MARIATI BT RAHMAN

WET 990184

SISTEM PENAPISAN IMEJ

(IMAGE FILTERING SYSTEM)

Abstrak

Sistem Penapisan Imej merupakan sebuah sistem yang melaksanakan proses penapisan imej iaitu suatu proses menyah-kebisingan yang terdapat pada imej. Pengguna boleh memasukkan imej yang mereka kehendaki ke dalam sistem. Pengguna boleh membuat pilihan samada ingin menggunakan penapisan ruang atau penapisan frekuensi. Proses penapisan imej akan dilaksanakan oleh sistem ini mengikut kemahuan pengguna. Imej akan dipaparkan selepas setiap proses penapisan dijalankan. Pengguna akan dapat melihat hasil setiap penapisan. Pengguna boleh mencetak hasil akhir imej seperti yang diingini. Pengguna boleh membuat pilihan samada inginkan gambar yang lebih jelas atau yang lebih kabur (gambar lebih realistik). Dengan adanya sistem ini, proses membuang kebisingan dan peningkatan mutu imej dapat dilaksanakan dengan lebih cepat dan efektif. Sistem ini juga imej dapat diubah dengan mudah mengikut citarasa pengguna. Di dalam sistem ini terdapat proses input data atau imej, proses penapisan, paparan imej dan cetakan imej. Sistem ini memudahkan pengguna mengubahsuai mutu imej daripada tidak berkualiti menjadi lebih berkualiti dan dapat digunakan dengan sebaik-baiknya. Bagi mewujudkan sistem yang menarik, cekap dan efektif, perisian yang digunakan bagi implementasi sistem ini adalah Matlab 6.0. Dengan kemunculan sistem ini para pelajar, pensyarah, doktor, jurufoto majalah, jurufoto akhbar, wartawan media dan para pengkaji astronomi serta mereka yang terlibat dengan imej atau gambar akan lebih mudah menjalankan aktiviti mereka dan mendapat manfaat daripada sistem yang dibangunkan ini.

Penghargaan

Alhamdulillah, bersyukur kita ke hadrat Ilahi kerana dengan limpah dan kurniaNya dapatlah saya menyiapkan satu kajian yang bertajuk "Image Filtering System" bagi memenuhi syarat bagi mata pelajaran Projek Ilmiah Tahap II (WXES 3182) dengan sempurna dan lancar di samping dapat menyiapkannya dalam jangka masa yang telah ditetapkan. Sesungguhnya tiada sesuatu kerja yang mudah, namun usaha yang berterusan perlu untuk mencapai apa yang dikehendaki walaupun terpaksa berkorban masa, tenaga, wang ringgit dan sebagainya. Setiap detik pasti mengundang rintangan-rintangan yang kadang-kala begitu menguji tahap kesabaran, kemahiran dan keimanan kita.

Namun kejayaan yang dicapai ini tidak akan hadir dengan sendirinya tanpa bantuan dan tunjuk ajar daripada semua pihak terutamanya daripada penasihat saya yang saya hormati iaitu Puan Nornazlita Hussin dan juga moderator saya iaitu Encik Phang Keat Keong. Tidak dilupakan juga kepada semua rakan seperjuangan yang banyak memberi nasihat, tunjuk ajar dan sokongan padu yang walaupun sedikit sumbangannya, namun amat bermakna bagi saya. Jutaan terima kasih diucapkan kepada mereka semua dan juga kepada semua yang terlibat secara langsung atau tidak langsung dalam menjayakan projek ini.

Akhir kata, saya berharap agar kajian ini akan dapat memberikan banyak manfaat kepada semua dan menjadi panduan dalam mencapai matlamat yang ditetapkan.

Sekian, terima kasih.

Senarai Isi Kandungan

Senarai Jadual

Senarai Rajah

1.0 PENGENALAN

1.1 Pengenalan Projek

Sistem Penapisan Imej ini merupakan suatu sistem yang dapat meningkatkan mutu imej. Pemindahan imej daripada gambar biasa kepada imej digital (imej yang telah diimbas) iaitu apabila dipindah masuk ke dalam komputer akan mewujudkan kebisingan. Kebisingan adalah suatu rentetan maklumat yang tidak diperlukan yang mencemari sesuatu imej. Ia dihasilkan oleh pelbagai sumber. Oleh itu, banyak teknik telah digunakan untuk melaksanakan penyelesaian kepada masalah ini.

Pelbagai jenis imej digunakan dalam aplikasi saintifik dan komersial. Oleh sebab itu, maka wujudlah sistem penapisan imej ini supaya imej atau gambar itu dapat diproses untuk kemudahan semua pihak yang memerlukannya. Sistem Penapisan Imej ini akan memperbaiki imej serta mempertingkatkan kualiti imej dengan menyah-kebisingan dan menggunakan teknik penapisan imej.

Imej-imej tersebut akan menjadi lebih jelas atau lebih kabur mengikut kemahuan pengguna. Sistem ini dapat mengubah imej mengikut citarasa pengguna. Dengan adanya sistem seperti ini diharapkan dapat mempermudahkan kerja mereka yang sering menggunakan imej dalam kehidupan harian mereka.

Penapisan memfokus kepada pengembalian imej yang mempunyai kebisingan dengan menapis keluar kebisingan. Pendekatan yang dibincangkan ini dibahagikan kepada 2 kategori iaitu Kaedah Domain Ruang dan Kaedah Domain Frekuensi.

Dalam kaedah domain ruang, penggunaan topeng ruang untuk pemprosesan imej dipanggil penapisan ruang dan topeng itu dipanggil penapis ruang. Tiga jenis penapis ruang dipersembahkan iaitu penapis mean, penapis median dan penapis peningkatan.

Dalam kaedah domain frekuensi, terdapat dua jenis penapis iaitu penapis lowpass dan penapis highpass. Penapis lowpass untuk melicinkan imej manakala penapis highpass untuk menajamkan imej.

1.2 Objektif Sistem

i) Membangunkan satu sistem penapisan imej iaitu suatu sistem menyah-kebisingan yang terdapat pada imej dengan cekap dan bersistematik.

ii) Sistem ini dapat menjimatkan masa pengguna dalam proses penapisan imej dan mempertingkatkan mutu imej

1.3 Skop Sistem

i) Proses menapis kebisingan dan meningkatkan mutu imej

ii) Menjalankan proses menginput imej daripada pengguna, melaksanakan penapisan imej, memaparkan semua imej sepanjang proses penapisan mengikut peringkat, mencetak imej awal, mencetak imej semasa penapisan dilakukan dan mencetak imej akhir yang diinginkan oleh pengguna.

iii) Sistem ini melaksanakan dua proses penapisan iaitu penapisan ruang dan penapisan frekuensi yang membolehkan pengguna membuat pilihan.

iv) Skop sistem ini meliputi proses penapisan iaitu menyah-kebisingan yang terdapat pada imej masukan oleh pengguna supaya imej yang lebih jelas dan licin diperolehi agar pengguna dapat mengaplikasikan imej tersebut dengan sebaik-baiknya.

1.4 Pengguna sasaran

- Para pelajar – para pelajar yang mempelajari sesuatu subjek yang melibatkan imej contohnya mata pelajaran 'Pemprosesan Imej Digital' seperti yang terdapat di Fakulti Sains Komputer dan Teknologi Maklumat. Begitu juga pelajar yang mengambil bidang Multimedia yang banyak menggunakan imej dalam pembelajaran mereka.

- Pensyarah – Para pensyarah yang menggunakan imej dalam pengajaran mereka dalam sesuatu subjek yang diajar kepada pelajar. Mereka memerlukan imej ini agar dapat diperlihatkan dengan lebih jelas kepada para pelajar.
- Doktor – Para doktor juga menggunakan imej ini contohnya dalam pengesanan sel-sel darah dan juga x-ray yang dijalankan ke atas pesakit mereka. Imej memang banyak digunakan dalam bidang perubatan. Oleh itu, sistem seperti ini memang amat diperlukan oleh para doktor.
- Jurufoto majalah, jurufoto akhbar – Kedua-dua jurufoto majalah dan akhbar memang memerlukan imej atau gambar yang bermutu untuk dimasukkan ke dalam majalah dan akhbar mereka. Dengan adanya imej yang baik dan menarik sudah tentu akan menjadi tarikan ramai pembaca.
- Wartawan media – para wartawan media juga akan menyelitkan gambar di dalam hasil karya atau penulisan mereka untuk menarik perhatian pembaca. Oleh itu imej yang bermutu diperlukan agar hasil karya itu akan lebih menarik. Ini kerana selalunya pembaca akan lebih tertarik apabila terdapat gambar yang diselitkan pada karya tersebut.
- Para pengkaji astronomi – Para pengkaji astronomi menggunakan sistem ini dalam mengkaji cakerawala seperti imej bulan, bintang, matahari dan planet-planet lain.

1.5 Penjadualan Projek

Jadual 1.1 : Jadual perancangan setiap fasa bagi Sistem Penapisan Imej

SISTEM PENAPISAN IMEJ									
FASA	TAHUN 2002							TAHUN 2003	
	JUN	JULAI	OGOS	SEPT	OKT	NOV	DIS	JAN	FEB
Kajian Awal dan Analisis Sistem									
Rekabentuk Sistem									
Rekabentuk Aturcara									
Pengekodan dan Pengujian Unit									
Integrasi dan Pengujian Sistem									
Penyelenggaraan Sistem									

1.6 Kesimpulan

Di sini dapat kita simpulkan bahawa sistem ini diwujudkan bagi kemudahan semua pihak yang memerlukan imej dalam kegunaan harian mereka. Dengan adanya sistem ini mutu imej dapat diperbaiki dan kualiti dipertingkatkan. Objektif sistem diharap akan dapat dicapai seperti yang dikehendaki. Skop sistem yang dipaparkan juga diharap dapat membantu pengguna dan penjadualan projek dapat diikuti sepenuhnya agar projek ini dapat disiapkan mengikut jangka masa yang ditetapkan mengikut perancangan dan tersusun.

2.0 KAJIAN LITERASI

Kajian literasi merupakan kajian permasalahan yang dijalankan sebelum projek dapat dilaksanakan. Kajian ini meliputi analisa ke atas sistem-sistem terdahulu, kajian teknik yang akan digunakan serta kajian terhadap domain projek ini.

2.1 Tujuan

i) Untuk mengumpul maklumat sistem yang akan dibangunkan

ii) Untuk mengkaji dan menilai sistem yang sama konsep atau berkaitan yang telah sedia ada dibangunkan bagi menentukan kekuatan dan kelemahan sistem tersebut di samping memperbaiki kelemahan sistem yang telah sedia ada

iii) Untuk mendapatkan pemahaman yang jelas tentang konsep yang terlibat dalam sistem yang akan dibangunkan, di samping membandingkan beberapa perisian, peralatan dan pendekatan yang akan digunakan bagi mendapatkan hasil dan penyelesaian yang terbaik

2.2 Kajian literasi / penyelidikan

Berpandukan pada kertas kerja lepas, jurnal-jurnal, pembacaan buku-buku rujukan samada secara media cetak atau elektronik seperti laman web Internet. Tujuan kajian ini adalah untuk mengenalpasti masalah-masalah dan cuba memahami sistem pengurusan masakini dalam usaha membina satu sistem yang lebih berkesan dan membaiki sebarang kelemahan yang wujud.

2.2.1 Sumber pengumpulan maklumat

Terdapat beberapa sumber pengumpulan maklumat yang telah dijalankan bagi memastikan penemuan maklumat yang lengkap dan jelas diperolehi.

2.2.1.1 Enjin Carian (Search Engine)

Melalui pelayaran Internet, pelbagai sumber informasi dapat diperolehi. Internet membantu individu memperolehi banyak maklumat yang merangkumi semua aspek melalui enjin carian. Terdapat beberapa enjin carian yang digunakan untuk mendapatkan maklumat iaitu melalui laman web :

i- http://www.yahoo.com

ii- http://www.catcha.com

iii- http://www.msn.com

Tinjauan yang dilaksanakan ke atas laman web ini memberikan banyak maklumat-maklumat sistem semasa yang telah dibangunkan oleh syarikat-syarikat perisian yang terkenal.

2.2.1.2 Bilik Dokumen / Perpustakaan Fakulti Sains Komputer dan Teknologi Maklumat

Bilik dokumen atau lebih dikenali sebagai perpustakaan FSKTM ini merupakan sumber untuk mengumpul maklumat dan juga tempat bagi dokumentasi atau laporan latihan ilmiah pelajar terdahulu. Ia dapat membantu pembangunan sistem berdasarkan maklumat yang relevan dan memang diperlukan bagi panduan pelajar yang sedang menyiapkan kertas kerja ini. Rujukan ke atas dokumentasi projek terdahulu memberikan sedikit sebanyak lakaran atau gambaran serta panduan dalam penghasilan kertas kerja ini.

2.2.1.3 Perpustakaan Utama Universiti Malaya

Maklumat mengenai sistem ini juga diperolehi daripada sumber rujukan di perpustakaan. Pembacaan buku-buku rujukan memberikan pemahaman secara umum tentang cabang kajian yang dibuat berkaitan Sistem Penapisan Imej. Melalui pembacaan ini akan memberi gambaran dan perspektif yang agak luas. Selain daripada itu, beberapa definisi juga diperolehi daripada rujukan perpustakaan. Banyak maklumat diperolehi daripada perpustakaan utama Universiti Malaya. Sumber ini penting dalam proses mendapatkan pemahaman yang jelas tentang sistem yang akan dibangunkan dan juga pengetahuan tentang pemprosesan imej.

2.2.2 Definisi

2.2.2.1 Sistem

Pada umumnya "sistem" didefinisikan sebagai suatu set yang mengandungi dua atau lebih unsur yang bergantungan antara satu sama lain yang berfungsi untuk mencapai sesuatu objektif tertentu; cara atau kaedah untuk melakukan sesuatu. Menurut definisi lain, sistem adalah sekumpulan unsure dan tatacara yang berkaitan, yang bekerjasama untuk melakukan sesuatu tugas. Bagi sistem yang berasaskan komputer, terdapat enam elemen yang terlibat iaitu perkakasan, perisian, pengguna, prosedur, data dan maklumat. Kesemua elemen ini akan berinteraksi untuk menukarkan data menjadi maklumat yang boleh digunakan oleh pengguna.

2.2.2.2 Penapisan

Proses penapisan mengurangkan data dan membenarkan pengekstrakan output yang diperlukan untuk analisa. Penapisan biasanya digunakan untuk membuang kebisingan atau menghasilkan peningkatan imej.

- Penapisan dalam domain ruang

Penapis dalam domain ruang dipanggil penapis ruang. Terdapat tiga jenis penapis yang dibincangkan iaitu :

i) Penapis mean

ii) Penapis median

iii) Penapis peningkatan

Kebanyakan penapis ruang diimplementasi menggunakan topeng convolution. Satu aspek menarik pada topeng convolution adalah kesan keseluruhan boleh dikesan berdasarkan corak umumnya.

i) Penapis mean digunakan untuk mengira nilai purata. Penapisan ini dilaksanakan dengan mencari nilai purata untuk nilai di dalam tetingkap NxN. Salah satu jenis penapis berjenis adalah penapis purata aritmetik yang merupakan asas kepada penapis pemurataan. Ia melicinkan atau melancarkan variasi tempatan dalam sesuatu imej iaitu seolah-olah mengaburkan imej tersebut tetapi dalam masa yang sama ia menyahkan kebisingan. Satu lagi jenis penapis pemurataan adalah yang dipanggil penapis pemurataan harmonik.

Penapis mean beroperasi dalam satu lengkungan piksel yang dipanggil jiran dan menukarkan nilai piksel yang di tengah-tengah dalam skop kejiranan itu tadi kepada nilai purata antara piksel-piksel dalam kejiranan itu. Penukaran ini dilakukan dengan menggunakan topeng convolution dengan matriks 3x3 iaitu :

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

ii) Penapis median adalah penapis non-linear. Penapisan ini dilaksanakan dengan membuat pengaturan nilai ahli-ahli kejiranan mengikut tahap terendah sehingga tahap tertinggi dalam nilai tahap kelabu dan kemudian menggunakan aturan ini untuk memilih nilai yang dikehendaki. Ianya berdasarkan kepada statistik imej yang dikenali sebagai statistik aturan.

Ia dilaksanakan dengan menggunakan subimej ataupun tetingkap dan menggantikan piksel di tengah-tengah. Statistik aturan adalah satu teknik menyusun piksel mengikut aturan yang berturutan berdasarkan nilai tahap kelabu. Penapis median akan mengambil nilai di tengah-tengah daripada set aturan tersebut. Langkah yang digunakan adalah bersamaan dengan proses convolution. Penapis ini boleh menggunakan pelbagai jenis matriks iaitu 3x3, 5x5, 7x7 dan sebagainya.

iii) Penapisan peningkatan digunakan untuk meningkatkan kualiti imej yang hendak diproses. Contoh penapis ini adalah penapis laplacian. Dua jenis matriks 3x3 bagi penapis laplacian adalah :

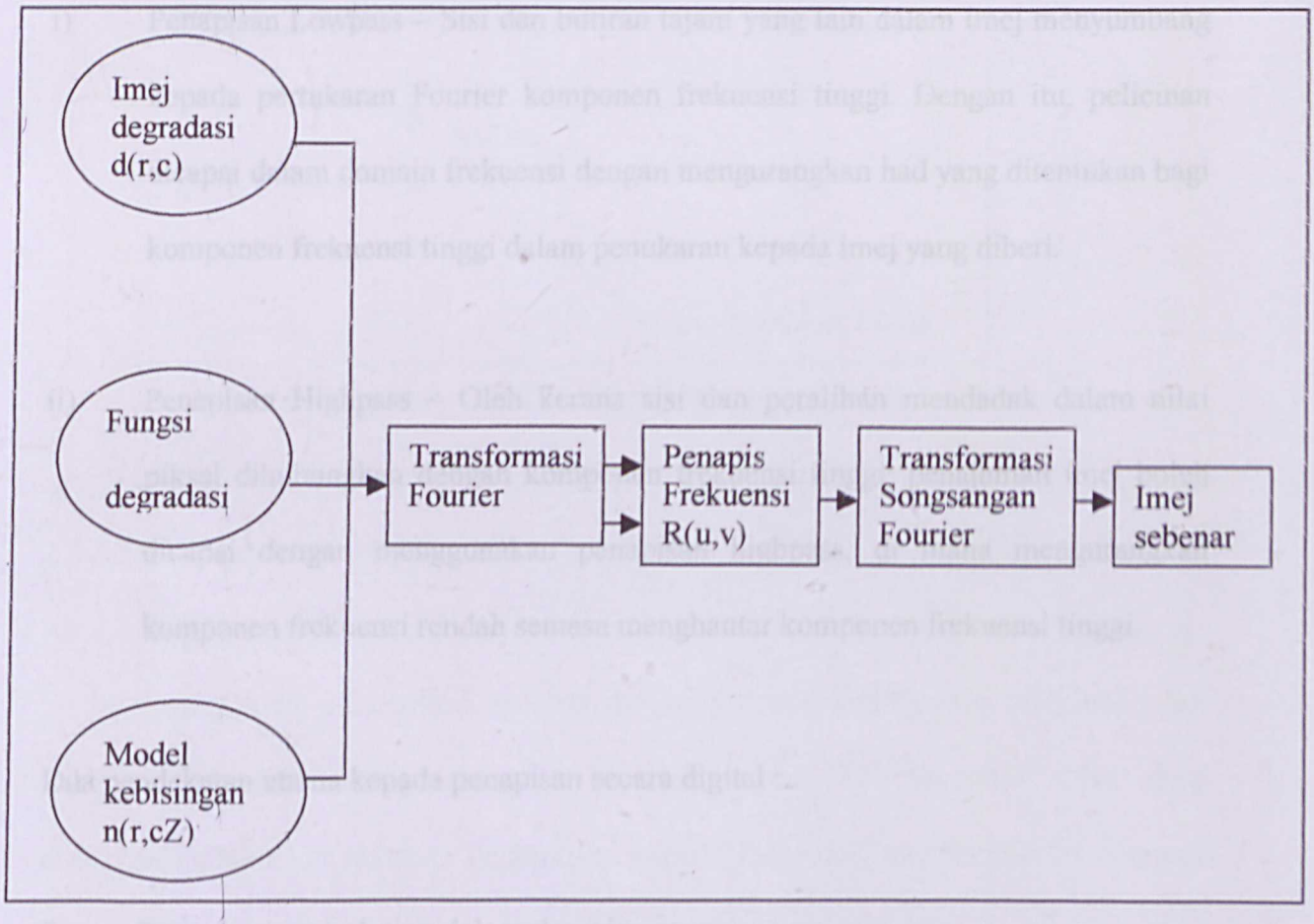
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Penapis laplacian akan meningkatkan keseluruhan imej dalam semua arah secara sama rata. Terdapat 4 topeng convolution penapis yang berbeza iaitu vertical, horizontal dan 2 diagonal iaitu :

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

- Penapisan dalam domain frekuensi

Ianya berfungsi dengan menggunakan perwakilan transformasi Fourier bagi sesuatu imej itu. Yakni dengan lebih tepat lagi, maklumat spektrum digunakan. Transformasi Fourier perlu dilakukan terhadap ketiga-tiga entiti berikut iaitu imej yang didegradasi, fungsi degradasi dan model kebisingan. Ianya akan membentuk satu penapis frekuensi yang mana ianya akan dilalukan kepada songsangan transformasi Fourier untuk mendapatkan imej sebenar. Ini dapat ditunjukkan dalam gambarajah di bawah :



Rajah 2.1 : Proses penapisan dalam domain frekuensi

Pengaburan adalah salah satu fungsi degradasi. Fungsi ini boleh berlaku secara linear dalam satu arah iaitu alam mendatar, menegak, pepenjuru ataupun dalam cara bersimetri (pengkaburan adalah sama dalam semua arah). Jumlah pengkaburan boleh didapati dengan membuat pemerhatian terhadap sumber titik ataupun garis dalam sesuatu imej.

Terdapat dua jenis penapis frekuensi iaitu :

i) Penapisan Lowpass – Sisi dan butiran tajam yang lain dalam imej menyumbang kepada pertukaran Fourier komponen frekuensi tinggi. Dengan itu, pelicinan dicapai dalam domain frekuensi dengan mengurangkan had yang ditentukan bagi komponen frekuensi tinggi dalam penukaran kepada imej yang diberi.

ii) Penapisan Highpass – Oleh kerana sisi dan peralihan mendadak dalam nilai piksel dihubungkan dengan komponen frekuensi tinggi, penajaman imej boleh dicapai dengan menggunakan penapisan highpass, di mana mengurangkan komponen frekuensi rendah semasa menghantar komponen frekuensi tinggi.

Dua pendekatan utama kepada penapisan secara digital :

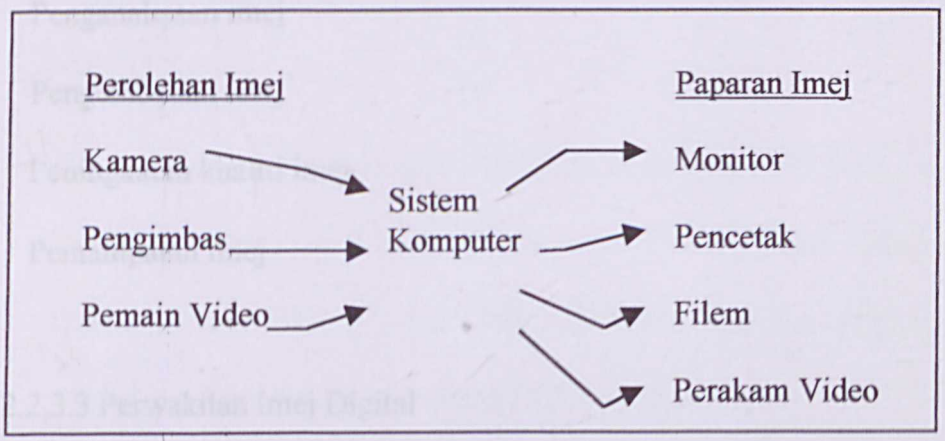
i) Penapisan convolution dalam domain ruang

ii) Analisis Fourier dalam domain frekuensi

2.2.2.3 Imej

2.2.2.3.1 Imej Digital

Imej dalam perwakilan komputer memerlukan beribu-ibu data dan tanpa amaun data yang betul, medium ini mungkin tidak akan berkesan.



Rajah 2.2 : Rajah menunjukkan bagaimana imej diperolehi dan dipaparkan

2.2.2.3.2 Pemprosesan Imej

Imej merupakan satu bentuk gambar dan imej digital adalah imej yang disimpan dalam bentuk digital yang seterusnya akan digunakan oleh komputer. Imej akan dimanipulasikan oleh manusia bergantung kepada kehendak mereka sendiri. Berikut adalah beberapa proses yang biasa dilakukan ke atas sesuatu imej iaitu :

- Pengembalian imej sebenar
- Pembaikpulihan imej
- Pemampatan imej

Hasil daripada imej yang telah diproses, outputnya akan digunakan kembali untuk analisa seterusnya. Pemprosesan imej digital seringkali ditafsirkan kepada pemprosesan imej dalam dua dimensi oleh komputer. Secara amnya ianya boleh ditunjukkan seperti di bawah :

- Perwakilan imej
- Penganalisaan imej
- Pengembalian imej
- Peningkatan kualiti imej
- Pemampatan imej

2.2.2.3.3 Perwakilan Imej Digital

Secara kebiasaannya, imej digital diwakili oleh perwakilan I(r,c) di mana r dan c merupakan kedudukan dan piksel tersebut yang menandakan keamatan untuk imej hitam putih. Tetapi, bagi imej berwarna, perwakilan I(r,c) mewakili fungsi yang berlainan. Berikut adalah beberapa jenis imej :

- Perduaan
  - Perwakilan yang paling senang kerana mempunyai dua nilai sahaja iaitu putih atau hitam, '0' atau '1'. Kerapkali digunakan dalam aplikasi penglihatan komputer.

- Penskalaan kelabu
  - Perwakilan khas yang mengandungi maklumat mengenai keamatan imej. Bilangan bit yang ada menentukan tahap-tahap keamatan sesuatu imej tersebut. Contohnya 8 bit/piksel mengandungi tahap 256(0-255) keamatan.

- Warna
  - Ia boleh diwakilkan dengan 3-band monochrome imej yang mana setiap band mewakili warna yang berlainan. Sekiranya setiap band mempunyai 8 bit per piksel, maka, untuk 1 piksel dalam perwakilan warna mempunyai 24 bit per piksel. Ini merupakan perwakilan RGB(red, green,blue)

- Pelbagai spektral
  - Maklumat yang disimpan menjangkau darjah penglihatan manusia. Contohnya seperti ultraviolet, x-ray dan radar. Imej seperti ini selalunya ditukarkan kepada perwakilan yang lebih bermakna seperti pemetaan kepada RGB.

2.2.2.3.4 Format Fail Imej Digital

Secara amnya, boleh dibahagikan kepada 2 kategori utama iaitu :

- Bitmap diwakili oleh pemodelan I(r,c)

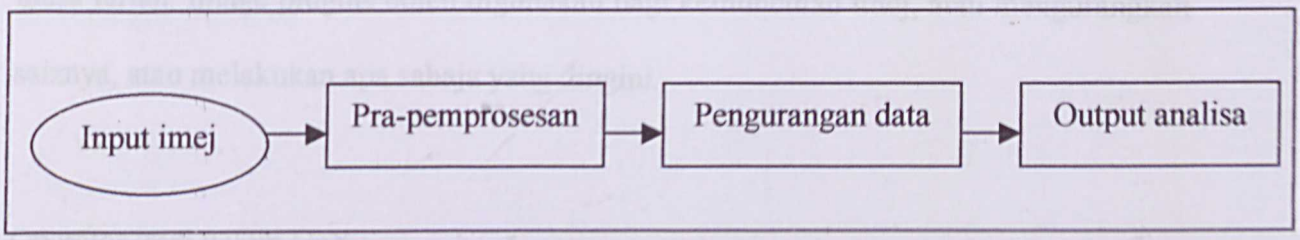
- Imej vektor iaitu kaedah yang digunakan untuk mewakili garis, lengkuk dan bentuk-bentuk lain yang mana ianya menyimpan titik-titik ataupun ciri-ciri penting sahaja.

Secara amnya, format fail yang berjenis bitmap mempunyai 2 bahagian iaitu :

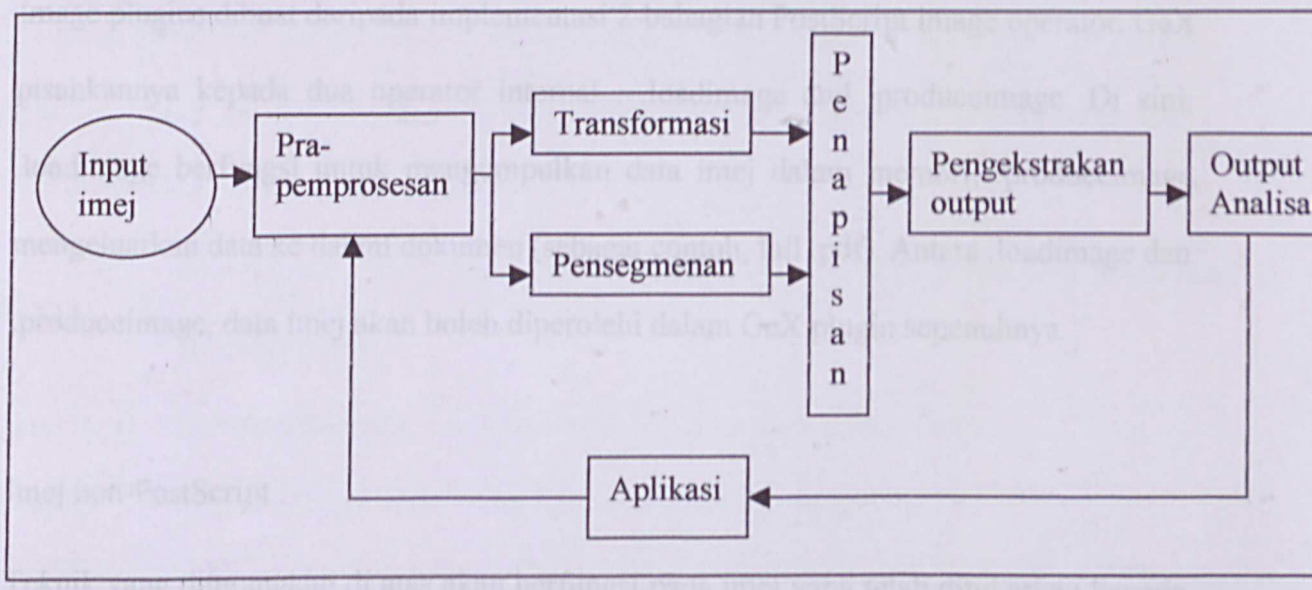
i) Maklumat kepala yang menyimpan bilangan baris(tinggi), bilangan lajur(lebar), bilangan jalur, bilangan bit per piksel dan jenis fail.

ii) Data mentah

2.2.2.3.5 Analisa Imej



Rajah 2.3 : Model Analisa Imej (Secara am)



Rajah 2.4 : Model Analisa Imej (Secara Terperinci)

2.2.3 Kajian yang telah dijalankan :

2.2.3.1 Kajian 1 :

VTeX/GeX image plugins

Ringkasan :

Dimulakan dengan versi 6.4, VTeX/GeX menyokong penapisan imej plugins.

'Image filtering' plugins membenarkan pemanipulasian ke atas imej dokumen semasa masa larian. Image plugins boleh digunakan bagi kemunculan imej, atau mengurangkan saiznya, atau melakukan apa sahaja yang dingini.

Operator imej dalam GeX :

Image plugins dibuat daripada implementasi 2-bahagian PostScript image operator. GeX pisahkannya kepada dua operator internal : .loadimage and .produceimage. Di sini, .loadimage berfungsi untuk mengumpulkan data imej dalam memori; .produceimage mengeluarkan data ke dalam dokumen (sebagai contoh, fail .pdf). Antara .loadimage dan .produceimage, data imej akan boleh diperolehi dalam GeX plugin sepenuhnya.

Imej non-PostScript :

Teknik yang diterangkan di atas akan berfungsi pada imej yang telah ditukarkan kepada fail .eps. Bagaimanapun, majoriti imej yang ingin ditapis adalah dalam format bitmap (VTeX kini mampu menampung BMP, PCX, GIF, JPEG, TARGA, TIFF, and PNG).

Untuk memudahkan penggunaan format-format seperti ini, GeX mempunyai operator tambahan, .readimage. Operator .readimage menerima single string argument yang mana merupakan fail bitmap; ia mengembalikan struktur imej sesuai-GeX yang mana boleh memberikan secara terus kepada .produceimage atau diproses melalui penapisan plugin. Sebagai contoh,

```
\special{pS: (mypic.gif) .readimage toGray .produceimage}
```

akan membaca imej GIF, menukarkannya kepada grayscale, dan kemudian mengeluarkannya. Didapati bahawa dalam penambahan kepada penapisan, operator ini mempunyai aplikasi yang lain; salah satu akan menggunakan ikon bitmap di dalam persekitaran PStricks.

Graphicx support :

Memandangkan penerangan di atas kelihatan menakutkan, penggunaan penapis imej sebenarnya adalah sangat mudah. Fungsi tahap rendah telah diintegrasikan ke dalam graphicx package, dan semua yang perlu diketahui adalah beberapa pilihan tambahan. Beberapa katakunci yang baru adalah :

viagex	Memproses imej bitmap melalui GeX; kunci ini diperlukan bagi mana-mana penapisan imej.
Brightness=B	Membuatkan imej lebih terang (B>0) atau lebih gelap (B<0)
Contrast=C	Membuatkan kontras imej lebih (C>0) atau kurang (C<0)
Colorspace=S	Menukarkan imej kepada ruang warna yang lain (contohnya, menjadikannya grayscaled)
Degrade=D	Downsample imej (biasanya, untuk mengurangkan saiz fail output).

Sebagai contoh,

\includegraphics[viagex,brightness=0.1,degrade=0.5]{mypic.gif}

Sampel plugins :

Segala fungsian yang diterangkan di atas disediakan oleh dua sampel penapis imej iaitu :

- Degrade plugin yang melaksanakan downsampling
- TransBit plugin melakukan manipulasi imej seperti yang dijelaskan di atas

GeX memungkinkan kita untuk mengimplementasi penapis imej kita sendiri.

2.2.3.2 Kajian 2 :

Dental Informatics - Spring 1996

Course director: Dr. Titus K. L. Schleyer, DMD, Ph. D

E-mel: titus@eniac.dental.temple.edu

Sesi Ringkasan – Imej dalam pergigian

Hasilnya:

Sesi ini akan membolehkan pelajar untuk :

1. Memahami sifat-sifat imej digital : bagaimana imej ini boleh dicipta, disimpan dimanipulasi dan dipaparkan.

Pemanipulasian Pameran :

Menggunakan dental radiographic dan imej klinik yang sebenar, imej digital pemanipulasian pameran akan diperjelaskan dan didemonstrasikan : look-up tables, windowing, thresholding, lengkungan gamma dan pengubahsuaian gamma, histogram equalization, histogram matching, dan histogram stretching, manipulasi geometrik, dan pseudocolorization.

Convolution dan Operasi Penapisan Digital :

Menggunakan dental radiographic dan imej klinik yang sebenar, operasi penapisan imej digital akan diperjelaskan dan didemonstrasikan : pengaburan dan penajaman, penapisan median, teknik peningkatan sisi, dilation and erosion imej binari.

Transformasi Fourier :

Konsep domain frekuensi akan dipersembahkan dan transformasi Fourier akan dilaksanakan bagi beberapa imej. Manipulasi imej dalam domain frekuensi akan dibandingkan dengan manipulasi dalam domain imej.

2.2.3.3 Kajian 3 :

Perisian Thumbs Plus

Versi 4.50-S (evaluation version) and 4.50-R (registered/retail)

Thumbs Plus telah memenangi banyak anugerah, menyertai angkasawan NASA pada semua pengembaraan angkasa lepas, dan digunakan oleh Intel, Microsoft, HP, Tentera Darat US, Air Force dan tentera laut dan sebagainya. Ia mempunyai beberapa ciri-ciri dan pilihan termasuklah :

- Pengubahsuaian Imej – Kecerahan, Kontras, Hue, Saturation, Keterangan, Keseimbangan warna dan gamma
- Penapisan imej digital – Menajamkan, Mengaburkan, Embos
- Membuat salinan, memindahkan, menamakan semula dan memadam fail dan folder
- Paparkan imej yang pelbagai
- Menyimpan komen dalam fail imej
- Menetapkan katakunci dan melakukan carian katakunci
- User-defined fields untuk mengkategorikan dan pencarian
- Mencari imej yang sama dan penduaannya berdasarkan kandungan imej
- Banyak fail menyusun pilihan (kesamaan, orientasi, saiz imej, lebar, tinggi, tarikh, nama, nama numerik, jenis dan saiz imej)

- Watermarking imej digital (untuk melindungi imej) atau untuk mengesan watermarks dalam imej
- Synchronised image scrolling, panning and zooming (untuk membandingkan imej)
- Galeri imej bagi tayangan slaid pesanan dan organisasi dengan segera
- Multiple customisable thumbnail views (small, large, custom, list and report)
- Pemprosesan berkumpulan berasaskan langkah
- Image stamping (overlay text or image)
- Freehand select, paste into
- Extensive search capabilities with named search sets
- Lokasi kegemaran bagi capaian segera ke folder
- Penamaan semula fail secara automatik (Pernomboran)

2.2.3.4 Kajian 4 :

Pengimbas Canon CanoCraft CS-P 3.7

Beberapa kajian dan penganalisaan telah dijalankan terhadap pengimbas Canon CanoCraft CS-P 3.7. Hasil yang didapati setelah beberapa kesan seperti 'Sharpen', 'Soften' dan 'Despeckle' digunakan adalah seperti di bawah :

i) 'Sharpen'

- Menggunakan kesan 'sharpen' apabila imej adalah keluar sedikit daripada fokus atau anda berhajat untuk menjelaskan garisan yang membentuk objek.
- Caranya adalah dengan mengklik menu [Effect] dan pilih [Sharpen].

- Kesan 'sharpen' akan dijalankan ke atas imej dalam tetingkap 'preview'.

'Sharpen' adalah kesan khas yang menimbulkan kontras di antara piksel bersebelahan untuk menunjukkan garisan yang membentuk objek lebih jelas lagi, terutama bagi objek yang kabur. Pemprosesan imej yang berulang kali menggunakan kesan 'sharpen' ini akan menyebabkan pendedahan kepada kekotoran atau goresan pada dokumen. Setiap pengguna perlu berhati-hati untuk tidak menggunakan kesan 'sharpen' ini beberapa kali.

Rajah 2.5 : Imej Bangunan Asal

Rajah 2.6 : Imej lebih jelas setelah menggunakan kesan 'Sharpen'

ii) 'Soften'

- Menggunakan kesan 'soften' untuk membentuk imej yang kabur dan sedikit tidak fokus.
- Caranya adalah dengan mengklik menu [Effect] dan pilih [Soften].
- Kesan 'soften' akan dilaksanakan ke atas imej dalam tetingkap 'preview'.

Pemprosesan yang berulangkali menggunakan kesan 'soften' akan menyebabkan imej terlalu kabur. Perlu berhati-hati agar kesan ini tidak digunakan banyak kali.

Rajah 2.7 : Bangunan Asal

Rajah 2.8 : Imej menjadi kabur setelah 'Soften' digunakan

iii) 'Despeckle'

- Menggunakan kesan 'despeckle' untuk melembutkan imej yang kasar. Kesan ini memindahkan kebisingan pada data imej.
- Kebisingan merupakan satu atau lebih piksel dengan kecerahan dan warna yang berbeza daripada sekumpulan piksel.
- Piksel adalah titik-titik yang membentuk imej.
- Caranya adalah dengan mengklik menu [Effect] dan pilih [Despeckle].
- Kesan 'despeckle' akan dijalankan ke atas imej dalam tetingkap 'preview'.

Pemprosesan imej dengan kesan 'despeckle' yang diulang beberapa kali akan menyebabkan imej kelihatan tebal.

Rajah 2.9 : Imej Asal

Rajah 2.10: Imej selepas kesan 'Despeckle'

2.2.3.5 Kajian 5 :

Adobe Photoshop 6.0

Satu lagi kajian telah dijalankan pada sistem yang sedia ada iaitu perisian Adobe Photoshop. Tiga jenis penapisan yang diambil adalah 'Despeckle', 'Sharpen' dan 'Sharpen More'.

Rajah 2.11 : Imej Asal

Rajah 2.12 : Imej selepas penggunaan penapis berjenis 'Despeckle'

Penapis kebisingan menambah atau membuang kebisingan, atau piksel dengan sebaran tahap warna secara rawak. Ini membantu percampuran pemilihan ke dalam persekitaran piksel. Penapis kebisingan boleh membentuk tekstur yang luar biasa atau memindahkan kawasan bermasalah seperti habuk atau goresan pada imej.

Salah satu contoh adalah penapis berjenis 'Despeckle'. Penapis 'Despeckle' akan mengesan sisi imej (kawasan di mana wujud perubahan warna) dan mengaburkan semua kawasan yang dipilih kecuali sisi-sisi tersebut. Pengaburan ini memindahkan kebisingan semasa memelihara butir-butir lain.

Rajah 2.13 : Imej selepas penggunaan penapis berjenis 'Sharpen'

Rajah 2.14 : Imej selepas penggunaan penapis berjenis 'Sharpen More'

Penapis 'Sharpen' memfokus imej-imej kabur dengan meningkatkan kontras piksel-piksel bersebelahan.

Jenis 'Sharpen' dan 'Sharpen More' memfokus kawasan pilihan dan meningkatkan kejelasannya. Penapis berjenis 'Sharpen More' lebih kuat kesan menajamkan imej berbanding penapis jenis 'Sharpen'.

2.3 Kesimpulan :

Daripada kajian yang telah dijalankan, memang banyak penapisan imej yang dijalankan iaitu bertujuan dalam bidang perubatan, pergigian, pemetaan, pendidikan dan sebagainya. Terdapat perisian yang boleh melaksanakan proses penapisan imej terdapat ianya masih tidak sesuai bagi semua golongan. Di harap sistem penapisan imej yang bakal dibangunkan ini akan memberikan banyak manfaat kepada semua yang menggunakan sistem ini iaitu pengguna sasaran bagi sistem ini.

3.0 METODOLOGI

3.1 Perancangan

Bahagian ini akan mengenengahkan perancangan yang akan dibuat terhadap sistem yang akan dibangunkan nanti. Sesuatu projek atau tugas yang hendak dilakukan memerlukan perancangan bagi memudahkan sesuatu matlamat yang dikehendaki tercapai menggunakan cara yang terbaik dan berkesan. Dalam tempoh perancangan, penentuan dan perancangan segala aktiviti-aktiviti yang terlibat dan kesan daripada aktiviti tersebut akan dibuat dalam usaha melaksanakan sesuatu tugas atau projek.

Perancangan untuk projek ini terdiri daripada dua perkara penting iaitu :

i) Kaedah yang digunakan untuk membangunkan aplikasi

ii) Jadual-jadual aktiviti yang terlibat sepanjang menjalankan projek

Perancangan untuk aplikasi Sistem Penapisan Imej yang dibangunkan adalah dengan menggunakan kaedah Kitar Hayat Pembangunan Sistem (KHPS) "System Development Life Cycle(SDLC)" atau lebih dikenali sebagai Model Air Terjun.

3.2 Metodologi Pembangunan Sistem

Metodologi pembangunan sistem yang digunakan bagi Sistem Penapisan Imej adalah kaedah Pemodelan Air Terjun.

3.2.1 Kaedah Pemodelan Air Terjun

Kaedah Pemodelan Air Terjun ini menggambarkan setiap peringkat pembangunan sistem seperti kitar air terjun yang berubah dari satu peringkat ke peringkat lain secara bertingkat-tingkat tanpa pertindihan.

Kaedah ini adalah salah satu daripada kejuruteraan perisian yang digunakan bagi memastikan langkah-langkah pembangunan sistem berjaya. Sebenarnya terdapat banyak kaedah metodologi pembangunan sistem yang digunakan dalam kejuruteraan sistem ini seperti model prototaip, model transformasi, model spiral dan banyak lagi. Kesemua metodologi ini bertujuan untuk memastikan proses pembangunan sesebuah sistem itu teratur dan mengikut kehendak pengguna.

Dalam setiap peringkat kitar hayat Sistem Penapisan Imej ini akan menerangkan tentang aktiviti-aktiviti dalam proses pembangunan sistem. Pemilihan langkah-langkah pembangunan yang teratur bertujuan untuk memastikan bahawa tujuan pembangunan sistem dan tujuan setiap bidang kerja diketahui oleh mereka yang terlibat dalam pembangunan sistem.

Metodologi Pemodelan Air Terjun ini telah terbukti berkesan dan diterimapakai sehingga kini. Penggunaan metodologi ini mempunyai beberapa kelebihan seperti berikut:

a) Perjalanan bagi setiap fasa adalah jelas. Oleh itu, perjalanan projek akan menjadi lebih teratur dan sempurna serta membolehkan kawalan kualiti terhadap hasil-hasil yang dicapai daripada projek pembangunan sistem.

b) Tiada pencampuran proses berlaku. Dalam model air terjun ini proses seterusnya hanya akan dilakukan setelah proses sebelumnya selesai.

c) Setiap tugas yang perlu dilaksanakan dalam setiap peringkat telah diberikan huraian dengan sepenuhnya dan melibatkan beberapa aktiviti yang lebih terperinci. Aktiviti-aktiviti bergabung dalam satu fasa yang perlu diselesaikan mengikut turutan fasa.

d) Pada setiap fasa terdapat penyelenggaraan. Ini adalah untuk menghapuskan atau mengurangkan kesilapan sepanjang perjalanan projek. Ini adalah penting agar setiap kesilapan dapat dikesan dan diperbaiki dengan lebih awal. Memandangkan sistem yang dijalankan ini mempunyai keperluan pengguna yang agak jelas serta tidak melibatkan pengguna sepanjang proses dilakukan, maka adalah lebih sesuai model air terjun digunakan.

Dalam model air terjun terdapat 6 fasa utama :

Fasa 1 : Definisi dan Analisis

Fasa 2 : Rekabentuk Sistem

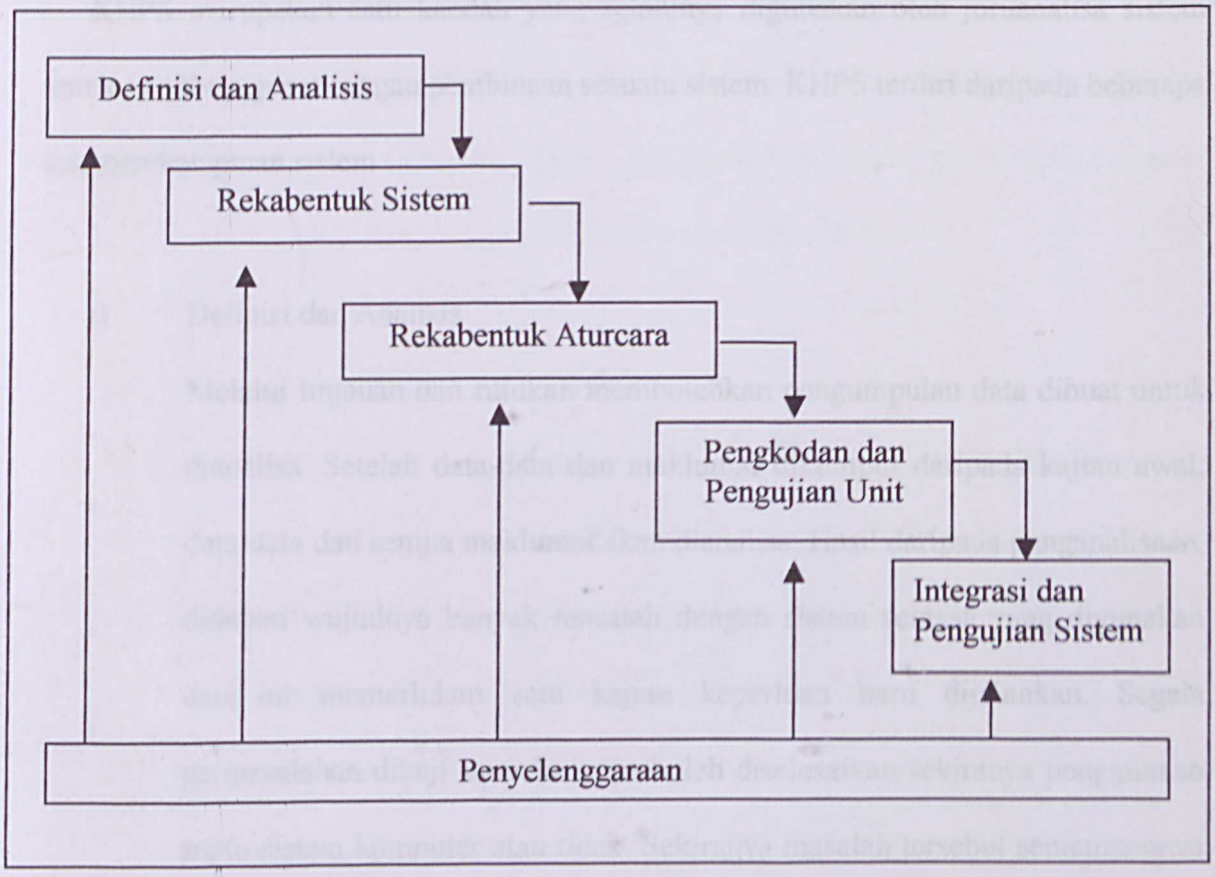
Fasa 3 : Rekabentuk Aturcara

Fasa 4 : Pengkodan dan Pengujian Unit

Fasa 5 : Integrasi dan Pengujian Sistem

Fasa 6 : Fasa Penyelenggaraan

Rajah di bawah menunjukkan secara gambaran aliran Kitar Hayat Pembangunan Sistem atau Pemodelan Air Terjun :



Rajah 3.1 : Gambaran Aliran Kitar Hayat Pembangunan Sistem (KHPS) atau Pemodelan Air Terjun

Berdasarkan kepada Rajah 3.1 dapat dinyatakan bahawa satu peringkat pembangunan sistem mestilah disempurnakan terlebih dahulu sebelum peringkat seterusnya dimulakan. Apabila semua keperluan diperolehi daripada pengguna,

maklumat dianalisa dari segi kekonsistenan dan kesempurnaan dan didokumentasikan dalam keperluan dokumen. Kemudian aktiviti rekabentuk akan mula dijalankan. Pemodelan Air Terjun ini boleh digunakan secara efektif untuk membantu pembangunan sistem dengan menakrifkan apa yang perlu dilakukan .

KHPS merupakan satu kaedah yang selalunya digunakan oleh juruanalisa sistem untuk membuat perancangan pembinaan sesuatu sistem. KHPS terdiri daripada beberapa fasa pembangunan sistem :

i) Definisi dan Analisis

Melalui tinjauan dan rujukan membolehkan pengumpulan data dibuat untuk dianalisa. Setelah data-data dan maklumat dikumpul daripada kajian awal, data-data dan semua maklumat akan dianalisa. Hasil daripada panganalisaan, didapati wujudnya banyak masalah dengan sistem semasa yang digunakan dan ini memerlukan satu kajian keperluan baru dijalankan. Segala permasalahan dikaji samada ianya boleh diselesaikan sekiranya penggunaan suatu sistem komputer atau tidak. Sekiranya masalah tersebut sememangnya boleh diselesaikan dengan suatu sistem komputer yang baik, maka ia diteruskan dengan fasa seterusnya.

ii) Rekabentuk Sistem

Fasa ini merupakan fasa yang agak sukar di mana data-data yang telah dikumpul dan dianalisa digunakan. Berdasarkan maklumat-maklumat dan data-data tersebut, rekabentuk sistem dibina. Dengan ini, aplikasi yang

dibangunkan nanti akan dapat dilaksanakan dengan sebaiknya tanpa sebarang masalah yang besar ditemui.

iii) Rekabentuk Aturcara

Fasa ini akan merekabentuk aturcara pula setelah selesai aktiviti merekabentuk sistem dilaksanakan. Rekabentuk aturcara dijalankan dengan sebaik-baiknya bagi mendapatkan hasil yang memenuhi seperti yang telah dirancangkan daripada awal. Rekabentuk aturcara amat penting untuk memastikan sistem dapat dilaksanakan dengan baik.

iv) Pengekodan dan Pengujian Unit

Di dalam fasa ini pengekodan program dilaksanakan selepas fasa rekabentuk aturcara. Proses pengekodan dibuat secara berhati-hati dan dilakukan secara berulang kali sehingga output yang dikehendaki dapat memenuhi apa yang dirancangkan. Pengujian unit perlu dijalankan untuk memastikan unit itu berjaya apabila pelbagai ujian dijalankan ke atasnya. Pengujian ini dijalankan untuk melihat sekukuh mana unit ini dapat bertahan jika pelbagai nilai masukan dimasukkan ke dalamnya.

v) Integrasi dan Pengujian Sistem

Fasa kelima adalah integrasi dan pengujian sistem. Di dalam fasa ini semua fasa sebelum ini diintegrasikan menjadi sebuah sistem. Kemudian, pengujian dijalankan ke atas sistem ini. Sistem diuji untuk memastikan sistem betul-

betul boleh digunakan sepenuhnya dan tiada sebarang masalah akan timbul selepas sistem digunakan.

vi) Penyelenggaraan

Fasa penyelenggaraan dilaksanakan untuk menyemak dan memperbaiki kesilapan sepanjang perjalanan projek. Ini memudahkan pengesanan kepada setiap kesilapan yang dibuat dari mula hingga akhir projek.

3.2.2 Perancangan Pembangunan Projek

Perancangan pembangunan projek telah ditunjukkan oleh Carta Gantt dalam Jadual 1.1.

4.0 ANALISA SISTEM

4.1 Definisi Analisa Sistem

Secara amnya, analisa sistem merupakan satu proses penganalisaan terhadap keperluan-keperluan sistem. Analisa keperluan sistem memerlukan kajian ke atas kelemahan sistem semasa yang sedang dijalankan dan seterusnya menjanakan satu perancangan dan konsep untuk menangani masalah-masalah yang timbul dari kelemahan tersebut.

Analisa keperluan sistem juga terdiri daripada analisa tentang khidmat-khidmat yang disediakan oleh sistem, kekangan-kekangan sistem dan matlamat sistem yang akan dijalankan. Di sini kefahaman kepada perisian juga diperlukan bagi memahami maklumat-maklumat domain perisian yang mana termasuklah fungsi yang diperlukan dan prestasi sesuatu perisian yang dipilih.

Fasa analisis sistem adalah satu fasa untuk mengkaji apa yang dapat dilakukan oleh sistem dan keperluan sistem yang sedia ada. Ini bermakna fasa ini dapat menentukan dua jenis keperluan sistem iaitu keperluan fungsian ("functional requirements") dan bukan fungsian ("nonfunctional requirements"). Serta keperluan perkakasan dan perisian untuk menyokong fungsi-fungsi yang telah dikenalpasti. Di dalam fasa ini juga segala objektif dan masalah sistem dikenalpasti untuk menghasilkan satu huraian berorientasikan pengguna ("user-oriented description").

4.2 Objektif Analisa Sistem

Objektif analisa sistem yang dijalankan adalah berdasarkan kepada tujuan berikut :-

i) Mengenalpasti apakah keperluan sebenar pengguna terhadap sistem yang akan dibangunkan

ii) Membolehkan pemilihan perisian yang baik dan benar-benar berkesan dalam memastikan matlamat dan objektif pembangunan sistem tercapai

iii) Membangunkan sebuah sistem yang benar-benar berkuasa dan cekap

iv) Menghasilkan analisa sistem yang lebih ekonomi dan teknikal

v) Menilai sejauh mana sistem yang akan dibina berbeza dengan sistem-sistem lain yang telah digunakan mahupun yang dicadangkan

4.3 Keperluan Sistem

Bagi memastikan sistem ini dapat beroperasi sepenuhnya dengan lancar, bahagian ini akan membincangkan tentang keperluan perkakasan dan perisian untuk perlaksanaan sistem. Keperluan pemilihan perisian dan perkakasan ini perlu bagi menjamin kemampuan sistem memenuhi objektif-objektif yang telah digariskan. Berikut merupakan perkakasan serta pemilihan perisian yang sesuai.

4.3.1 Spesifikasi Perisian

- Pemproses Mikro : Pentium, Pentium Pro, Pentium II, Pentium III, Pentium IV atau AMD Athlon based personal comp
- Microsoft Windows 95, Windows 98 (Original & second edition), Windows millenium Edition (ME), Windows NT 4.0 (with service Pack 5 for Y2K compliancy or Service Pack 6a) or Windows 2000.
- CD-ROM drive (bagi proses installation)
- 64 MB RAM minimum, 128 MB RAM
- 8-bit graphics adapter & display (untuk 256 simultaneous colors)

Aplikasi Pembangunan Sistem :

MATLAB 6.0

Matrix Laboratory atau MATLAB, mengintegrasikan kiraan, gambaran dan pengaturcaraan bersama-sama antaramuka pengguna. Masalah dan penyelesaian diekspresikan sebagai notasi matematik dalam MATLAB. Ia juga mempunyai ciri-ciri grafik untuk menggambarkan data. MATLAB menyediakan had untuk pembinaan struktur grafik 2 dimensi (2-D) dan 3 dimensi (3-D). MATLAB hanya bekerja dengan tatasusunan. MATLAB juga membenarkan penggunaan tatasusunan pelbagai dimensi secara fleksibel. MATLAB juga menyelesaikan masalah yang melibatkan tatasusunan lebih cepat berbanding dengan pengekodan dalam bahasa pengaturcaraan C atau Fortran. MATLAB mempunyai 'toolboxes' yang mengandungi aplikasi yang spesifik. Antara 'toolboxes' yang terdapat dalam MATLAB adalah pemprosesan imej. Perisian ini dipilih

kerana ia amat sesuai digunakan bagi pemprosesan imej. Perisian ini dapat menjelaskan imej, mengaburkan imej dan memperbaikinya di samping dapat digunakan seterusnya untuk tujuan pemprosesan imej tersebut.

4.3.2 Analisis Keperluan

Terdapat dua jenis keperluan yang perlu dititikberatkan iaitu keperluan fungsian dan keperluan bukan fungsian.

4.3.2.1 Keperluan Fungsian

Keperluan fungsian adalah fungsi atau kebolehan-kebolehan yang boleh dilakukan oleh sesuatu sistem itu. Keperluan fungsian juga menerangkan interaksi antara sistem dan persekitaran sistem atau penyelesaian implementasi ke atas masalah yang dihadapi oleh pengguna. Bagi sistem ini terdapat beberapa bahagian iaitu :-

i) Modul Pemilihan Domain

- Modul ini akan mengenalpasti jenis domain yang dipilih iaitu samada domain ruang atau domain frekuensi dan jenis penapis yang ingin digunakan.

ii) Modul Masukan Imej

- Modul ini membenarkan imej dimasukkan oleh pengguna ke dalam sistem iaitu gambar yang telah diimbas atau gambar daripada kamera digital.

iii) Modul Penapisan Imej

- Modul ini akan melaksanakan proses penapisan ke atas imej.

iv) Modul Cetakan

- Modul ini akan mencetak hasil akhir imej yang telah dijalankan proses penapisan ke atasnya. Ini adalah mengikut keinginan pengguna samada mahu mencetaknya atau tidak.

4.3.2.2 Keperluan Bukan Fungsian

Keperluan bukan fungsian adalah kekangan atau halangan yang menghadkan penyelesaian ke atas masalah sistem tetapi ia tidak akan menjejaskan fungsi sistem ini. Oleh itu, sistem ini mesti beroperasi untuk mengatasi kekangan ini.

Keperluan bukan fungsian bagi sistem ini adalah seperti berikut :

i) Kecekapan

- Kecekapan dalam teknologi komputer bermaksud sesuatu prosedur boleh dipanggil atau dicapai beberapa kali akan menghasilkan output yang sama bukannya berlainan

ii) Keberkesanan

- Keberkesanan bermaksud skrin input dan output mempunyai tujuan yang khusus dalam sistem

iii) Penjimatan masa

- Sekiranya sistem dapat menjimatkan masa, maka sisitem itu akan lebih menarik minat pengguna untuk menggunakan sistem itu. Ini kerana masyarakat masa kini sangat menghargai masa dan adalah sia-sia jika masa dibazirkan begitu sahaja.

5.0 REKABENTUK SISTEM

5.1 Rekabentuk Antaramuka Pengguna

Rekabentuk Antaramuka Pengguna merupakan salah satu aktiviti yang terkandung dalam rekabentuk sistem yang mana ia bermatlamat untuk menyediakan cara terbaik untuk pengguna berinteraksi dengan komputer, atau yang dikenali sebagai interaksi manusia-komputer (HCI). Rekabentuk antaramuka yang baik merupakan satu elemen yang dititikberatkan dalam pembangunan sistem ini kerana ia memberi kesan kepada kejayaan sistem dan kepuasan pengguna sistem. Pengguna tidak mahu menggunakan masa yang banyak untuk mempelajari sesuatu sistem. Mereka hanya cenderung untuk menjadikan komputer sebagai alat pemudah kerja. Oleh itu, antaramuka yang baik dapat membantu mencapai sasaran ini.

Terdapat beberapa panduan dalam merekabentuk antaramuka pengguna yang baik, antaranya adalah :

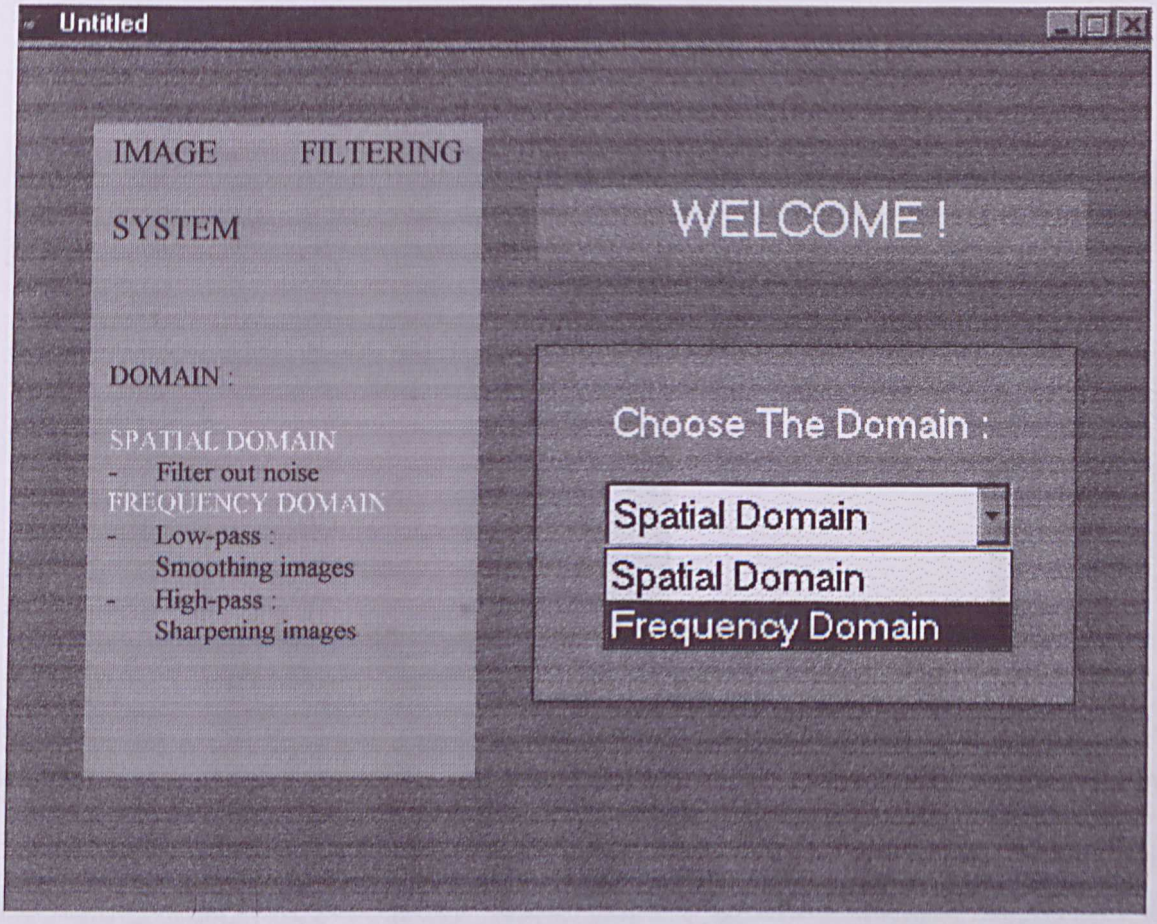
i) Memastikan antaramuka adalah mudah

ii) Memastikan persembahan antaramuka yang konsisten

iii) Menyediakan perkhidmatan pergerakan antaramuka pengguna yang jelas

iv) Mencipta antaramuka yang menarik

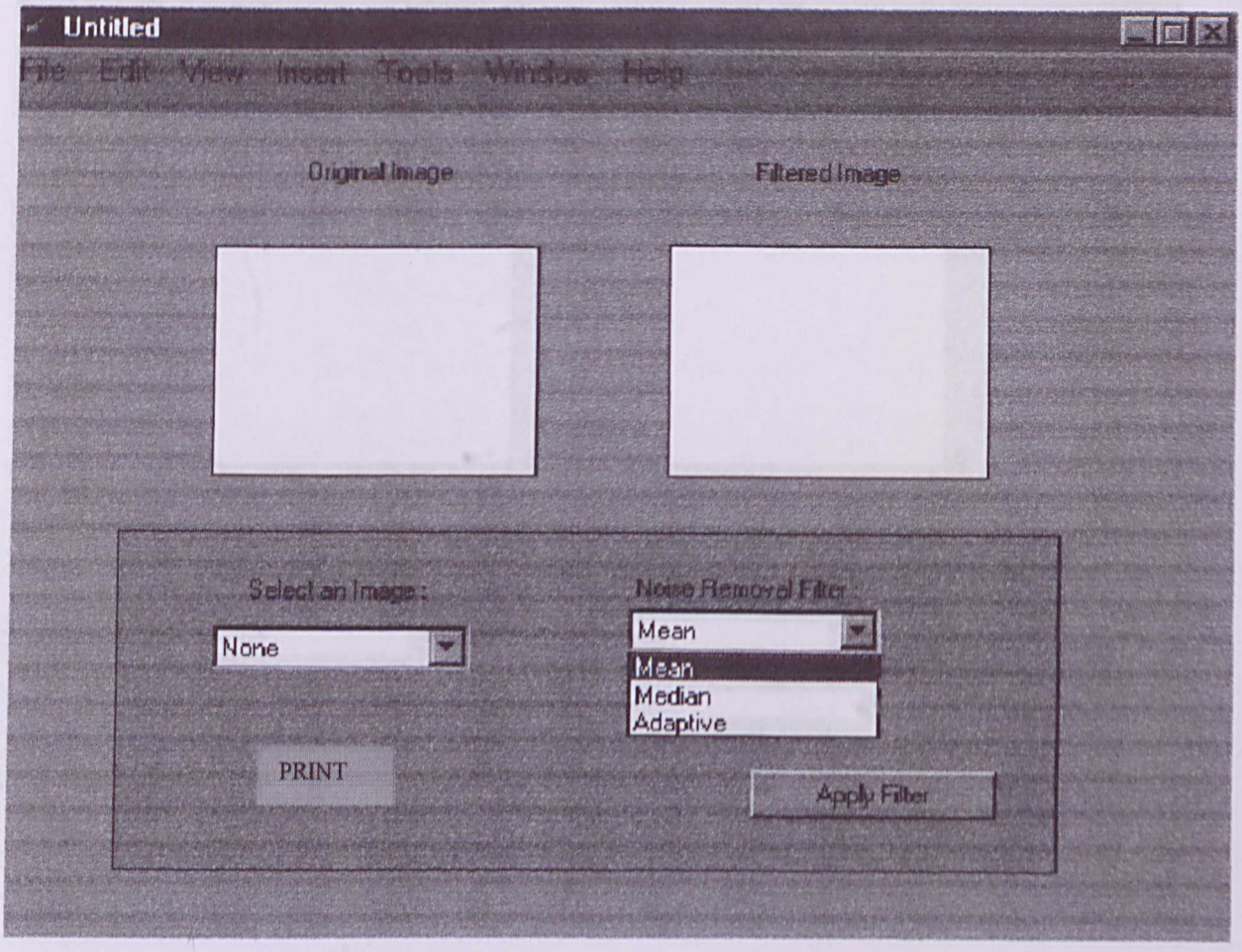
5.1.1 Skrin antaramuka utama



Rajah 5.1 : Skrin antaramuka pengguna bagi Sistem Penapisan Imej

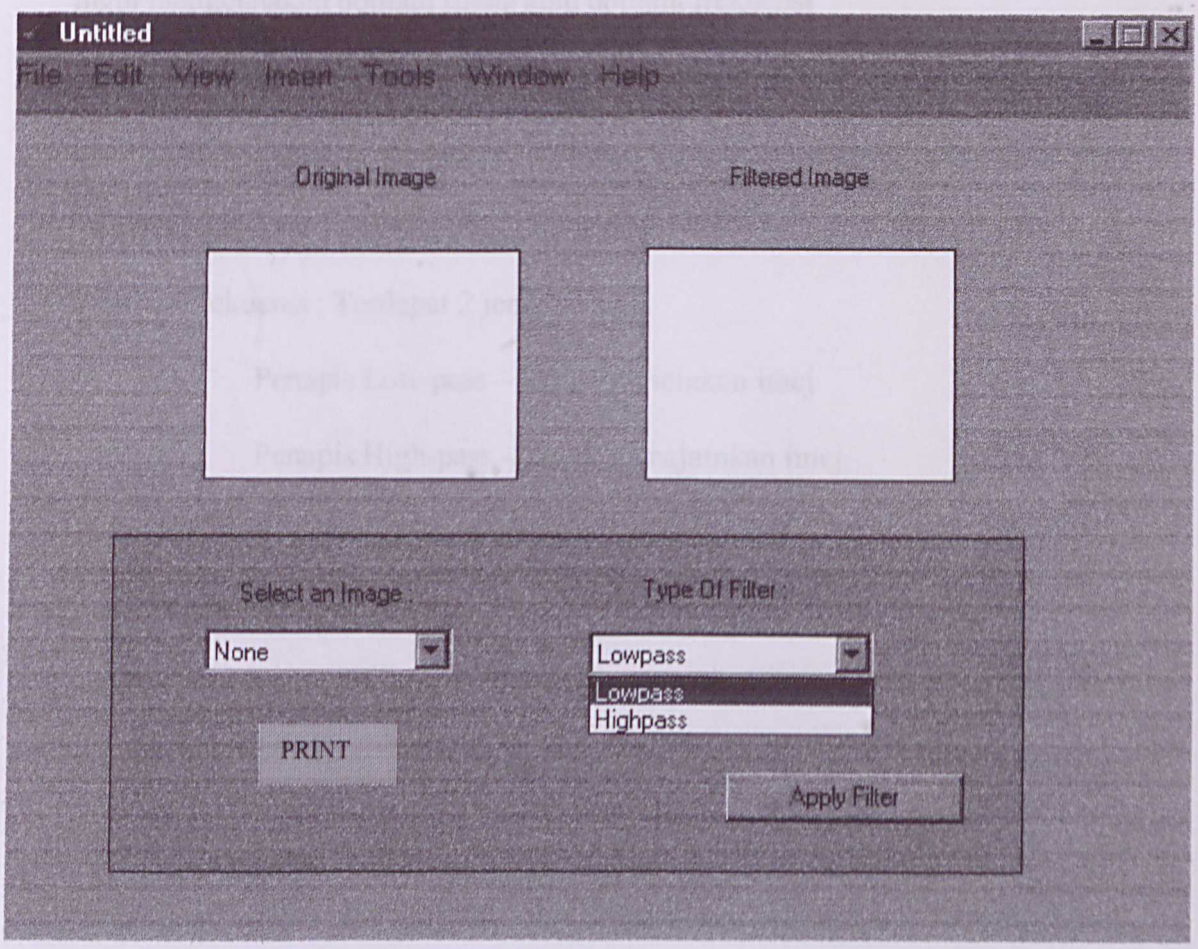
5.1.2 Skrin antaramuka mengikut domain

5.1.2.1 Skrin antaramuka mengikut domain ruang



Rajah 5.2 : Skrin antaramuka bagi domain ruang

5.1.2.2 Skrin antaramuka mengikut domain frekuensi



Rajah 5.3 : Skrin antaramuka bagi domain frekuensi

5.2 Proses-proses dalam sistem

Penerangan tentang proses-proses yang dijalankan oleh Sistem Penapisan Imej :

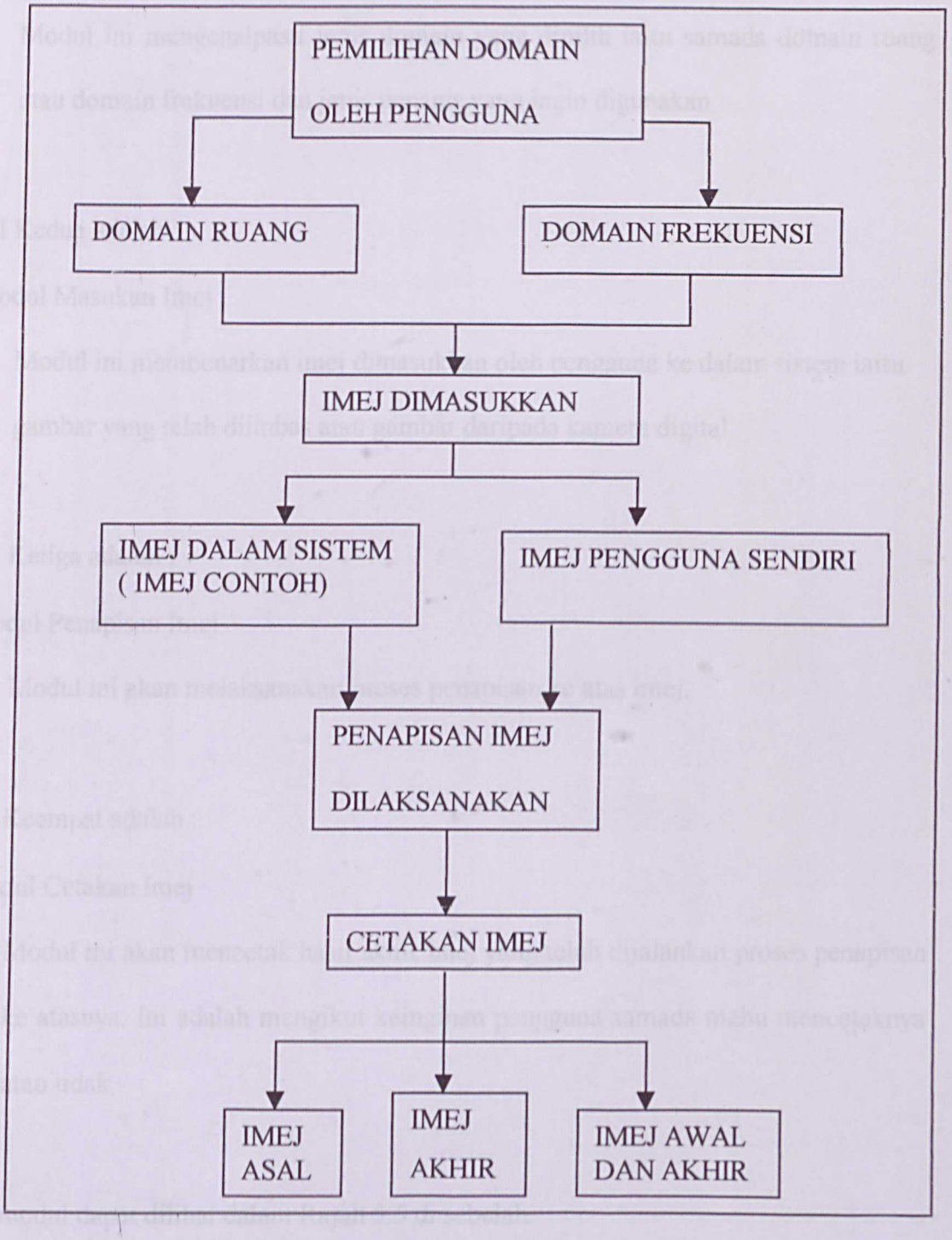
- Pemilihan domain oleh pengguna
  - Pada peringkat permulaan, pengguna akan membuat pilihan mereka iaitu samada ingin menggunakan domain ruang atau domain frekuensi
  - Domain Ruang : Terdapat 3 jenis penapis iaitu penapis mean, median dan peningkatan. Domain ini adalah bertujuan untuk membuang kebisingan yang terdapat pada imej
  - Domain Frekuensi : Terdapat 2 jenis penapis
    i) Penapis Low-pass – untuk melicinkan imej
    ii) Penapis High-pass – untuk menajamkan imej

- Imej dimasukkan oleh pengguna
  - Apabila selesai memilih domain yang diingini, pengguna akan memasukkan imej ke dalam sistem. Imej boleh diperolehi di dalam sistem ini iaitu imej yang disediakan oleh sistem sebagai percubaan. Atau pengguna boleh terus menggunakan imej sendiri yang ingin ditapis samada daripada kamera digital atau gambar yang diimbas.

- Proses penapisan imej dijalankan
  - Kemudian, proses penapisan dijalankan apabila butang "Apply Filter" digunakan. Imej yang dimasukkan oleh pengguna akan ditapis iaitu samada menyah-kebisingan, melicinkan atau menajamkan imej untuk tujuan peningkatan mutu imej.

- Proses cetakan imej dilaksanakan

  - Akhir sekali, proses mencetak imej dilaksanakan. Pengguna boleh membuat pilihan samada ingin mencetak imej asal atau imej setelah penapisan dijalankan atau kedua-dua imej sekali.



Rajah 5.4 : Proses-proses dalam Sistem Penapisan Imej

5.3 Modul-modul dalam Sistem Penapisan Imej

Modul Pertama adalah :

- Modul Pemilihan Domain
  - Modul ini mengenalpasti jenis domain yang dipilih iaitu samada domain ruang atau domain frekuensi dan jenis penapis yang ingin digunakan.

Modul Kedua adalah :

- Modul Masukan Imej
  - Modul ini membenarkan imej dimasukkan oleh pengguna ke dalam sistem iaitu gambar yang telah diimbas atau gambar daripada kamera digital.

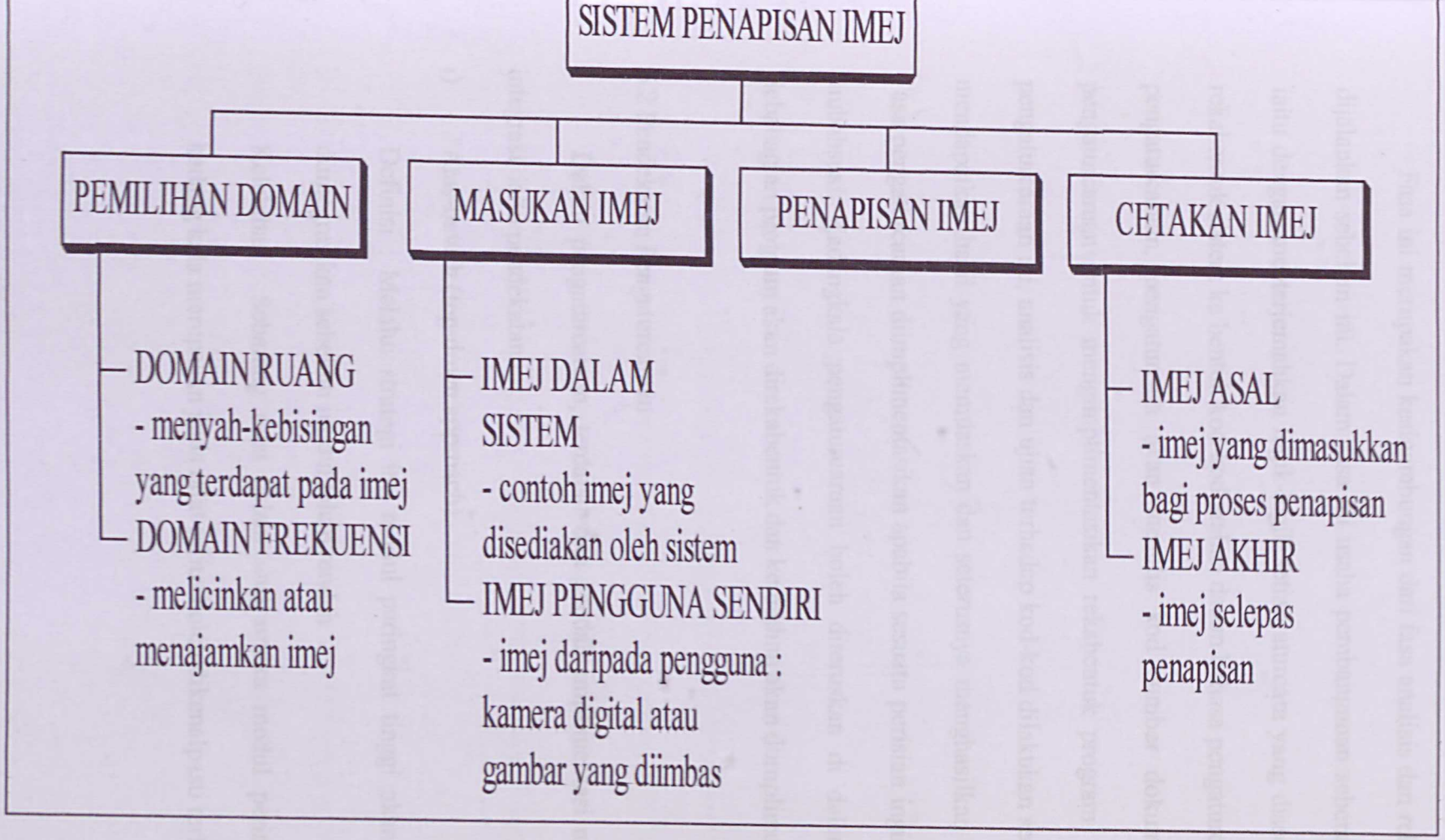
Modul Ketiga adalah :

- Modul Penapisan Imej
  - Modul ini akan melaksanakan proses penapisan ke atas imej.

Modul Keempat adalah :

- Modul Cetakan Imej
  - Modul ini akan mencetak hasil akhir imej yang telah dijalankan proses penapisan ke atasnya. Ini adalah mengikut keinginan pengguna samada mahu mencetaknya atau tidak.

Semua modul dapat dilihat dalam Rajah 5.5 di sebelah.



Rajah 5.5 : Modul-modul dalam Sistem Penapisan Imej

6.0 PERLAKSANAAN / PEMBANGUNAN SISTEM

6.1 Pengenalan

Fasa ini merupakan kesinambungan dari fasa analisis dan rekabentuk yang telah dijalankan sebelum ini. Dalam fasa ini usaha pembangunan sebenar sistem dilakukan iaitu dengan menterjemahkan logik-logik setiap aturcara yang disediakan semasa fasa rekabentuk sistem ke bentuk kod-kod arahan dalam bahasa pengaturcaraan. Semasa fasa pengaturcaraan, pengaturcara akan menulis kod sumber dokumen dalam bahasa pengaturcaraan untuk mengimplimentasikan rekabentuk program. Di sepanjang fasa pengaturcaraan ini, analisis dan ujian terhadap kod-kod dilakukan secara konsisten bagi mendapatkan hasil yang memuaskan dan seterusnya menghasilkan sistem tanpa ralat. Fasa pengaturcaraan diimplimentasikan apabila sesuatu perisian ingin dibina atau ingin diubahsuai. Kadangkala pengaturcaraan boleh diteruskan di dalam fasa pengujian. Sebahagian program akan direkabentuk dan kemudian akan diimplimentasikan dan diuji.

6.2 Pendekatan Pengaturcaraan

Dalam pengaturcaraan, terdapat tiga kaedah implimentasi modul dan strategi integrasi, iaitu pendekatan :

i) Atas-bawah (top-down approach)

Definisi : Melalui strategi ini, modul peringkat tinggi akan dikod, diuji dan diintegrasikan sebelum modul tahap rendah.

Kelebihan : Sebarang ralat dalam antaramuka modul peringkat tinggi yang kadang kala merupakan jenis ralat serius boleh dikenalpasti terlebih dahulu.

Kekurangan : Ujian program mungkin menjadi sukar apabila modul peringkat rendah mempersembahkan fungsi-fungsi kritikal operasi.

ii) Bawah-atas (bottom-up approach)

Definisi : Pendekatan ini bermula dengan pengekodan, pengujian dan integrasi modul peringkat bawah manakala modul peringkat tinggi hanya bertindak sebagai rangka sahaja.

Kelebihan : Modul peringkat rendah yang kritikal kepada operasi program akan diimplimentasi dan diuji terlebih dahulu. Dalam beberapa kes, modul yang dibangunkan untuk aplikasi lain mungkin diguna semula atau diubahsuai.

Kekurangan : Sukar untuk membuat pemerhatian terhadap keseluruhan operasi program selagi ianya tidak sempurna dan diintegrasikan dengan modul peringkat atas.

iii) Bebenang (threads approach)

Keputusan dibuat dalam turutan di mana fungsi program harus diimplimen. Modul yang menyokong setiap fungsi akan ditentukan dan setiap set akan diimplimentasikan dalam turutan menurun mengikut kepentingan fungsian.

Kelebihan : Fungsi yang paling penting akan diimplimentasi terlebih dahulu.

Kekurangan : Integrasi modul yang berikutnya mungkin sukar jika dibandingkan dengan kedua-dua pendekatan di atas.

Di dalam sistem ini, saya telah menggunakan pendekatan bawah-atas iaitu bermula dengan modul peringkat bawah kemudian barulah diintegrasikan dengan modul peringkat atas.

6.3 Dokumentasi pengaturcaraan :

Dokumentasi suatu aturcara merupakan suatu set penjelasan yang menerangkan kepada pembaca tentang apa yang dilakukan oleh aturcara dan bagaimana aturcara tersebut melaksanakannya.

Dokumentasi kod sumber berkualiti tinggi adalah penting untuk mengurangkan ralat pengekodan apabila suatu program ditulis dan memudahkan penyelenggaraan program yang berikutnya dilaksanakan.

6.4 Pendekatan Pengaturcaraan

Kemahiran pengaturcaraan yang baik akan menghasilkan sistem yang mudah diselenggarakan. Pendekatan pengaturcaraan yang baik kebiasaannya memerlukan beberapa kriteria yang penting. Kerana itulah, sistem ini berlandaskan beberapa perkara seperti di bawah :

a) Kebolehbacaan

   Kod aturcara hendaklah boleh dibaca oleh pengaturcara lain tanpa menghadapi sebarang masalah. Ini memerlukan pemilihan nama pembolehubah, komen yang disertakan dan penyusunan keseluruhan aturcara

b) Teknik penamaan yang baik

   Penamaan yang baik bermaksud bahawa nama yang diberikan kepada pembolehubah, kawalan dan modul dapat menyediakan identifikasi yang mudah kepada pengaturcara. Penamaan ini dilakukan dengan kod yang konsisten dan seragam.

c) Dokumentasi dalaman

Dokumentasi dalaman di dalam kod pengaturcaraan adalah penting untuk menambahkan pemahaman. Ini biasanya merujuk kepada komen dalaman yang disediakan sebagai panduan untuk memahami aturcara terutamanya dalam fasa penyelenggaraan.

6.5 Permulaan pembangunan sistem

Pada awalnya saya bercadang menggunakan MATLAB di dalam mengimplimentasikan sistem saya ini iaitu sistem penapisan imej. Saya telah membuat kajian ke atas perisian MATLAB dan cuba sedaya upaya mempelajari perisian ini sehingga berjaya. Saya juga telah menambah Matlab Add-in ke dalam Microsoft Visual C++ untuk menjadikan sistem saya suatu sistem stand-alone. Saya berjaya melaksanakan sistem ini di dalam MATLAB tetapi tidak dapat menjayakannya di dalam Microsoft Visual C++. Ini adalah kerana terdapat beberapa fungsi MATLAB yang tidak menyokongnya apabila ia menjadi suatu sistem stand-alone. Penukaran aturcara MATLAB kepada Visual C++ menyebabkan berlaku banyak ralat. Contonya, di dalam MATLAB, aturcara itu berjaya dilarikan, tetapi setelah diterjemah kepada Visual C++ terdapat banyak ralat. Saya terus mencuba mengkaji segala ralat yang berlaku tetapi tidak mampu untuk menyelesaikannya. Kemudian, saya telah membuat kajian tentang Photoshop®. Adobe Photoshop mempunyai bahasa pengaturcaraannya sendiri yang membenarkan pengguna untuk mencipta penapis mereka sendiri. Saya menjumpai beberapa sumber dan beberapa laman web dengan penapis yang direka menggunakan Photoshop® Filter Factory. Kelebihan yang saya dapati dalam penggunaan Filter Factory adalah mempelajari algoritma penapisan imej

seperti yang boleh digunakan dalam Photoshop®. Sistem yang saya bangunkan ini adalah sistem penapisan imej atau dipanggil FilterExplorer, yang membenarkan imej dibuka, kemudian dipaparkan pada skrin, algoritma penapisan dijalankan ke atas imej dan hasilnya dapat dilihat. Pengimplimentasian program Windows® GUI yang memudahkan imej dibuka dengan cepat menjadi beban sebelum saya perlu fikirkan tentang implimentasi penapis. Ini menyebabkan saya perlu mempelajari teknik dalam Visual C++®. Saya telah menjumpai sumber yang baik dalam Internet yang memberikan kod untuk membuka dan menyimpan imej JPEG dan BMP dalam aplikasi MFC (Microsoft® Foundation Class) Visual C++®. Selain daripada perubahan daripada penggunaan perisian MATLAB kepada MFC, perubahan lain yang saya lakukan adalah jenis-jenis penapis yang digunakan dalam sistem saya ini.

6.5.1 Penerangan tentang jenis penapis yang digunakan

Terdapat empat kategori algoritma penapisan atau pemprosesan imej iaitu proses titik, proses kawasan, proses kerangka dan proses geometrik. Proses titik mengubah setiap piksel di dalam imej hanya berdasarkan nilai sebenar piksel. Proses kawasan mengubah setiap piksel berdasarkan nilai piksel sebenar dan nilai piksel di sekelilingnya. Piksel di sekelilingnya selalunya didapati dengan memilih piksel radius dan segiempat sama 'kernel' (dengan radius yang dipilih) di sekitar piksel yang berada di tengah-tengah. Kesemua piksel di dalam 'kernel' diproses mengikut turutan untuk mendapatkan nilai baru bagi piksel yang di tengah. Proses kerangka digunakan untuk menggabungkan imej; nilai piksel yang baru ditentukan dengan hubungan antara satu atau lebih imej yang

berkaitan dengan nilai piksel. Proses geometrik adalah sama seperti proses titik (iaitu hanya satu nilai piksel digunakan untuk menentukan nilai piksel baru), tetapi nilai piksel yang baru itu adalah berdasarkan beberapa transformasi geometrik. Sistem yang dibina ini merangkumi kesemua kategori di atas kecuali proses kerangka. Ini kerana penapisan kerangka melibatkan penggunaan lebih daripada satu imej dan ianya akan menjadikan sistem ini bertambah kompleks, jadi, ianya tidak dilaksanakan. Berpandukan keterangan di atas, penapis titik hanya akan mengubah warna, atau nilai imej, contohnya, dalam penukaran imej berwarna kepada imej penskalaan kelabu atau sebaliknya. Penapis kawasan selalunya mengekalkan bentuk imej asasnya tetapi mengubah imej lebih daripada menukarkan warna imej tersebut. Penapis geometrik selalunya membiarkan sahaja warna imej tersebut tetapi memindahkan pikselnya. Saya telah membahagikan penerangan tentang penapis ini kepada empat seksyen iaitu penapis titik, penapis kawasan, penapis geometri dan penapis artistik. Penapis artistik adalah lebih kepada penapis kawasan yang lebih tegas atau penggunaan kombinasi beberapa teknik.

6.5.2 Implimentasi Sistem Penapisan Imej / FilterExplorer

Sekarang kita akan lihat kepada implimentasi aplikasi FilterExplorer, termasuk beberapa perilaku yang melibatkan di mana imej disimpan dan dimanipulasi dan fungsi sokongan imej dalam implimentasi penapis. Implimentasi dan fungsi-fungsi sokongan FilterExplorer adalah aplikasi Microsoft® Windows® 32-bit GUI yang dilarikan pada versi Windows® selepas Windows® 95. Aplikasi ini ditulis dalam Microsoft® Visual C++® Version 6.0

Standard Edition menggunakan MFC AppWizard. AppWizard menghasilkan aplikasi asas Windows® GUI secara automatik menggunakan MFC bagi antaramuka dokumen tunggal. Ini bermaksud saya telah menulis kod untuk mengisi dokumen tunggal, dalam kes ini ia dimaksudkan dengan paparan imej (antaramuka dokumen tunggal bermaksud hanya satu imej boleh dibuka pada satu-satu masa). Menggunakan kod yang dicari di dalam laman web, saya boleh membuka dan menyimpan imej JPEG dan BMP kepada dan daripada aplikasi saya. Seperti yang dibaca daripada fail, kod menyimpan tinggi dan lebar imej sebagai 'unsigned integers' dan imej sebagai bait-bait penimbal. Penimbal imej boleh dianggap sebagai tatasusunan satu dimensi bait-bait, tiga darab panjang darab lebar imej. Piksel adalah dengan panjang tiga bait, satu bait bagi setiap nilai merah, hijau dan biru. Penimbal imej boleh ditulis seperti di bawah :

buf = (BAIT*)BAIT baru(tinggi * lebar * 3);

Di dalam dua fail iaitu filters.h dan filters.cpp, saya telah isytihar jenis : pixel, hsvpixel, and imagematrix, dan kelas : CImage dan CFilter (sebenarnya, pixel dan hsvpixel adalah diimplimen sebagai kelas juga). Pixel mengandungi tiga bait, satu bagi setiap komponen merah, hijau dan biru. Operasi telah ditakrifkan bagi pixel untuk menetapkan warna piksel (komponen RGB), mendapatkan intensiti piksel dan mencerahkan atau menggelapkan piksel. Hsvpixel bukan untuk komponen RGB tetapi ditakrifkan untuk Hue, Saturation, dan Value (semua diimplimentasi sebagai berjenis double). Imagematrix ditakrifkan sebagai tataasusunan dua dimensi (menggunakan notasi penunjuk):

typedef pixel** imagematrix;

Untuk memudahkan pengaturcaraan (dan kebolehbacaan), manipulasi imej hanya dilakukan apabila imej adalah di dalam ruang imagematrix. Contohnya, piksel putih mempunyai komponen RGB yang ditetapkan kepada 255. Dengan itu, piksel pada koordinat (x,y) dalam imagematrix img kepada putih akan melaksanakan perkara di bawah:

```
img[y][x].r = 255;
img[y][x].g = 255;
img[y][x].b = 255;
```

Atau, terus memanggil fungsi color(r, g, b) :

```
img[y][x].color(255,255,255);
```

Atau dengan lebih ringkas lagi menggunakan fungsi color() :

```
img[y][x].color(255);
```

yang mana telah menetapkan komponen kepada 255. Perhatikan bahawa contoh yang diberi di atas bahawa koordinat (x,y) telah diterbalikkan. Beginilah cara bagaimana imej disimpan. (Juga, secara terus daripada fail, imej JPEG dan BMP disimpan dalam susunan bait BGR). Kelas CImage mengandungi ahli yang menerangkan dan memanipulasi imej daripada ruang penimbal. Fungsi ini memuat dan menyimpan imej, memaparkan imej pada skrin atau memanggil penapis. Fungsi yang memanggil penapis membina satu objek Cfilter berjenis seperti yang dipanggil dan memanggil fungsi ahli CFilter go() untuk melarikan penapisan itu :

```
void CImage::ProcessFilter(enum filtertype type)
{
CFilter currentfilter(this, type);
```

```
currentfilter.go();
}
```

Apa yang tersembunyi di sini adalah pembina CFilter menukarkan ruang penimbal imej kepada imagematrix dan menetapkan jenis penapis. Terdapat satu fungsi umum iaitu go(). Ia memanggil fungsi yang sesuai mengikut jenis objek CFilter. Apabila go() kembali, jika berjaya, penimbal CImage mengandungi imej yang telah ditapis dan penimbal 'undo' (juga adalah ahli pembolehubah CImage) direka dengan salinan imej itu sebelum ditapis. Ini membenarkan aplikasi atau sistem saya ini untuk persembahkan satu tahap 'undo' dengan menukarkan ruang penimbal dengan penimbal utama apabila pengguna membuat permintaan ke atas 'undo'.

Kelas CFilter adalah yang paling menarik sekali dalam kajian ini. Kesemua kod sebelum kod CFilter hanyalah implimentasi sahaja. CFilter mengandungi fungsi untuk menukarkan kepada dan daripada ruang penimbal dan imagematrix. Sebagai contoh, untuk menukarkan daripada penimbal kepada imagematrix, perlu diperuntukkan memori bagi imagematrix:

```
imagematrix CFilter::CreateImageMatrix(UINT row, UINT col)
{
imagematrix img = NULL;
img = (imagematrix)malloc(row * sizeof(imagerow));
if (img == NULL)
return(NULL);
for (unsigned int i = 0; i < m_height; i++)
```

```
{
img[i] = (imagerow)malloc(col * sizeof(pixel));
if (img[i] == NULL)
{
FreeImageMatrix(img, row);
return(NULL);
}
}
return(img);
}
```

and then do the simple conversion

```
void CFilter::BufferToMatrix(BYTE* buf, imagematrix img)
{
UINT col, row;
for (row=0; row < m_height; row++)
{
for (col=0; col < m_width; col++)
{
LPBYTE pRed, pGrn, pBlu;
pBlu = buf + row * m_width * 3 + col * 3;
pGrn = buf + row * m_width * 3 + col * 3 + 1;
pRed = buf + row * m_width * 3 + col * 3 + 2;
img[row][col].r = *pRed;
img[row][col].g = *pGrn;
```

```
img[row][col].b = *pBlu;
}
}
}
```

Apabila telah selesai dengan imagematrix, memori tidak perlu diperuntukkan. Ini amat penting dalam sistem ini, kerana bagi imej yang besar, amaun memori yang diperuntukkan kepada imagematrix adalah besar. Dengan memanggil beberapa rutin penapis tanpa membersihkan memori yang diperuntukkan akan menyebabkan penggunaan semua memori yang terdapat dalam sistem. Fungsi pemindahan ditunjukkan di bawah:

```
void CFilter::FreeImageMatrix(imagematrix img, unsigned int row)
{
for (unsigned int i = 0; i < row; i++)
free(img[i]);
free(img);
}
```

Penggunaan sistem yang dibangunkan ini iaitu FilterExplorer adalah bersesuaian kerana ia berfungsi seperti aplikasi Windows® yang lain. Imej akan dibuka dan disimpan daripada menu File. Apabila imej dibuka, ia akan dipaparkan pada skrin dan namanya akan terpapar pada bar tajuk. Mana-mana penapis pada menu Filter boleh dilaksanakan dengan memilih penapis daripada menu dan memilih pilihan dalam kotak dialog (sekiranya ada). Terdapat satu tahap 'undo', juga boleh didapati daripada menu Filter. Tambahan kepada penngunaan menu, butang toolbar boleh juga digunakan.

Langkah-langkah seterusnya muncul setelah fail dibuka dan pengguna memilih penapis dalam menu Filter:

· Dokumen paparkan ahli kelas (CFilterExplorerView) pembolehubah currentimage (berjenis CImage) telah sedia ada pada skrin (ini berlaku apabila fail dibuka).

· Ahli Fungsi ProcessFilter() currentimage dipanggil. Ia mengambil satu parameter, jenis (atau nama) penapis, di mana ditentukan dengan memilih penapis mana yang diklik daripada menu.

· ProcessFilter() mencipta satu objek berjenis CFilter yang mana telah dipindahkan ke dalam fungsi dan memanggil ahli objek fungsi go().

· Fungsi penapis (dalam Cfilter) yang bertanggungjawab dipanggil.

· Fungsi penapis memanggil kotak dialog yang bersesuaian dan menunggu input daripada pengguna. Apabila pengguna menekan butang OK, dua jenis imagematrix direka. Satu imagematrix (img) direka daripada penimbal imej sebenar (seperti yang dipaparkan pada skrin). Satu lagi (filt) adalah kosong, ia akan menjadi imej baru yang akan terhasil selepas penapisan dijalankan.

· Penapis memproses img dan membina filt.

· Sekiranya pengguna tidak membatalkan sebelum penapisan selesai dijalankan, penimbal imej semasa disalin kepada penimbal 'undo', matriks filt diterjemahkan kepada penimbal, dan dijadikan semasa (dan dipaparkan pada skrin). Perantaraan img dan matriks filt dimusnahkan.

6.5.3 Penerangan tentang penapis

Penapis disusun mengikut jenis, seperti yang diterangkan sebelum ini. Kebanyakan penapis ini dimodelkan daripada Photoshop®. Penapis yang terakhir disenaraikan iaitu 'raindrop effect' adalah dalam bahagian yang menggabungkan kombinasi 'hue/saturation', 'blur', 'fish eye lens', dan 'random blur filters'. Ini adalah senarai implimentasi penapis dalam FilterExplorer. Untuk melihat penerangan, sila lihat di Apendiks A. Implimentasi penuh bagi setiap penapis dijumpai dalam filters.h dan filters.cpp yang terdapat pada Apendiks B.

Penapis Titik (Point Filters) :

· Grayscale

· Hue / Saturation

· Equalize Intensity

Penapis Kawasan (Area Filters) :

· Blur

· Median Blur

· Custom Convolution

Penapis Geometrik (Geometric Filters) :

· Fish-Eye Lens

· Polar Coordinates

Penapis Artistik (Artistic Filters) :

· Oil Paint

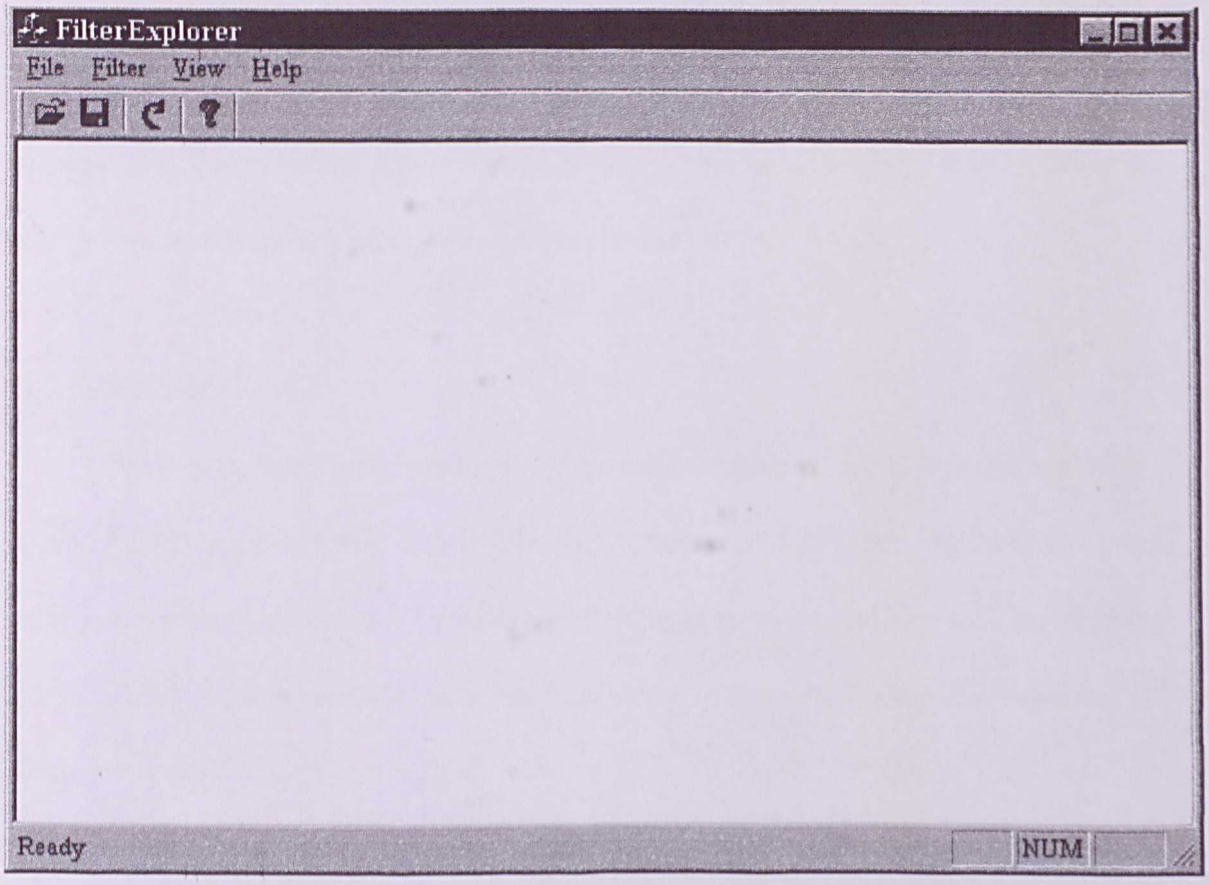
· Frosted Glass

· Random Blur

· Raindrops

6.5.4 Perubahan dalam antaramuka pengguna

Oleh kerana penukaran perisian berlaku, ia memberi kesan kepada antaramuka pengguna yang telah saya rancang pada awal iaitu dalam bab rekabentuk sistem. Modul pemilihan domain dan modul cetak juga tidak dapat dilaksanakan akibat kekurangan masa dalam mempelajari MFC. Di bawah ini ditunjukkan antaramuka bagi sistem yang telah dibangunkan.



6.6 Kesimpulan

Dalam bab perlaksanaan dan pembangunan sistem ini, banyak perubahan yang telah saya lakukan. Antara perubahannya adalah penukaran penggunaan perisian daripada MATLAB kepada MFC tetapi masih menggunakan Microsoft Visual C++ sebagai asasnya. Selain daripada itu, satu lagi perubahan adalah jenis-jenis penapis yang digunakan. Oleh kerana berlaku perubahan-perubahan ini, antaramuka pengguna juga turut berubah. Segala yang dirancang pada awalnya mengalami perubahan besar dalam sistem yang dibangunkan ini.

7.0 PENGUJIAN SISTEM

7.1 Pengenalan

Proses-proses pengujian akan dilakukan bagi mengesan sebarang ralat yang terdapat di dalam kod aturcara dan seterusnya membetulkannya sehingga dapat dilarikan dengan baik dan sempurna. Fasa ini merupakan satu fasa untuk memastikan objek-objek yang telah ditetapkan dan dikehendaki tercapai. Ia menerangkan tentang perlaksanaan dan langkah-langkah yang diperlukan dalam melakukan konfigurasi sistem. Pengujian sistem pula merupakan aspek penting bagi menentukan tahap kualiti sesuatu sistem dan ia mewakili dasar pertimbangan ke atas spesifikasi, rekabentuk dan pengkodan bagi memastikan sistem dilaksanakan mengikut spesifikasinya dan sejajar dengan keperluan pengguna. Ia merupakan satu proses pengesahan sistem.

7.2 Perlaksanaan :

Proses perlaksanaan merangkumi perubahan dari struktur rekabentuk sistem kepada kod program yang boleh dilarikan. Proses perlaksanaan ini berlaku secara ulangan sehingga memenuhi keperluannya, selepas proses penilaian ini dilaksanakan, pengumpulan maklumat-maklumat yang berkaitan dengan ralat akan dikumpulkan dan dianalisa.

Pembangun perlu membetulkan ralat ini dengan kaedah 'debug' atau pengubahsuaian aturcara program yang berkaitan dengan ralat yang dikesan. Setelah segala ralat telah dibetulkan dan aturcara telah berjaya dilarikan, sistem akan diuji.

7.3 Pengujian

Semasa fasa pengujian, program yang dibangun atau diperlukan akan dinilai untuk menentukan samada ianya memenuhi keperluan yang ditentukan. Pengujian boleh mengenalpasti ralat dalam program rekabentuk atau ralat dalam pengekodan program. Dalam kes tertentu, pengujian mungkin boleh menunjukkan dengan tepat spesifikasi yang tidak betul atau tidak lengkap.

Namun demikian, perlu diambil kira bahawa pengujian hanya boleh mengecam kehadiran ralat. Ianya tidak boleh mengecam ketiadaan ralat tersebut dalam program yang dibangunkan.

Antara beberapa peraturan yang perlu dipatuhi untuk mencapai objektif pengujian adalah :

i) Pengujian adalah proses melaksanakan aturcara untuk mengesan ralat

ii) Kes ujian yang baik perlu mempunyai kebarangkalian yang tinggi dalam mengesan ralat yang dijangka berlaku

iii) Ujian yang berjaya adalah ujian yang dapat mengatasi ralat yang dijangka berlaku

7.3.1 Langkah-langkah pengujian

Fasa pengujian adalah pelbagai, bergantung kepada faktor saiz program dan kebendaannya. Walaubagaimana pun, pengujian melibatkan tujuh langkah utama :

i) Pilih sempadan pengujian

ii) Tentukan matlamat pengujian

iii) Pilih pendekatan pengujian

iv) Bangunkan pengujian

v) Pimpin pengujian

vi) Penilaian keputusan pengujian

vii) Dokumentasikan pengujian

7.3.2 Pendekatan proses pengujian

Kebanyakan pembangun sistem menggunakan tiga pendekatan semasa proses pengujian, iaitu :

i) Pengujian fungsi (Black-box test)

-Pengujian berdasarkan spesifikasi program

ii) Pengujian struktur (White-box test)

-Pengujian berdasarkan kepada pengetahuan pembangun sistem mengenai struktur dan implimentasi program

iii) Pengujian antaramuka

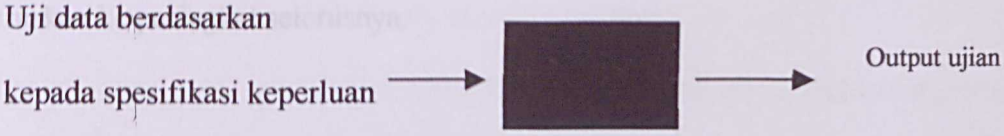
-Spesifikasi program dan pengetahuan berkenaan antaramuka dalaman diuji dan ianya sangat penting kepada pembangunan sistem berorientasikan objek.

Pendekatan tersebut boleh digunakan untuk sebarang tahap atau peringkat dalam proses pengujian. Dalam sistem ini, pendekatan yang ditekankan adalah Pengujian Fungsi dan Pengujian Antaramuka.

1) Pengujian Fungsi

Pengujian dijalankan berdasarkan kepada spesifikasi keperluan modul-modul yang dibina. Tumpuan pengujian adalah kepada input dan output yang dijangkakan bagi sesuatu sistem. Matlamat pengujian adalah untuk menentukan samada setiap fungsi

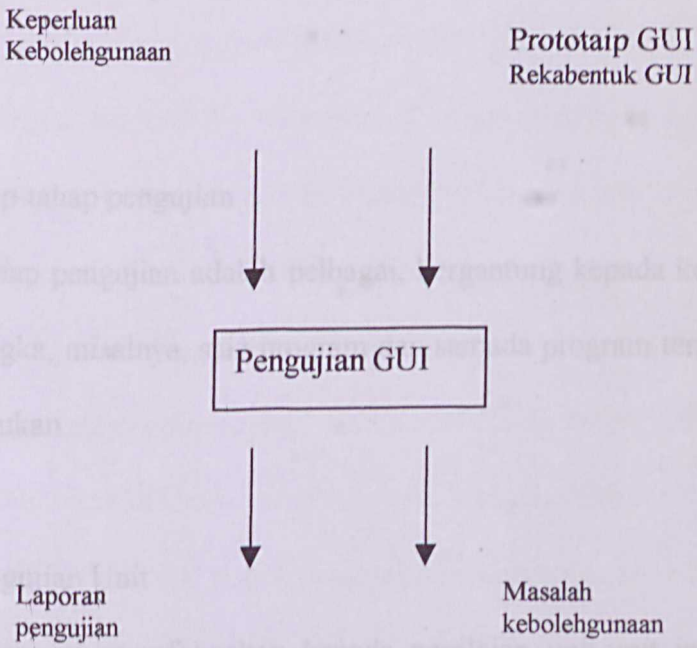
dapat dilaksanakan sepenuhnya atau tidak dan dalam masa yang sama pengujian mencari sebarang ralat yang mungkin wujud dalam setiap fungsi berkenaan.



Rajah 7.1 : Pengujian Black-box

2) Pengujian Antaramuka

Menekankan pengujian rekabentuk GUI dan prototaip yang dibina. Dalam proses GUIDE, sekurang-kurangnya perlu ada tiga kitaran pengujian.



Rajah 7.2 : Input dan output pengujian

Pengujian antaramuka memberikan asas maklumat untuk membuat membuat keputusan samada rekabentuk GUI dan prototaip adalah lengkap. Jika sasaran kebolehgunaan dapat dicapai dan tiada masalah serius kebolehgunaan ditemui, pembangunan GUI boleh diteruskan ke peringkat seterusnya.

Input utama pengujian GUI adalah :

i) Keperluan kebolehgunaan yang dipersetujui

ii) Prototaip GUI dan dokumentasi rekabentuk GUI

Manakala output bagi pengujian GUI merupakan laporan penilaian yang mengandungi masalah pengguna. Keperluan kebolehgunaan boleh diubahsuai berdasarkan pengalaman yang diperolehi semasa proses pengujian. Selain itu, model objek pengguna dan stail panduan aplikasi juga boleh diubahsuai sebagai satu keputusan masalah semasa proses pengujian.

7.3.3 Tahap-tahap pengujian

Tahap pengujian adalah pelbagai, bergantung kepada kepelbagaian situasi yang tidak dijangka, misalnya, saiz program dan samada program tersebut telah dibangunkan atau diperlukan.

7.3.3.1 Pengujian Unit

Pengujian ini memfokuskan kepada penilaian unit-unit individu dalam program. Ujian unit ini adalah melibatkan :

- Pengujian antaramuka untuk memastikan aliran maklumat yang betul dan lancar

- Memastikan bahagian tidak bersandar yang berada di dalam struktur kawalan diuji sekurang-kurangnya sekali

Langkah-langkah berikut dilakukan semasa melakukan ujian unit :

1) Kod diperiksa dengan cara membacanya, mencuba untuk melihat algoritma, data dan ralat sintaks
2) Kod dibandingkan dengan spesifikasi dan rekabentuk sistem untuk memastikan semua kes yang relevan telah dipertimbangkan
3) Akhir sekali, kod dikompil bagi menghapuskan semua ralat sintaks yang ada

7.3.3.2 Pengujian Modul

Proses pembangunan sistem ini dilakukan mengikut modul demi modul, maka pengujian dilakukan ke atas sesuatu modul sebaik sahaja ianya selesai dibangunkan. Seperti yang telah diterangkan di dalam bab yang terdahulu, sistem ini dibangunkan dengan menggunakan model air terjun di mana setiap modul aturcara perlu dihasilkan bermula dari peringkat awal dan kemudiannya diuji. Seterusnya apabila satu lagi fungsi ditambah ia akan diuji sekali lagi dan begitulah seterusnya. Setiap modul diuji supaya ia dapat melaksanakan fungsi-fungsi yang diingini. Ujian ini dilakukan bagi mengesan sebarang ralat dalam memasukkan data , pengeluaran output dan keberkesanan aturcara.

Pengujian ini cenderung diaplikasikan kepada program besar yang mana setiap modul individu menyumbang kepada cebisan-cebisan tetap tugasan dalam program tersebut.

Dalam sistem ini, modul yang terlibat adalah :

i) Pengujian setiap modul antaramuka untuk memastikan bahawa mesej input yang disampaikan diterima oleh pengguna

ii) Kawalan yang mana ianya penting untuk mengesan ralat dan memastikan semua modul beroperasi dengan betul pada sempadan modul yang ditetapkan

iii) Semua bahagian yang melalui struktur kawalan diuji untuk memastikan semua penyataan di dalam modul dilaksanakan sekurang-kurangnya sekali. Ianya adalah penting pada setiap modul kerana peninggalan salah satu bahagian akan menghasilkan kemungkinan ralat yang tinggi

7.3.3.3 Pengujian Integrasi

Di dalam fasa ini, proses pengujian akan dijalankan ke atas antaramuka-antaramuka bagi dua komponen yang saling berinteraksi antara satu sama lain dalam satu unit. Kemungkinan-kemungkinan seperti wujudnya ralat-ralat yang mana ia boleh menyebabkan fail-fail tidak berjaya untuk dikompilasikan akan berlaku di sini kerana terdapat banyak modul dan unit di dalam sistem ini. Justeru itu, adalah amat penting untuk melakukan proses ini dengan sebaik mungkin bagi memastikan bahawa sistem ini dapat diintegrasikan dengan baik dan lancar secara keseluruhannya.

Pada peringkat ini juga, satu pendekatan yang dipanggil sebagai pendekatan bawah-atas telah diaplikasikan. Menerusi pendekatan ini sesuatu modul yang terbawah akan diintegrasikan dengan modul yang terletak lebih atas daripadanya. Di samping itu, pada masa yang sama, pengujian ke atas proses penghantaran parameter juga turut dilaksanakan.

Pengujian ini memfokus kepada penilaian berkumpulan modul-modul program untuk menentukan :

i) samada antaramuka rosak atau terbengkalai fungsinya dan

ii) secara keseluruhan, samada modul program gagal memenuhi spesifikasi keperluan yang ditetapkan

Strategi pengujian yang digunakan adalah ujian peningkatan yang mana subset bagi setiap modul dikumpulkan secara iteratif dan diuji sehingga keseluruhan program adalah lengkap

7.3.3.4 Pengujian Sistem

Fasa pengujian sistem merupakan satu fasa yang amat penting dalam membangunkan sesuatu sistem. Apa yang perlu dilakukan adalah memastikan bahawa sistem ini boleh berjalan seperti yang diharapkan. Percubaan ini dilakukan dengan cuba mencari ralat yang mungkin berlaku serta mencari tahap keberkesanan sistem. Selain itu, fasa ini juga amat penting bagi menghasilkan sistem yang benar-benar mantap. Setelah keseluruhan sistem disahkan pengujiannya, ia digabungkan dengan elemen-elemen sistem yang lain seperti perkakasan, pengguna akhir dan sistem pengoperasian. Ujian sistem memastikan kesemua elemen berfungsi dengan betul dan perlaksanaan keseluruhan sistem mencapai objektif yang dikehendaki. Antara proses pengujian yang dilakukan adalah seperti berikut :

Empat jenis ujian :

i) Fungsi : Tentukan samada program integrasi memenuhi keperluan

ii) Persembahan : Tentukan samada program integrasi memenuhi kriteria persembahan tertentu

iii) Penerimaan : Maklum balas daripada pengguna.

iv) Pemasangan : Persembahan sistem dalam persekitaran operasi

Proses pengujian ini sebenarnya hanya dijalankan apabila kesemua aturcara yang ditulis telah berjaya dilarikan dengan jayanya tanpa ralat semasa pengujian integrasi. Pengujian ini merupakan pengujian peringkat terakhir yang penting dalam memastikan bahawa sistem ini akan dapat beroperasi dan menjalankan fungsi-fungsinya dengan baik sebelum dipaparkan secara rasmi untuk kegunaan umum. Antara objektif ujian sistem ini adalah untuk :

- Mengukur prestasi, kelemahan dan keupayaan sistem, secara keseluruhannya samada ia dapat mencapai tahap yang boleh diterima
- Mengesahkan ketepatan dan kejituan semua komponen sistem yang dibangunkan berdasarkan spesifikasi-spesifikasi sistem yang telah direkabentuk. Setiap subsistem dipastikan akan boleh dilarikan dengan baik dan sistem penggunaan ini akan berfungsi sebagaimana yang dikehendaki dalam keadaan yang sempurna dengan persekitaran operasi yang sebenar.
- Mengukur sejauh mana sistem yang dibangunkan dapat memenuhi objektif yang telah ditentukan

7.4 Kesimpulan

Perlaksanaan, penyelenggaraan dan pengujian ini adalah satu fasa yang terpenting dalam pembangunan sesuatu sistem. Ini disebabkan kerana fasa inilah yang menentukan apakah sistem yang akan terhasil nanti. Melalui fasa perlaksanaan kod-kod sumber, konfigurasi sistem dan pengujian sistem, ia dapat memastikan bahawa sistem yang terhasil mengikut garis-garis dan objektif yang ditetapkan semasa rekabentuk sistem. Di samping itu, konfigurasi sistem adalah penting kerana tanpanya, sistem tidak dapat dilarikan dengan sewajarnya.

8.0 PERBINCANGAN

8.0 Pengenalan

Bab ini membincangkan beberapa aspek iaitu keputusan yang diperolehi, masalah dan penyelesaian, kelebihan dan kelemahan perisian multimedia yang dibangunkan dan peningkatan yang boleh dibuat kepada sistem ini. Sebagai tambahan, bab ini juga membincangkan kekangan-kekangan yang dihadapi semasa pembangunan sistem dan penyelesaian yang mungkin boleh diambil sebagai langkah untuk mengatasinya.

8.1 Keputusan yang diperolehi

Setelah melalui proses pengimplimentasian dan pengujian, kini keputusan diperolehi daripada sistem yang telah dibangunkan. Hasil yang didapati adalah sistem ini dapat menjalankan tugasnya dengan baik mengikut jenis penapisnya. Oleh itu segala imej yang dimasukkan dapat menjalankan semua fungsi yang terdapat di dalam sistem ini. Sistem ini berjaya memanipulasi imej dengan cepat dan berkesan.

8.2 Masalah dan penyelesaian

Banyak masalah yang timbul sepanjang pembangunan sistem ini. Antara masalah-masalah yang dihadapi dan penyelesaiannya adalah :

i) MATLAB

Masalah : saya mengalami kesukaran ketika ingin menjadikan sistem ini stand-alone dan penyelesaiannya adalah dengan menggunakan Microsoft Visual C++. Setelah itu, ada lagi halangan iaitu, ada fungsi dalam MATLAB yang tidak menyokongnya apabila

diterjemah ke dalam Visual C++. Program boleh dilarikan dalam MATLAB tetapi tidak dapat dilarikan di dalam Visual C++.

Penyelesaian : Saya telah menggunakan Microsoft Foundation Class (MFC) bagi menggantikan MATLAB. Dan hasilnya adalah memuaskan.

ii) MFC dan Visual C++

Masalah : Memerlukan masa yang agak panjang untuk mempelajari benda baru. Agak sukar untuk mempelajari MFC dan Visual C++.

Penyelesaian : Mendapatkan rujukan daripada laman web yang berkaitan dan penggunaan HELP di dalam Microsoft Visual C++.

iii) Algoritma bagi pembinaan penapis

Masalah : Kesukaran untuk mencari algoritma bagi setiap jenis penapis yang digunakan

Penyelesaian : Akhirnya, didapati daripada Internet

iv) Kod-kod program

Masalah : Oleh kerana terlalu banyak kod yang boleh didapati daripada laman web, menyebabkan saya tidak tahu ingin menggunakan yang mana satu.

Penyelesaian : Kaedah try and error digunakan bagi setiap kod yang dijumpai. Akhirnya kod-kod digabungkan menjadi satu untuk penghasilan sistem ini.

v) Kekurangan masa

Masalah : Oleh kerana masa yang diberikan adalah cukup singkat, terdapat beberapa modul yang tidak terdapat dalam sistem saya. Antara modul yang tidak dapat dilaksanakan adalah cetak imej.

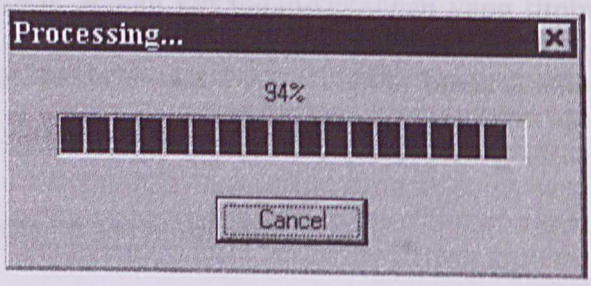
Penyelesaian : Perlu diberikan tempoh yang lebih lama dalam penghasilan sistem kerana ia mengambil masa yang lama untuk mempelajari sesuatu yang baru dan memandangkan ini adalah kali pertama saya membangunkan satu sistem.

8.3 Penilaian Sistem

Penilaian sistem adalah suatu proses mengenalpasti kelebihan dan kelemahan sistem yang dibangunkan.

8.3.1 Kelebihan sistem :

- Sistem mampu berfungsi dengan baik tanpa sebarang masalah dan menjalankan fungsi dengan sempurna
- Sistem ini mudah, ringkas dan senang difahami dan digunakan oleh pengguna
- Sistem ini memudahkan mereka yang ingin mengedit gambar
- Sistem ini mempunyai kotak yang dipaparkan semasa proses penapisan sedang dijalankan. Dengan adanya kotak ini, kita dapat mengetahui samada penapisan ini sedang dijalankan atau tidak. Kotak tersebut dipaparkan di bawah :



Rajah 8.1 : Proses sedang dilaksanakan

- Sekiranya kita hendak membatalkan penapisan yang hendak dijalankan, kita boleh klik pada butang cancel sebelum mencapai 100%.
- Sesiapa sahaja boleh menggunakan imej ini tanpa bantuan daripada sesiapa. Setelah memilih jenis penapis yang dikehendaki, mereka akan terus dapat melihat hasilnya. Dan ianya sungguh menarik.

8.3.2 Kelemahan sistem :

- Sistem ini tidak mempunyai modul cetak imej. Oleh itu, imej yang dihasilkan tidak dapat dicetak secara terus.
- Sistem ini tidak dapat dilarikan sekiranya tiada librarynya. Iaitu, ia bukan dalam bentuk pakej. Sekiranya tiada library yang diperlukan di dalam komputer itu, sistem ini tidak dapat digunakan.
- Butang 'undo' hanya akan memulangkan imej sebelumya sahaja. Sekiranya dua jenis penapis digunakan, butang 'undo' akan memaparkan imej sebelumya iaitu imej dengan penapis pertama. Imej asal (imej sebenar tanpa penapis) tidak akan kelihatan lagi.

8.4 Kekangan dan penyelesaian

Senarai yang berikut adalah kekangan-kekangan yang timbul semasa fasa-fasa pembangunan sistem dan penyelesaian yang mungkin boleh diambil dalam mengatasi masalah tersebut :

i) Tempoh penghasilan pakej multimedia

Kekangan : Jangkamasa yang terhad untuk membangunkan keseluruhan sistem menyebabkan modul terhad

Penyelesaian : Tempoh yang sesuai perlu diberi agar pembangunan sistem lebih lengkap dan banyak modul boleh diselitkan dalam suatu pakej

8.5 Kesimpulan

Secara keseluruhannya, bab ini menerangkan tentang keputusan yang diperolehi, apakah masalah-masalah yang dihadapi di sepanjang proses membangunkan sistem ini serta penyelesaiannya. Selain itu, kelebihan dan kelemahan juga dibincangkan serta kekangan yang dihadapi. Dengan ini, diharap agar sistem ini dapat digunakan oleh pengguna dengan sebaiknya dan dapat dipertingkatkan lagi dengan penambahan modul-modul yang lain serta penambahan dari segi jenis penapis yang digunakan.

9.0 KESIMPULAN DAN CADANGAN

9.1 Kesimpulan

Secara keseluruhannya, sistem ini berjaya mencapai objektif pembangunannya. Projek ini telah memberi peluang untuk membina aplikasi yang sebenar daripada suatu lakaran. Membangunkan sistem ini sahaja telah memberi cabaran yang besar kepada saya. Di sepanjang pembangunan sistem ini, saya telah menghadapi cabaran dari segi fizikal dan mental berhubung pembinaan sistem ini, walaubagaimanapun, pengalaman yang diperolehi adalah amat berharga dan berbaloi dengan apa yang telah diusahakan.

Segala pengetahuan, kemahiran dan pengalaman yang diperolehi sepanjang menyiapkan kertas cadangan dan pembangunan sistem merupakan ilmu yang cukup bernilai bagi saya. Ia juga mengajar saya betapa pentingnya pengurusan masa yang baik. Di samping itu, saya amat menghargai apa yang dipelajari sepanjang pembangunan sistem. Pada masa yang sama, ia telah menyedarkan saya bahawa pendidikan di Universiti hanya menyediakan asas di dalam bidang sains dan teknologi maklumat kepada mahasiswa, tetapi suatu proses pembelajaran yang menghasilkan pengetahuan tidak akan membawa apa-apa makna selagi pengetahuan itu tidak diaplikasikan. Saya berharap agar sistem ini akan dapat dimajukan dan dipertingkatkan kepada sistem yang lebih baik di kemudian hari nanti.

Secara keseluruhannya, saya berpuas hati dan berbangga dengan projek tahunan akhir ini kerana telah memberikan pengetahuan dan pengalaman yang sangat berguna. Saya berharap supaya apa yang diperolehi di sepanjang pembangunan sistem ini akan menyediakan diri saya kepada aplikasi serupa yang mugkin akan digunakan untuk projek-projek lain di masa hadapan. Saya yakin bahawa aplikasi yang digunakan dalam sistem ini boleh digunakan untuk membina sistem yang lebih kompleks kelak.

9.2 Cadangan dan Perancangan masa depan

Terdapat beberapa cadangan untuk mempertingkatkan sistem ini. Antaranya adalah :

- Penambahan fungsi, modul atau sub-modul – Penambahan lebih banyak fungsi, modul atau sub-modul misalnya membenarkan pengguna mencetak imej yang telah ditapis.
- Penambahan jenis penapis – Penambahan pelbagai jenis penapis ke dalam sistem boleh memantapkan lagi sistem ini dan menyebabkan lebih ramai pengguna yang tertarik menggunakannya.
- Latihan ilmiah sesuai kepada semua mahasiswa tidak mengira fakulti. Seharusnya mahasiswa digalakkan membuat sistem agar mudah mendapatkan peluang kerjaya kelak.
- Selain itu, jangka masa yang diberi perlu dipanjangkan agar sistem yang dibina mampu setanding dengan mereka yang telah bekerja dalam bidang perkomputeran dan Teknologi Maklumat.

Apendiks A

Penapis : Grayscale

Penskalaan kelabu membawa satu isu yang paling penting tentang pemprosesan warna di mana akan digunakan dalam beberapa penapis. Isu tentang warna melibatkan bagaimana dengan mudahnya menkategorikan warna. Iaitu dengan cara bagaimana menukarkan tiga nilai RGB pada piksel ke dalam satu nilai. Kebanyakan algoritma penapisan menggunakan imej hitam dan putih. Setiap piksel adalah satu bait yang mana boleh mempunyai 256 nilai yang berbeza iaitu daripada 0 sehingga 255 atau 256 tahap kelabu. Apabila algoritma memanggil warna hanya terdapat 256 kemungkinan yang berbeza tetapi dengan imej berwarna terdapat 2563 = 16,777,216 kemungkinan yang berbeza. Ia adalah mustahil yang setiap warna mempunyai warna yang berbeza, mesti ada yang sama.

Jadi, jawapannya adalah intensiti warna atau keterangan warna. Dengan itu, jika kita boleh tukarkan piksel RGB ke dalam nilai tunggal, berpandukan pada intensiti, ia adalah cara yang baik untuk menerangkan tentang piksel yang menggunakan had yang lebih kecil daripada 16.7 juta. Didapati bahawa hijau mempunyai intensiti yang lebih berbanding merah, yang mana lebih intensiti berbanding biru. Di bawah adalah fungsi ahli piksel FilterExplorer bagi intensiti :

```
BYTE pixel::intensity()
{
return BYTE(0.3*r + 0.59*g + 0.11*b);
}
```

Di bawah ini pula disediakan pseudokod berdasarkan intensiti:

```
for every pixel in the image do
```

```
{
intensity = currentpixel.intensity();
filterpixel.r = intensity;
filterpixel.g = intensity;
filterpixel.b = intensity;
}
```

Diingatkan bahawa untuk piksel RGB kepada kelabu, semua komponen perlu mempunyai nilai yang sama.

FilterExplorer mempunyai tambahan dua jenis penskalaan kelabu. Satu berasaskan hue atau warna dan satu lagi berdasarkan saturation atau kepekatan. Hue dikira berpandukan darjah. Dalam model ini, 0 darjah adalah merah, 120 darjah adalah hijau dan 240 darjah adalah biru. Dengan itu, penukaran kepada penskalaan kelabu, nilai piksel 0 (hitam) adalah untuk warna merah yang sebenar, 85 (kelabu gelap) akan mewakili warna hijau sebenar dan 170 (kelabu cair) akan mewakili warna biru. (Begitu juga dengan nilai piksel 1 (putih) akan juga mewakili warna merah sebenar). Kedua-dua penapis ini menggunakan model warna HSV:

```
// grayscale by hue
for every pixel in the image do
{
hsvpixel = currentRGBpixel;
filterpixel.r = hsvpixel.h;
filterpixel.g = hsvpixel.h;
filterpixel.b = hsvpixel.h;
}
```

```
// grayscale by saturation
for every pixel in the image do
{
hsvpixel = currentRGBpixel;
filterpixel.r = hsvpixel.s;
filterpixel.g = hsvpixel.s;
filterpixel.b = hsvpixel.s;
}
```

Setiap penapis menggunakan komponen piksel RGB 'hue' atau 'saturation' yang ditukarkan kepada HSV dalam nilai piksel penapis.

Penapis : Hue/Saturation

Hue/saturation meupakan satu lagi penapis proses titik yang diimpilen berdasarkan fungsi yang sama dalam Photoshop®. Penapis ini membenarkan pengubahsuaian bagi setiap komponen H(ue) S(aturation) V(alue) piksel HSV. Di sini, kita akan menggunakan RGB / HSV menerangkan tentang penapis penskalaan kelabu.

Algoritmanya adalah :

```
for every pixel in the image do
{
hsvpixel = currentpixel;
hsvpixel.h += somevalue1;
hsvpixel.s += somevalue2;
hsvpixel.v += somevalue3;
filterpixel = hsvpixel;
```

}

Kotak dialog penapis membenarkan pengguna untuk meningkatkan atau mengurangkan setiap komponen hsvpixel. Kesemua program harus menukarkan piksel semasa kepada HSV, melaksanakan peningkatan atau penurunan dan penterjemahan semula kepada RGB. Terdapat pilihan untuk 'Colorize' imej di dalam kotak dialog hue/saturation FilterExplorer. Dengan menandakan kotak ini menggunakan 'hue' yang sama (seperti yang terdapat pada dialog) bagi setiap piksel. Jadi, selain untuk meningkatkan atau mengurangkan 'hue' piksel, ia dengan mudah mengumpukkan ia sebagai hue yang baru.

Penapis : Equalize Intensity

Dalam pembahagian yang sama yang sebenar mana-mana secara rawak mengambil warna akan berfungsi seperti skema warna, menyediakan semua warna yang mempunyai intensiti atau nilai yang sama. Program akan menyuruh pengguna memasukkan input intensiti antara 1 hingga 100, di mana dipetakan ke dalam had nilai bagi komponen nilai hsvpixel 0 hingga 1. Setiap piksel dalam imej telah ditetapkan kepada nilai itu dan membiarkan 'hue' dan 'saturation' :

```
for every pixel in the image do
{
hsvpixel = currentpixel;
hsvpixel.v = value;
filterpixel = hsvpixel;
}
```

Bagi beberapa imej, keputusan skema warna adalah sangat baik, kadangkala tidak.

Penapis : Blur

Kabur merupakan penapis kawasan yang paling mudah untuk difahami dan diimplimentasikan. Bagi penapis kawasan, nilai piksel yang telah ditapis adalah bukan sahaja berdasarkan nilai sebenar piksel, tetapi juga nilai piksel sekeliling piksel sebenar. Kabur secara tenikalnya dipanggil penapis purata bergerak kerana cara ia dilaksanakan. Kabur direka dengan kaedah melihat pada setiap piksel dan mengumpukkan nilai piksel yang di tengah-tengah sebagai purata nilai piksel di sekitarnya. Di bawah adalah pseudokod bagi algoritma tersebut :

```
for every pixel in the image do
{
R = G = B = 0;
numpixels = 0;
for every pixel in radius r surrounding the center pixel do
{
R += currentpixel.r;
G += currentpixel.g;
B += currentpixel.b;
numpixels++;
}
filterpixel.r = R / numpixels;
filterpixel.g = G / numpixels;
filterpixel.b = B / numpixels;
}
```

Kod di atas menggunakan kawasan segi empat sama untuk mengira purata bagi beberapa piksel yang di tengah. Dalam kotak dialog kabur dalam FilterExplorer terdapat 'check box' yang dipanggil 'Fast Blur'. Jika kotak itu ditandakan, piksel dalam kawasan segi empat sama digunakan. Kabur sesuai digunakan untuk membuang kebisingan dalam imej, bagaimana pun, kesan yang paling jelas adalah kehilangan nilai sebenar.

Penapis : Median Blur

Seperti yang telah disebutkan tadi, penapis kabur digunakan untuk membuang kebisingan atau ralat dalam imej. Kabur akan memindahkan ralat dalam imej (contohnya habuk atau kesan calar pada gambar yang diimbas atau negatif), tetapi nilai terperinci akan hilang dalam proses ini. Ianya adalah lebih baik sekiranya segala yang terperinci itu kekal selepas penapis kabur dilaksanakan dengan ralat yang telah dibuang, yang mana akan dilaksanakan oleh 'median blur'.

Median blur mempunyai impilentasi yang sama dengan kabur. Bagaimanapun, selain menggunakan nilai purata piksel disekelilingnya untuk mendapatkan nilai baru, penapis ini menyatakan bahawa nilai piksel ralat dalam data adalah berbeza daripada nilai piksel sebenar yang baik. Dengan itu, sekiranya kita mengambil setiap piksel dalam kawasan segiempat sama di sekeliling piksel yang berada di tengah, dan menyusun nilai, ralat seharusnya adalah satu di penghujung senarai yang disusun.

Bagi ralat yang kecil, nilai piksel di tengah mempunyai nilai ralat yang paling rendah. Sekiranya kita memilih di tengah ataupun median nilai di dalam senarai, kemungkinan kita tidak akan memilih nilai yang salah, jadi kita umpukkan nilai median kepada piksel di tengah. Keputusan ini akan mempunyai nilai yang hampir dengan gambar sebenar. Warna akan disusun mengikut intensiti. Seperti yang telah dinyatakan

sebelum ini, intensiti merupakan perkara asas bagi analisa imej. Algoritmanya adalah ringkas dan masalah implimentasi sebenar adalah mencari cara menyusun intensiti piksel dengan pantas dan melibatkan nilai RGB. Fungsi My MedianPixel() mengambil pernyataan tatasusunan intensiti dan tatasusunan RGB yang terlibat. Fungsi ini mencipta tatasusunan sementara bagi integer yang mewakili penunjuk kepada tatsusunan intensiti dan piksel. MedianPixel() memulangkan piksel median RGB.

Di sini adalah algoritma bagi penapis ini :

```
for every pixel in the image do
{
numpixels = 0;
for every pixel in radius r surrounding the center pixel do
{
intensity = currentpixel.intensity;
intbuffer[nextlocation] = intensity;
colorbuffer[nextlocation] = currentpixel;
numpixels++;
}
filterpixel = MedianPixel(intbuffer, colorbuffer, numpixels);
}
```

Penapis : Custom Convolution

Cara yang paling mudah untuk memahami penapis 'convolution' adalah memikirkan tentangnya sebagai satu penambahan yang berat atau purata. 'Convolution' dijalankan seperti proses kawasan penapisan yang lain, oleh itu, piksel di sekeliling piksel sebenar digunakan untuk mengira nilai piksel berkaitan yang ditapis. Dalam

penapis 'convolution', 'convolution kernel' digunakan, biasanya matriks 3x3 atau 5x5. Kemasukan (integer positif atau negatif) dalam matriks adalah berat yang diberi kepada piksel yang mengelilingi piksel yang berada di tengah (yang mana bertanggungjawab kepada elemen matriks pusat). Secara asasnya, bagi piksel, piksel yang berada di sekelilingnya adalah 'multiplied' oleh 'coefficient' yang tertentu dalam 'convolution kernel' dan dicampur bersama-sama.

Dalam implimentasi saya, penambahan ini dibahagikan oleh hasil tambah 'coefficient' bagi 'convolution kernel' dalam penghasilan purata berat (jika penambahan 'coefficient' adalah kurang atau sama dengan sifar, pembahagian tidak dijalankan). Purata berat adalah nilai yang diberi kepada piksel tertentu yang telah ditapis. Ini diulang kepada setiap piksel di dalam imej :

```
sum = sum of coefficients in kernel;
divide each element in kernel by sum;
for every pixel in the image do
{
R = G = B = 0;
for every pixel in radius r surrounding the center pixel do
{
R += currentpixel.r * corrspd kernel value;
G += currentpixel.g * corrspd kernel value;
B += currentpixel.b * corrspd kernel value;
}
filterpixel.r = R;
filterpixel.g = G;
```

filterpixel.b = B;

}

Saya telah memilih untuk memanggil 'custom convolution' kerana pengguna mentakrifkan 'kernel coefficients'. Dialog 'custom convolution' adalah matriks 5x5 dalam kotak edit. Nilai integer positif atau negatif boleh dimasukkan dalam matriks; matriks ini adalah 'kernel'. Di bawah ini disenaraikan beberapa 'convolution kernels' dan kesan yang dihasilkan. Rekabentuk 'kernels' ini adalah berasaskan pada matematik tahap tinggi.

Blur: 1 1 1 1 2 1

1 1 1 2 4 2

1 1 1 1 2 1

Sharpen: -1 -1 -1 0 -1 0

-1 9 -1 -1 5 –1

-1 -1 -1 0 -1 0

Edge Enhancement: 0 0 0 0 -1 0 -1 0 0

-1 1 0 0 1 0 0 1 0

0 0 0 0 0 0 0 0 0

Find Edges: 0 1 0 -1 -1 -1 1 -2 1

1 -4 1 -1 8 -1 -2 4 –2

0 1 0 -1 -1 -1 1 -2 1

Emboss: -2 -1 0

-1 1 1

0 1 2

Penapis : Fish-Eye Lens

Ide ini bermula apabila kanta mata ikan digunakan dalam fotografi untuk menghasilkan imej dengan pandangan 180 darjah. Kesan yang didapati adalah imej muncul seperti hendak keluar, seperti ada sfera di nahagian belakang yang menolak imej keluar.

Kesan ini telah dijalankan oleh transformasi geometrik piksel dalam beberapa kawasan. Dalam kes ini, kawasannya adalah bulatan besar yang boleh muat apabila diletakkan di tengah-tengah imej. Fungsi di dalam penapis ini melibatkan penukaran koordinat dalam piksel imej saya daripada koordinat 'cartesian' kepada 'polar'. Fungsi ini dimodelkan pada struktur optik mata ikan sebenar, yang melibatkan resolusi adalah tinggi di tengah dan menurun apbila ke bahagian tepi :

new_radius = s log(1 + w(old_radius))

w muncul menjadi amaun kelengkungan, atau berapa banyak bahagian tengah imej yang ditolak ke hadapan. Saya menjumpai nilai yang boleh diterima untuk w untuk berada dalam had [0.001..0.1]. Dengan mengetahui jika old_radius adalah radius yang maksimum (R), new_radius akan dipetakan kepada R yang sama, dan dengan mempunyai w yang telah dipilih, didapati bahawa s dalam terma R dan w dalam mencari parameter bagi persamaan di atas. Memandangkan R 'will vary depending' pada imej, dan w 'will vary depending' pada input pengguna (kelengkungan adalah pilihan di dalam kotak dialog fish-eye), s dikira pada masa-larian seperti di bawah :

s = R / log(w*R+1)

Apa yang akan berlaku jika fungsi 'inverse of the fish-eye' digunakan?. Mungkin ada yang menyangka yang hasil imej akan muncul seperti masukan sfera ke dalam imej.

Itulah hasil yang akan didapati sekiranya pengguna menandakan inverse pada kotak dialog fish-eye. Di bawah adalah fungsi inverse:

new_radius = (e^(old_radius / s) – 1) / w)

Di bawah adalah algoritma asas bagi kesan fish-eye.

```
R = (min(image_width, image_height)) / 2;
Get w from user;
s = R / log(w*R+1);
for every pixel in the image do
{
polarpixel = currentpixel in polar coordinates (r(adius), a(ngle));
if (polarpixel.r < R)
{
polarpixel.r = s * log(1+w*r);
filterpixel = polarpixel in Cartesian coordinates;
}
else
filterpixel = currentpixel; // leave pixels outside R alone
}
```

Penapis : Polar Coordinates

Anggap bahawa piksel dalam imej berasaskan Cartesian adalah koordinat polar dan memaparkannya menggunakan system Cartesian. Dengan itu, piksel pada koordinat (x,y) mempunyai radius koordinat polar dan sudut (r,a), menggunakan bahagian pusat imej sebagai polar yang sebenar. Di sini, untuk mencari nilai imej yang telah ditapis,

didapati (r,a), kemudian menggunakan r = x, dan a = y dan plot titik dalam system Cartesian. Langkah daripada Cartesian kepada koordinat polar adalah berasaskan geometri dan trigonometri:

radius = sqrt(x*x + y*y)

angle = arctan(x / y)

Walaupun tidak digunakan dalam penapis, langkah kembali kepada Cartesian diberikan oleh :

x = radius * cos(angle)

y = radius * sin (angle)

Menggunakan kaedah ini, tidak semua lokasi koordinat dalam imej yang ditapis, diplot dengan nilai piksel daripada imej sebenar. Untuk penyelesaian semula isu ini, mula-mula tetapkaan setiap piksel dalam imej yang ditapis kepada putih.

Seperti yang telah dimaklumkan bahawa idenya adalah mudah tetapi implimentasi memerlukan kerja yang sungguh berhati-hati dan teliti dalam memanipulasi kedua-dua system koordinat yang bekerja bersama untuk menghasilkan keputusan yang betul. Pseudokod yang ringkas disediakan di bawah, bagaimanapun, implementasi sebenar adalah mengelirukan.

```
for every pixel in the filtered image do
currentpixel.color(255); // set to white
for every pixel in the original image do
{
// x and y are the coordinates of the currentpixel
// the pixel in the center of the image x = 0, y = 0
r = sqrt(x*x + y*y);
```

```
a = atan(x/y);
x = r * image_height / R;
y = a * image_width / 6.2832;
filterpixel.x = x;
filterpixel.y = y;
}
```

Penapis : Oil Paint

Kesan lukisan cat adalah proses kawasan, seperti kabur dan 'convolution' yang telah diterangkan tadi. Kesannya direka dengan mengumpukkan piksel di tengah ke dalam kawasan kernel yang paling boleh memunculkan warna pada kawasan itu. Dengan cara ini, warna yang penting 'clumped' bersama-sama. Algoritma asas adalah mudah untuk kesan yang menarik :

```
for every pixel in the image do
filterpixel = MostFreqentColor(currentpixel);
```

Kekompleksan terdapat dalam fungsi MostFrequentColor(). Ia berfungsi seperti ini: Bagi setiap piksel yang mengelilingi koordinat yang diberi iaitu piksel semasa, MostFrequentColor() menentukan intensiti piksel itu dan memerhatikan berapa banyak piksel mempunyai mana-mana intensiti itu. Komponen RGB bagi piksel yang mempunyai intensiti yang sama sebagai paling kerap muncul adalah purata dan dikembalikan sebagai warna yang kerap digunakan.

Di dalam proses kawan yang lain, jumlah piksel yang mengelilingi piksel semasa adalah berasaskan pada radius yang diisikan oleh pengguna pada dialog lukisan cat. Hasilnya imej adalah sedikit 'coarse' dan tidak licin. Dialog lukisan cat mempunyai bar

'slider' untuk menetapkan nombor kemungkinan intensity warna piksel yang mungkin dipetakan yang mana menukarkannya kepada kesan imej yang dilicinkan.

Penapis : Frosted Glass

Algoritmanya adalah sama dengan lukisan cat. Perbezaan di antara keduanya adalah di dalam fungsi helper yang dipanggil. Di sini, selain memanggil MostFrequentPixel(), fungsi RandomColor() dipanggil. Piksel yang intensitinya muncul selalu akan mempunyai peluang yang lebih baikuntuk dipilih berbanding yang piksel yang intensitinya adalah kurang dalam kawasan tempatan. Perubahan ini meningkatkan output yang terhasil.

Fungsi RandomColor() diimplimen dengan menyimpan tatasusunan 256 integer, indeks tatasusunan mewakili intensiti; nilai indeks mewakili berapa banyak piksel yang mempunyai intensiti. Nombor rawak dipilih di antara 1 dan jumlah nombor piksel dalam kawasan tempatan. Bermula daripada intensiti 1 dan melalui tatasusunan, fungsi akan mengira nombor piksel yang mempunyai intensity yang sesuai (nilai tatasusunan) sehingga pengiraan itu mencapai nombor rawak. Intensiti ini dipilih dan kesannya adalah memberi berat yang lebih kepada warna dengan intensiti yang biasa. Purata nilai piksel RGB dengan intensiti itu mengembalikan purata berat.

Penapis : Random Blur

Ia adalah hampir dengan hubungan penapis kawasan, ini kerana piksel-pikselnya diubah berdasarkan piksel di sekelilingnya. Perkara yang membuatkan penapis ini berbeza adalah tidak semua piksel diproses dan pemprosesan ini tidak muncul dalam susunan linear.

Kesan random blur ini memilih koordinat (x,y) di dalam imej dan mengaburkan kawasan segiempat di sekitar piksel. Kabur yang wujud adalah sama seperti penapis blur. Memandangkan kabur adalah penapis yang ringkas, ini mungkin adalah cara yang terbaik untuk bermula.

```
for every pixel in the image do
// copy the image since the entire image won't be affected
filterpixel = currentpixel;
RAD = blocksize * blocksize;
repeat num_of_blurs times
{
x = rand() * (image_height / RAND_MAX);
y = rand() * (image_width / RAND_MAX);
for every pixel in radius blocksize surrounding (x,y) do
{
R = G = B = 0;
for every pixel in radius r surrounding the center pixel do
{
R += currentpixel.r;
G += currentpixel.g;
B += currentpixel.b;
numpixels++;
}
filterpixel.r = R / RAD;
filterpixel.g = G / RAD;
```

filterpixel.b = B / RAD;

}

}

Nombor kabur, saiz blok dan radius kabur adalah kesemua nilai yang dimasukkan oleh pengguna melalui kotak dialog random blur. Hasil yang didapati agak menarik.

Penapis : Raindrops

'Raindrops' adalah bulatan kecil berbeza saiz yang secara rawak diletakkan pada imej, jadi tiada bulatan yang bertindih antara satu sama lain. Setiap kawasan bulatan diproses dengan kesan fish-eye lens yang kecil, dan 'highlight' dan baying-bayang ditambah untuk menghasilkan kesan tiga dimensi. Selepas penambahan kesan-kesan itu, bulatan itu akan dikaburkan untuk memindahkan kesan piksel daripada fish-eye yang dihasilkan dan melembutkan 'highlight' dan baying-bayang. Melalui kotak dialog, pengguna menentukan saiz raindrop yang paling besar dan 'density'nya. Jika algoritma tidak boleh memenuhi permintaan pembolehubah 'density' raindrops (berpandukan kepada kecil atau besar saiz raindrops) ia akan berhenti, dengan kebanyakan kawasan (tetapi tidak semua) yang diselaputi 'water drops'.

Penapis raindrop menggunakan kombinasi ketiga-tiga jenis penapis yang telah diterangkan : titik, kawasan dan geometrik dan menggunakan bebrapa penapis yang lain iaitu : hue/saturation (untuk 'highlight' dan bayang-bayang), blur (untuk melicinkan water drop), fish-eye lens (untuk imej yang lebih menakjubkan), dan random blur (untuk letakkan raindrops secara rawak).

Algoritma yang digunakan adalah seperti di bawah:

w = 0.4; // this value seemed to produce the right

```
// amount of curvature (see fish-eye lens)
for every pixel in the image do
// copy the imahe since the entire image won't be affected
filterpixel = currentpixel;
for num_of_raindrops do
{
size = rand() * (MAX_SIZE / RAND_MAX);
R = size / 2;
s = R / log(w*R+1); // recall fish-eye variable
do
{
x = rand() * image_height-1 / RAND_MAX;
y = rand() * image_width-1 / RAND_MAX;
} while (this (x,y) value will produce a raindrop that will overlap an existing raindrop);
for pixels inside the circle centered at (x,y) of radius R do
{
convert (x,y) to polar;
do fish-eye lens transformation;

if (currentpixel is in highlight area)
{
filterpixel.r += highlight_increment;
filterpixel.g += highlight_increment;
filterpixel.b += highlight_increment;
```

```
}
else if (currentpixel is in shadow area)
{
filterpixel.r -= shadow_decrement;
filterpixel.g -= shadow_decrement;
filterpixel.b -= shadow_decrement;
}
}
for pixels inside the circle centered at (x,y) of radius R do
{
blur the pixel as done before;
}
}
```

Apendiks B

Filters.h

```
////////////////////////////////////////////////////////////////////////////////
//
// Filename.....: Filters.h
// Compilation..: Microsoft Visual C++ 6.0.
// Description..: interface for pixel, hsvpixel, CFilter, CImage classes
//
////////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_FILTER_H__D53069AA_EA08_4687_9202_7CAF73FE0011__INCLUDED_)
```

```
#define AFX_FILTER_H__D53069AA_EA08_4687_9202_7CAF73FE0011__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
////////////////////////////////////////////////////////////////////
// Pixel Classes
////////////////////////////////////////////////////////////////////
class hsvpixel;      // forward reference
class pixel
{
public:
void operator= (pixel& p);
void operator= (hsvpixel& hsv);
void color(BYTE R, BYTE G, BYTE B);
void color(BYTE X);
void color(float R, float G, float B);
BYTE intensity();
void lighten(UINT i);
void darken(UINT i);

BYTE r;
BYTE g;
BYTE b;
};
class hsvpixel
{
public:
void operator= (hsvpixel& hsv);
```

```
    void operator= (pixel& p);
    double h;                   // hue [0,360]
    double s;                   // saturation [0,1]
    double v;                   // value/brightness [0,1]
    };
    ////////////////////////////////////////////////////////////
    // Type Definitions
    ////////////////////////////////////////////////////////////
    typedef pixel*  imagerow;
    typedef pixel** imagematrix;
    typedef bool*   boolrow;
    typedef bool**  boolmatrix;
    #define UNDEFINED -1
    // list of all filters
    enum filtertype
    {
    UNDO, //  undo
    // POINT PROCESSES
    GRAYINT, //  convert to grayscale by intensity
    GRAYHUE, //  convert to grayscale by hue
    GRAYSAT, //  convert to grayscale by saturation
    HUESAT, //  alter hue, saturation, and brightness
    EQINTENSITY, //  make all color intensities (value) equal
    // AREA PROCESSES
    BLUR, //  normal blur
    MEDIANBLUR, //  median blur
    CONVOLUTION, //  convolution filters
    // AREA PROCESSES (ARTISTIC)
```

```
    OIL, //   oil paint effect
    FROSTGLASS, //   frosted glass effect
    RANDBLUR, //   random blured squares
    RAINDROPS, //   looking through a rained on window
    // GEOMETRIC PROCESSES
    FISHEYE, //   distort to look like fish eye lens was used
    POLAR //   convert cartesian to polar coordinates
    };
    ////////////////////////////////////////////////////////////////
    // Image Class Definition
    ////////////////////////////////////////////////////////////////
    class CImage
    {
    friend class CFilter; //everything in filter can access private here
    public:
    CImage();
    virtual ~CImage();
    public:
    // inline functions
    void SetFilename(CString sFilename) { m_filename = sFilename; };
    bool OK() { return (m_buf != NULL); };
    UINT GetWidth() { return m_width;  };
    UINT GetHeight() { return m_height;  };
    CString GetFilename() { return m_filename;   };
    public:
    // member functions
    void LoadJPG();
    void LoadBMP();
```

```
    void DrawBMP(CDC* dc, CRect rectClient);
    void SaveJPG(CString sFilename);
    void SaveBMP(CString sFilename);
    void ProcessFilter(enum filtertype type);
    private:
    // private member variables
    BYTE*   m_buf;
    BYTE*   m_undobuf;
    UINT    m_width;
    UINT    m_height;
    UINT    m_widthDW;
    CString m_filename;
    };
    ////////////////////////////////////////////////////////////////
    // Filter Class Definition
    ////////////////////////////////////////////////////////////////
    class CFilter
    {
    public:
    CFilter(CImage* img, filtertype type);
    virtual ~CFilter();
    void go();
    private:
    // private member functions
imagematrix CreateImageMatrix(unsigned int row, unsigned int col);
void        FreeImageMatrix(imagematrix img, unsigned int row);
boolmatrix  CreateBoolMatrix(unsigned int row, unsigned int col);
void      FreeBoolMatrix(boolmatrix bmtx, unsigned int row);
```

```
    void  BufferToMatrix(BYTE* buf, imagematrix img);
    void   MatrixToBuffer(imagematrix img, BYTE* buf);
    void   CopyImageBuffer(BYTE* buf, BYTE* bufcopy);
    void   FilterInit();
    pixel MostFrequentColor(unsigned int x, unsigned int y, unsigned int r, unsigned int I);
    pixel   RandomColor(unsigned int x, unsigned int y, unsigned int r);
    void   MPQ(BYTE* buf, UINT* index, int lb, int ub);
    pixel   MedianPixel(BYTE* buf, pixel* pixbuf, int length);
    bool   grayint();
    bool   grayhue();
    bool   graysat();
    bool   huesat();
    bool   eqintensity();
    bool   blur();
    bool   medianblur();
    bool   convolution();
    bool   oil();
    bool   frostglass();
    bool   randblur();
    bool   raindrops();
    bool   fisheye();
    bool   polar();
private:
// private member variables
CImage*          m_pImage;      // pointer to image calling this filter
UINT   m_height;       // image height
UINT   m_width;        // image width
```

```
filtertype          m_type;// what filter?
imagematrix  m_img;       // original image in matrix form (input to filter)
imagematrix  m_filt;      // filtered image in matrix form (output from filter)
};
#endif //
!defined(AFX_FILTER_H__D53069AA_EA08_4687_9202_7CAF73FE0011__INCLUDED_
```

Filters.cpp

```
//////////////////////////////////////////////////////////////
//
// Filename.....: Filters.cpp
// Compilation..: Microsoft Visual C++ 6.0.
// Description..: implementation  for pixel, hsvpixel,
// CFilter, CImage classes
//
//////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "FilterExplorer.h"
#include "Filters.h"
#include "jpegfile.h"
#include "bmpfile.h"
#include "dialogs.h"
#include <math.h>
#include <algorithm>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////
// PIXEL PROCESSING SUPPORT FUNCITONS
//////////////////////////////////////////////////////////////

void pixel::operator =(pixel &p)
//
// Desc:   RGB pixel assignment
//
{
      r = p.r;
      g = p.g;
      b = p.b;
}

void pixel::operator =(hsvpixel& hsv)
//
```

```
// Desc:   Converstion of Hue-Saturation-Value pixel to RGB
// Pre:    hsv is an HSV pixel
// Post:   the equavalent RGB pixel is returned
//
{
    double R, G, B;

    if (hsv.s == 0)
    {
        R = hsv.v;
        G = hsv.v;
        B = hsv.v;
    }
    else
    {
        int i;
        double f, p, q, t;

        if (hsv.h == 360)
            hsv.h = 0;
        hsv.h /= 60;                    // h now in [0,6)
        i = int(floor(hsv.h));
        f = hsv.h - i;
        p = hsv.v * (1 - hsv.s);
        q = hsv.v * (1 - (hsv.s * f));
        t = hsv.v * (1 - (hsv.s * (1 - f)));

        switch (i)
        {
            case 0: R = hsv.v;
                    G = t;
                    B = p;
                    break;
            case 1: R = q;
                    G = hsv.v;
                    B = p;
                    break;
            case 2: R = p;
                    G = hsv.v;
                    B = t;
                    break;
            case 3: R = p;
                    G = q;
                    B = hsv.v;
                    break;
            case 4: R = t;
                    G = p;
                    B = hsv.v;
                    break;
            case 5: R = hsv.v;
                    G = p;
                    B = q;
        }
    }

    r = (BYTE)(R * 255);
    g = (BYTE)(G * 255);
```

```
    b = (BYTE)(B * 255);
}

void pixel::color(BYTE R, BYTE G, BYTE B)
//
// Desc:   Assigns pixel the values R, G, and B
//
{
    r = R;
    g = G;
    b = B;
}

void pixel::color(BYTE X)
//
// Desc:   Makes gray pixel with intensity X
//
{
    r = X;
    g = X;
    b = X;
}

void pixel::color(float R, float G, float B)
//
// Desc:   Colors a pixel given the float values R, G, and // B.
//      If the float is < 0 or > 255, the color will be set
//         to 0 or 255 respectively.
//
{
    if (R < 0) R = 0;
    if (R > 255) R = 255;
    if (G < 0) G = 0;
    if (G > 255) G = 255;
    if (B < 0) B = 0;
    if (B > 255) B = 255;

    r = (BYTE)R;
    g = (BYTE)G;
    b = (BYTE)B;
}

BYTE pixel::intensity()
//
// Desc:Returns the intensity of a RGB pixel.  This is the
// Luminance (Y)component of the YIQ color model (used in
// TV displays).  Intensity is most commonly used as the
// basis for color image analysis.  This intensity
// 'formula' appears to produce the best results when using // this
basis.
//
{
    return BYTE(0.3*r + 0.59*g + 0.11*b);
}

void pixel::lighten(UINT i)
//
```

```
// Desc:   Lightens a pixel by i.
//
{
      if (r >= 255-i) r = 255;
      else r += i;
      if (g >= 255-i) g = 255;
      else g += i;
      if (b >= 255-i) b = 255;
      else b += i;
}

void pixel::darken(UINT i)
//
// Desc:   Darkens a pixel by i.
//
{
      if (r <= i) r = 0;
      else r -= i;
      if (g <= i) g = 0;
      else g -= i;
      if (b <= i) b = 0;
      else b -= i;
}

void hsvpixel::operator =(hsvpixel& hsv)
//
// Desc:   HSV pixel assignment.
//
{
      h = hsv.h;
      s = hsv.s;
      v = hsv.v;
}

void hsvpixel::operator =(pixel& p)
//
// Desc:   Converstion of RGB pixel to Hue-Saturation-Value // pixel
// Pre:    p is a RGB pixel
// Post:   the equavalent HSV pixel is returned
//
{
      //convert p to range [0,1]
      double r = p.r / 255.0;
      double g = p.g / 255.0;
      double b = p.b / 255.0;

      double Max = max(max(r, g), b);
      double Min = min(min(r, g), b);

      // set value/brightness v
      v = Max;

      // calculate saturation s
      if (Max != 0)
            s = (Max - Min) / Max;
      else
            s = 0;
```

```
	// calculate hue h
	if (s == 0)
		h = UNDEFINED;
	else
	{
		double delta = Max - Min;
		if (r == Max)
			h = (g - b) / delta;//resulting color is between
yellow and magenta
		else if (g == Max)
			h = 2 + (b - r) / delta;// resulting color is between
cyan and yellow
		else if (b == Max)
			h = 4 + (r - g) / delta;// resulting color is between
magenta and cyan

		h *= 60;			// convert hue to degrees

		if (h < 0)			// make sure positive
			h += 360;
	}

}

////////////////////////////////////////////////////////////
// CIMAGE Construction/Destruction
////////////////////////////////////////////////////////////

CImage::CImage()
{
	m_buf	  = NULL;
	m_undobuf = NULL;
	m_width   = 0;
	m_height  = 0;
	m_widthDW = 0;
	m_filename.Empty();
}

CImage::~CImage()
{
	if (m_buf != NULL)
	{
		delete[] m_buf;
		m_buf = NULL;
	}
	if (m_undobuf != NULL)
	{
		delete[] m_undobuf;
		m_undobuf = NULL;
	}
}

////////////////////////////////////////////////////////////
// CIMAGE Member Functions
////////////////////////////////////////////////////////////
```

```
void CImage::LoadJPG()
//
// Desc:   Loads a JPG image from a file into the buffer
// Pre:    m_filename is the name of the file containting // the JPG
image
// Post:   m_buf contains the file; m_width and m_height
// are set;
//         m_undobuf has been allocated enough space to hold a copy of
// an image the size of m_buf.
//
{
      if (m_buf != NULL)              // m_buf is the global buffer
      {
            delete[] m_buf;
            m_buf = NULL;
      }

      if (m_undobuf != NULL)
      {
            delete[] m_undobuf;
            m_undobuf = NULL;
      }

      m_buf = JpegFile::JpegFileToRGB(m_filename, &m_width, &m_height);
      m_undobuf = (BYTE*)new BYTE[m_height * m_width * 3];

      JpegFile::BGRFromRGB(m_buf, m_width, m_height);
      JpegFile::VertFlipBuf(m_buf, m_width * 3, m_height);
}

void CImage::LoadBMP()
//
// Desc:   Loads a BMP image from a file into the buffer
// Pre:    m_filename is the name of the file containting the BMP image
// Post:   m_buf contains the file; m_width and m_height are set;
//         m_undobuf has been allocated enough space to hold a copy of
// an image the size of m_buf.
//
{
      if (m_buf != NULL)
      {
            delete[] m_buf;
            m_buf = NULL;
      }

      if (m_undobuf != NULL)
      {
            delete[] m_undobuf;
            m_undobuf = NULL;
      }

      BMPFile theBmpFile;
      m_buf = theBmpFile.LoadBMP(m_filename, &m_width, &m_height);
      if ((m_buf == NULL) || (theBmpFile.m_errorText != "OK"))
      {
            AfxMessageBox(theBmpFile.m_errorText, MB_ICONSTOP);
            m_buf = NULL;
```

```
            return;
        }

        m_undobuf = (BYTE*)new BYTE[m_height * m_width * 3];

        JpegFile::BGRFromRGB(m_buf, m_width, m_height);
        JpegFile::VertFlipBuf(m_buf, m_width * 3, m_height);
}

void CImage::DrawBMP(CDC* dc, CRect rectClient)
//
// Desc:   Displays m_buf on the screen
// Pre:    dc is a handle to the screen device context; rectClient is
// the screen size
// Post:   m_buf is displayed, centered, on the screen
//
{
        if (m_buf == NULL)      // if nothing in the buffer, get out
            return;

        if (dc != NULL)
        {
            // center
            int left = max(rectClient.left, ((rectClient.Width()  -
(int)m_width)  / 2));
            int top  = max(rectClient.top,  ((rectClient.Height() -
(int)m_height) / 2));

            // a 24-bit DIB is DWORD-aligned, vertically flipped and
has Red and Blue bytes
            // swapped.  We already did the RGB->BGR and the flip when
we read the image,
            // now do the DWORD-align

            BYTE* tmp;
            tmp = JpegFile::MakeDwordAlignedBuf(m_buf, m_width,
m_height, &m_widthDW);

            // set up DIB
            BITMAPINFOHEADER bmiHeader;
            bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
            bmiHeader.biWidth = m_width;
            bmiHeader.biHeight = m_height;
            bmiHeader.biPlanes = 1;
            bmiHeader.biBitCount = 24;
            bmiHeader.biCompression = BI_RGB;
            bmiHeader.biSizeImage = 0;
            bmiHeader.biXPelsPerMeter = 0;
            bmiHeader.biYPelsPerMeter = 0;
            bmiHeader.biClrUsed = 0;
            bmiHeader.biClrImportant = 0;

            // send it to the CDC
            // lines returns the number of lines actually displayed

            int lines = StretchDIBits(
                dc->m_hDC,
```

```
                    left,
                    top,
                    bmiHeader.biWidth,
                    bmiHeader.biHeight,
                    0,
                    0,
                    bmiHeader.biWidth,
                    bmiHeader.biHeight,
                    tmp,
                    (LPBITMAPINFO)&bmiHeader,
                    DIB_RGB_COLORS,
                    SRCCOPY);

            delete[] tmp;
        }
}

void CImage::SaveJPG(CString sFilename)
//
// Desc:   Saves the image in m_buf to file sFilename as a JPG
// Pre:    m_buf contains an image; sFilename is the filename to save
// m_buf to
// Post:   file has been saved as JPG to sFilename
//
{
      if (m_buf == NULL)
      {
            AfxMessageBox ("No Image to Save!");
            return;
      }

      // we vertical flip for display, undo that.
      JpegFile::VertFlipBuf(m_buf, m_width * 3, m_height);

      // we swap red and blue for display, undo that.
      JpegFile::BGRFromRGB(m_buf, m_width, m_height);

      BOOL ok = JpegFile::RGBToJpegFile(
            sFilename,
            m_buf,
            m_width,
            m_height,
            TRUE,                  // save in color
            100);                  // quality value [1..100]

      if (!ok)
            AfxMessageBox("Write Error!", MB_ICONSTOP);
      else
      {
            m_filename = sFilename;
            LoadJPG();
      }
}

void CImage::SaveBMP(CString sFilename)
//
// Desc:   Saves the image in m_buf to file sFilename as a BMP
```

```
// Pre:    m_buf contains an image; sFilename is the filename to save
// m_buf to
// Post:   file has been saved as BMP to sFilename
//
{
      if (m_buf == NULL)
      {
            AfxMessageBox ("No Image to Save!");
            return;
      }

      // image in m_buf is already BGR and vertically flipped
      // so we don't need to do that for this function

      BMPFile theBmpFile;
      theBmpFile.SaveBMP(
            sFilename,
            m_buf,
            m_width,
            m_height);

      if (theBmpFile.m_errorText != "OK")
            AfxMessageBox(theBmpFile.m_errorText, MB_ICONSTOP);
      else
      {
            m_filename = sFilename;
            LoadBMP();
      }
}

void CImage::ProcessFilter(enum filtertype type)
//
// Desc:   Creates a CFilter object attached to this image (i.e.
//         the filter will process this image) of filtertype type.
// Pre:    type is the filtertype you wish to use to filter m_buf
// Post:   If a filter took place, m_buf is the filtered image and
// m_undobuf is a copy of m_buf before the filter.
//         If there was no filter, m_buf is unchanged and m_undobuf is
//         undefined.
// Source: custom
//
{
      CFilter currentfilter(this, type);
      currentfilter.go();
}


////////////////////////////////////////////////////////////////////////
// CFILTER Notes:
//   CFilter is declared as a friend class in CImage.  Everything in
//   CFilter can access private members in CImage.  The function
//   descriptions and pre/postconditions below may mention members not
//   declared in CFilter (e.g. m_buf).  These members are declared in
//   CImage.
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
```

```
// CFILTER Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFilter::CFilter(CImage* img, filtertype type)
{
      m_pImage = img;
      m_height = img->m_height;
      m_width  = img->m_width;
      m_type   = type;
}

CFilter::~CFilter()
{

}

//////////////////////////////////////////////////////////////////////
// CFILTER Member Functions
//////////////////////////////////////////////////////////////////////

void CFilter::go()
//
// Desc:   Calls the specified filter function
// Pre:    m_type is the filter to be called
// Post:   If a filter took place, m_buf is the filtered image and
// m_undobuf is a copy of m_buf before the filter.
//         If there was no filter, m_buf is unchanged and m_undobuf is
//         undefined.
// Source: custom
//
{
      bool ok = false;

      if (m_type == UNDO)
            CopyImageBuffer(m_pImage->m_undobuf, m_pImage->m_buf);
      else
      {
            switch (m_type)
            {
                  // POINT PROCESSES
                  case GRAYINT:                          ok = grayint();
                  break;
                  case GRAYHUE:                          ok = grayhue();
                  break;
                  case GRAYSAT:                          ok = graysat();
                  break;
                  case HUESAT:                           ok = huesat();
                  break;
                  case EQINTENSITY:                ok = eqintensity();
            break;

                  // AREA PROCESSES
                  case BLUR:                             ok = blur();
                  break;
                  case MEDIANBLUR:                 ok = medianblur();
            break;
```

```
                case CONVOLUTION:               ok = convolution();
            break;

                // AREA PROCESSES (ARTISTIC)
                case OIL:                               ok = oil();
                break;
                case FROSTGLASS:                ok = frostglass();
            break;
                case RANDBLUR:                          ok = randblur();
            break;
                case RAINDROPS:                         ok = raindrops();
            break;

                // GEOMETRIC PROCESSES
                case FISHEYE:                           ok = fisheye();
                break;
                case POLAR:                             ok = polar();
                break;

                default: ok = false;
            }

            if (ok)
            {
                // Allow undo
                CopyImageBuffer(m_pImage->m_buf, m_pImage-
>m_undobuf);
                CWnd* pMain = AfxGetMainWnd();
                if (pMain != NULL)
                {
                    CMenu* pMenu = pMain->GetMenu();
                    pMenu->EnableMenuItem(ID_UNDO, MF_ENABLED);
                }

                MatrixToBuffer(m_filt, m_pImage->m_buf);  // Send our
filtered image to the buffer to display
                FreeImageMatrix(m_img, m_height);               //
Free image matrix memory
                FreeImageMatrix(m_filt, m_height);              // "
            }
        }
}


////////////////////////////////////////////////////////////////////////
// CFILTER Private Member Functions
////////////////////////////////////////////////////////////////////////

// IMAGE MATRIX-BUFFER SUPPORT FUNCTIONS ////////////////////////////

// imagematrix CFilter::CreateImageMatrix(unsigned int row, unsigned
// int col)
//
// Desc:   Allocates memory for an image matrix of pixels, of size row
// by col
// Pre:    row are col are size for matrix allocation
// Post:   Pointer to allocated memory returned.
```

```
//
{
      // dynmaic allocation of a two-dimensional array
      imagematrix img = NULL;
      img = (imagematrix)malloc(row * sizeof(imagerow));

      if (img == NULL)
            return(NULL);

      for (unsigned int i = 0; i < m_height; i++)
      {
            img[i] = (imagerow)malloc(col * sizeof(pixel));
            if (img[i] == NULL)
            {
                  FreeImageMatrix(img, row);
                  return(NULL);
            }
      }
      return(img);
}

void CFilter::FreeImageMatrix(imagematrix img, unsigned int row)
//
// Desc:   Deallocates memory from an image matrix of pixels, of row
// rows
// Pre:    img was allocated by CreateImageMatrix(...); row is number
//         of rows in imagematrix img
// Post:   memory deallocated
//
{
      for (unsigned int i = 0; i < row; i++)
            free(img[i]);
      free(img);
}

boolmatrix  CFilter::CreateBoolMatrix(unsigned int row, unsigned int
col)
//
// Desc:   Allocates memory for an matrix of bool, of size row by col
// Pre:    row are col are size for matrix allocation
// Post:   Pointer to allocated memory returned.
//
{
      // dynmaic allocation of a two-dimensional array
      boolmatrix bmtx = NULL;
      bmtx = (boolmatrix)malloc(row * sizeof(boolrow));

      if (bmtx == NULL)
            return(NULL);

      for (unsigned int i = 0; i < m_height; i++)
      {
            bmtx[i] = (boolrow)malloc(col * sizeof(bool));
            if (bmtx[i] == NULL)
            {
                  FreeBoolMatrix(bmtx, row);
                  return(NULL);
```

```
            }
        }
        return(bmtx);
}

void CFilter::FreeBoolMatrix(boolmatrix bmtx, unsigned int row)
//
// Desc:    Deallocates memory from an bool matrix, of row rows
// Pre:     bmtx was allocated by CreateBoolMatrix(...); row is number
//          of rows in boolmatrix bmtx
// Post:    memory deallocated
//
{
        for (unsigned int i = 0; i < row; i++)
             free(bmtx[i]);
        free(bmtx);
}

void CFilter::BufferToMatrix(BYTE* buf, imagematrix img)
//
// Desc:    Converts string of BYTEs (buf) to imagematrix img
// Pre:     buf != NULL, img created from CreateImageMatrix,
//          m_height and m_width are dimensions of imagematrix img
// Post:    img contains RGB pixels translated from buf
//
{
        UINT col, row;
        for (row=0; row < m_height; row++)
        {
             for (col=0; col < m_width; col++)
             {
                  LPBYTE pRed, pGrn, pBlu;
                  pBlu = buf + row * m_width * 3 + col * 3;
                  pGrn = buf + row * m_width * 3 + col * 3 + 1;
                  pRed = buf + row * m_width * 3 + col * 3 + 2;
                  img[row][col].r = *pRed;
                  img[row][col].g = *pGrn;
                  img[row][col].b = *pBlu;
             }
        }
}

void CFilter::MatrixToBuffer(imagematrix img, BYTE* buf)
//
// Desc:    Converts imagematric img to string of BYTEs (buf)
// Pre:     img contains an image of pixels, buf != NULL,
//          m_height and m_width are dimensions of imagematrix img
// Post:    buf contains BTYEs translated from img pixels
//
{
        UINT col, row;
        for (row = 0; row < m_height; row++)
        {
             for (col = 0; col < m_width; col++)
             {
                  BYTE Red, Grn, Blu;
                  Red = img[row][col].r;
```

```
                Grn = img[row][col].g;
                Blu = img[row][col].b;
                buf[row * m_width * 3 + col * 3] = Blu;
                buf[row * m_width * 3 + col * 3 + 1] = Grn;
                buf[row * m_width * 3 + col * 3 + 2] = Red;
            }
        }
}

void CFilter::CopyImageBuffer(BYTE* buf, BYTE* bufcopy)
//
// Desc:   Copies buf to bufcopy.
// Pre:    buf, buffcopy != NULL both must have been allocated
//         the same memory space (this is done in CImage::LoadJPG or
//         in CImage::LoadBMP).
// Post:   Both buf and bufcopy exist on successful return.  If buf ==
// NULL upon entry, buf and bufcopy are undefined on return.
// Source: custom
//
{
      if (buf==NULL)
            return;

      if (bufcopy==NULL)
            return;

      int size = m_height * m_width * 3;

      for (int i = 0; i < size; i++)
      {
            bufcopy[i] = buf[i];
      }
      return;
}

// FILTER SUPPORT FUNCTIONS ////////////////////////////////////

void CFilter::FilterInit()
//
// Desc:   Prepares m_img and m_filt so the image in the buffer m_buf
// can be filtered to the matrix for m_filt
// Pre:    m_buf != NULL, m_height and m_width are size of image in
// m_buf
// Post:   m_img created from m_buf, m_filt created the same size as
// m_img but not initialized.
//
// Source: custom
//
{
      // place the buffer in a dynamic image matrix
      m_img = CreateImageMatrix(m_height, m_width);
      BufferToMatrix(m_pImage->m_buf, m_img);

      // create blank 'to-be-filtered' image
      m_filt = CreateImageMatrix(m_height, m_width);
}
```

```
pixel CFilter::MostFrequentColor(unsigned int x, unsigned int y,
unsigned int r, unsigned int I)
//
// Desc:   Determines the most frequently appearing color in a sub-
// matrix of m_img according to the pre/postconditions below.
// Pre:    (x,y) is the 'center' pixel in a sub-matrix of m_img with
// radius r.I is the number of different intensities to distinguish in
// the sub-matrix.
// Post:   The 'average' color of the pixels with the most frequently
// appearing intensity in the sub-matrix of pixels centered at (x,y)
//          with radius r in m_img.
//
{
      int i, j;
      pixel q;
      BYTE intensity;
      BYTE MAXINTENSITY = I;
      double scale = MAXINTENSITY / 255.0;

      // dynamic array allocations:
      //    e.g. BYTE intensitycount[I+1];

      BYTE* intensitycount = NULL;
      intensitycount = (BYTE*)malloc((I+1) * sizeof(BYTE)); // index =
intensity

                        // value at index = count;
      UINT* averagecolorR = NULL;
      averagecolorR = (UINT*)malloc((I+1) * sizeof(UINT));

      UINT* averagecolorG = NULL;
      averagecolorG = (UINT*)malloc((I+1) * sizeof(UINT));

      UINT* averagecolorB = NULL;
      averagecolorB = (UINT*)malloc((I+1) * sizeof(UINT));

      for (i = 0; i <= MAXINTENSITY; i++)
            intensitycount[i] = 0;
            // set all counters to 0

      for (i = int(x-r); i <= int(x+r); i++)
            for (j = int(y-r); j <= int(y+r); j++)
            {
                  if (i >= 0 && i < (int)m_height && j >= 0 && j <
(int)m_width)
                  {
                        q = m_img[i][j];
                        intensity = q.intensity() * scale;
      // find intensity
                        intensitycount[intensity]++;

                        if (intensitycount[intensity] == 1)
                        {
                              averagecolorR[intensity] = q.r;
                              averagecolorG[intensity] = q.g;
                              averagecolorB[intensity] = q.b;
                        }
```

```
                        else
                        {
                                averagecolorR[intensity] += q.r;
        // keep a running total
                                averagecolorG[intensity] += q.g;
        //   we'll divide later
                                averagecolorB[intensity] += q.b;
                        }
                }
        }

        // find max intensity

        intensity = 0;
        int maxinstance = 0;

        for (i = 0; i <= MAXINTENSITY; i++)
        {
                if (intensitycount[i] > maxinstance)
                {
                        intensity = i;
                        maxinstance = intensitycount[i];
                }
        }

        pixel mostfrequent;
        mostfrequent.r = averagecolorR[intensity] / maxinstance;
        mostfrequent.g = averagecolorG[intensity] / maxinstance;
        mostfrequent.b = averagecolorB[intensity] / maxinstance;

        free(intensitycount);
        free(averagecolorR);
        free(averagecolorG);
        free(averagecolorB);

        return mostfrequent;
}

pixel CFilter::RandomColor(unsigned int x, unsigned int y, unsigned int
r)
//
// Desc:   Chooses a random color from those appearing in a sub-matrix
// of m_img according to the pre/postconditions below.
// Pre:    (x,y) is the 'center' pixel in a sub-matrix of m_img with
// radius r.
// Post:   The 'average' color of the pixels of a random intensity in
// the sub-matrix of pixels centered at (x,y) with radius r.  Intensity
//         is chosen by giving weight to the more common intensities.
//
{
        int i, j;
        pixel q;
        BYTE intensity;
        const BYTE MAXINTENSITY = 255;

        BYTE intensitycount[MAXINTENSITY+1];        // index = intensity,
value = count;
```

```
      UINT averagecolorR[MAXINTENSITY+1];
      UINT averagecolorG[MAXINTENSITY+1];
      UINT averagecolorB[MAXINTENSITY+1];

      for (i = 0; i <= MAXINTENSITY; i++)
      {
            intensitycount[i] = 0;  // set all counters to 0
      }

      int numint = 0;

      for (i = int(x-r); i <= int(x+r); i++)
            for (j = int(y-r); j <= int(y+r); j++)
            {
                  if (i >= 0 && i < (int)m_height && j >= 0 && j <
(int)m_width)
                  {
                        q = m_img[i][j];
                        intensity = q.intensity();
                        intensitycount[intensity]++;
                        numint++;

                        if (intensitycount[intensity] == 1)
                        {
                              averagecolorR[intensity] = q.r;
                              averagecolorG[intensity] = q.g;
                              averagecolorB[intensity] = q.b;
                        }
                        else
                        {
                              averagecolorR[intensity] += q.r;
                              averagecolorG[intensity] += q.g;
                              averagecolorB[intensity] += q.b;
                        }

                  }
            }
      int randnum, count, indx;
      do
      {
            // weighted random
            randnum = int((rand()+1) * ((double)numint /
(RAND_MAX+1)));
            count = 0;
            indx = 0;
            do
            {
                  count += intensitycount[indx];
                  indx++;
            } while (count < randnum);

            intensity = indx-1;

      } while (intensitycount[intensity] == 0);

      pixel randpix;
```

```
	randpix.r = averagecolorR[intensity] / intensitycount[intensity];
	randpix.g = averagecolorG[intensity] / intensitycount[intensity];
	randpix.b = averagecolorB[intensity] / intensitycount[intensity];

	return randpix;
}

void CFilter::MPQ(BYTE* buf, UINT* index, int lb, int ub)
//
// Desc:   Quicksort implementation for MedianPixel(...)
//
{
	int i, j;
	BYTE pivot;

	if (ub > lb)
	{
		j = lb - 1;                              // make the partitions
		pivot = buf[index[ub]];
		for (i = lb; i <= ub; i++)      // main loop
		{
			if (buf[index[i]] <= pivot)
			{
				j++;
				if (i != j)
					std::swap(index[i], index[j]);
			}
		}
		MPQ(buf, index, lb, j-1);
		MPQ(buf, index, j+1, ub);
	}
}

pixel CFilter::MedianPixel(BYTE* buf, pixel* pixbuf, int length)
//
// Desc:   Modification of the Quicksort sorting algorithm.
// Pre:        buf and pixbuf != NULL; buf is intensities of the
// corresponding pixels in pixbuf; length is number of entries in both
// buf and pixbuf to use to find the median (beginning with element 0).
// Post:   Returns the 'median' pixel (based on intensity) of pixbuf.
//         buf and pixbuf are unmodified.
//
{
	// dynamic array allocation
	UINT* index = NULL;
	index = (UINT*)malloc(length * sizeof(UINT));

	int x;

	for (x = 0; x < length; x++)        // initialize index with
indicies of buf/pixbuf
		index[x] = x;                                  // we will sort
the indicies, not the elements

and pixbuf                                                  // of buf
	UINT lb = 0;
	UINT ub = length - 1;
```

```
      MPQ(buf, index, lb, ub);                    // call quicksort

      pixel p = pixbuf[index[length/2]];  // use index to get the
median from pixbuf

      free(index);

      return p;
}

// FILTERS ////////////////////////////////////////////////////

// bool CFilter::grayint()
//
// Desc:   Convert to grayscale using color intensity
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
      // Init Matrices m_img and m_filt
      FilterInit();

      // Setup and Call Progress Dialog
      CProgressDlg pdlg;
      pdlg.Create();
      unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
      unsigned int pixelsProcessed = 0;
      bool canceled = FALSE;

      // Filter Algorithm ///////////////////////////////////

      unsigned int i, j;
      BYTE x;

      for (i = 0; i < m_height; i++)
      {
            if (pdlg.CheckCancelButton())
                  if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
                  {
                        canceled = true;
                        break;
                  }

            for (j = 0; j < m_width; j++)
            {
                  x = m_img[i][j].intensity();
                  m_filt[i][j].color(x);

                  pixelsProcessed++;
                  if (pixelsProcessed % pixels == 0)
                        pdlg.StepIt();
            }
```

```
	}

	pdlg.DestroyWindow();
	return !canceled;
}

bool CFilter::grayhue()
//
// Desc:   Convert to grayscale using color hue
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
	// Init Matrices m_img and m_filt
	FilterInit();

	// Setup and Call Progress Dialog
	CProgressDlg pdlg;
	pdlg.Create();
	unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
	unsigned int pixelsProcessed = 0;
	bool canceled = FALSE;

	// Filter Algorithm ///////////////////////////////////

	unsigned int i, j;
	BYTE x;
	hsvpixel hsvpix;

	for (i = 0; i < m_height; i++)
	{
		if (pdlg.CheckCancelButton())
			if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
			{
				canceled = true;
				break;
			}

		for (j = 0; j < m_width; j++)
		{
			hsvpix = m_img[i][j];
			x = (BYTE)(hsvpix.h * (255.0 / 360));
			m_filt[i][j].color(x);

			pixelsProcessed++;
			if (pixelsProcessed % pixels == 0)
				pdlg.StepIt();
		}
	}

	pdlg.DestroyWindow();
	return !canceled;
}
```

```
bool CFilter::graysat()
//
// Desc:   Convert to grayscale using color saturation
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
      // Init Matrices m_img and m_filt
      FilterInit();

      // Setup and Call Progress Dialog
      CProgressDlg pdlg;
      pdlg.Create();
      unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
      unsigned int pixelsProcessed = 0;
      bool canceled = FALSE;

      // Filter Algorithm ////////////////////////////////////

      unsigned int i, j;
      BYTE x;
      hsvpixel hsvpix;

      for (i = 0; i < m_height; i++)
      {
            if (pdlg.CheckCancelButton())
                  if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
                  {
                        canceled = true;
                        break;
                  }

            for (j = 0; j < m_width; j++)
            {
                  hsvpix = m_img[i][j];
                  x = (BYTE)(hsvpix.s * 255);
                  m_filt[i][j].color(x);

                  pixelsProcessed++;
                  if (pixelsProcessed % pixels == 0)
                        pdlg.StepIt();
            }
      }

      pdlg.DestroyWindow();
      return !canceled;
}

bool CFilter::huesat()
//
// Desc:   Modify Hue, Saturation, and Value (Brightness) of the image.
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
```

```
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
      // Call Dialog
      CHueSatDlg dlg;
      if (dlg.DoModal() != IDOK)
            return false;

      // Init Matrices m_img and m_filt
      FilterInit();

      // Setup and Call Progress Dialog
      CProgressDlg pdlg;
      pdlg.Create();
      unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
      unsigned int pixelsProcessed = 0;
      bool canceled = FALSE;

      // Filter Algorithm /////////////////////////////////////

      unsigned int i, j;
      hsvpixel hsvpix;

      for (i = 0; i < m_height; i++)
      {
            if (pdlg.CheckCancelButton())
                  if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
                  {
                        canceled = true;
                        break;
                  }

            for (j = 0; j < m_width; j++)
            {
                  hsvpix = m_img[i][j];

                  if (dlg.m_bColorize)
                        hsvpix.h = dlg.m_iHue + 180;
                  else
                  {
                        hsvpix.h += dlg.m_iHue;
                        if (hsvpix.h < 0)
                              hsvpix.h += 360;
                        else if (hsvpix.h > 360)
                              hsvpix.h -=360;
                  }

            if (dlg.m_iLight < 0)
                  hsvpix.v = (dlg.m_iLight + 100) * (hsvpix.v / 100.0);
            else if (dlg.m_iLight > 0)
                  hsvpix.v = ((dlg.m_iLight) * ((1 - hsvpix.v) /
100.0)) + hsvpix.v;
```

```
            if (dlg.m_iSat < 0)
                  hsvpix.s = (dlg.m_iSat + 100) * (hsvpix.s / 100.0);
            else if (dlg.m_iSat > 0)
                  hsvpix.s = ((dlg.m_iSat) * ((1 - hsvpix.s) / 100.0))
+ hsvpix.s;

                  m_filt[i][j] = hsvpix;

                  pixelsProcessed++;
                  if (pixelsProcessed % pixels == 0)
                        pdlg.StepIt();
            }
      }

      pdlg.DestroyWindow();
      return !canceled;
}

bool CFilter::eqintensity()
//
// Desc:   Convert every pixel to the same intensity
//         (value/brightness).  Theoretically,based on color theory,
// this should produce a 'pleasing' color scheme given the right
//         proportion of color (hue).
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
      // Call Dialog
      CEqIntensityDlg dlg;
      if (dlg.DoModal() != IDOK)
            return false;

      // Init Matrices m_img and m_filt
      FilterInit();

      // Setup and Call Progress Dialog
      CProgressDlg pdlg;
      pdlg.Create();
      unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
      unsigned int pixelsProcessed = 0;
      bool canceled = FALSE;

      // Filter Algorithm /////////////////////////////////////

      unsigned int i, j;
      hsvpixel hsvpix;

      for (i = 0; i < m_height; i++)
      {
            if (pdlg.CheckCancelButton())
                  if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
                  {
```

```
                        canceled = true;
                        break;
                    }

                for (j = 0; j < m_width; j++)
                {
                    hsvpix = m_img[i][j];
                    hsvpix.v = dlg.m_iIntensity / 100.0;
                    m_filt[i][j] = hsvpix;

                    pixelsProcessed++;

                    if (pixelsProcessed % pixels == 0)
                        pdlg.StepIt();
                }
            }

            pdlg.DestroyWindow();
            return !canceled;
}

bool CFilter::blur()
//
// Desc:   Blur Filter.  Simple average of all surrounding pixels in a
// given radius to the center pixel.  Also called moving average.
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
            // Call Dialog
            CBlurDlg dlg;
            if (dlg.DoModal() != IDOK)
                return false;

            // Init Matrices m_img and m_filt
            FilterInit();

            // Setup and Call Progress Dialog
            CProgressDlg pdlg;
            pdlg.Create();
            unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
            unsigned int pixelsProcessed = 0;
            bool canceled = FALSE;

            // Filter Algorithm ////////////////////////////////////

            float R, G, B;
            int i, j, k, l, x, y;

            int numpixels;                  // number of pixels in the
'window' for a particular window
                                            //   will be smaller
than normal for pixels around the border
```

```
	int RAD = dlg.m_nRadius;
	int circumference = RAD * 2;
	int ksq;                                    // k^2
	double  r;                                  // polar coordinate radius

	for (i = 0; i < (int)m_height; i++)
	{
		if (pdlg.CheckCancelButton())
			if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
			{
				canceled = true;
				break;
			}

		for (j = 0; j < (int)m_width; j++)
		{
			R = G = B = 0;
			numpixels = 0;

			for (k = -RAD; k <= RAD; k++)
			{
				if (!dlg.m_bFast) ksq = k*k;
				for (l = -RAD; l <= RAD; l++)
				{
					// convert this square to polar
					if (!dlg.m_bFast) r = sqrt(ksq + l*l);

					// if we're inside R (i.e. inside the
circle, inside the square) do...
					if ((dlg.m_bFast) || (!dlg.m_bFast && r
<= RAD))
					{
						x = i + k;
						y = j + l;

						if (x >= 0 && x < (int)m_height &&
y >= 0 && y < (int)m_width)
						{
							R += (float)m_img[x][y].r;
							G += (float)m_img[x][y].g;
							B += (float)m_img[x][y].b;
							numpixels++;
						}
					}
				}
			}
			m_filt[i][j].color(
				(BYTE)(R/numpixels), (BYTE)(G/numpixels),
(BYTE)(B/numpixels));

			pixelsProcessed++;
			if (pixelsProcessed % pixels == 0)
				pdlg.StepIt();
		}
	}
```

```
      pdlg.DestroyWindow();
      return !canceled;
}

bool CFilter::medianblur()
//
// Desc:   Median blur filter.  For each pixel in the original image,
// assigns the cooresponing pixel in the filtered image to the pixel
// with the median intensity of the surrounding pixels.  Radius
// determined by CMedianBlurDlg.  Similar to blur() but preserves edges
// better.
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
      // Call Dialog
      CMedianBlurDlg dlg;
      if (dlg.DoModal() != IDOK)
            return false;

      // Init Matrices m_img and m_filt
      FilterInit();

      // Setup and Call Progress Dialog
      CProgressDlg pdlg;
      pdlg.Create();
      unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
      unsigned int pixelsProcessed = 0;
      bool canceled = FALSE;

      // Filter Algorithm ////////////////////////////////////

      float R, G, B;
      int i, j, k, l, x, y;
      BYTE intensity;
      pixel q;

      int numpixels;     // number of pixels in the 'window' for a
particular window
                         //   will be smaller than normal for pixels
around the border

      int circumference = dlg.m_iRadius * 2 + 1;
      int circumference2 = circumference * circumference;

      pixel* colorsbuf = NULL;
      colorsbuf = (pixel*)malloc(circumference2 * sizeof(pixel));

      BYTE* intbuf = NULL;
      intbuf = (BYTE*)malloc(circumference2 * sizeof(BYTE));

      for (i = 0; i < (int)m_height; i++)
      {
            if (pdlg.CheckCancelButton())
```

```
                    if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
                    {
                          canceled = true;
                          break;
                    }

              for (j = 0; j < (int)m_width; j++)
              {
                    R = G = B = 0;
                    numpixels = 0;
                    for (k = 0; k < circumference; k++)
                    {
                          for (l = 0; l < circumference; l++)
                          {
                                x = i + k - (dlg.m_iRadius);
                                y = j + l - (dlg.m_iRadius);

                    if (x >= 0 && x < (int)m_height && y >= 0 && y <
(int)m_width)
                    {
                          q = m_img[x][y];
                          intensity = q.intensity();
                          intbuf[circumference*k+l] = intensity;
                          colorsbuf[circumference*k+l] = m_img[x][y];
                          numpixels++;
                                }
                          }
                    }
               m_filt[i][j] = MedianPixel(intbuf, colorsbuf, numpixels);

                    pixelsProcessed++;
                    if (pixelsProcessed % pixels == 0)
                          pdlg.StepIt();
              }
        }

        free(colorsbuf);
        free(intbuf);

        pdlg.DestroyWindow();
        return !canceled;
}

bool CFilter::convolution()
//
// Desc:   Custom convolution filter.  For each pixel in the original
// image, assigns the cooresponing pixel in the filtered image to the
// pixel with a weighted average of the surrounding pixels (the
// convolution kernel).  The kernel is determined by CConvolutionDlg.
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
        // Call Dialog
```

```
	CConvolutionDlg dlg;
	if (dlg.DoModal() != IDOK)
		return false;

	// Init Matrices m_img and m_filt
	FilterInit();

	// Setup and Call Progress Dialog
	CProgressDlg pdlg;
	pdlg.Create();
	unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
	unsigned int pixelsProcessed = 0;
	bool canceled = FALSE;

	// Filter Algorithm ////////////////////////////////////

	float hcoe[5][5];
	hcoe[0][0] = (float)dlg.m_iPos11;
	hcoe[0][1] = (float)dlg.m_iPos12;
	hcoe[0][2] = (float)dlg.m_iPos13;
	hcoe[0][3] = (float)dlg.m_iPos14;
	hcoe[0][4] = (float)dlg.m_iPos15;
	hcoe[1][0] = (float)dlg.m_iPos21;
	hcoe[1][1] = (float)dlg.m_iPos22;
	hcoe[1][2] = (float)dlg.m_iPos23;
	hcoe[1][3] = (float)dlg.m_iPos24;
	hcoe[1][4] = (float)dlg.m_iPos25;
	hcoe[2][0] = (float)dlg.m_iPos31;
	hcoe[2][1] = (float)dlg.m_iPos32;
	hcoe[2][2] = (float)dlg.m_iPos33;
	hcoe[2][3] = (float)dlg.m_iPos34;
	hcoe[2][4] = (float)dlg.m_iPos35;
	hcoe[3][0] = (float)dlg.m_iPos41;
	hcoe[3][1] = (float)dlg.m_iPos42;
	hcoe[3][2] = (float)dlg.m_iPos43;
	hcoe[3][3] = (float)dlg.m_iPos44;
	hcoe[3][4] = (float)dlg.m_iPos45;
	hcoe[4][0] = (float)dlg.m_iPos51;
	hcoe[4][1] = (float)dlg.m_iPos52;
	hcoe[4][2] = (float)dlg.m_iPos53;
	hcoe[4][3] = (float)dlg.m_iPos54;
	hcoe[4][4] = (float)dlg.m_iPos55;

	float R, G, B;
	int a, b, i, j, k, l, x, y;

	const int radius = 2;
	const int circumference = 4;  // actually 5, but the algorithm
takes care of the 'center'
                                                  // pixel in
implementation; this notion is taken from the
                                                  // blur() filter
	float sum = 0;                          // convolution kernal sum
	for (a = 0; a < 5; a++)
		for (b = 0; b < 5; b++)
```

```
                sum += hcoe[a][b];

        if (sum <= 0)
                sum = 1;
        else
        {
                for (a = 0; a < 5; a++)
                        for (b = 0; b < 5; b++)
                                hcoe[a][b] /= sum;
        }


        for (i = 0; i < (int)m_height; i++)
        {
                if (pdlg.CheckCancelButton())
                        if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
                        {
                                canceled = true;
                                break;
                        }

                for (j = 0; j < (int)m_width; j++)
                {
                        R = G = B = 0;
                        for (k = 0; k <= circumference; k++)
                        {
                                for (l = 0; l <= circumference; l++)
                                {
                                        x = i + k - radius;
                                        y = j + l - radius;

                                        if (x >= 0 && x < (int)m_height && y >= 0
&& y < (int)m_width)
                                        {
                                        R += (float)m_img[x][y].r * hcoe[k][l];
                                        G += (float)m_img[x][y].g * hcoe[k][l];
                                        B += (float)m_img[x][y].b * hcoe[k][l];
                                        }
                                }
                        }
                        m_filt[i][j].color(R, G, B);

                        pixelsProcessed++;
                        if (pixelsProcessed % pixels == 0)
                                pdlg.StepIt();
                }
        }

        pdlg.DestroyWindow();
        return !canceled;
}

bool CFilter::oil()
//
// Desc:   Oil paint effect filter.  For each pixel, sets the
// cooresponding pixel in the filtered image to the 'most frequently
```

```
// appearing color' of the surrounding pixels.  Calls
// MostFrequentColor(..).  Radius (brush size)
// determined [1..5] from COilDlg.  Smoothness [10..255] also
// determined from COilDlg.
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
      // Call Dialog
      COilDlg dlg;
      if (dlg.DoModal() != IDOK)
            return false;

      // Init Matrices m_img and m_filt
      FilterInit();

      // Setup and Call Progress Dialog
      CProgressDlg pdlg;
      pdlg.Create();
      unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
      unsigned int pixelsProcessed = 0;
      bool canceled = FALSE;

      // Filter Algorithm ////////////////////////////////////

      unsigned int i, j;

      for (i = 0; i < m_height; i++)
      {
            if (pdlg.CheckCancelButton())
                  if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
                  {
                        canceled = true;
                        break;
                  }

            for (j = 0; j < m_width; j++)
            {
                  m_filt[i][j] = MostFrequentColor(i, j, dlg.m_iBrush,
dlg.m_iSmooth);
                  pixelsProcessed++;

                  if (pixelsProcessed % pixels == 0)
                        pdlg.StepIt();
            }
      }

      pdlg.DestroyWindow();
      return !canceled;
}

bool CFilter::frostglass()
//
```

```
// Desc:   Produces an image that appears the viewer is looking through
//         frosted glass.  For each pixel, sets the cooresponding pixel
//         in the filtered image to a 'weighted (by intensity) random
// color'of the surrounding pixels.  Calls RandomColor(...).  Radius
//         (frost amount) determined [1..10] from CFrostGlassDlg.
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
      // Call Dialog
      CFrostGlassDlg dlg;
      if (dlg.DoModal() != IDOK)
            return false;

      // Init Matrices m_img and m_filt
      FilterInit();

      // Setup and Call Progress Dialog
      CProgressDlg pdlg;
      pdlg.Create();
      unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
      unsigned int pixelsProcessed = 0;
      bool canceled = FALSE;

      // Filter Algorithm ////////////////////////////////////

      unsigned int i, j;

      for (i = 0; i < m_height; i++)
      {
            if (pdlg.CheckCancelButton())
                  if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
                  {
                        canceled = true;
                        break;
                  }

            for (j = 0; j < m_width; j++)
            {
                  m_filt[i][j] = RandomColor(i, j, dlg.m_iFrost);
                  pixelsProcessed++;

                  if (pixelsProcessed % pixels == 0)
                        pdlg.StepIt();
            }
      }

      pdlg.DestroyWindow();
      return !canceled;
}

bool CFilter::randblur()
//
```

```
// Desc:   Radomly blurs square sections in the image.  User chooses
// options from CRandBlurDlg: m_iSize is dimension of each square,
// m_iRadius is the blur radius (how much each block is blurred),
// m_iAmount is how many sqaures sections to blur.
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
      // Call Dialog
      CRandBlurDlg dlg;
      if (dlg.DoModal() != IDOK)
            return false;

      // Init Matrices m_img and m_filt
      FilterInit();

      // Setup and Call Progress Dialog
      CProgressDlg pdlg;
      pdlg.Create();
      pdlg.SetRange(0, dlg.m_iAmount);
      bool canceled = FALSE;

      // Filter Algorithm ////////////////////////////////////

      unsigned int i, j, k, l;
      unsigned int x, y;                  // beginning coord (random)
      unsigned int ex, ey;                // end coord (from beginning)

      int rad = dlg.m_iRadius;            // blur radius (bluriness)
      int RAD = rad * rad;     // total num pix processed per block
      int rad2 = rad / 2;                 // half of the radius

      float R, G, B;

      // make copy of image
      for (i = 0; i < m_height; i++)
            for (j = 0; j < m_width; j++)
                  m_filt[i][j] = m_img[i][j];

      // and blur some of it
      for (int numblurs = 0; numblurs < dlg.m_iAmount; numblurs++)
      {
            if (pdlg.CheckCancelButton())
                  if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
                  {
                        canceled = true;
                        break;
                  }

            do
            {
                  x = int(rand() * ((double)m_height / RAND_MAX));
                  y = int(rand() * ((double)m_width / RAND_MAX));
                  ex = x + dlg.m_iSize - 1;
```

```
                ey = y + dlg.m_iSize - 1;
        } while (ex >= m_height || ey >= m_width);

        for (i = x + rad2; i < ex - rad2; i++)
            for (j = y + rad2; j < ey - rad2; j++)
            {
                R = G = B = 0;
                for (k = 0; (int)k < rad; k++)
                {
                    for (l = 0; (int)l < rad; l++)
                    {
                    R += (float)m_img[i+k-rad2][j+l-rad2].r;
                    G += (float)m_img[i+k-rad2][j+l-rad2].g;
                    B += (float)m_img[i+k-rad2][j+l-rad2].b;
                    }
                }
        m_filt[i][j].color(BYTE(R/RAD), BYTE(G/RAD), BYTE(B/RAD));
            }
            pdlg.StepIt();
    }

    pdlg.DestroyWindow();
    return !canceled;
}

bool CFilter::raindrops()
//
// Desc:   Produces effect of water drops on a surface.  Number of
// water drops (dlg.m_iAmount) and maximum size (dlg.m_iSize) are
// determined from CRaindropsDlg.
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
    // Call Dialog
    CRaindropsDlg dlg;
    if (dlg.DoModal() != IDOK)
        return false;

    // Init Matrices m_img and m_filt
    FilterInit();

    // Setup and Call Progress Dialog
    CProgressDlg pdlg;
    pdlg.Create();
    pdlg.SetRange(0, dlg.m_iAmount);
    bool canceled = FALSE;

    // Filter Algorithm /////////////////////////////////////

    int i, j, k, l, m, n;// for loop processing (image coordinates)

    int         x, y;       // center coord of raindrop (random)
```

```
      double  r, a; // polar coordinates of raindrop (radius, angle)
      double  oldr;     // polar radius before fish eye transformation

      bool  findanother = false; // used to find 'good' random
coordinate of raindrop
      int         count = 0;

      int SIZE = dlg.m_iSize; // size of largest raindrop (from dialog)
      int         size;       // size of current raindrop
      int         size2;      // half of the current raindrop size
      int         R;          // maxium raindrop radius (same as size2)

      double  w = 0.4;        // fish eye coefficients
      double  s;

      int   blurrad;// blur radius (half size of blur blur kernel
      float   red, gre, blu;                    // red, green, blue
values used in blur of raindrop
      int         num;              // num of pixels in blur kernel

      boolmatrix bmtx;
      bmtx = CreateBoolMatrix(m_height, m_width);

      // set up bool matrix (to test raindrop location)
      //    and make copy of image
      for (i = 0; i < m_height; i++)
            for (j = 0; j < m_width; j++)
            {
                  bmtx[i][j] = false;
                  m_filt[i][j] = m_img[i][j];
            }

      // do raindrops
      for (int numblurs = 0; numblurs < dlg.m_iAmount; numblurs++)
      {
            if (pdlg.CheckCancelButton())
                  if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
                  {
                        canceled = true;
                        break;
                  }

            // determine size
            size = rand() * ((double)(SIZE-5) / RAND_MAX) + 5;
            size2 = size / 2;
            R = size2;
            s = R / log(w*R+1);

            // find random coord to make raindrop
            count = 0;
            do
            {
                  findanother = false;
                  x = int(rand() * ((double)(m_height-1) / RAND_MAX));
                  y = int(rand() * ((double)(m_width-1) / RAND_MAX));
```

```
                if (bmtx[x][y])
                        findanother = true;
                else
                {
                        for (i = x-size2; i <= x+size2; i++)
                                for (j = y-size2; j <= y+size2; j++)
                                {
                                        if (i >= 0 && i < (int)m_height &&
j >= 0 && j < (int)m_width)
                                                if (bmtx[i][j])
                                                        findanother = true;
                                }
                }
                count++;

        } while (findanother && (count < 10000));

        if (count >= 10000)
        {
         numblurs = dlg.m_iAmount;// this will kill the for loop
                pdlg.SetPos(100);
                break;                  // stop us from doing this raindrop
        }


        // do fisheye for square of sides size around that point

        // double for loop around center of raindrop
        for (i = -1 * size2; i < size - size2; i++)
        {
                for (j = -1 * size2; j < size - size2; j++)
                {
                        // convert this square to polar
                        r = sqrt(i*i + j*j);
                a = atan2((float)i, j);// calculates arctan(i/j)

// if we're inside R (i.e. inside the circle, inside the square) do...
                        if (r <= R)
                        {
                                oldr = r;
      // make transformation using Basu and Licardie model [DEVE01]
                                r = (exp(r/s)-1)/w;

                        // convert to cartesian; displace to (y,x)
                                k = x + int(r * sin(a));
                                l = y + int(r * cos(a));

                        // double for loop around raindrop to (y,x)
                                m = x + i;
                                n = y + j;

      if (k >= 0 && k < (int)m_height && l >= 0 && l < (int)m_width)
      if (m >= 0 && m < (int)m_height && n >= 0 && n < (int)m_width)
                                        {
                                                m_filt[m][n] = m_img[k][l];
                                                bmtx[m][n] = true;
```

```
if (dlg.m_bHighlight) // if the user wants highlight/shadow
                                        {
                                                if (oldr >= 0.9 * R)
                                                {
                                    if ((a <= 0) && (a > -2.25))
m_filt[m][n].darken(80);
                            else if ((a <= -2.25) && (a > -2.5))
m_filt[m][n].darken(40);
                            else if (( a <= 0.25) && ( a > 0))
m_filt[m][n].darken(40);
                                                }
                                    else if (oldr >= 0.8 * R)
                                                {
                            if ((a <= -0.75) && (a > -1.50))
m_filt[m][n].darken(40);
                            else if ((a <= 0.10) && (a > -0.75))
m_filt[m][n].darken(30);
                            else if ((a <= -1.50) && (a > -2.35))
m_filt[m][n].darken(30);
                                                }
                                    else if (oldr >= 0.7 * R)
                                                {
                                    if ((a <= -0.10) && (a > -2.0))
m_filt[m][n].darken(20);
                            else if ((a <= 2.50) && (a > 1.90))
m_filt[m][n].lighten(60);
                                                }
                                    else if (oldr >= 0.6 * R)
                                                {
                            if ((a <= -0.50) && (a > -1.75))
m_filt[m][n].darken(20);
                            else if ((a <= 0) && ( a > -0.25))
m_filt[m][n].lighten(20);
                            else if ((a <= -2.0) && (a > -2.25))
m_filt[m][n].lighten(20);
                                                }
                        else if (oldr >= 0.5 * R)
                                                {
                            if ((a <= -0.25) && (a > -0.50))
m_filt[m][n].lighten(30);
                            else if ((a <= -1.75 ) && (a > -2.0))
m_filt[m][n].lighten(30);
                                                }
                                            else if (oldr >= 0.4 * R)
```

```
                                        {
                                    if ((a <= -0.5) && (a > -1.75))
m_filt[m][n].lighten(40);
                                        }
                                    else if (oldr >= 0.3 * R)
                                        {
                                    if ((a <= 0) && (a > -2.25))
m_filt[m][n].lighten(30);
                                        }
                                    else if (oldr >= 0.2 * R)
                                        {
                                if ((a <= -0.5) && (a > -1.75))
m_filt[m][n].lighten(20);
                                        }
                                    }
                                }
                            }
                        }
        if (dlg.m_bHighlight)//don't blur the water unless using
highlight/shadow
{
blurrad = size / 25 + 1;
for (i = -1 * size2 - blurrad; i < size - size2 + blurrad; i++)
{
for (j = -1 * size2 - blurrad; j < size - size2 + blurrad; j++)
                    {
                            // convert this square to polar
                            r = sqrt(i*i + j*j);
                            if (r <= R*1.1)
                            {
                                    red = gre = blu = 0;
                                    num = 0;
                            for (k = -blurrad; k < blurrad + 1; k++)
                    {
                            for (l = -blurrad; l < blurrad + 1; l++)
                                    {
                                            m = x+i+k;
                                            n = y+j+l;
if (m >= 0 && m < (int)m_height && n >= 0 && n < (int)m_width)
                                            {
                                        red += m_filt[m][n].r;
                                        gre += m_filt[m][n].g;
                                        blu += m_filt[m][n].b;
                                                num++;
                                            }
                                    }
                            }
                            m = x+i;
                            n = y+j;
if (m >= 0 && m < (int)m_height && n >= 0 && n < (int)m_width)
m_filt[m][n].color(BYTE(red/num), BYTE(gre/num), BYTE(blu/num));
```

```
                        }
                    }
                }
            }

            pdlg.StepIt();
        }

        FreeBoolMatrix(bmtx, m_height);

        pdlg.DestroyWindow();
        return !canceled;
}

bool CFilter::fisheye()
//
// Desc:   Makes center of image appear as it was pushed out by a
// sphere or pushed in by a cone.  User options from CFishEyeDlg:
// m_bInverse == true, 'pushed in by cone' effect; m_iBackround, 1 ->
// leave background alone, 2 -> make background white, 3 3-> make
// background black; m_iCurvature, amount to push out or in.
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
        // Call Dialog
        CFishEyeDlg dlg;
        if (dlg.DoModal() != IDOK)
                return false;

        // Init Matrices m_img and m_filt
        FilterInit();

        // Setup and Call Progress Dialog
        CProgressDlg pdlg;
        pdlg.Create();
        unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
        unsigned int pixelsProcessed = 0;
        bool canceled = FALSE;

        // Filter Algorithm ////////////////////////////////////

        int                 i, j;           // cartesian coordinates
        int                 x, y;       // converted cartesian coordinates

        double              r, a;// polar coordinates (radius, angle)
        unsigned int R  = (min(m_width, m_height)) / 2; // maxium radius
of polar coordinates

        double w = 0.001 * dlg.m_iCurvature; // curvature [0.001,0.1]

                //   m_iCurvature will return [1,100]
        double      s = R / log(w*R+1);// transformation coefficient
```

```
            //   set according to largest radius

            //   and curvature w

      int w2 = m_width  / 2;
      int h2 = m_height / 2;

      enum background {leave, white, black};
      int  backgroundtype = dlg.m_iBackground;
      BOOL inverse = dlg.m_bInverse;

      // for loops with origin in center of image
      // [i,j] in terms of filtered image

      for (i = -1*h2; i < ((int)m_height-h2); i++)
      {
            if (pdlg.CheckCancelButton())
            if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
                  {
                        canceled = true;
                        break;
                  }
            for (j = -1*w2; j < ((int)m_width-w2); j++)
            {
                  // convert to polar
                  r = sqrt(i*i + j*j);
                  a = atan2((float)i, j); // calculates arctan(i/j)

                  // if we're inside R (i.e. inside the circle) do...
                  if (r <= R)
                  {
      // make transformation using Basu and Licardie model [DEVE01]
                        if (!inverse)
                              r = (exp(r/s)-1)/w;
                        else
                              r = s * log(1+w*r);

                        // convert back to cartesian
                        x = int  (r * cos(a));
                        y = int  (r * sin(a));

                        // move origin back to bottom left
                        x += w2;
                        y += h2;

                        m_filt[i+h2][j+w2].b = m_img[y][x].b;
                        m_filt[i+h2][j+w2].g = m_img[y][x].g;
                        m_filt[i+h2][j+w2].r = m_img[y][x].r;

                  }
                  // if we're outside R, do not curve
                  else
                  {
                        switch (backgroundtype)
```

```
                    {
            case leave: m_filt[i+h2][j+w2].b = m_img[i+h2][j+w2].b;

      m_filt[i+h2][j+w2].g = m_img[i+h2][j+w2].g;

      m_filt[i+h2][j+w2].r = m_img[i+h2][j+w2].r;
                                                        break;
                        case white:       m_filt[i+h2][j+w2].b = 255;

      m_filt[i+h2][j+w2].g = 255;

      m_filt[i+h2][j+w2].r = 255;
                                                        break;
                        default:          m_filt[i+h2][j+w2].b = 0;

      m_filt[i+h2][j+w2].g = 0;

      m_filt[i+h2][j+w2].r = 0;
                        }
                  }

                  pixelsProcessed++;

                  if (pixelsProcessed % pixels == 0)
                        pdlg.StepIt();

            }
      }

      pdlg.DestroyWindow();
      return !canceled;
}

bool CFilter::polar()
//
// Desc:   Pretends the coordinates of the image are polar instead of
// cartesian and plots the polar coordinates as cartesian.  No user
// option dialog.
// Pre:    m_buf != NULL; must call FilterInit() to initialize m_img
// and m_filt
// Post:   true:  m_filt contains the filtered image in matrix form
//         false: m_filt is undefined (user pressed cancel in a dialog)
//
{
      // Init Matrices m_img and m_filt
      FilterInit();

      // Setup and Call Progress Dialog
      CProgressDlg pdlg;
      pdlg.Create();
      unsigned int pixels = (unsigned int)(m_height * m_width * 0.01);
// 1% of total pixels
      unsigned int pixelsProcessed = 0;
      bool canceled = FALSE;

      // Filter Algorithm ////////////////////////////////////
```

```
	int		i, j;
		// cartesian coordinates
	int		x, y;
		// converted cartesian coordinates
	int		k, l, m;

	double		r, a;
		// polar coordinates (radius, angle)
	unsigned int R  = (min(m_width, m_height))/2;	// maxium radius
of polar coordinates

	int w2 = m_width  / 2;
	int h2 = m_height / 2;

	for (k = 0; k < (int)m_height; k++)
		for (l = 0; l < (int)m_width; l++)
			m_filt[k][l].color(255);

	for (i = 0; i < (int)m_height; i++)
	{
		if (pdlg.CheckCancelButton())
			if(AfxMessageBox("Are you sure you want to Cancel?",
MB_YESNO)==IDYES)
			{
				canceled = true;
				break;
			}

		for (j = 0; j < (int)m_width; j++)
		{
			// move x, y so centerpixel is (0,0) -- for polar
conversion
			x = i - h2;
			y = j - w2;

			// convert [x,y] to polar
			r = sqrt(x*x + y*y);
			a = atan2((float)x, y);

			// scaling
			x = int(r*m_height/R);
			y = int(a*m_width/6.2832);

			//move origin to top center (visually)
			x = m_height - x - 1;
			y += w2;

			k = i;
			l = j;

			//rotate filtered image 90-degrees clockwise
			m = l;
			l = k;
			k = m_height - m;

			//displace back to center origin
			k += (w2 - h2);
```

```
                l += (w2 - h2);

                // plot the pixel, if it exists
        if (l >= 0 && l < (int)m_width && k >= 0 && k < (int)m_height)
        if (x >= 0 && x < (int)m_height && y >= 0 && y < (int)m_width)
        m_filt[k][l].color(m_img[x][y].r, m_img[x][y].g, m_img[x][y].b);

                pixelsProcessed++;
                if (pixelsProcessed % pixels == 0)
                    pdlg.StepIt();
            }
        }
        pdlg.DestroyWindow();
        return !canceled;
}
```

Rujukan :

1. Rafael C. Gonzalez & Richard E. Woods, Digital Image Processing, Addison-Wesley Publishing Company, 1993
2. Fu.K.S & Mui J.K., "A Survey of Image Segmentation" Pattern Recognition, vol. 13, 1981
3. A.K. Jain, Fundamentals of Image Processing, Prentice-Hall Inc, Eaglewood Cliffs 1989
4. Pfleeger, Shari Lawrence; Software Engineering; Theory And Practice, Prentice-Hall International, Inc; Washington; 1998. Pg 48-58
5. Hawryszkiewycz, Igor; Introduction to Systems Analysis and Design, 2nd ed, Prentice-Hall Australia Pty Ltd; Australia; 1998. Pg 120-131.
6. MATLAB Application Program Interface Guide, The Mathwork Inc, 1998.
7. Computer Vision & Image Processing a practical approach using CVIP tools, Scott E. Umbaugh, Prentice-Hall, Inc 1998
8. Introductory remote sensing : digital image processing & applications, Paul J. Gibson & Clare H. Power, 2000, St. Edmundsbury Press, Bury St. Edmundssuffolk
9. Angel, E. *Interactive Computer Graphics: A Top-Down Approach with OpenGL*. Reading, MA: Addison-Wesley, 2000.
10. Davies, A., and P. Fennessy. *Digital Imaging for Photographers*. Boston: Focal Press, 1998.
11. Foley, J., A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Reading, MA: Addison-Wesley, 1990.

12. Gonzalez, R. C., and P. Wintz. *Digital Image Processing*. Reading, MA: Addison-Wesley, 1977.

13. Hall, E. L. *Computer Image Processing and Recognition*. New York: Academic Press, 1979.

14. Hill, F. S. *Computer Graphics*. New York: Macmillan, 1990.

15. Holzmann, G. J. *Beyond Photography: The Digital Darkroom*. Englewood Cliffs, NJ: Prentice Hall, 1988.

16. Hough, T., ed. *The Joy of Photography*. Reading, MA: Addison-Wesley, 1991.

17. Kruglinski, D. J., G. Shepherd, and S. Wingo. *Programming Microsoft Visual C++ Fifth Edition*. Redmond, WA: Microsoft Press, 1998.

18. Lindley, C. A. *Practical Image Processing in C*. New York: John Wiley & Sons, Inc., 1991.

19. Lyon, D. A. *Image Processing in Java*. Upper Saddle River, NJ: Prentice Hall PTR, 1999.

20. Martinez, B. and J. Block. *Visual Forces: An Introduction to Design*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1995.

21. Parsons, T. W. *Introduction to Algorithms in Pascal*. New York: John Wiley & Sons, Inc., 1995.

22. Pitas, I. *Digital Image Processing Algorithms and Applications*. New York: John Wiley & Sons, Inc., 2000.

23. Seul, M., L. O'Gorman, and M. Sammon. *Practical Algorithms for Image Analysis: Description, Examples, and Code*. Cambridge: University Press, 2000.

24. Sphar, C. *Learn Microsoft Visual C++ 6.0 Now*. Redmond, WA: Microsoft Press, 1999.

25. Teuber, J. *Digital Image Processing*. New York: Prentice Hall, 1993.

26. http://www.mathworks.com

27. http://hwr.nici.kun.nl

28. http://peipa.essex.ac.uk

29. http://www.cs.cmn.edu/afs/cs/project/cil/ftp/html/vision.html

30. http://www.sci.lib.uci.edu/HSG/MedicalImage.html

31. http://www.rz.go.dlr.de:8081/softarch.html

32. http://www.eecs.wsu.edu/Ipdb/title.html

33. http://www-isis.ecs.soton.ac.uk/research/visinfo/rgroup.html

34. http://george.lbl.gov/computer_vision.html

35. http://george.lbl.gov/ITG.html

36. http://www.video.eecs.berkeley.edu/

37. http://www.cg.tuwien.ac.at/studentwork/CESCG97/boros/

38. http://www.ping.be/~ping1339/polar.htm

39. http://www-sop.inria.fr/chir/personnel/devernay/publis/distcalib/

40. http://www.pcigeomatics.com/cgi-bin/pcihlp/IHS

41. http://www.pcigeomatics.com/cgi-bin/pcihlp/RGB

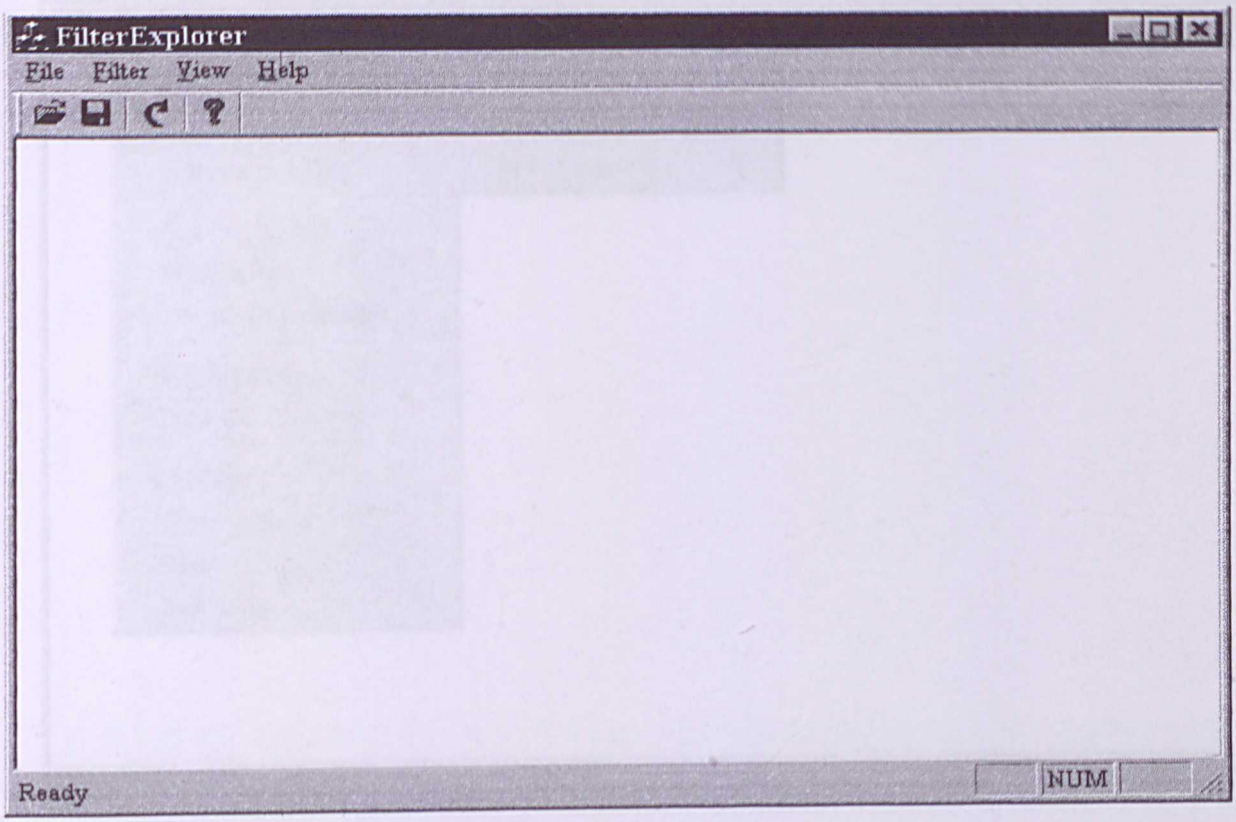
42. http://www.persci.com/~schulze/java/

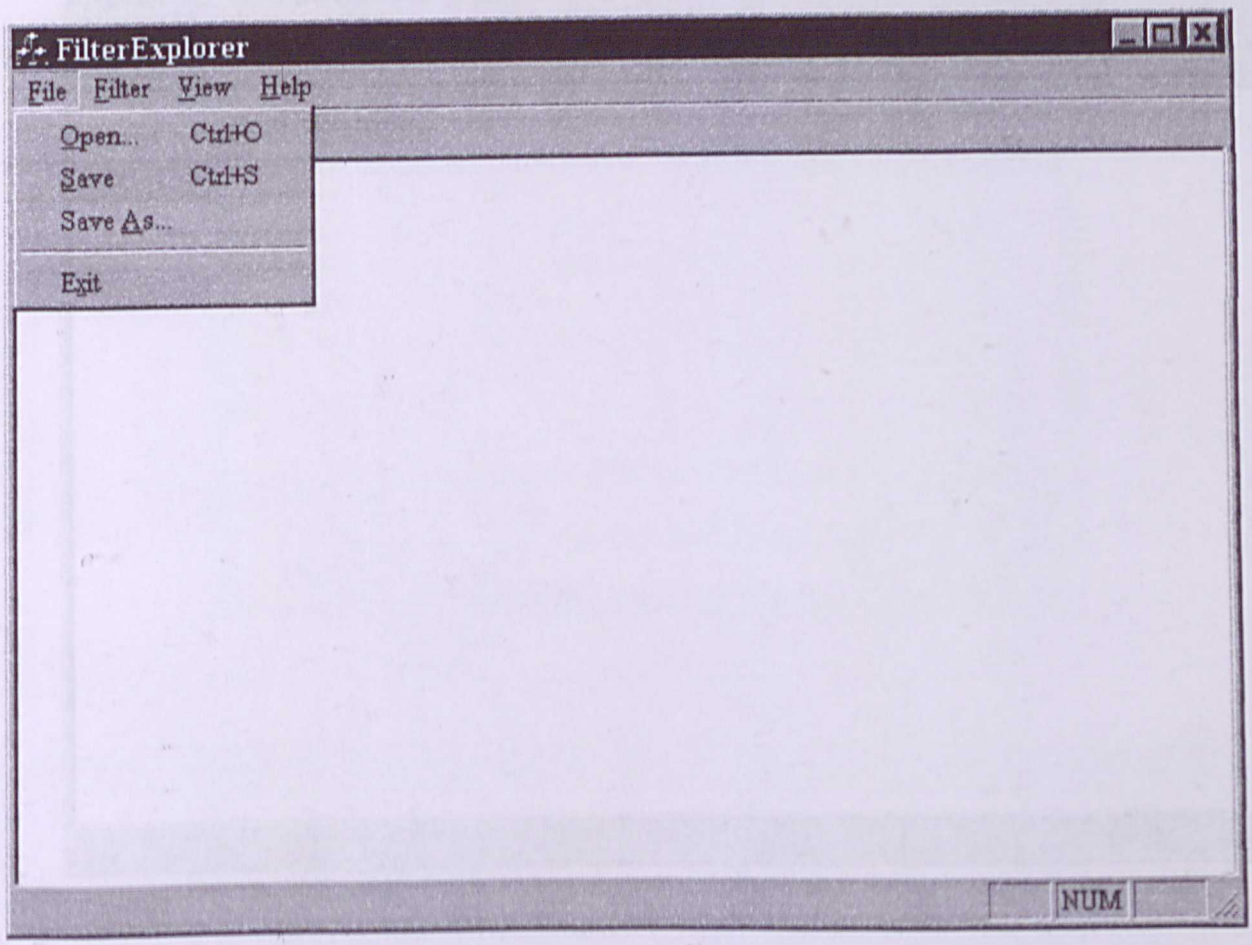
43. http://www.smalleranimals.com/jpegfile.htm

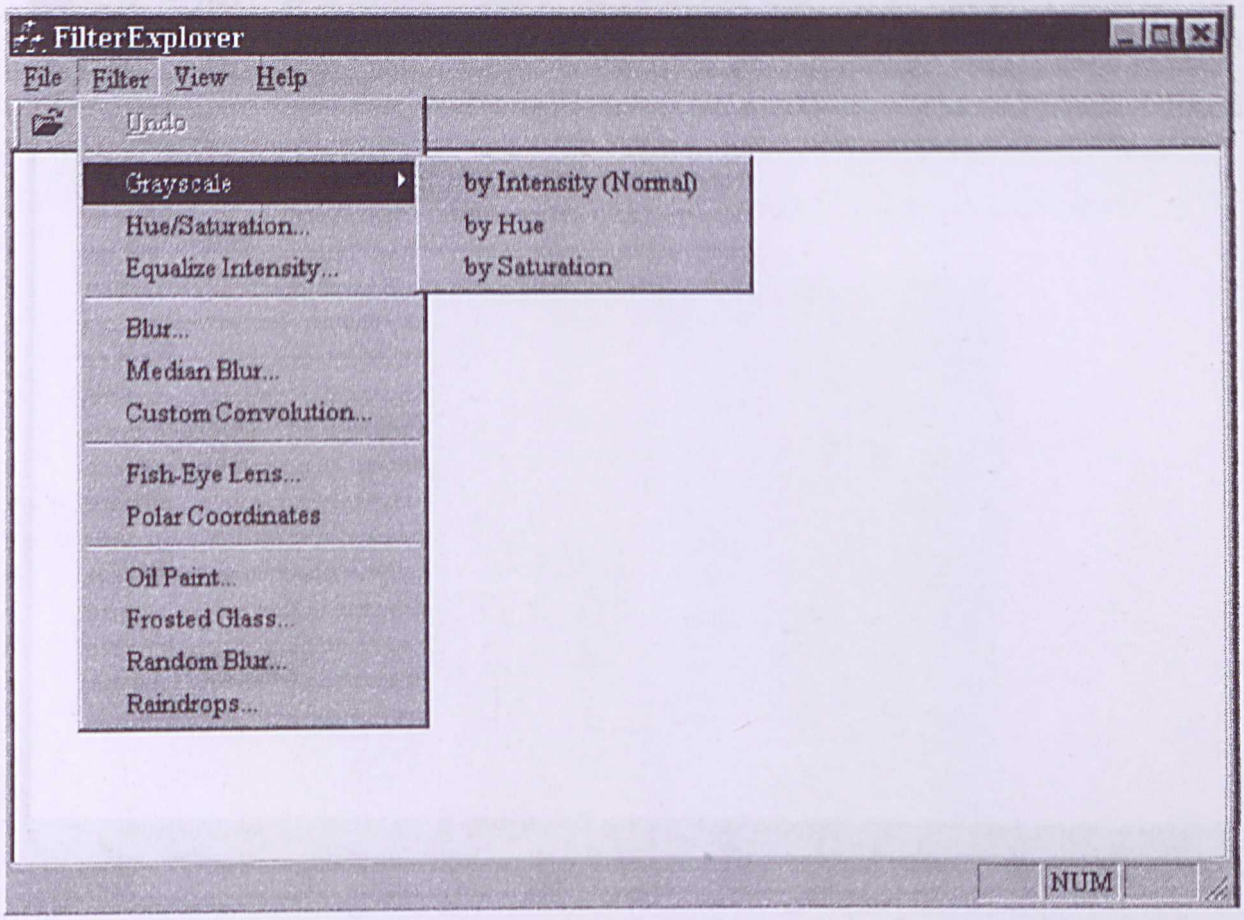
44. http://www.ctr.columbia.edu/~jrsmith/html/pubs/tatfcir/node8.html

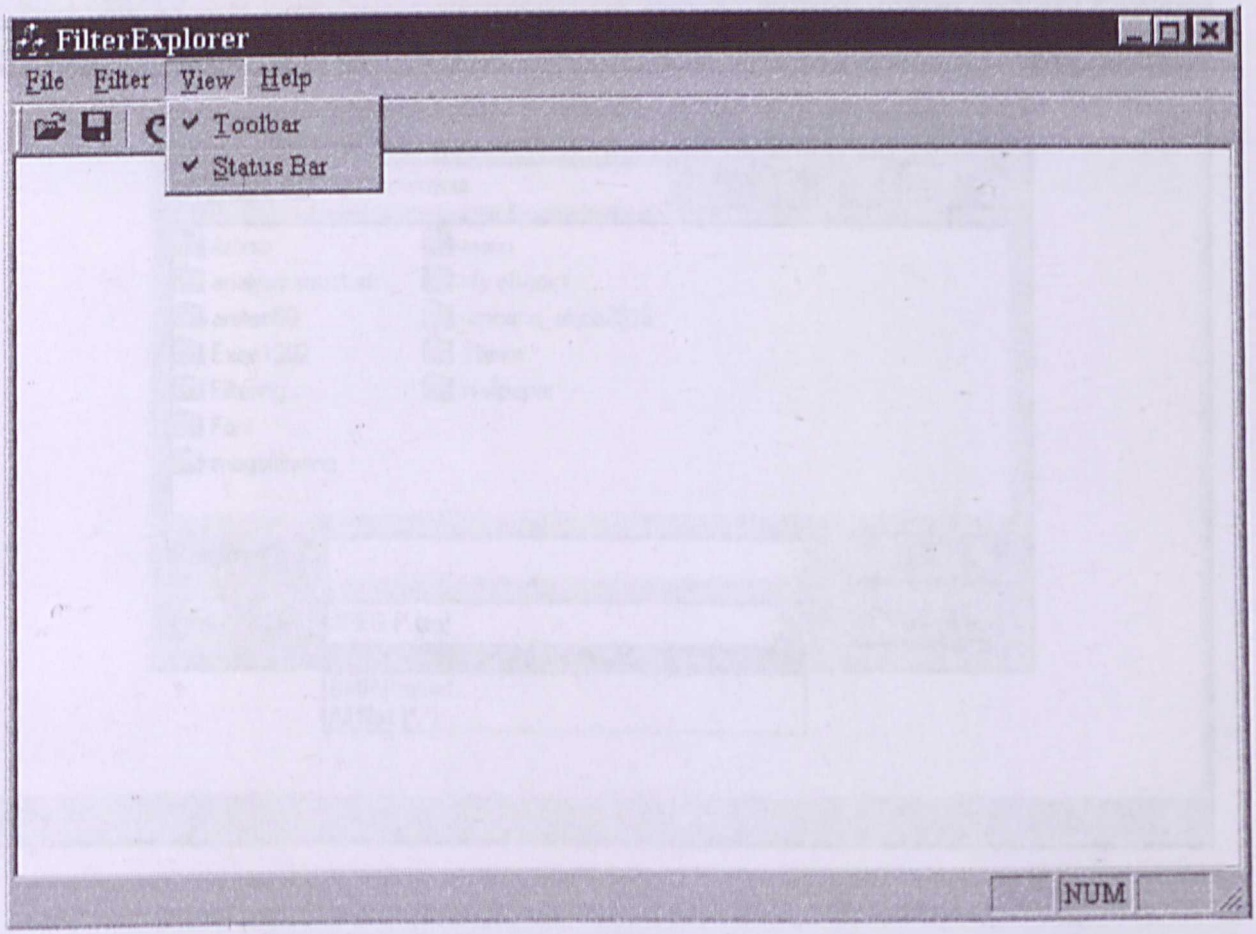
45. http://pico.i-us.com/prog/ffpg.html

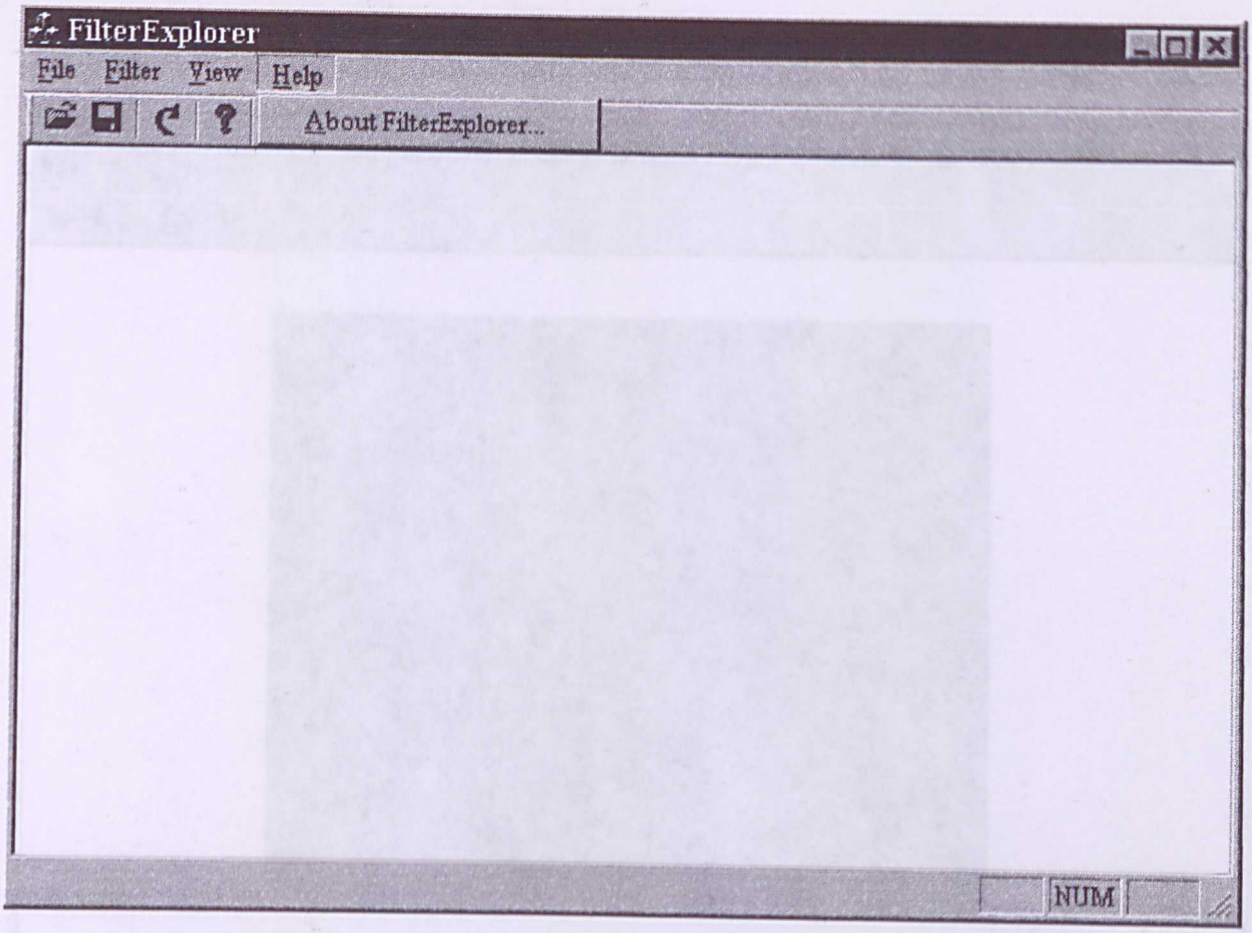
LAMPIRAN

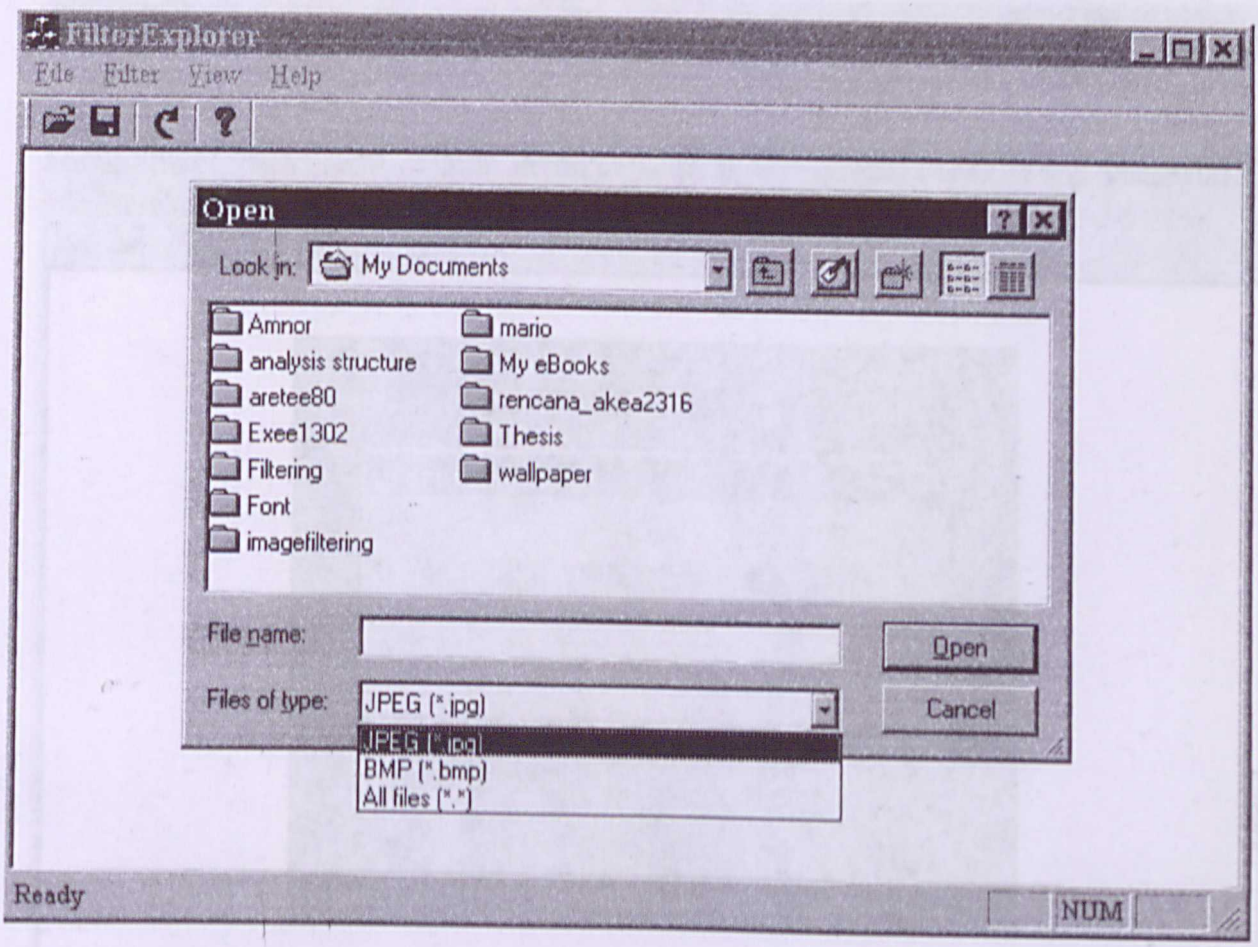

FilterExplorer
File Filter View Help
Ready
NUM

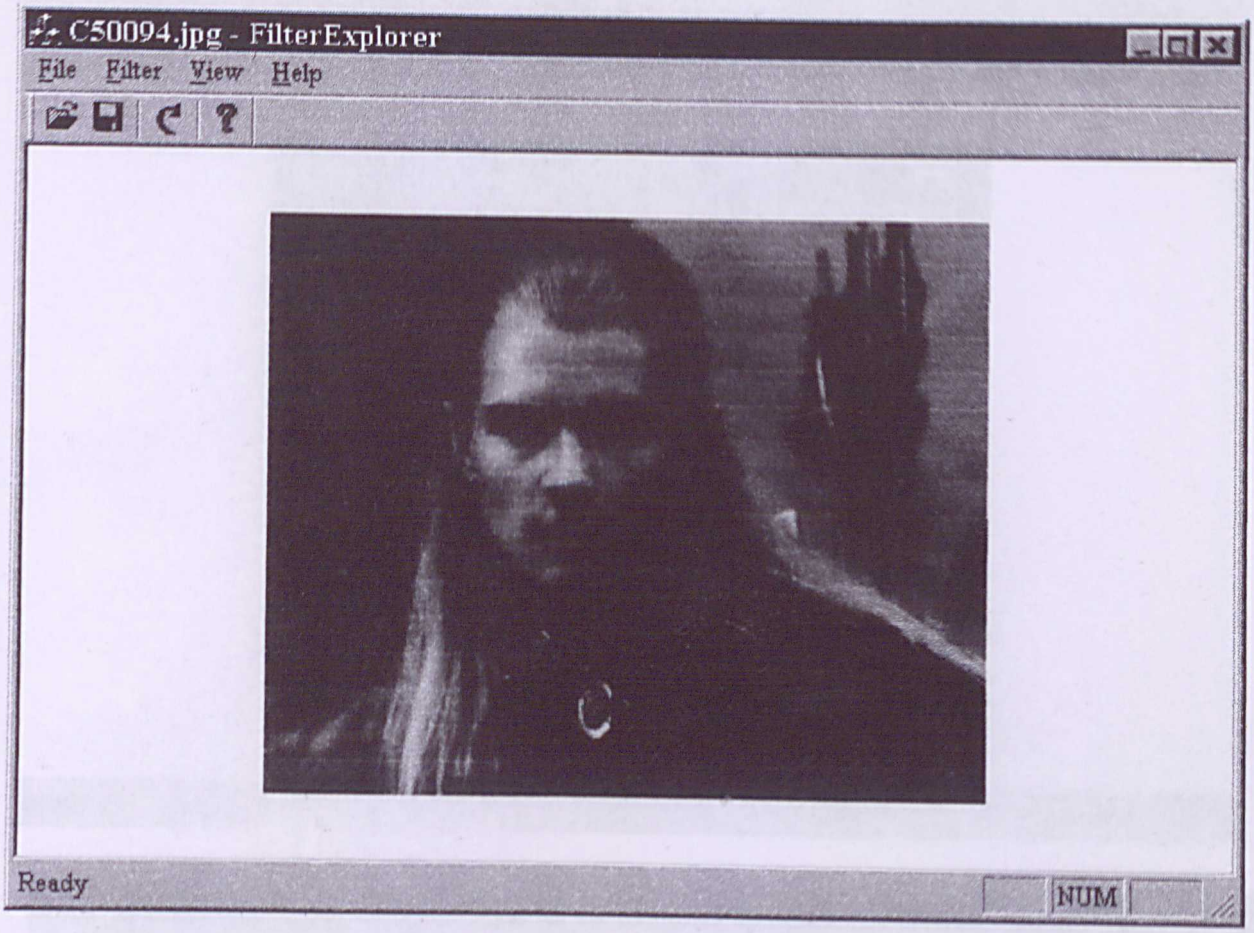

FilterExplorer
File Filter View Help
Open... Ctrl+O
Save Ctrl+S
Save As...
Exit
NUM

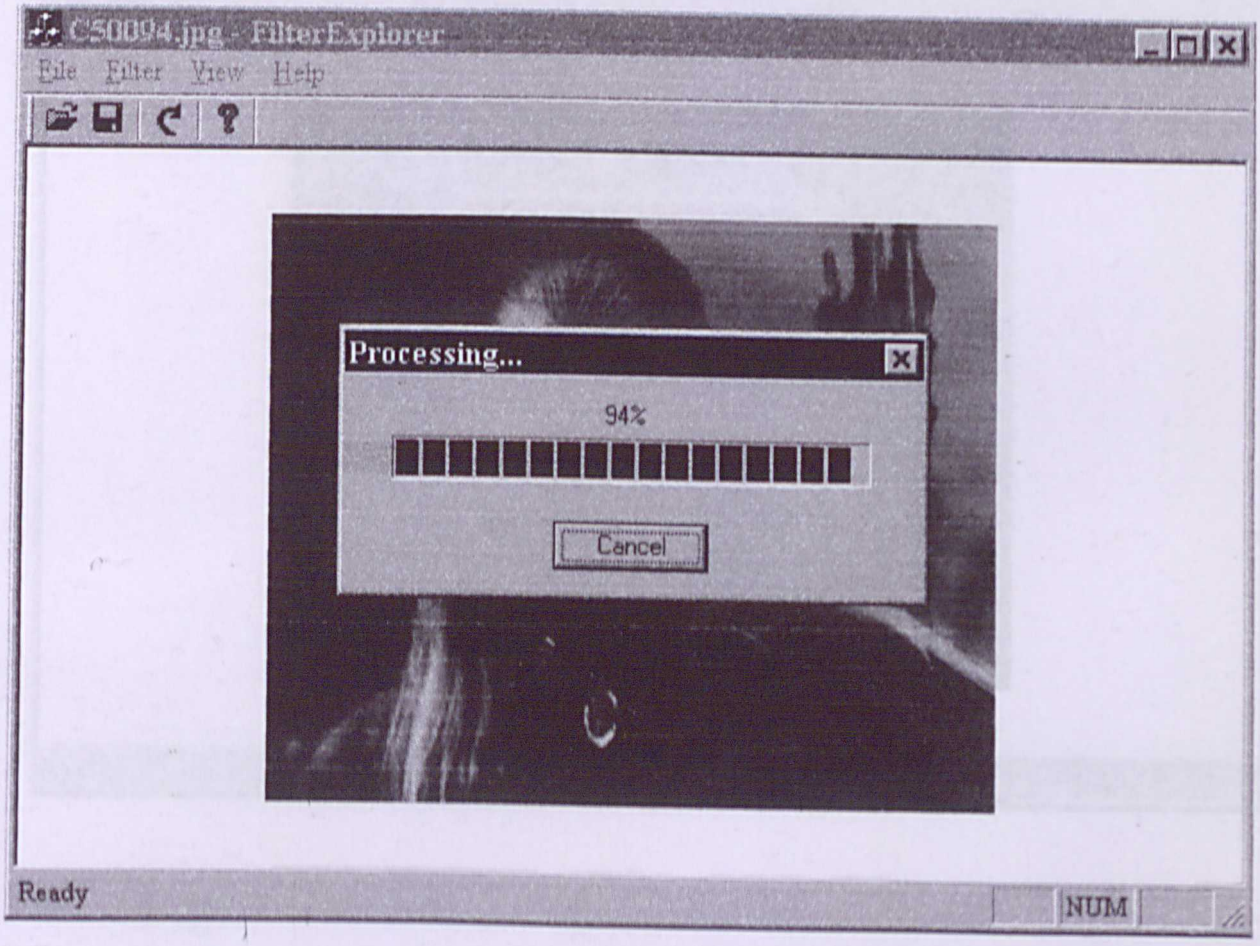

FilterExplorer
File
Filter
View
Help
Undo
Grayscale
Hue/Saturation...
Equalize Intensity...
Blur...
Median Blur...
Custom Convolution...
Fish-Eye Lens...
Polar Coordinates
Oil Paint...
Frosted Glass...
Random Blur...
Raindrops...
by Intensity (Normal)
by Hue
by Saturation
NUM

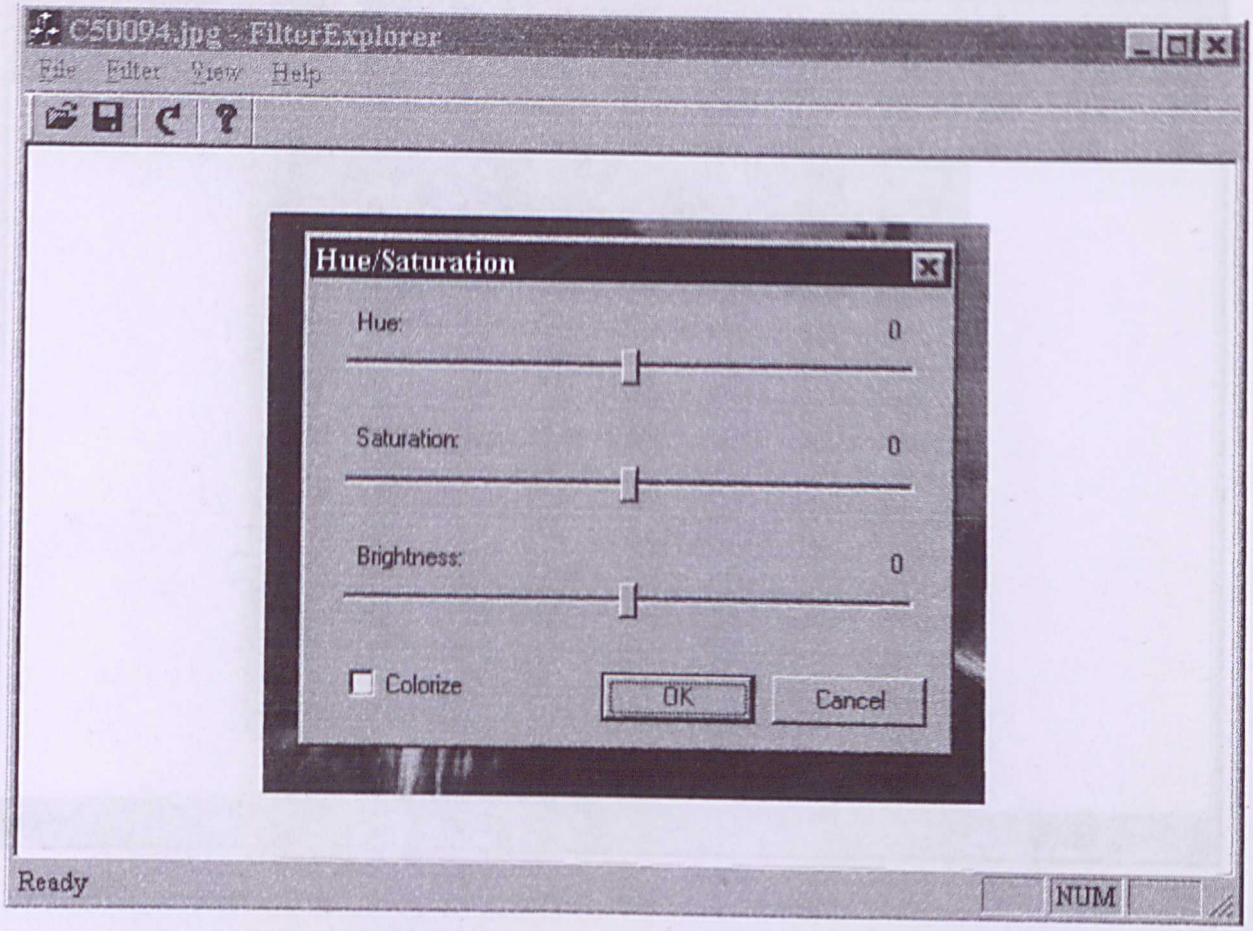

FilterExplorer
File
Filter
View
Help
Toolbar
Status Bar
NUM

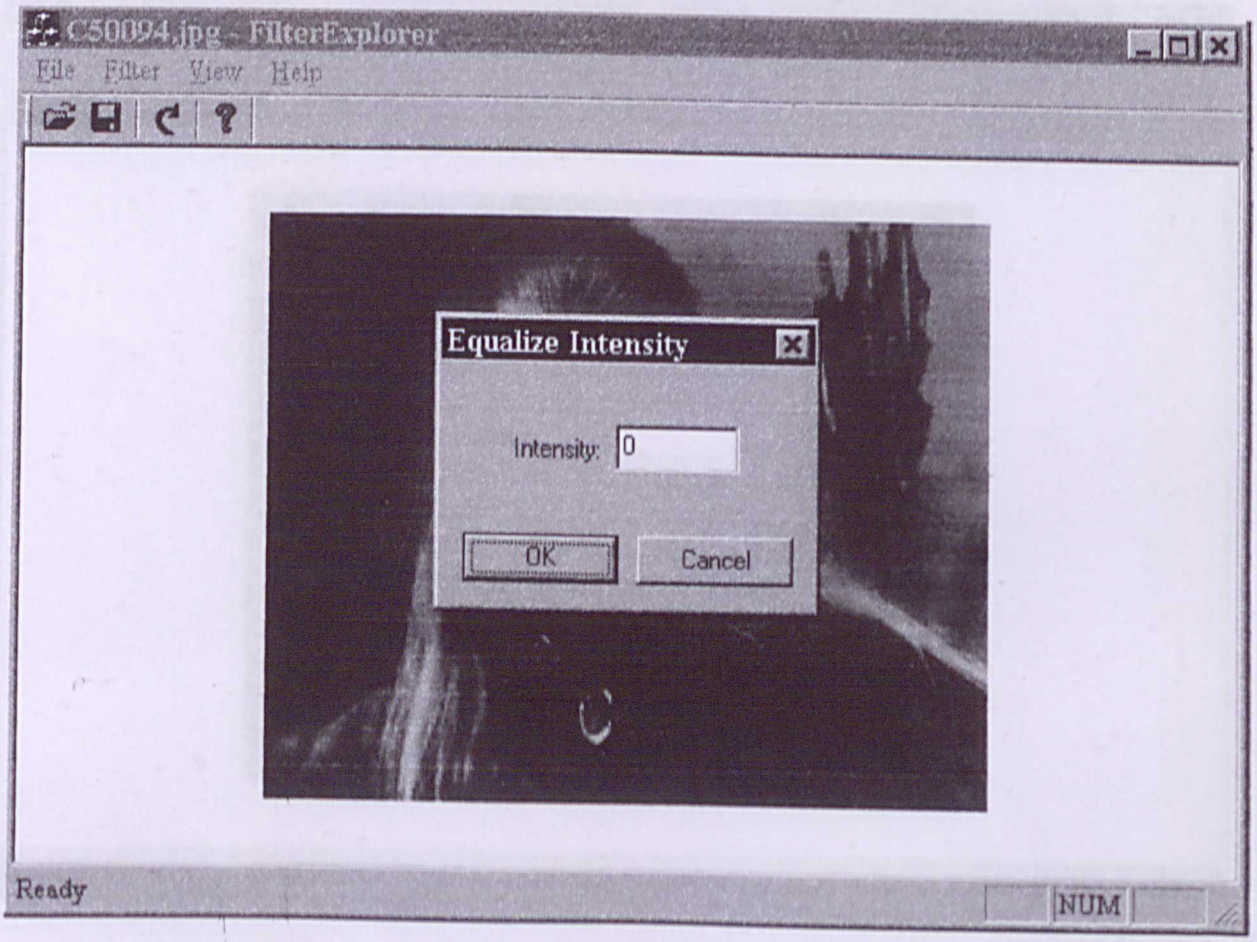

FilterExplorer
File Filter View Help
About FilterExplorer...
NUM

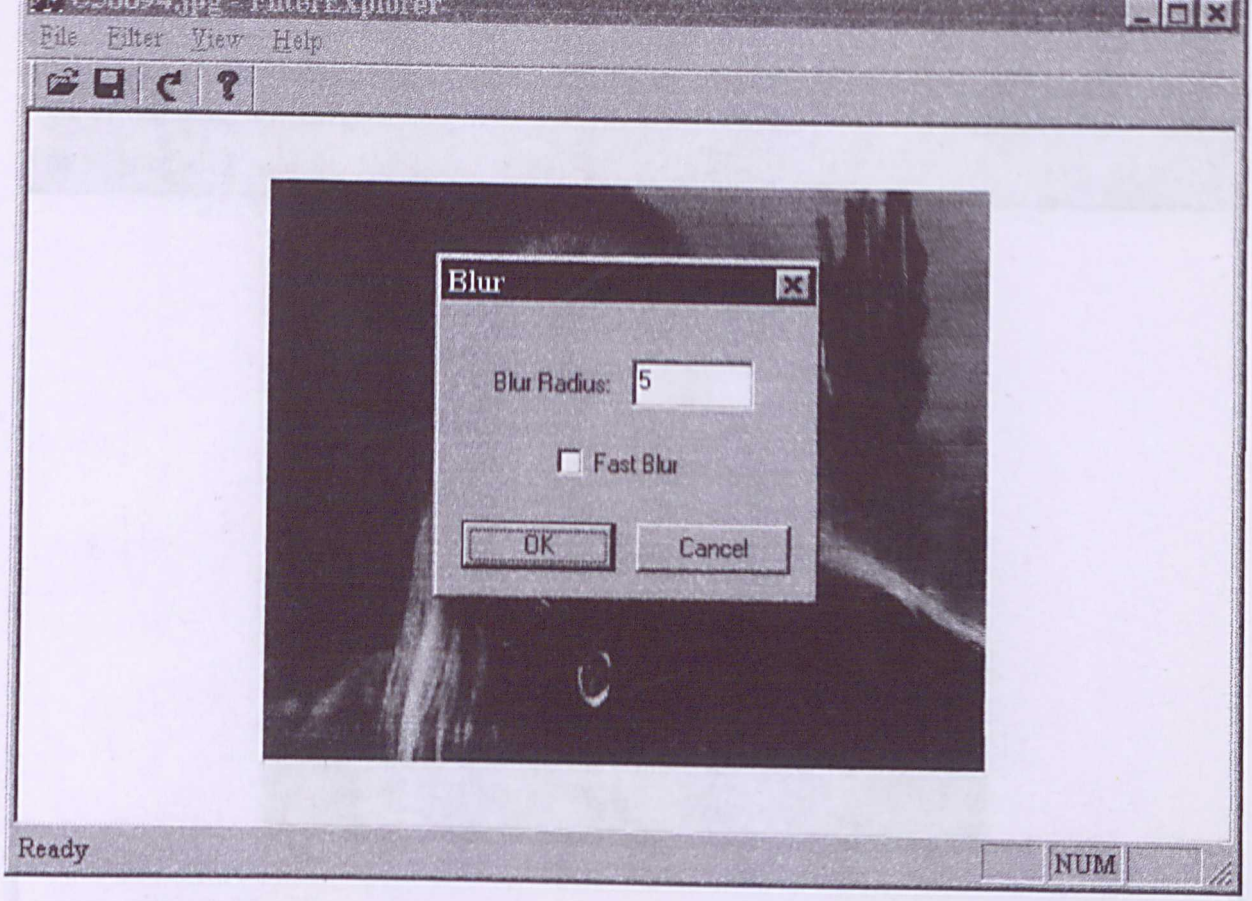

FilterExplorer
File Filter View Help
Open
Look in: My Documents
Amnor
analysis structure
aretee80
Exee1302
Filtering
Font
imagefiltering
mario
My eBooks
rencana_akea2316
Thesis
wallpaper
File name:
Open
Files of type: JPEG (*.jpg)
Cancel
JPEG (*.jpg)
BMP (*.bmp)
All files (*.*)
Ready
NUM

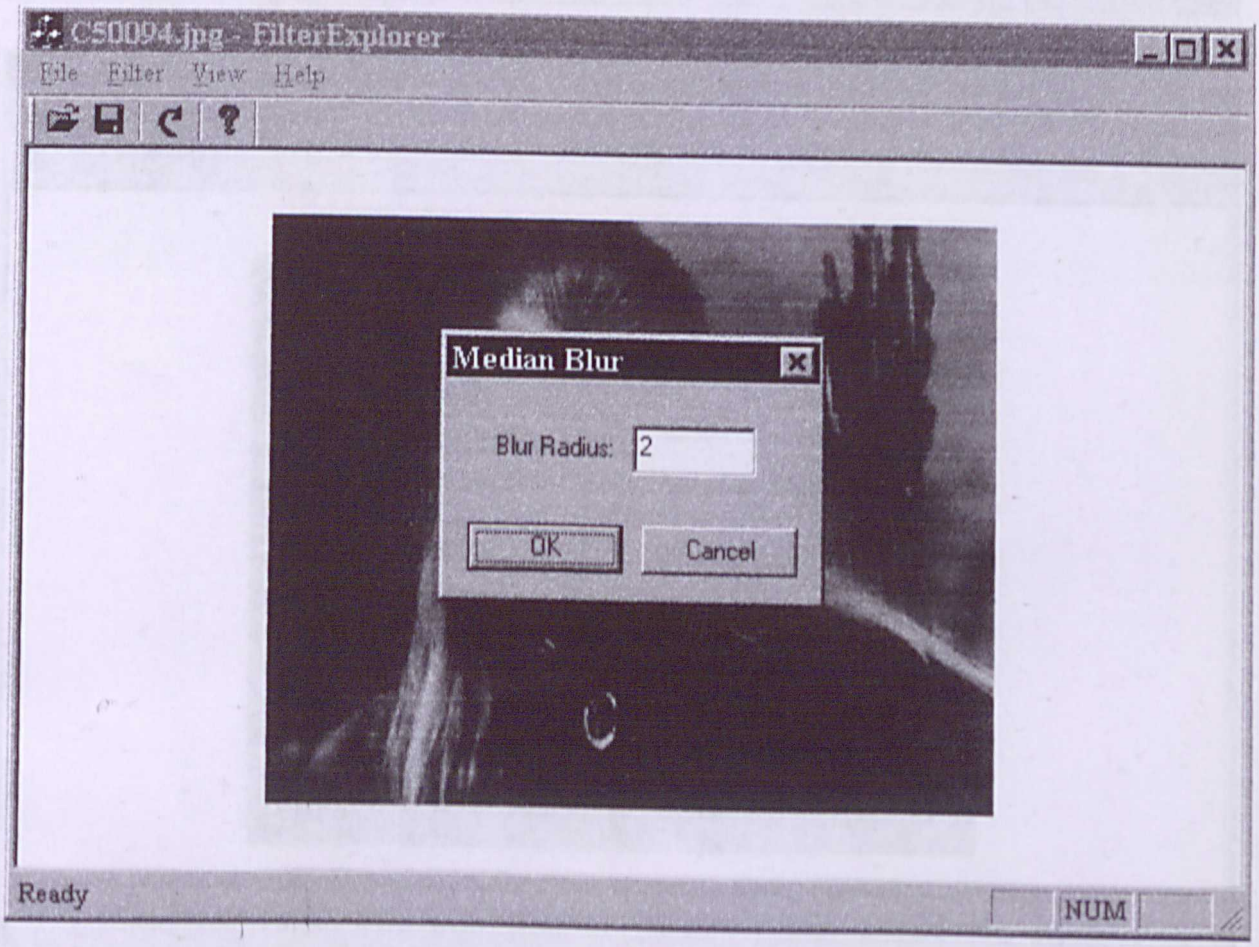

C50094.jpg - FilterExplorer
File Filter View Help
Ready
NUM

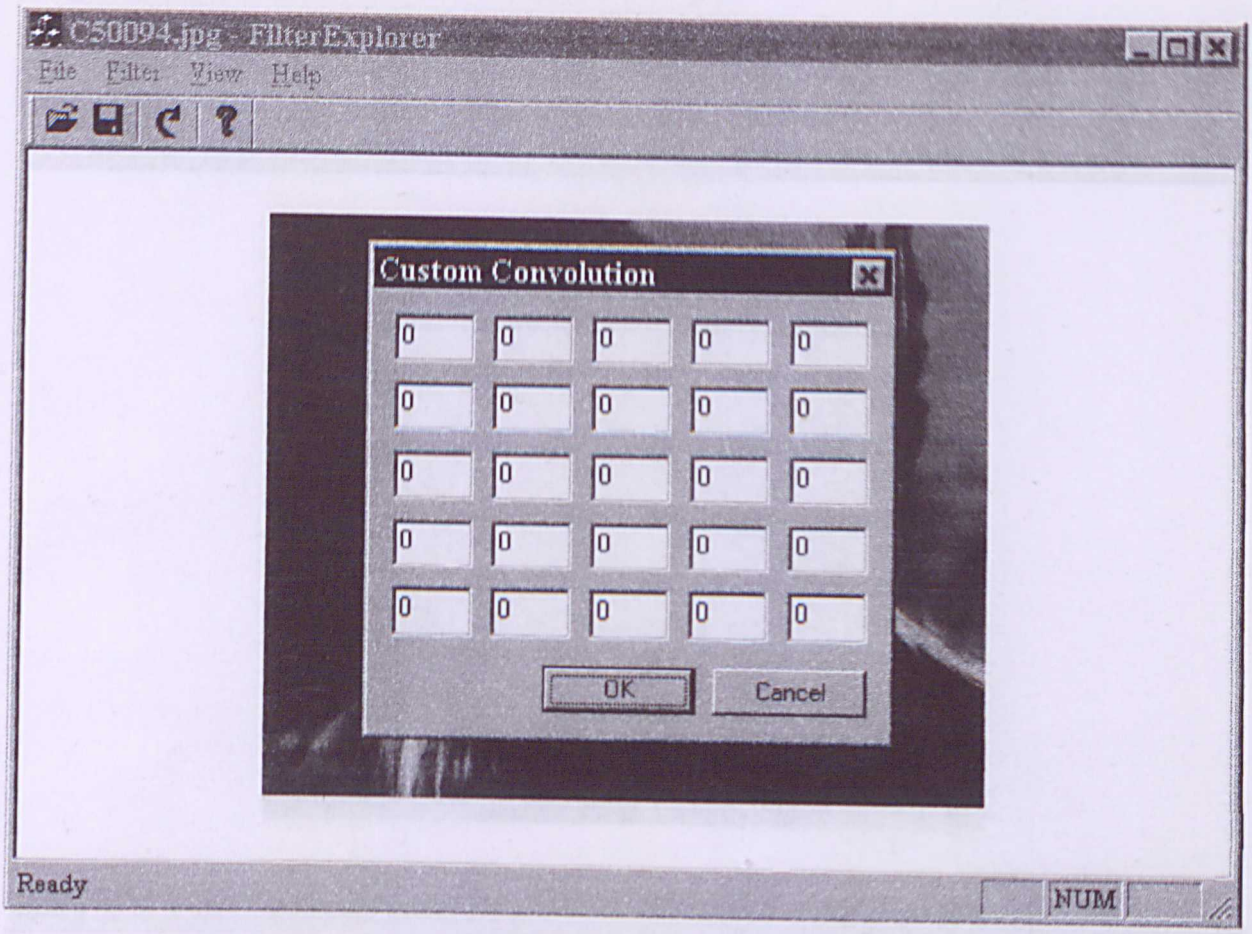

C50094.jpg - FilterExplorer
File Filter View Help
Processing...
94%
Cancel
Ready
NUM

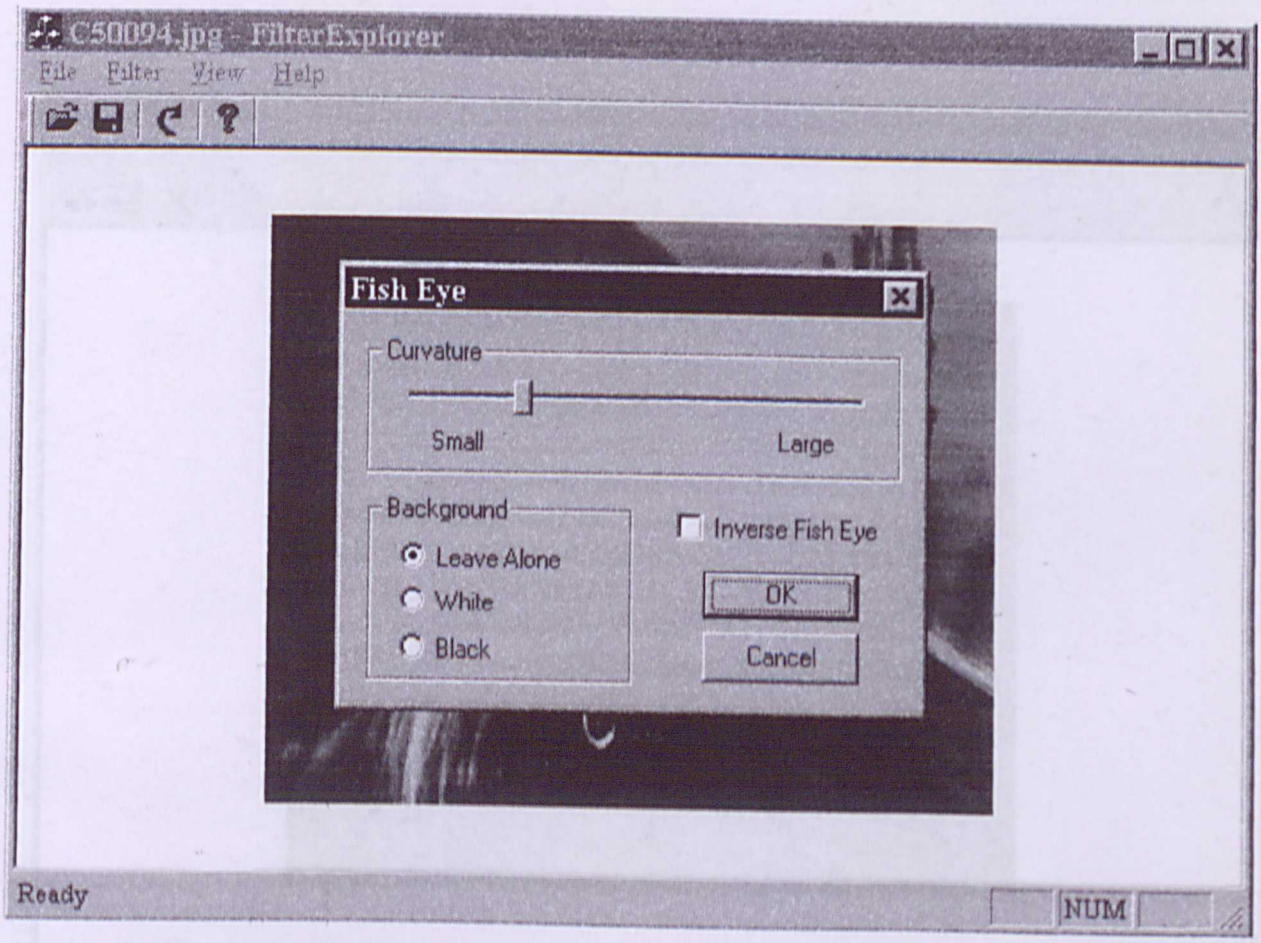

C50094.jpg - FilterExplorer
File Filter View Help
Hue/Saturation
Hue:
0
Saturation:
0
Brightness:
0
Colorize
OK
Cancel
Ready
NUM

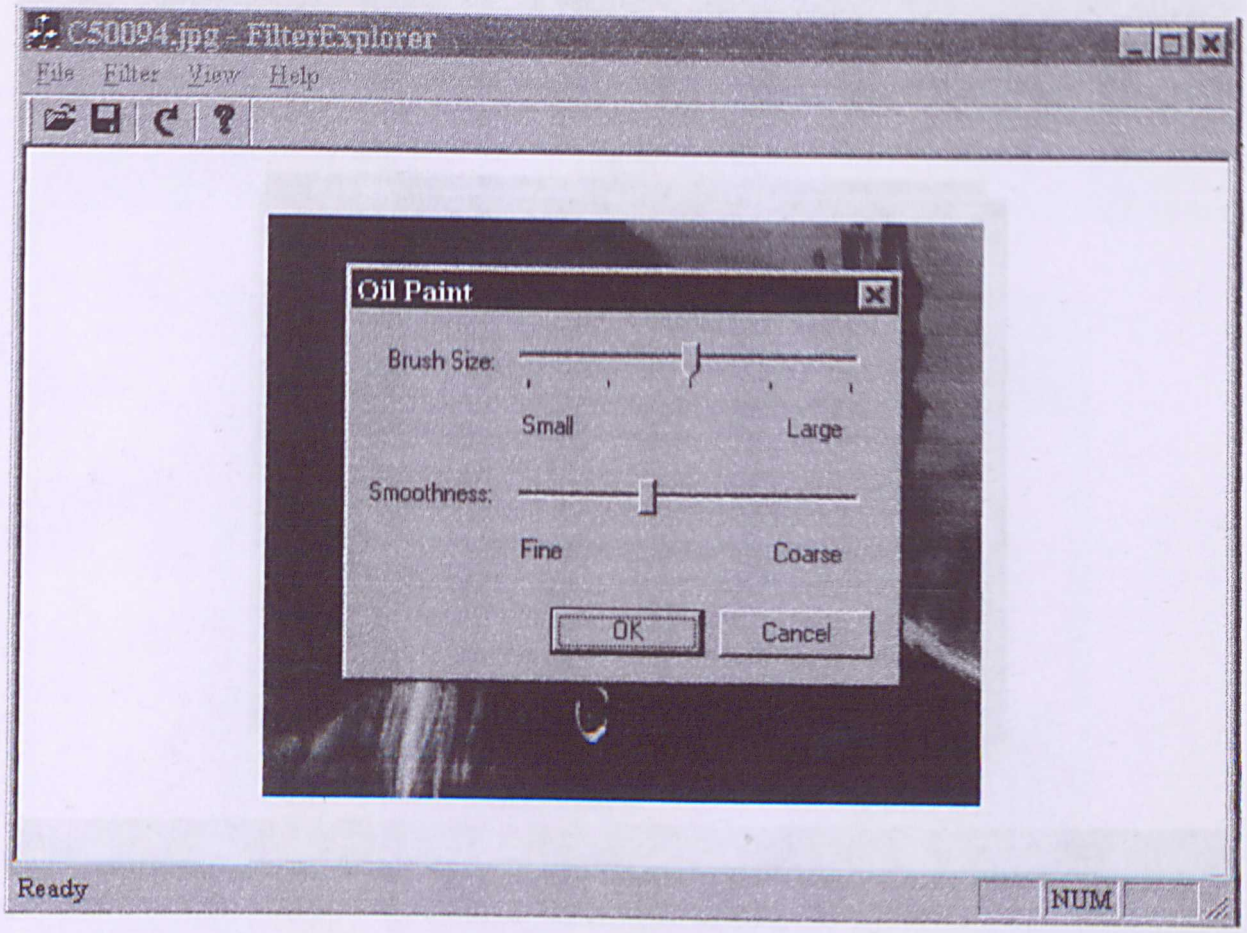

C50094.jpg - FilterExplorer
File Filter View Help
Equalize Intensity
Intensity:
0
OK
Cancel
Ready
NUM

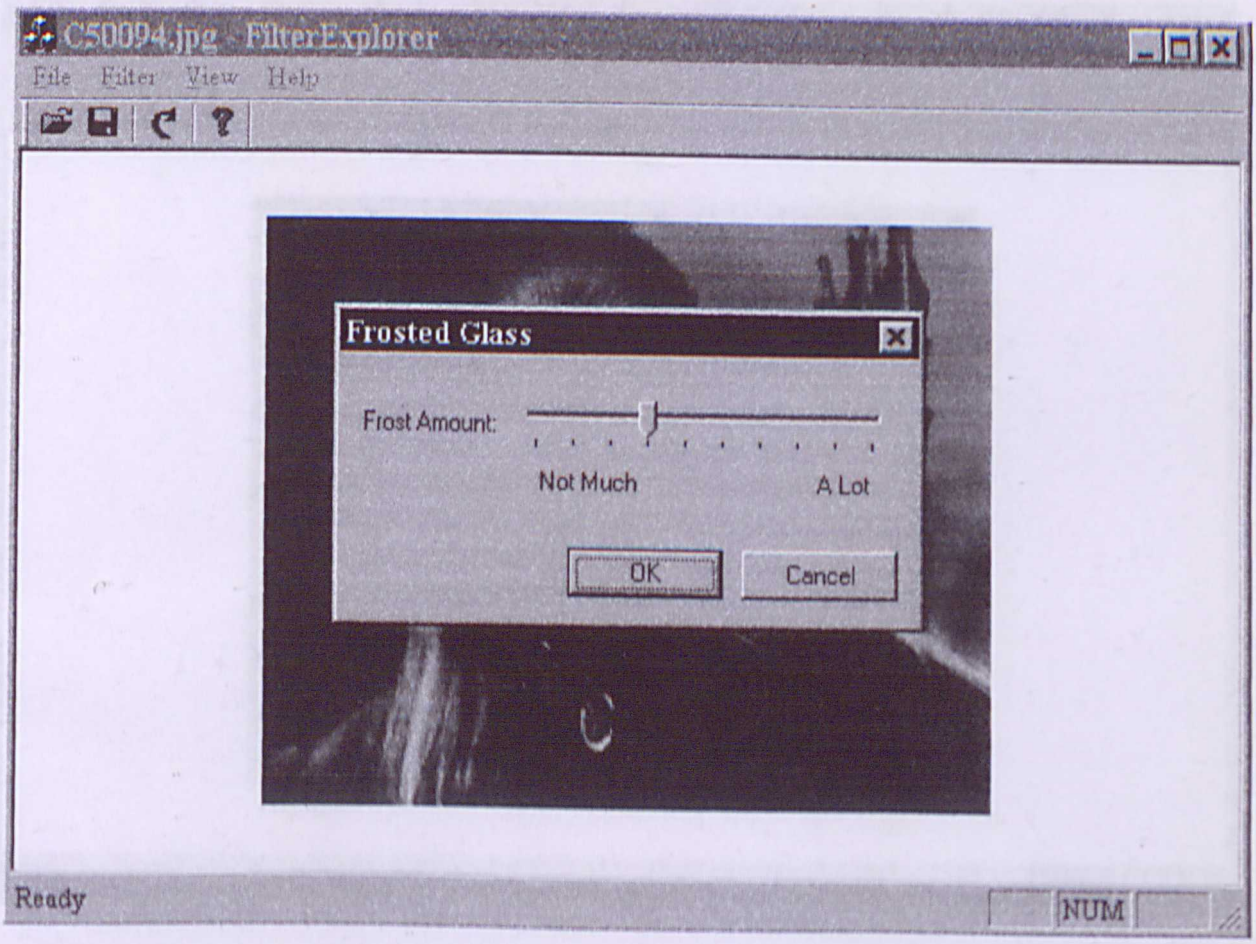

File Filter View Help
Blur
Blur Radius: 5
Fast Blur
OK
Cancel
Ready
NUM

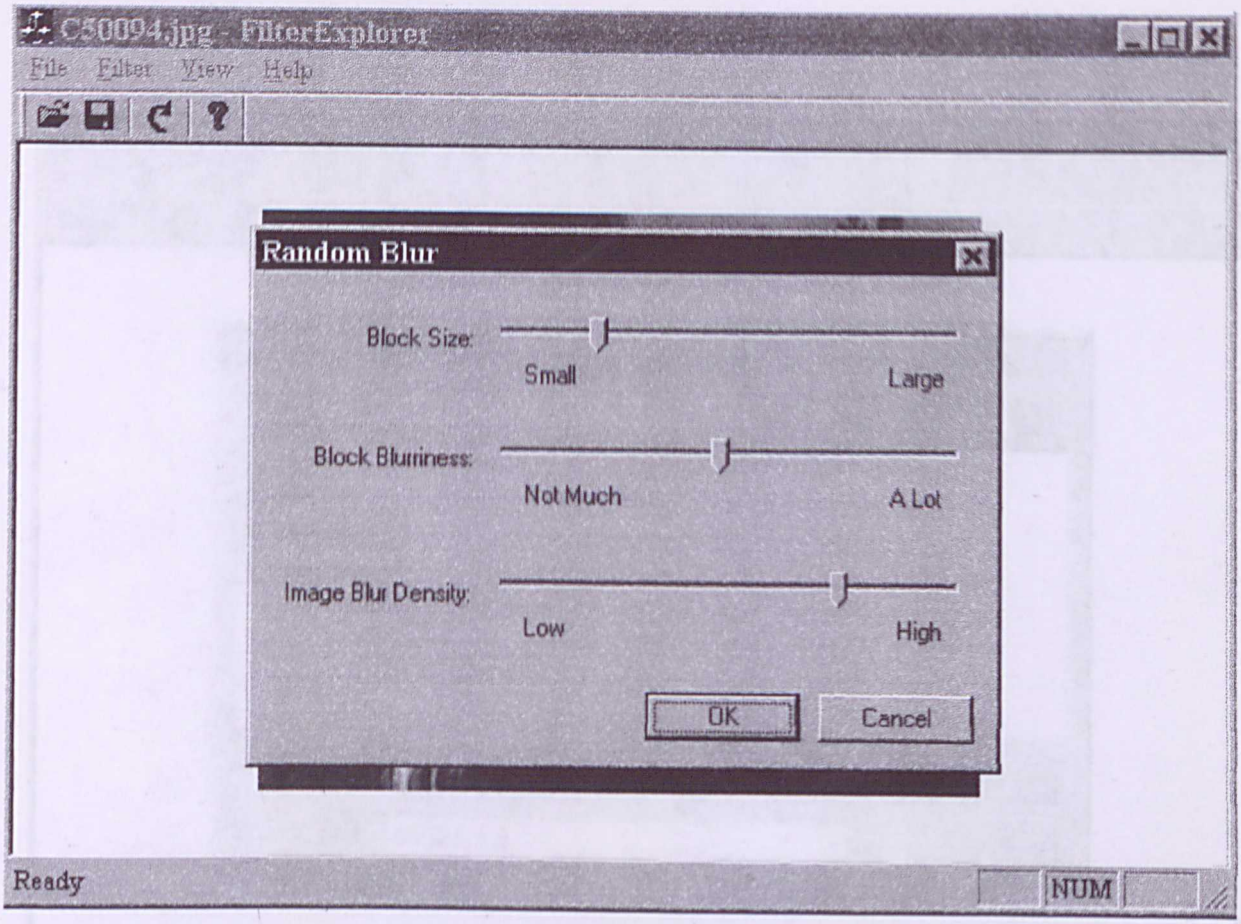

C50094.jpg - FilterExplorer
File Filter View Help
Median Blur
Blur Radius: 2
OK
Cancel
Ready
NUM

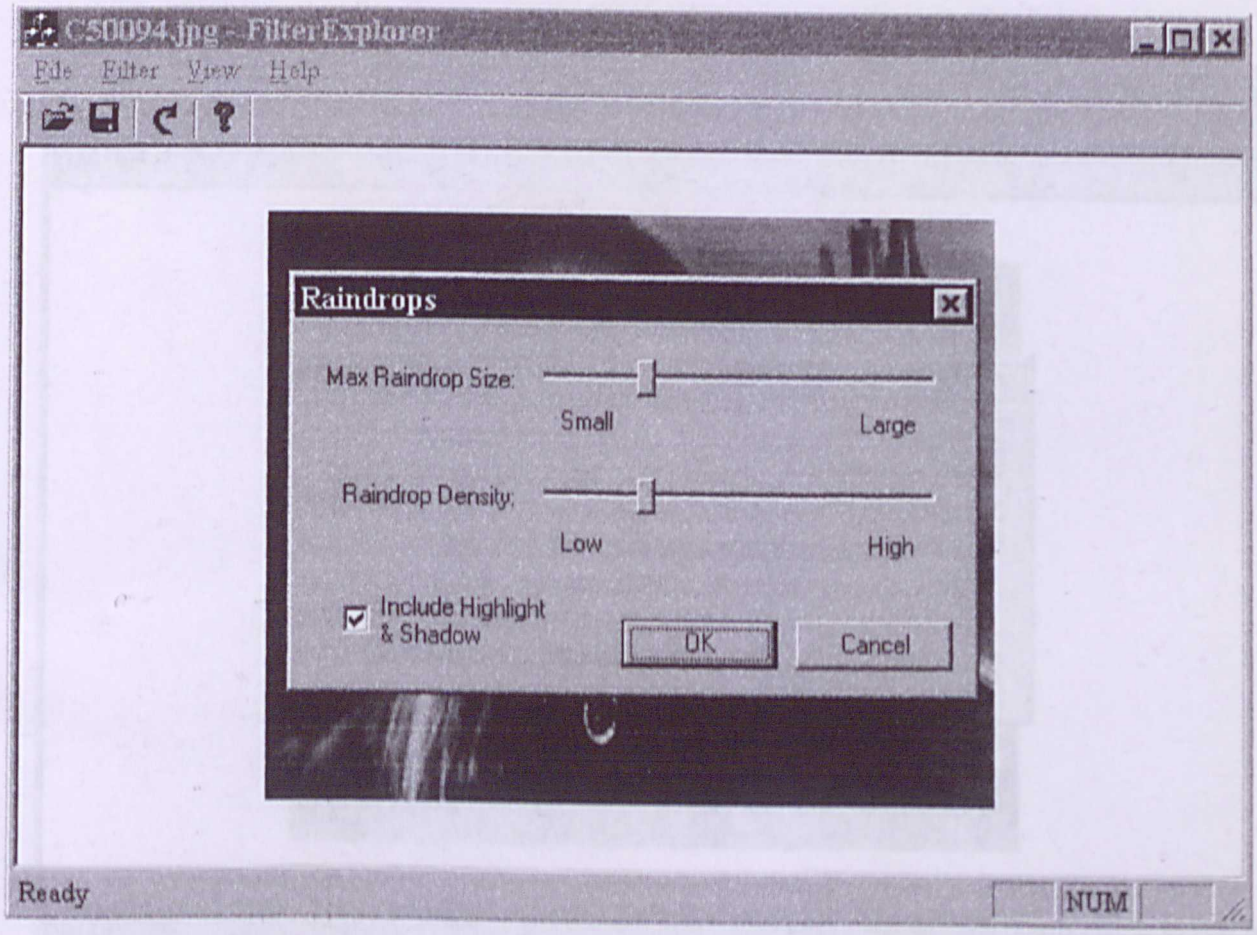

C50094.jpg - FilterExplorer
File Filter View Help
Custom Convolution
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
OK
Cancel
Ready
NUM

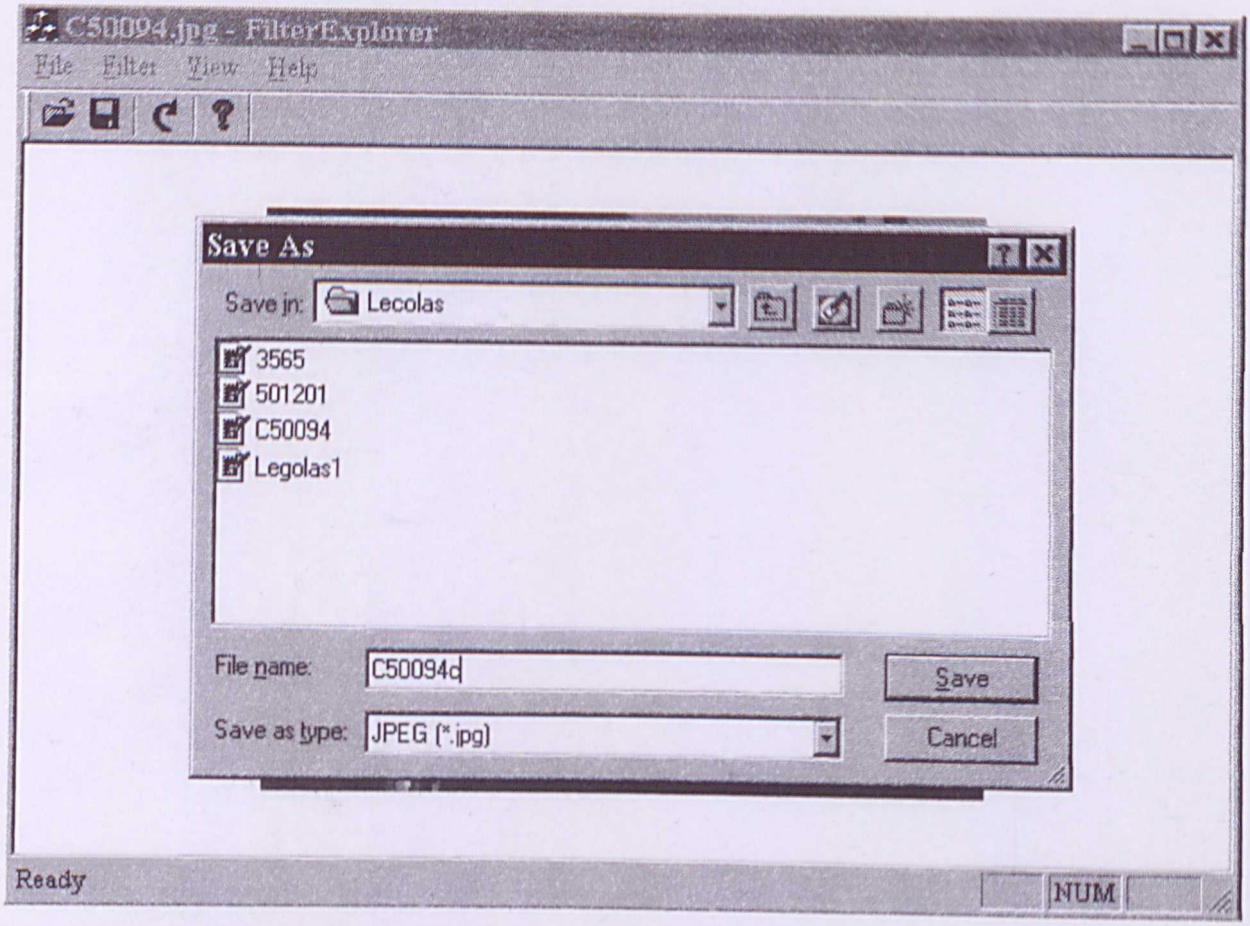

C50094.jpg - FilterExplorer
File Filter View Help
Fish Eye
Curvature
Small
Large
Background
Leave Alone
White
Black
Inverse Fish Eye
OK
Cancel
Ready
NUM

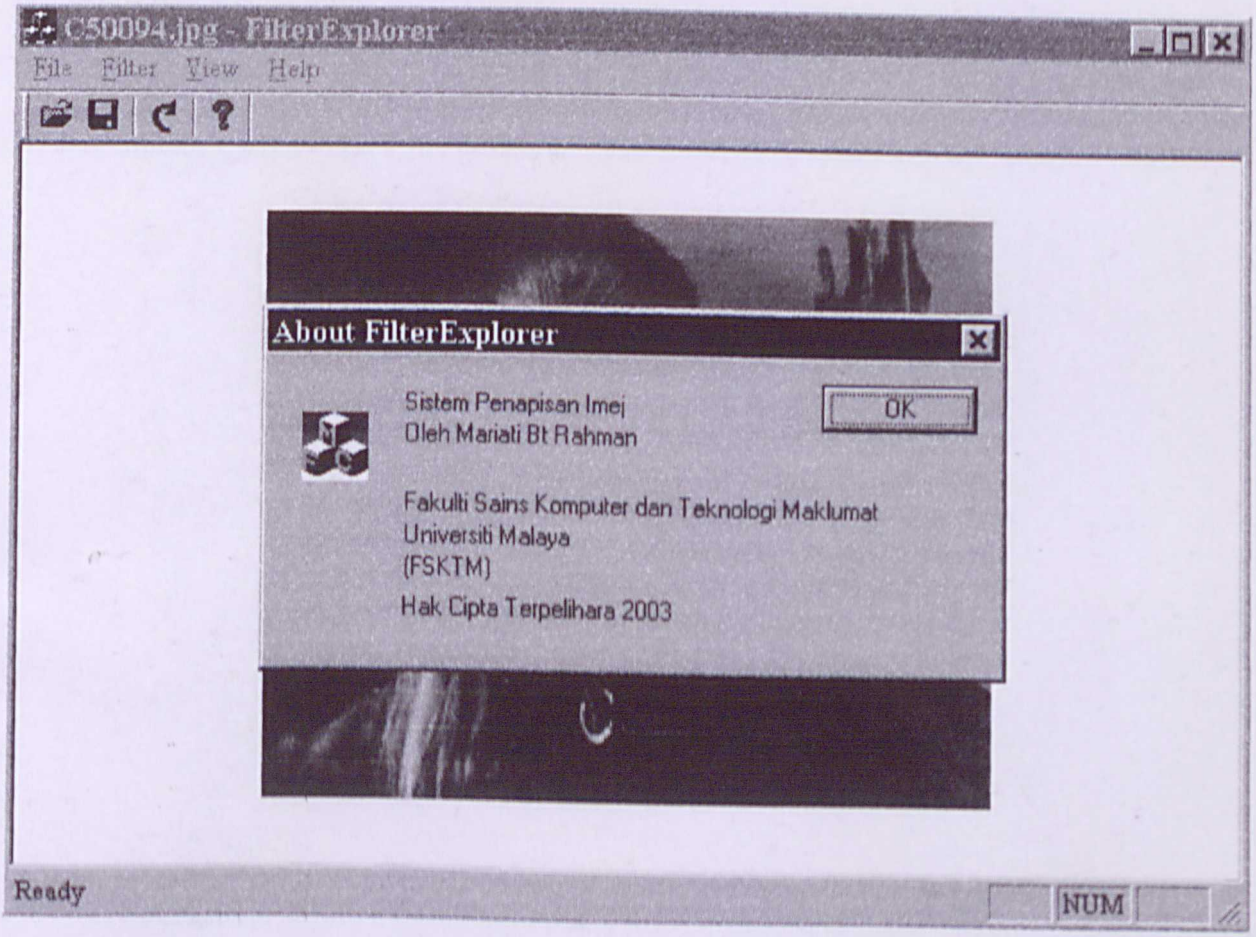

C50094.jpg - FilterExplorer
File Filter View Help
Oil Paint
Brush Size:
Small
Large
Smoothness:
Fine
Coarse
OK
Cancel
Ready
NUM


C50094.jpg - FilterExplorer
File Filter View Help
Frosted Glass
Frost Amount:
Not Much
A Lot
OK
Cancel
Ready
NUM


C50094.jpg - FilterExplorer
File Filter View Help
Random Blur
Block Size:
Small
Large
Block Bluriness:
Not Much
A Lot
Image Blur Density:
Low
High
OK
Cancel
Ready
NUM


C50094.jpg - FilterExplorer
File Filter View Help
Raindrops
Max Raindrop Size:
Small
Large
Raindrop Density:
Low
High
Include Highlight & Shadow
OK
Cancel
Ready
NUM


C50094.jpg - FilterExplorer
File Filter View Help
Save As
Save in: Lecolas
3565
501201
C50094
Legolas1
File name: C50094d
Save as type: JPEG (*.jpg)
Save
Cancel
Ready
NUM


C50094.jpg - FilterExplorer
File Filter View Help
About FilterExplorer
Sistem Penapisan Imej
Oleh Mariati Bt Rahman
OK
Fakulti Sains Komputer dan Teknologi Maklumat
Universiti Malaya
(FSKTM)
Hak Cipta Terpelihara 2003
Ready
NUM