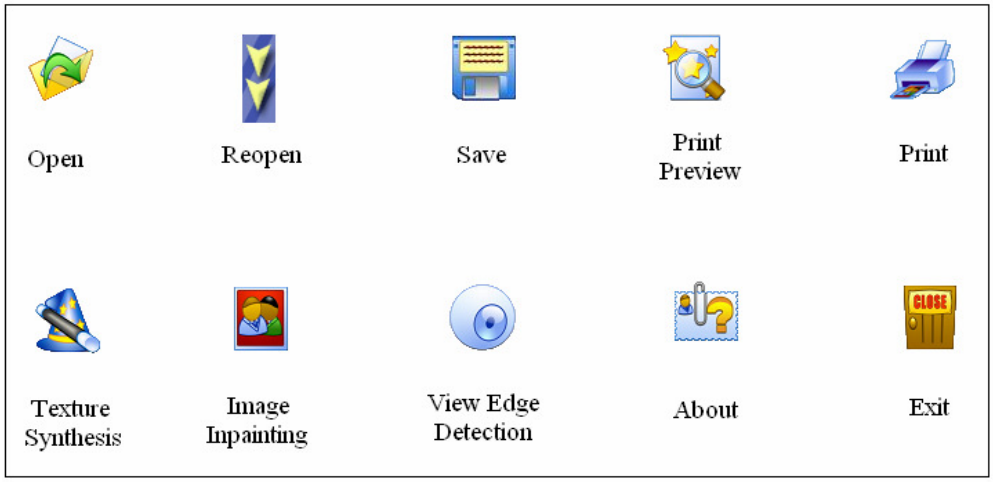
Pada bab ini akan dijelaskan tentang implementasi serta pengujian sistem perangkat lunak yang dibuat pada penelitian ini. Implementasi aplikasi dilakukan dengan menggunakan bantuan bahasa pemrograman Delphi 7.0 [2,3]. Pengujian dilakukan dengan menggunakan komputer dengan spesifikasi sebagai berikut :

- *Processor Intel Pentium Dual Core 2.66 GHz*
- *Memory 512 DDR2 RAM*
- *VGA Card GeForce7300*
- Sistem operasi *Microsoft Windows XP Service Pack 2*

Saat perangkat lunak dijalankan pertama masuk pada *form* utama yang berisi menu dan tombol-tombol untuk melakukan proses yang diinginkan oleh *user*. Gambar tampilan *form* utama dapat dilihat pada Gambar 4.1 sedangkan tampilan dari menu-menu yang ada dapat dilihat pada Gambar 4.2



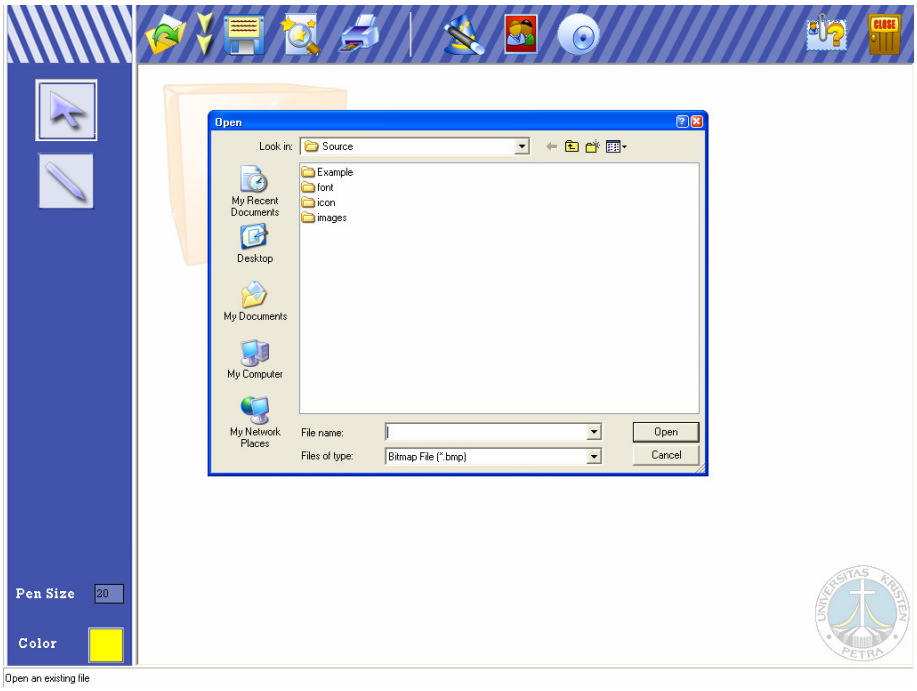
Gambar 4.1 Tampilan Pertama Perangkat Lunak



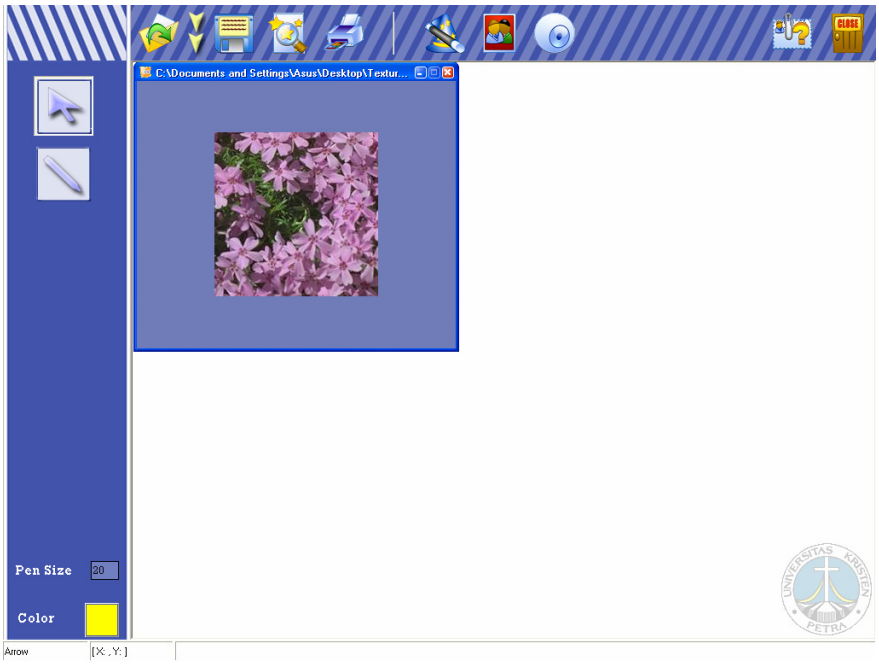
Gambar 4.2 Tampilan Menu

4.1 Proses Open File Image

Untuk membuka sebuah *file image*, *user* dapat memilih menu *open*. Gambar yang mewakili menu pilihan ini dapat dilihat pada Gambar 4.2. Tampilan *form open file* dapat dilihat pada Gambar 4.3 sedangkan tampilan *form* setelah proses *open file* dapat dilihat pada Gambar 4.4.



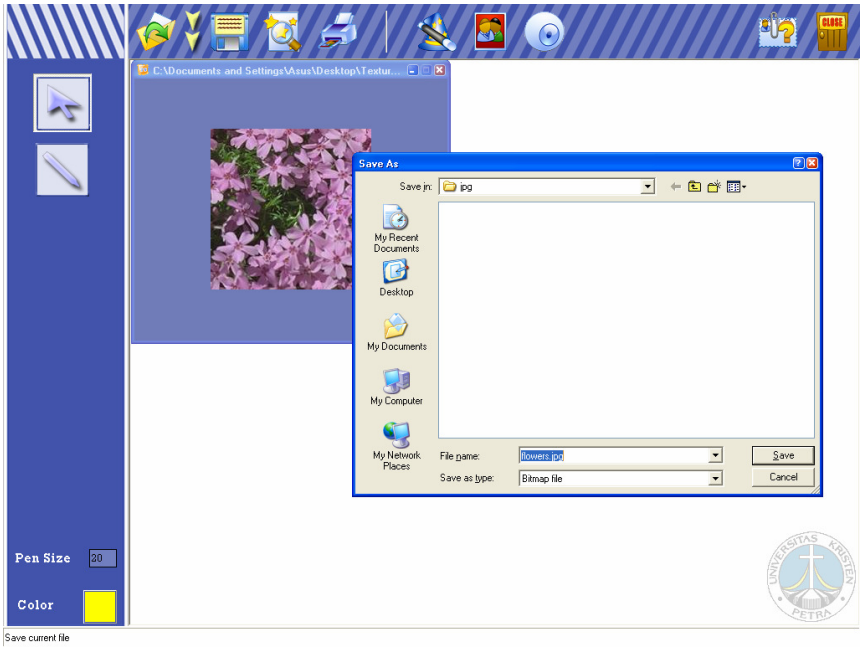
Gambar 4.3 Tampilan *Form Open File*



Gambar 4.4 Tampilan Setelah Proses *Open File*

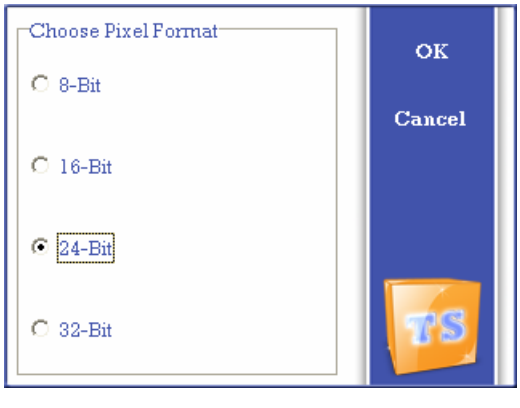
4.2 Proses *Save File*

Untuk menyimpan sebuah *file image*, *user* dapat memilih menu *save*. Jika terdapat beberapa *file image* yang dibuka, maka *file image* yang akan disimpan adalah *file image* yang berada pada status *active window*. Tampilan *form save file* dapat dilihat pada Gambar 4.5.



Gambar 4.5 Tampilan *Save File*

User dapat menyimpan gambar dalam 2 format *file* yang berbeda yaitu BMP dan JPG. Jika *user* memilih untuk menyimpan *file image* dalam format BMP, maka aplikasi akan menampilkan *form dialog Choose Pixel Format*. Gambar *form dialog* ini dapat dilihat pada Gambar 4.6.



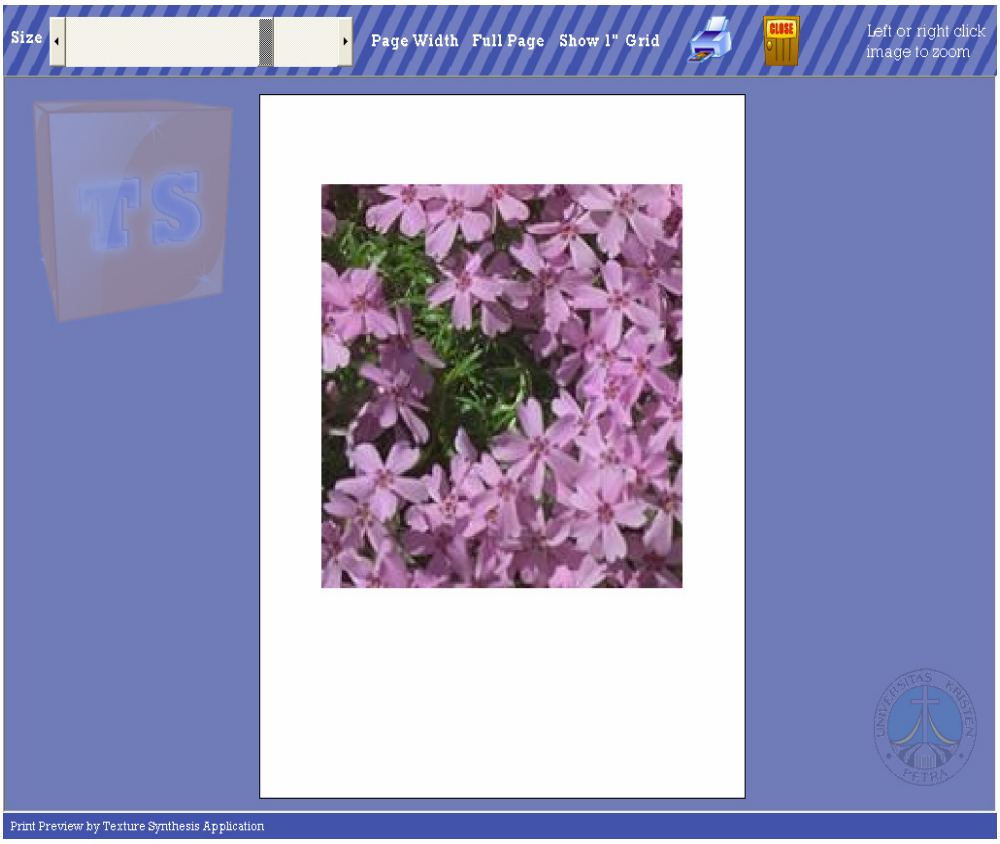
Gambar 4.6 Form *Dialog Choose Pixel Format*

Setelah user memilih *pixel format* dan mengkonfirmasinya dengan menekan tombol ”OK” maka *file* akan disimpan pada alamat yang sudah ditentukan oleh *user* pada *save dialog* sebelumnya. Sedangkan jika user memilih

untuk menyimpan file image dalam format JPG, maka file akan langsung di simpan.

4.3 Proses *Print Preview*

Untuk melihat tampilan *print* dari *image* yang sudah dibuka, *user* dapat memilih menu *print preview*. Jika terdapat beberapa *file image* yang dibuka, maka tampilan *print preview* akan menampilkan *file image* yang berada pada status *active window*. Tampilan *form print preview* dapat dilihat pada Gambar 4.7.

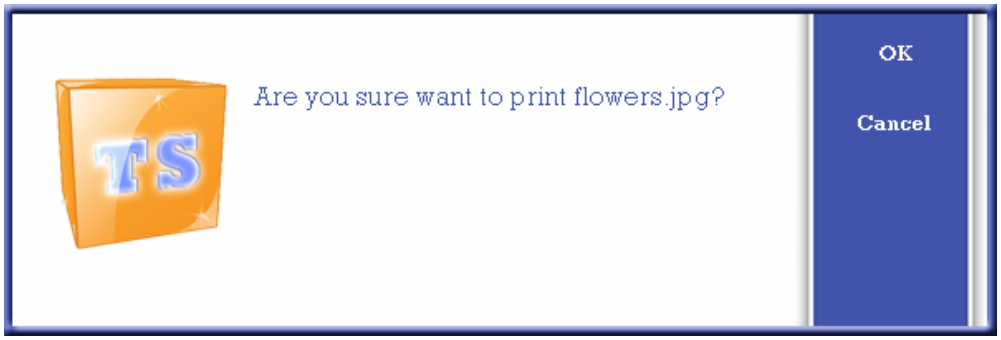


Gambar 4.7 Tampilan *Form Print Preview*

4.4 Proses *Print*

Untuk mencetak gambar, *user* dapat memilih menu *print*. Jika terdapat beberapa *file image* yang dibuka, maka *file image* yang akan dicetak adalah *file image* yang berada pada status *active window*. Setelah *user* memilih menu print, maka aplikasi akan menampilkan *form dialog* konfirmasi *print* jika user

mengkonfirmasinya dengan menekan tombol ”OK”, maka aplikasi akan mencetak *file image* yang berada pada status *active window* tersebut. Tampilan *form dialog* konfirmasi *print* dapat dilihat pada Gambar 4.8.



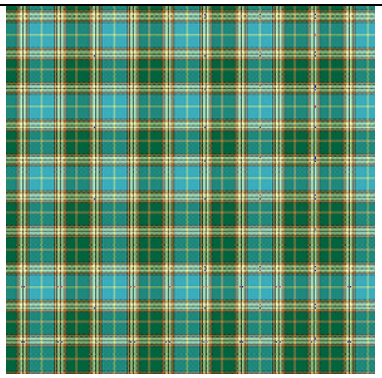
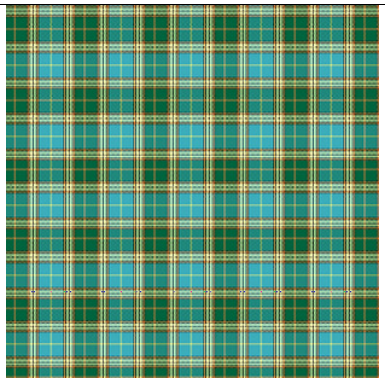
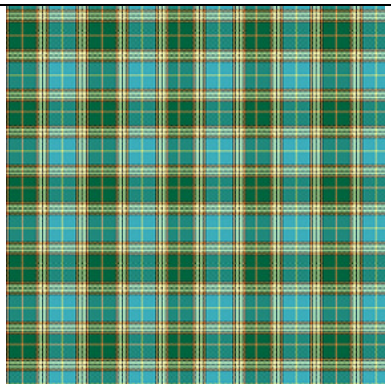
Gambar 4.8 Form *Dialog* Konfirmasi *Print*

4.5 Proses *Texture Synthesis*

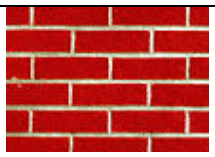
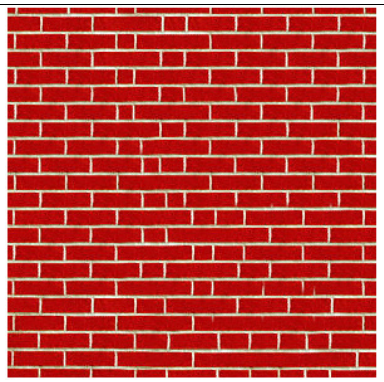
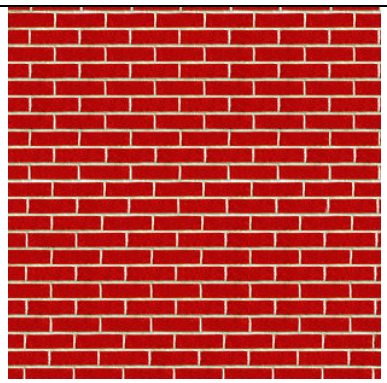
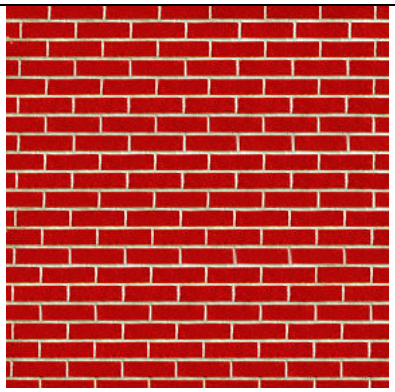
Sebelum melakukan proses *texture synthesis* ini, *user* terlebih dahulu harus sudah membuka minimal sebuah *file image*, dan hanya *image* yang berada pada status *active window* saja yang akan diproses. Setelah *user* memilih menu *texture synthesis*, maka aplikasi akan menampilkan *form dialog texture synthesis parameters* yang berisi *parameter name*, *parameter find matches method* dimana terdapat 2 pilihan yaitu metode *Sum of Square Differences* (SSD) dan metode Bantuan *Histogram*, *parameter* O*utput size* yang terdiri dari *height* dan *width*, *parameter Quilter* yang terdiri *patch size* dan *overlap size*, dan *parameter increment.* Setelah *user* menentukan *parameter* yang diinginkan dan mengkonfirmasinya dengan menekan tombol ”OK” aplikasi akan melakukan proses *texture synthesis* terhadap *image* yang berada pada status *active window*.

Beberapa hasil percobaan proses *texture synthesis* terhadap beberapa *file image* dengan *parameter* yang berbeda-beda dapat dilihat pada Tabel 4.1 hingga Tabel 4.5.

Tabel 4.1 Uji Coba *Texture Synthesize File* regular001.bmp

Keterangan	Gambar
Input Image nama file : regular001.bmp size : 121 x 121	
Output Image Method : SSD Size : 294 x 294 Patch size : 30 Overlap size : 5 Increment : 1	
Output Image Method : SSD Size : 309 x 309 Patch size : 45 Overlap size : 8 Increment : 1	
Output Image Method : SSD Size : 285 x 285 Patch size : 60 Overlap size : 12 Increment : 1	

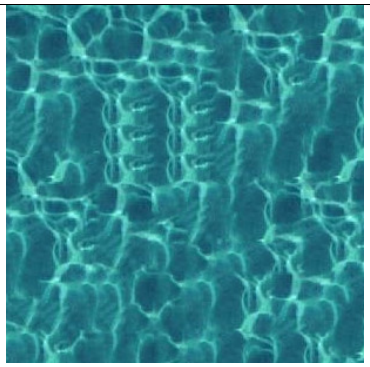
Tabel 4.2 Uji Coba *Texture Synthesize File* near-regular002.bmp

Keterangan	Gambar
Input Image Nama file : near-regular002.bmp Size : 107 x 77	
Output Image Method : SSD Size : 299 x 299 Patch size : 30 Overlap size : 5 Increment : 1	
Output Image Method : SSD Size : 309 x 309 Patch size : 45 Overlap size : 8 Increment : 1	
Output Image Method : SSD Size : 285 x 285 Patch size : 60 Overlap size : 12 Increment : 1	

Tabel 4.3 Uji Coba *Texture Synthesize File* irregular003.bmp

Keterangan	Gambar
Input Image Nama file : irregular003.bmp Size : 124 x 124	
Output Image Method : SSD Size : 299 x 299 Patch size : 30 Overlap size : 5 Increment : 1	
Output Image Method : SSD Size : 309 x 309 Patch size : 45 Overlap size : 8 Increment : 1	
Output Image Method : SSD Size : 285 x 285 Patch size : 60 Overlap size : 12 Increment : 1	

Tabel 4.4 Uji Coba *Texture Synthesize File* n-stochastic001.bmp

Keterangan	Gambar
Input Image Nama file : n-stochastic001.bmp Size : 128 x 128	
Output Image Method : SSD Size : 305 x 305 Patch size : 30 Overlap size : 5 Increment : 1	
Output Image Method : SSD Size : 309 x 309 Patch size : 45 Overlap size : 8 Increment : 1	
Output Image Method : SSD Size : 285 x 285 Patch size : 60 Overlap size : 12 Increment : 1	

Tabel 4.5 Uji Coba *Texture Synthesize File* stochastic.jpg

Keterangan	Gambar
Input Image Nama file : stochastic.jpg Size : 119 x 115	
Output Image Method : SSD Size : 294 x 294 Patch size : 30 Overlap size : 5 Increment : 1	
Output Image Method : SSD Size : 309 x 309 Patch size : 45 Overlap size : 8 Increment : 1	
Output Image Method : SSD Size : 285 x 285 Patch size : 60 Overlap size : 12 Increment : 1	

4.6 Proses *Image Inpainting*

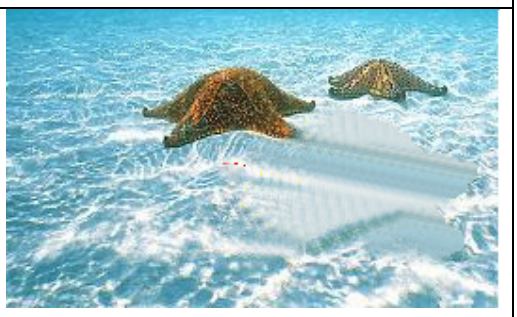
Berikut ini adalah pengujian aplikasi terhadap *image impainting*. *Image impainting* adalah proses untuk mengisi bagian tertentu dari sebuah *image* dengan menggunakan informasi *texture* yang berada di sekeliling bagian yang akan diisi tersebut. Pengujian ini dilakukan dengan melakukan penghapusan secara manual pada bagian tertentu dari *image* dan kemudian dilakukan pengisian *texture* dengan *texture synthesis.* Hal ini dapat dilihat pada Tabel 4.6 dan Tabel 4.7.

Sebelum melakukan proses *image inpainting* ini, *user* terlebih dahulu harus sudah membuka minimal sebuah *file image*, dan hanya *image* yang berada pada status *active window* saja yang akan diproses. Setelah *user* membuka *file image* yang akan diproses, *user* dapat melakukan penghapusan terhadap bagian yang diinginkan dari *image* dengan menggunakan *pen tool*. Pengaturan ukuran *pen* dan warna penanda perubahan terletak di sebelah kiri bawah form utama. Setelah *user* selesai melakukan perubahan, maka *user* dapat memilih menu *image inpainting*, maka aplikasi akan menampilkan *form dialog image inpainting parameters* yang berisi *parameter name*, *parameter find matches method* dimana terdapat 2 pilihan yaitu metode *Sum of Square Differences* (SSD) dan metode Bantuan *Histogram*, *parameter Scanning Order* yang terdiri dari *regular vertical, regular horizontal, surrounding* dan *edge priority* lalu *parameter Quilter* yang terdiri *patch size* dan *overlap size*, dan *parameter increment.* Setelah *user* menentukan *parameter* yang diinginkan dan mengkonfirmasinya dengan menekan tombol ”OK” aplikasi akan melakukan proses *image inpainting* terhadap *image* yang berada pada status *active window*. Pada Tabel 4.6 dan Tabel 4.7 ditampilkan beberapa hasil percobaan proses *image inpainting* terhadap beberapa *file image* dengan *parameter* yang berbeda-beda.

Tabel 4.6. Uji Coba *Image Inpainting File* bintanglaut.bmp

Keterangan	Gambar
Input Image Nama file : bintanglaut.bmp	
Image setelah masking	
Output Image Method : SSD Scanning order : Vertical Patch size : 10 Overlap size : 3 Increment : 5	
Output Image Method : SSD Scanning order : Vertical Patch size : 20 Overlap size : 5 Increment : 5	

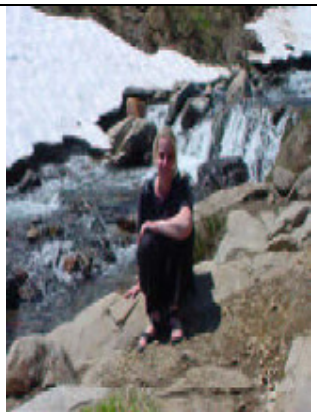
Tabel 4.6 Uji Coba *File Image Inpainting* bintanglaut.bmp (lanjutan)

Keterangan	Gambar
Output Image Method : SSD Scanning order : Horizontal Patch size : 10 Overlap size : 3 Increment : 5	
Output Image Method : SSD Scanning order : Horizontal Patch size : 20 Overlap size : 5 Increment : 5	
Output Image Method : SSD Scanning order : Surrounding Patch size : 10 Overlap size : 3 Increment : 5	
Output Image Method : SSD Scanning order : Surrounding Patch size : 20 Overlap size : 5 Increment : 5	

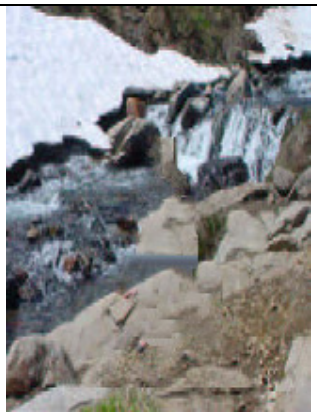
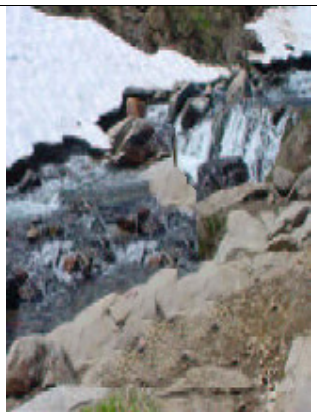
Tabel 4.6 Uji Coba *Image Inpainting File* bintanglaut.bmp (lanjutan)

Keterangan	Gambar
Output Image Method : SSD Scanning order : Edge priority Patch size : 10 Overlap size : 3 Increment : 5	
Output Image Method : SSD Scanning order : Edge priority Patch size : 20 Overlap size : 5 Increment : 5	

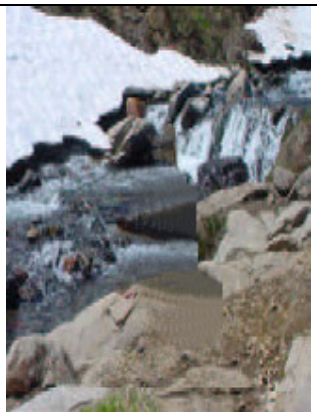
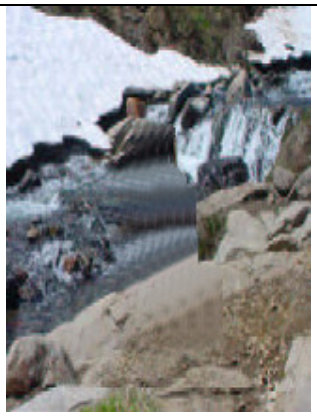
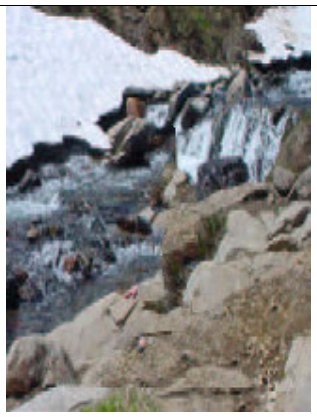
Tabel 4.7. Uji Coba *Image Inpainting File* sungai.bmp

Keterangan	Gambar
Input Image Nama file : sungai.bmp	

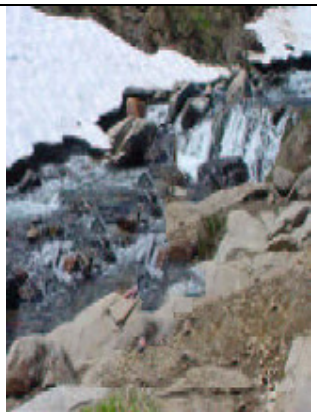
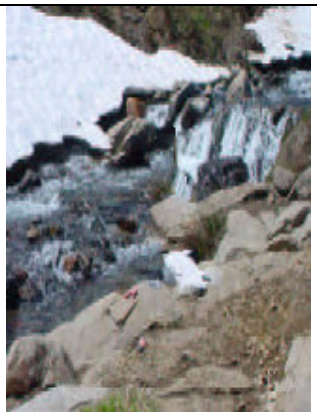
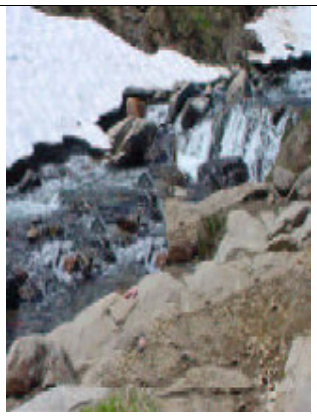
Tabel 4.7 Uji Coba *Image Inpainting File* sungai.bmp (lanjutan)

Keterangan	Gambar
Image setelah masking	
Output Image Method : SSD Scanning order : Vertical Patch size : 10 Overlap size : 3 Increment : 5	
Output Image Method : SSD Scanning order : Vertical Patch size : 20 Overlap size : 5 Increment : 5	

Tabel 4.7 Uji Coba *Image Inpainting File* sungai.bmp (lanjutan)

Keterangan	Gambar
Output Image Method : SSD Scanning order : Horizontal Patch size : 10 Overlap size : 3 Increment : 5	
Output Image Method : SSD Scanning order : Horizontal Patch size : 20 Overlap size : 5 Increment : 5	
Output Image Method : SSD Scanning order : Surrounding Patch size : 10 Overlap size : 3 Increment : 5	

Tabel 4.7 Uji Coba *Image Inpainting File* sungai.bmp (lanjutan)

Keterangan	Gambar
Output Image Method : SSD Scanning order : Surrounding Patch size : 20 Overlap size : 5 Increment : 5	
Output Image Method : SSD Scanning order : Edge priority Patch size : 10 Overlap size : 3 Increment : 5	
Output Image Method : SSD Scanning order : Edge priority Patch size : 20 Overlap size : 5 Increment : 5	

BAB 5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari keseluruhan aplikasi yang telah dibuat dan pengujian yang telah dilakukan dapat disimpulkan beberapa hal dibawah ini :

- Pencarian *patch* terbaik dengan menggunakan metode bantuan *histogram* tidak dapat menghasilkan *patch* yang maksimal dikarenakan pencarian *patch* tersebut hanya berdasarkan distribusi warna pada *image*. Sedangkan pada metode *sum of square differences* (SSD) sudah dapat menghasilkan *patch* yang lebih baik dari bantuan *histogram* karena pencarian *patch* dilakukan dengan memperhitungkan posisi pixel dan warna.
- *Image quilting* sudah dapat melakukan proses *texture synthesis* dengan baik (tidak terlihat bahwa *image output* berasal dari potongan-potongan *image*) pada beberapa macam *texture* baik pada *texture regular, irregular, maupun stochastic*. Untuk *texture regular*, supaya mendapatkan hasil yang maksimal maka besar *patch* sebaiknya lebih besar dari *object texture image* tersebut yang berulang. Sedangkan untuk texture irregular, *image quilting* mempunyai kesulitan pada *texture-texture irregular* tidak mempunyai informasi *texture* yang lengkap. Dan untuk texture *stochastic*, *image quilting* sudah dapat melakukan proses *texture synthesis* dengan baik terlepas dari ukuran *patch* yang dipakai.
- Penanganan *overlap* dengan metode *feather blending* dapat berhasil baik (tidak terlihat bahwa *image output* berasal dari potongan-potongan *image*) pada *image* yang mempunyai degradasi warna hampir sama. Akan tetapi *feather blending* mengalami kesulitan pada penanganan *patch* yang memiliki ukuran *overlap* yang besar, karena hasil *feather blending* akan tampak semakin *blurring*.
- *Scanning order edge priority* sudah dapat melakukan proses *image inpainting* dengan baik (daerah yang terhapus telah dapat diisi dengan *pixel-pixel* yang sesuai) melebihi *image inpainting* dengan *scanning order surrounding.*

5.2 Saran

Beberapa saran yang dapat berguna untuk pengembangan aplikasi ini:

- Menambahkan *image segmentation* saat pengolahan *overlap* sehingga hasil penggabungan *overlap* mengurangi *boundary mismatch.*
- Menambahkan fitur *texture synthesis* yang melakukan *synthesis* dengan memperhitungkan aspek *perspektif* suatu *image*.

DAFTAR PUSTAKA

[1] Efros, A. (2006). *Computational Photography : Image Pyramids and Blending*. Juli 14, 2007. http://www.cb.uu.se/~lucia/dbb/pyramids.ppt

[2] Malik, Jaja J. (2000). *Seri trip & trik unik delphi lanjutan*. Madiun: MADCOMS

[3] Malik, Jaja J. (2002). *Seri panduan pemrograman : pemrograman borland delphi7 (jilid2)*. Madiun: MADCOMS.

[4] Nealen, A & Alexa, M. (2004). *Discrete Geometric Modeling : Texture Synthesis*. Juli 14, 2007. http://cg.cs.tu-berlin.de/papers/hts.ppt

[5] Nealen, A & Alexa, M. (2003). *Discrete Geometric Modeling : Fast and High Quality Overlap Repair for Patch-Based Texture Synthesis.* Juli 14, 2007. http://cg.cs.tu-berlin.de/papers/for.ppt

[6] Sigit, Riyanto. (2001). *Digital Image Processing with MFC*. Jakarta: PT. Gramedia Pustaka Utama.

Lampiran :

Rekapitulasi Anggaran

No	Uraian	Penggunaan	Jumlah	Harga satuan (Rp)	Total (Rp)
1.	Biaya Pembuatan Perangkat Lunak	Pembuatan dan implementasi sistem	1 modul	1.250.000	1.250.000
2.	USB Flash Disk	Penyimpanan Data (160 GB)	1 buah	440.000	440.000
3.	Biaya Survey	Biaya perjalanan ke tempat kegiatan (Makassar)	3 pertemuan	1.500.000	4.500.000
4.	Biaya Survey	Biaya Penginapan (Makasar)	3 pertemuan 2 malam	400.000	2.400.000
4.	Kertas	Cetak Laporan	2 rim	30.000	60.000
5.	Fotocopy dan Penjilidan	Penggadaan Laporan	5 eks	20.000	100.000
Sub Total					8.750.000