# The Development of a Fragment-Based Image Completion Plug-in for the GIMP

Submitted in partial fulfilment
of the requirements of the degree
Bachelor of Arts (Honours)
of Rhodes University

Cathy Irwin

5th November 2004

## Abstract

Recent developments in the field of image manipulation and restoration have seen the merging of techniques that reconstruct texture and those that reconstruct structure to create algorithms that are effective for a far greater variety of image completion scenarios than were previously possible. Few of these innovative and useful algorithms are available to ordinary users however.

We describe the implementation of a version of one such technique, the fragment-based image completion algorithm developed by Drori, Cohen-Or and Yeshuran [2003]. This automatic image completion algorithm uses the known parts of an image to infer the unknown parts, thus reconstructing the missing region as realistically as possible. We implement this algorithm as a plug-in for the GIMP making it freely accessible and easy to use in conjunction with other image manipulation tools. Furthermore, the process of developing plug-ins for the GIMP is evaluated and an on-line tutorial for writing general plug-ins for the GIMP is developed alongside the image completion plug-in implementation.

Results achieved by this plug-in show that it is highly effective for completing regions that have random texture and it reconstructs the general structure of objects well. Geometric shapes are not necessarily completed as expected however because the algorithm does not take mathematical principles of shapes into account. It is concluded that the image completion algorithm is suitable for most of the scenarios put forward by the original authors. The tutorial provides a starting point for writing general plug-ins for the GIMP, covers common problems and lists useful resources. The process of plug-in development for the GIMP is well documented and supported.

**Acknowledgements**

Many thanks go to my project supervisors, Shaun Bangay and Adele Lobb for guiding me through the development process and offering much constructive citicism along the way.

Thank you also to my mother for proof reading, both my parents for being a willing audience for practising my presentations on, and my brother Barry for his comments and suggestions borne from having done it all before me.

Also thanks to the Andrew Mellon Scholarship Foundation for providing me with the funds to complete my Honours degree.

# Contents


**1 Introduction** **6**

1.1 Background . . . . . 6

1.2 Document Structure . . . . . 7

**2 Related Work** **8**

2.1 Texture Synthesis . . . . . 8

2.2 Image Inpainting . . . . . 10

2.3 Other types of image manipulation . . . . . 10

2.4 Mathematical exploration into completion fields . . . . . 11

2.5 GIMP Development . . . . . 12

2.6 Other implementations . . . . . 13

2.7 Summary . . . . . 14

**3 Design** **15**

3.1 Fragment-Based Image Completion . . . . . 15

3.2 Fast Approximation . . . . . 15

3.3 Confidence Map and Calculating candidate regions . . . . . 16

3.4 Searching . . . . . 19

3.5 Compositing Fragments . . . . . 19

3.6 Summary . . . . . 21

**4 Implementation in GIMP** **22**

4.1 Setting up the Plug-in . . . . . 22

4.1.1 Resources Used . . . . . 22

4.1.2 Modifying the template . . . . . 23

4.1.3 The User Interface . . . . . 24

4.2 Fast Approximation implementation . . . . . 25

4.3 Confidence map and Candidate region map implementation . . . . . 27
4.4 Search implementation . . . . . 28
4.5 Composite implementation . . . . . 29
4.6 How to use the Image Completion Plug-in . . . . . 31
4.7 Summary . . . . . 31

**5 Tutorial Development** **33**
5.1 Topics covered . . . . . 33
5.2 Web page Design . . . . . 34
5.3 Summary . . . . . 35

**6 Results** **36**
6.1 Testing Accuracy . . . . . 36
6.2 Testing Efficiency . . . . . 37
6.3 Summary . . . . . 40

**7 Conclusion** **45**
7.1 Future Work and Possible Extensions . . . . . 45
7.2 Conclusion . . . . . 46

**References** **46**

**A Tutorial: Writing a plug-in for the GIMP in C** **49**

**B Image Completion Results** **55**

# List of Figures


3.1 (a) Original Image. (b) Inverse matte $\bar{\alpha}$. (c) Image $\bar{C}$. (d) Image $Y$ after two blur iterations. (e) Final result of the fast approximation phase with blur radius set to 9 pixels. . . . . 17

3.2 (a) Initial confidence map. (b) Confidence map after 325 iterations. (c) Initial candidate position map. . . . . 18

3.3 (a) Completed image of the runner after 374 search and composite iterations. (b) Detail showing only the reconstructed area. . . . . 20

4.1 User Interface for the Automatic Image Completion Plug-in . . . . . 25

4.2 Channels and Layers dialogs which show how each processing phase can be viewed separately. . . . . 26

4.3 a) The original image. b) The areas reconstructed by the completion process. c) The completed image showing matching target and source regions are outlined with circles with the red lines pointing to the centre of the target regions. . . . . 30

5.1 Starting page for the on-line plug-in tutorial . . . . . 35

6.1 Results for geometric (a-b) and stochastic textures (c-d) showing the source image on the left and the completed image on the right with reconstructed region outlined. . . . . 38

6.2 Completion results for a variety of parameter settings: First two rows: gap size 30 x 35 pixels (a) Large blur radius, small neighbourhood size. (b) Smaller blur radius. (c) Large blur radius, larger neighbourhood size. Third row: (d) Smaller version of image with gap still 30 x 35 pixels, blur radius and neighbourhood size as in (a). Fourth row: (e) smaller gap size of 15 x 35 pixels. . . . . 41

6.3 (a) Original image. (b) Best reconstruction of missing area (outlined by blue square). . . . . 42

6.4 Impact of Neighbourhood Region Size on Image Completion Time . . . . . 42
6.5 Completion results for photographs. . . . . 43

A.1 Getting Started . . . . . 50
A.2 Accessing the PDB and putting your plug-in in the GIMP menu . . . . . 51
A.3 Modifying the template and Using existing GIMP procedures and plug-ins . . . . 52
A.4 Making a simple interface and Hints and tips page . . . . . 53
A.5 List of Resources . . . . . 54

B.1 Further image completion results . . . . . 56

# List of Tables

6.1 Results for images in Figures 6.2 and 6.3. N indicates the size of the neighbourhood region. . . . . 40

# Chapter 1

# Introduction

**Problem Statement:**

The primary aim of this project is to provide extra functionality to the GNU Image Manipulation Program (GIMP) by developing an automatic image completion plug-in for it. This plug-in is based on recent research into fragment-based image completion algorithms. A secondary aim is the development of an on-line tutorial for writing general plug-ins for the GIMP to be a guide for other wanting to write plug-ins in the future.

## 1.1 Background

The manipulation and modification of images in an undetectable manner has in the past required skillful and time-consuming artistic endeavour. Applications for such a process include the removal of scratches, defects or writing from photographs or films, the restoration and reconstruction of damaged images, and the removal of objects from images. Filling in the missing regions in a realistic manner is an artistic challenge. Computer technology has made this process significantly easier with many widely available image-editing packages such as Adobe PhotoShop or the GIMP. These provide tools such as erasing, blurring and clone brushing. These manual techniques still require skillful application however. Recent developments in the field of image manipulation and restoration have focused on automating the image completion process where the only user input is the selection of the region to be removed. The particular technique we implement is called fragment-based image completion and is based on the work of Drori, Cohen-Or and Yeshuran [2003]. Their algorithm focusses in particular on the filling-in of large, textured regions once objects have been removed and is used as a basis for the development of the auto-

matic image completion plug-in for the GIMP. A plug-in is a program that extends the operation of a parent application such as the GIMP which is an open source application intended for tasks such as photograph retouching, image compositing and image authoring.

While we describe a re-implementation of the techniques described in the original paper, the approach is novel because of its incorporation into the GIMP. Although there will inevitably be some differences in the results achieved due to the choice of implementation language, development environment and coding style, the general claims of the authors can be verified and the algorithm tested to see whether it works in all the scenarios that the original authors claim it does. For example, does this technique work as well on highly textured regions as smooth regions and can it reconstruct geometric shapes?

The process of writing plug-ins for the GIMP is also investigated. An on-line tutorial on writing general plug-ins for the GIMP has been developed alongside the implementation of the image completion plug-in. This tutorial aims to be a guide for anyone wishing to write a simple plug-in for the GIMP for the first time. It deals with some of the issues investigated during the implementation of the image completion plug-in including: hints on where to start, how to incorporate existing GIMP plug-ins and procedures in your code, an explanation of GIMP image components (layers, drawables, channels), a list of useful functions and a list of further resources and tutorials.

## 1.2 Document Structure

Chapter 2 outlines various techniques related to the field of image completion including those of texture synthesis, image inpainting and other related image manipulation techniques. This chapter also describes some of the other on-line tutorials for writing GIMP plug-ins that are available and that influenced the design of the tutorial developed as part of this project. Chapter 3 gives an overview of the fragment-based image completion algorithm and the design issues at each of the steps. This is followed by a detailed description of how steps in this algorithm have been implemented as a plug-in for the GIMP in chapter 4. Chapter 5 discusses in detail the on-line tutorial for writing plug-ins for the GIMP that has been developed in conjunction with the actual plug-in and a discussion of results takes place in chapter 6. The document concludes with a discussion on the findings and limitations of the process and conclusions regarding the success of the results.

# Chapter 2

# Related Work

The broad influences on the development of the fragment-based image completion algorithm [Drori et al. 2003] are those of texture synthesis, image inpainting, the recent combination of these two methods, related types of image manipulation and some of the more purely mathematical models of predicting completion fields.

In terms of plug-in development for the GIMP, on-line documentation and other tutorials have been the main resources used.

## 2.1 Texture Synthesis

Texture Synthesis is when a sample texture is used to generate a new texture of potentially unlimited size that is perceived to be the same texture as the original. New textures must be able to tile seamlessly. This is useful where large missing regions need to be filled. Within this broad category there are several approaches. These include a technique of incrementally building up new texture based on similar neighbourhoods in the sample texture [Wei and Levoy 2000]; using a given mask to replace regions with synthesized texture [Igehy and Pereira 1997]; producing new texture by stitching together blocks of sample texture [Efros and Freeman 2001]; producing new textures by means of pyramid-based texture analysis [Heeger and Bergen 1995]; and creating fundamentally new textures by means of applying editing operations to the original texture [Brooks and Dodgson 2002]. Some of these methods are more suited to reproducing stochastic (random) textures [Heeger and Bergen 1995, Igehy and Pereira 1997] while others work better for more structured, deterministic textures [Efros and Freeman 2001].

The work done by [Heeger and Bergen 1995] using pyramid-based texture matching models is an important influence on many later algorithms [Igehy and Pereira 1997, Brooks and Dodgson

2002]. Their method involves analysing a digitized image to compute various texture parameters. These parameters are then used in the synthesizing phase to generate the new texture so that it resembles the original. An image pyramid is generated which is a set of versions of the original texture image of varying sizes that correspond to different frequency bands [Heeger and Bergen 1995]. Synthesis is performed by matching the histograms (distributions) of the image pyramid to modify a noise image until it has a similar distribution to the example texture. Igehy and Pereira [Igehy and Pereira 1997] extend this algorithm by adding a composition step which uses a mask to turn the noise image into a combination of the original texture and its own synthesized texture. The result is a smoother transition between the original texture and the newly generated texture. This reduces boundary problems where the synthesized texture needs to tile with the original and also avoids obvious repetition of artifacts. As with Heeger and Bergens' original algorithm [Heeger and Bergen 1995], this technique is best suited to areas with stochastic textures and fails in areas of structured texture.

Markov Random Fields (MRF) is another commonly used texture model upon which many other texture synthesis algorithms are based [Wei and Levoy 2000]. This uses a method of probability sampling to generate new texture and is effective for a wide variety of texture types. However the sampling process is computationally very expensive, so Wei and Levoy [Wei and Levoy 2000] have adapted the process to use deterministic searching instead to greatly speed up the process and still achieve comparable results. The inputs consist of a sample texture and a random noise image of any desired size. Similar to the two previously mentioned algorithms, this one modifies the noise image until it matches the sample texture. This works particularly well on stochastic textures, but not where depth, 3D structure or lighting is important.

An entirely different approach is that employed by Efros and Freeman [Efros and Freeman 2001]. Instead of modifying a noise image to create texture, they do what they call image quilting which is effectively taking small samples of the original texture and stitching them together in such a way as to generate unlimited amounts of new texture. This is particularly effective for generating semi-structured textures, but works well on stochastic textures too. The basic algorithm consists of firstly patching together randomly chosen blocks of the original texture, then introducing some overlap according to some measure of how that block agrees with its neighbours. Finally, looking at the error in the overlap region, a new boundary for the block is determined by calculating a minimum cost path through that error surface.

Closely related to texture synthesis is texture editing. Brooks and Dodgson [Brooks and Dodgson 2002] have developed a technique of assessing the similarity of pixels within a texture to then apply global operations which affect the colour and brightness of all similar pixels. This

warps the original texture to create a fundamentally new one. Response times are improved by using pyramids (similar to Heeger & Bergen [Heeger and Bergen 1995]) and neighbourhood matching (similar to Wei & Levoy [Wei and Levoy 2000] ) to take a multi-scaled approach.

## 2.2 Image Inpainting

Image inpainting is based on the way professional art restorators retouch paintings. Significant work on this technique has been done by Bertalmio, Sapiro, Caselles & Ballester [Bertalmio et al. 2000]. Their algorithm aims to smoothly propagate the information on the boundary areas of the selection inwards to fill the gap. This is achieved by iteratively prolonging the isophote lines, which are lines with equal grey values in the image, inwards from the boundary while taking into account their angle of arrival and possible curvature. This reproduces the structure of the region but does not take into account the texture of the region at all. Therefore this method is best suited to filling in scratches and small regions of disocclusion (where an unwanted object obscures an area). Application of this algorithm on large, textured areas produces flat, unrealistic results, something which the fragment-based image completion algorithm [Drori et al. 2003] attempts to address using samples from neighbourhood regions to reproduce similar texture inside the gap. The only user input required is the selection of the regions to be inpainted.

The simultaneous filling in of texture and structure into missing regions has also recently been explored [Bertalmio et al. 2004]. The image is first decomposed into the sum of two functions, one capturing the structure and one the texture. The first of these is then reconstructed using image inpainting and the other is reconstructed using texture synthesis. The original image is then reconstructed by adding these two modified images. Theoretically, any automated texture synthesis or inpainting technique could be used in this method. This algorithm is intended for a far greater variety of image reconstruction scenarios, making it more versatile than any of the previously mentioned methods.

## 2.3 Other types of image manipulation

Closely related to the topic of texture synthesis and transfer are other techniques such as image analogies [Hertzmann et al. 2001], transferring colour to greyscale images [Welsh et al. 2002], and certain other image based modeling and editing techniques [Oh et al. 2001]. Common concepts utilized by these techniques and the fragment-based image completion algorithm are their use of pyramids and the methods of getting neighbourhood samples in order to manipulate

target pixels.

Transferring colour to greyscale images [Welsh et al. 2002] is an extension of texture transfer since a training colour is used similarly to the way in which a training texture would be used to generate new texture. By matching luminance and texture information in the source colour image and the target greyscale image, corresponding chromatic information is then mapped to the target image. Finding texture matches is handled similarly to the way Efros and Freeman [Efros and Freeman 2001] have described. The method for finding neighbourhood samples is particularly relevant to the fragment-based image completion algorithm [Drori et al. 2003] which also needs to search for appropriately coloured and textured samples from which to generate filler information.

Generating image analogies is another variation of texture manipulation. This involves the application of any one of a variety of complex image filters to modify the style of one image to match that of an example image such as an oil painting, watercolour, embossing or blur to name a few. Hertzmann [Hertzmann et al. 2001] proposes such a technique based on the recent developments in texture synthesis [Heeger and Bergen 1995, Wei and Levoy 2000, Efros and Freeman 2001]. Firstly, multi-scale pyramids of the source and target images are constructed. Then, from the coarsest resolution to the finest, neighbourhood statistics of each pixel are compared to find the best match between the unfiltered source and target image. The features in the corresponding place in the filtered source are then applied to the target, generating the desired effect. This method is similar to the way the fragment-based image completion algorithm [Drori et al. 2003] searches for appropriate source pixels.

Some editing techniques [Oh et al. 2001] deal particularly with overcoming the limitations of common clone brushing tools used to fill in gaps in an image. Clone brushing is when a piece of source region is copied directly to the destination region. This can however cause texture foreshortening and can produce extraneous artifacts unless the intensity, orientation and depth of the source and target regions match exactly. They have developed a way to use the 3D information in an image to overcome these problems by means of a distortion free clone brush and a texture-illuminance decoupling filter.

## 2.4 Mathematical exploration into completion fields

Other relevant investigations into image completion include those which focus on shape and curve completions across gaps [Sharon et al. 2000, Williams and Jacobs 1997], those which explicitly relate shape completion to image restoration [Hirani and Totsuka 1996], and those

which use a novel lines level structure to resolve disocclusion [Masnou and Morel 1998]. The latter works well for images which have prominent discontinuities.

Williams and Jacobs [Williams and Jacobs 1997] exploit the randomness of stochastic textures by assuming that the probability distribution of possible boundary completion shapes can be modelled by a random walk in a lattice across the image plane. The probability that such a walk will join two boundary fragments is computed as the product of two vector field convolutions. The smoothest resulting curve, called the curve of least energy, is used to complete the image. Similarly, Sharon et al. [Sharon et al. 2000] detect the curve of least energy but use multi-scale procedures to speed up the numerical analysis process.

In image processing terms, the frequency domain captures global features and large textures while the spatial domain captures local continuity and structure. As the previous examples highlight, many techniques are only effective on one of these domains (ie. they reproduce either texture or structure, not both). Hirani and Totsuka [Hirani and Totsuka 1996] have developed an algorithm which works in both of these domains in order to restore images in a natural way. Based on the theory of Projections onto Convex Sets (POCS), they also use sample regions from the source image to restore the noisy pixels. This method claims to be effective not only for stochastic regions [Williams and Jacobs 1997], but for areas containing randomly placed prominent lines.

## 2.5 GIMP Development

The main resource for any development in the GIMP is 'The GIMP Developers' website (`http://developer.gimp.org`). Among other things, this website provides information about plug-in development, mailing lists, submitting bug reports, standards, frequently asked questions about development and links to the GIMP reference manuals for the GIMP 2.0 API.

Also available on-line are a number of tutorials for writing plug-ins for the GIMP [Turner 1998, Neary 2004, Budig 2002]. Several of these tutorials follow a similar format in that there is a table of contents which can then be iterated through page by page. The tutorials that were most relevant to the development of the image completion plug-in and its accompanying tutorial are those that focus on writing plug-ins using C as opposed to Gimp-Perl or Gimp-Python which are other options. The design of these sites was an important consideration in the design of the plug-in tutorial developed for this project.

'Writing a GIMP plug-in' [Turner 1998] explains in detail the essential functions of a plug-in such as the main, query and run functions and how to include the plug-in in the Procedural

Database (PDB). It also covers details about working with images using pixel regions and tiles and gives an introduction to creating a user interface for a plug-in. Extensive code samples are provided. David Neary's paper, 'Writing a GIMP plug-in' [Neary 2004], provides an updated summary of Turner's manual and goes further to mention compilation and installation issues. 'Script-Fu and Plug-ins for the GIMP' [Budig 2002] covers the default scripting extension - Script-Fu, and some details about programming plug-ins in C. This tutorial also describes access to the PDB, covers topics on the organisation of image data and efficient access to pixel data and also provides code for a sample plug-in.

Another useful resource for GIMP development that has been used during the development of the image completion plug-in is the GIMP developers mailing list listed on the official GIMP website (`http://www.gimp.org`). Discussion on this list is about core program development as well as plug-in development.

## 2.6 Other implementations

Another implementation of the fragment-based image completion algorithm [Drori et al. 2003] is that by Daniel Crispell from Brown University in the United States [Crispell 2003]. Crispell implemented a version of the fragment-based image completion algorithm in MATLAB. He changed the algorithm in the following ways: firstly the source fragments are sampled from the original image only to avoid propagating errors. Secondly, the confidence values are 'reset' after the coarse level of completion to give the fine level compositions more power. There is also no fast approximation for the fine level phase to avoid blurring the results of the coarse level phase. Crispell concludes that while the fragment-based image completion algorithm is effective to some extent, it does not always synthesize believable natural patterns and does not take any underlying sense of structure into account so symmetrical objects are also not completed accurately.

While Crispell's implementation was in MATLAB, some of his code proved useful in designing the high level structure for the plug-in code, particularly for the fast approximation step. For the plug-in, the actual implementation of algorithm required an entirely different approach to that of Crispell however because of the use of GIMP functions and the need to write separate functions to do arithmetic operations on images, unlike the built-in operations available in MATLAB. For the search and composite steps, Crispell's design was very different to that of the plug-in so his code was not an influence at all.

## 2.7 Summary

Automated image restoration techniques mean that no particular training or skills are needed to perform complex image manipulations. This has significant advantages for the ordinary computer user. Current research in the field of image restoration and image completion is focusing on speedup and improving the efficiency of algorithms as well as further investigation into the combination of techniques to work more effectively for a broader variety of situations. In particular, there is much emphasis on developing algorithms that effectively complete both stochastic and structured regions. The fragment-based image completion algorithm [Drori et al. 2003] is a good representation of this direction of research and is as such an appropriate example of useful functionality that can be successfully incorporated into the GIMP. While other implementations of this algorithm have not managed to reproduce results of the same quality as those of the original authors, they are nonetheless able to reconstruct complex textures.

Due to the open-source nature of the GIMP, there are many on-line resources available about development in the GIMP besides what is available on the official GIMP website and the GIMP developer website. These resources include tutorials on writing plug-ins, reference manuals and examples of plug-in code.

# Chapter 3

# Design

## 3.1 Fragment-Based Image Completion

The fragment-based image completion algorithm [Drori et al. 2003] is an example of the current direction of development in the field of image manipulation and restoration. This algorithm uses the visible parts of an image as a training set to infer the unknown parts when a portion of the image is removed. It aims to synthesize a "complete, visually plausible and coherent image" [Drori et al. 2003] (pg 1). An outline of this process is as follows: firstly, an inverse matte that contains the entire extracted region is defined by the user. This inverse matte is used to mask off the regions to be removed and to define a confidence level for each pixel (called the confidence map) with those pixels that are closer to the known regions having higher confidence values. All the confidence values increase during the completion process. An approximation of the low confidence areas is generated using a simple smoothing process known as fast approximation. This rough region is then augmented with familiar details taken from areas of higher confidence. At each step a target fragment, which consists of a circular neighbourhood around the pixel, is completed by adding more detail from an appropriate source fragment which has higher confidence. The source fragments are selected from the immediate vicinity of the unknown region. As the process continues, the average confidence of the pixels converges to one, completing the image in a manner which takes both texture and some sense of structure into account.

## 3.2 Fast Approximation

The first phase of the image completion algorithm is the rough approximation of the unknown portion of the image, known as fast approximation. This step involves first multiplying the

original image *C* (Figure 3.1a) by its inverse matte $\bar{\alpha}$ (Figure 3.1b) which represents the portion of the image to be removed. This produces an image $\bar{C}$ (Figure 3.1c) where the region of the offending object is masked off by a black gap. $\bar{C}$ is added to $\alpha$ (the original matte of the image) and the resulting image is then repeatedly blurred, shrunk ($\downarrow$), expanded($\uparrow$) and blurred again and then has the known values $\bar{C}$ reintroduced at each iteration as well according to the following formula [Drori et al. 2003]:

$$Y_{t+1}^{l} = \left(Y_{t}^{l}\alpha + \bar{C}\right)\left(*K_{\varepsilon} \downarrow\right)^{l}\left(\uparrow *K_{\varepsilon}\right)^{l} \tag{3.1}$$

where Y is initially a plain white image and K represents a kernel filter that applies a Gaussian blur. A kernel filter works by applying a certain operation to every pixel in an image. The Gaussian blur applied here has a smoothing effect achieved by removing fine image detail and noise. The combination of blurring and scaling the image up and down has the effect of making the boundaries of the gap creep inwards until the missing region is filled with roughly the same colours as appeared on its borders (Figure 3.1d and e). These approximate colours guide the search for similar neighbourhoods during the search phase of the algorithm (Section 3.4). The known values in $\bar{C}$ are reintroduced at each iteration to prevent the blurred colours from becoming too diluted and to ensure that at the final iteration only the unknown region has been modified by the fast approximation process. The number of times that the resizing and blurring operations take place is determined by the number of pyramid levels *l* to be constructed from the image (ie. the number of different sizes of the image to be generated). The boundaries are considered to have converged sufficiently when the convergence error dips below a certain threshold called the convergence maximum. The convergence error is calculated on each iteration of the loop by taking the average difference between the image before the blurring process and after.

## 3.3 Confidence Map and Calculating candidate regions

The second phase of the algorithm is to use the inverse matte $\bar{\alpha}$, which specifies the areas to be completed, to calculate a confidence map $\beta$ of the image to define the level of certainty with which the colour value of each pixel *i* is known. The confidence map is calculated according to the following formula [Drori et al. 2003]:

$$\beta_i = \begin{cases} 1 & if \bar{\alpha} = 1 \\ \sum_{j \in N(i)} g_j \bar{\alpha_j^2} & otherwise \end{cases} \tag{3.2}$$

Here *N* is the size of the neighbourhood region around the pixel and *g* is a Gaussian falloff

[H]

(a) (b) (c) (d) (e)

Figure 3.1: (a) Original Image. (b) Inverse matte $\bar{\alpha}$. (c) Image $\bar{C}$. (d) Image $Y$ after two blur iterations. (e) Final result of the fast approximation phase with blur radius set to 9 pixels.

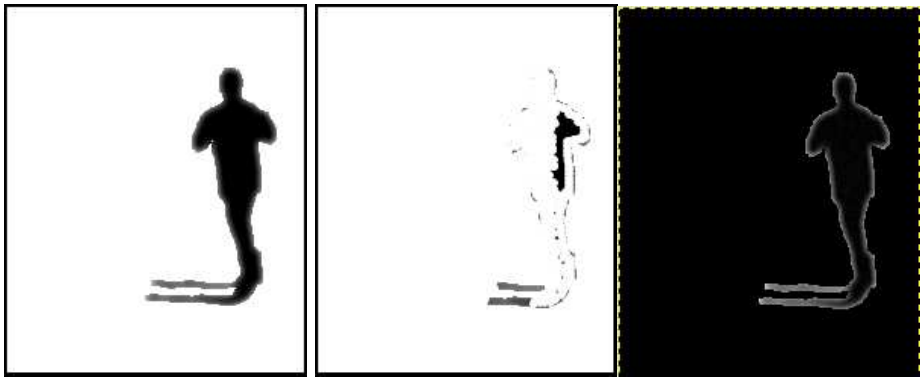
function which decreases the level of confidence toward the centre of the unknown region (Figure 3.2d). The white pixels (which are known) have a confidence value of 1 while the unknown pixels are black and grey depending on their distance from the border of the unknown region. Figure 3.2(a) shows the initial confidence map $\beta$ that is calculated with close-up detail of the runner's head in (d). During the completion process the black areas gradually turn to white as the confidence in those pixel values increases (Figure 3.2b). This confidence map is used later for comparing and selecting fragments during the search phase (Section 3.4).

The confidence map $\beta$ is then passed as input to the function which calculates the set of candidate positions $v$ to use in the search phase of the algorithm. Candidate positions are those in the unknown region that need to be filled in. This is done according to the following formula [Drori et al. 2003]:

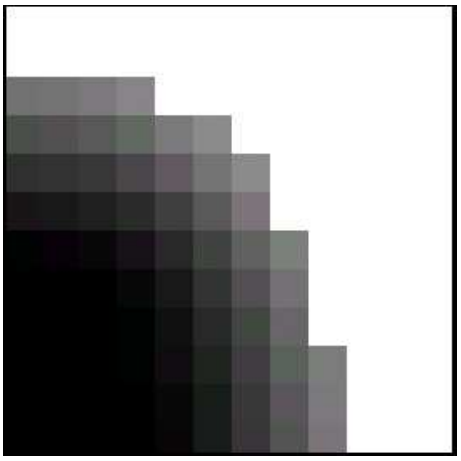
$$v_i = \begin{cases} 0 & if \beta_i > \mu(\beta) \\ \beta_i + \rho[0, \sigma(\beta)] & otherwise \end{cases} \tag{3.3}$$

This function sets pixels *i* which have a value greater than the mean confidence level, $\mu(\beta)$, to zero and adds a random noise between 0 and the standard deviation $\sigma(\beta)$ to the confidence values of the other pixels. The pixel with the maximum value in the candidate map is then concluded to be the next most appropriate pixel, *T*, to be used in the search and composite phases. This is to the lightest, or most known pixel in the unknown region. Figure 3.2(c) shows the initial map of candidate positions. The set of candidate positions is recalculated after every iteration of the

[h]



(a) (b) (c)



(d)

Figure 3.2: (a) Initial confidence map. (b) Confidence map after 325 iterations. (c) Initial candidate position map.
(d) Detail from the head of the runner in the confidence map showing how the pixels are lighter toward the edge of the unknown region, representing their higher confidence values.

image completion process to take into account the updated confidence map.

## 3.4 Searching

The search algorithm uses the confidence value of each pixel together with its colour value to find the best matches between unknown target regions and known source regions. The colour values of the pixels in the unknown region are obtained from the result of the fast approximation procedure and represent the similarity of features between the source and target regions. The search procedure takes the target pixel *T* and iterates through each pixel in the vicinity to find an appropriate source pixel *S* to use in the compositing phase of the algorithm where the detail from the known region is used to fill in the unknown regions. For each potential source pixel *S*, the neighbourhood region around it is examined to determine the difference *d* between the colour of *T* (in the unknown region) and *S* (in the known region) which is then used together with the difference between the values in the confidence map at the corresponding coordinates according to the following formula [Drori et al. 2003]:

$$r = \min \sum_{s=S_r(i),t=T(i),i\in N} \left(d\left(s,t\right)\beta_s\beta_t + \left(\beta_t - \beta_s\right)\beta_t\right) \tag{3.4}$$

*N* is the size of the neighbourhood region around *S* for which the average colour value and relative confidence values are determined. N is the same size as the neighbourhood used in the calculation of the confidence map (Eq. 3.2).

According to the original authors [Drori et al. 2003] this formula finds the pixels which have a higher confidence value in the source than in the target regions and correspond well in terms of colour. The similarity between the features of *T* and *S* is denoted by $d\left(s,t\right)$and their confidence is denoted by $\beta_s\beta_t$. Therefore the first term in the function, $d\left(s,t\right)\beta_s\beta_t$, penalizes different values in corresponding pixels which have high levels in both the source and target fragments. The second term of the function, $\left(\beta_t - \beta_s\right)\beta_t$, rewards pixels with a higher confidence value in the source than the target and penalizes those pixels with a lower confidence in the source than the target.

## 3.5 Compositing Fragments

Once a suitable source pixel, *S*, has been selected, the next step is to superimpose the fragment surrounding *S* over the target fragment so that the detail from the source merges seamlessly with

(a)

(b)

Figure 3.3: (a) Completed image of the runner after 374 search and composite iterations. (b) Detail showing only the reconstructed area.

the region around the target. The source and target fragments also need to be the same size and this is determined by the size of the neighbourhood region, $N$, as used in Eq. 3.4. It is also important to take the alpha values of the regions into consideration. The transparency of the selected source fragment needs to increase toward its edges in order to blend in convincingly with the target region that it is layered on top of. The design of this phase of the algorithm deviates from the method of the original authors [Drori et al. 2003] in that instead of using a Laplacian pyramid and OVER operator to composite the fragments, a more simple copy and paste technique is used which employs feathering of the source region to adjust the alpha values appropriately. It was decided that making use of the existing GIMP functionality is adequate to achieve the same sort of effect as the complex pyramid calculations which merge images according to binary masks. The only expected disadvantage of using the more simple technique is that the overlap of known and unknown regions does not vary according the size of the gap to be filled.

Once the compositing step is complete, the confidence map is updated accordingly to reflect that that target region is no longer unknown. The process of calculating the confidence map, selecting candidate pixels, searching and compositing the selections is then repeated until the image is determined to be complete (when all values in the confidence map converge to 1). Figure 3.3 (a) shows the result of this process with the detail of the reconstructed region shown in (b).

## 3.6 Summary

The design of the algorithm for the image completion plug-in for the GIMP follows that outlined in the paper "Fragment-Based Image Completion" [Drori et al. 2003] as closely as possible with minor simplifications and adaptations as necessary to suit implementation in the GIMP environment. The fast approximation step uses a smoothing process to create an image where the region to be reconstructed is filled with approximate colours; the generation of the confidence map and candidate region map allow the search procedure to match target regions in the unknown area with appropriate source regions in the known areas of the image; and the composite step superimposes these source regions over the target regions. The next chapter describes how each section of the algorithm is implemented as part of the image completion plug-in and the specific issues that arose during the development process.

# Chapter 4

# Implementation in GIMP

This chapter discusses the process of writing a plug-in for the GIMP in the context of the development of the image completion plug-in. How each phase of the image completion algorithm is implemented is then discussed in detail with special reference made to the GIMP-specific features employed.

## 4.1 Setting up the Plug-in

### 4.1.1 Resources Used

The plug-in works in GIMP 2.0. and version 1.3.3 of the freely available plug-in templates from the GIMP Developers' website (`http://developer.gimp.org/plug-in-template.html`) is used as a base for the implementation. The template is essentially an 'empty' plug-in which merely defines the structure needed. The implementation language used is C (as opposed to Perl or Python) because of its superior stability when programming plug-ins and because of the extensive documentation available for writing plug-ins for the GIMP in C. The GIMP Reference Library manual (`http://developer.gimp.org/api/2.0/libgimp/index.html`) and the GTK library (`http://developer.gnome.org/doc/API/2.0/gtk/index.html`) are two such extensive resources which provide C functions that can be incorporated into the plug-in code. The GIMP Reference Library manual in particular provides access to functions which manipulate GIMP-specific elements such as drawables, channels and layers as well as access to all the standard GIMP toolbox facilities and existing plug-ins and procedures. It also lists functions that manipulate colours, brushes, 'pixelregions', paths, selections and many other GIMP components. The GTK library has been used to create the interface for the plug-in.

### 4.1.2 Modifying the template

Among other files that come with the plug-in template, there are four folders, namely 'help', 'pixmaps', 'po' and 'src'. The 'help' folder provides html templates for writing on-line documentation for the plug-in. The 'pixmaps' folder contains some icons images and the 'po' folder contains the files required for translation of the plug-in into various languages including German, French, Swedish and Chinese. All the actual code for the image completion plug-in is written in the 'main.c' file of the 'src' folder which also contains interface and render classes.

In order to access the custom plug-in from within the GIMP, it first needs to be registered in the Procedural Database (PDB) which is queried by the GIMP at start up. This allows it to be accessed by other plug-ins and scripts. The PDB describes the functionality of the plug-in as well as its parameters. This means that each time the plug-in is recompiled GIMP needs to be restarted so that it can access the most up-to-date version of the plug-in. The PDB is queried by checking the value of the global PLUG_IN_INFO variable in the *main()* method.

```
/* Setting PLUG_IN_INFO */
GimpPlugInInfo PLUG_IN_INFO =
{
   NULL, /* init_proc */
   NULL, /* quit_proc */
   query, /* query_proc */
   run, /* run_proc */
};
```

The most important functions are the *query()* and *run()* functions. The *query()* function makes the call to the *gimp_install_procedure()* (shown below) which registers the plug-in in the PDB and specifies where in the menu structure the plug-in will occur.

```
gimp_install_procedure (PROCEDURE_NAME,
          "A fragment-
based automatic image completion plug-in",
          "Approximates detail in an unknown re-
gion by first smoothing the unknown area
            and then sampling detail from appropri-
ate source regions to composite into the
            target regions.",
```

```
"Cathy Irwin <ctn@rucus.net>",
"Cathy Irwin <ctn@rucus.net>",
"2004",
("<Image>/Filters/Misc/AutoComplete"),
"RGB*, GRAY*, INDEXED*",
GIMP_PLUGIN,
G_N_ELEMENTS (args), 0,
args, NULL);
```

The *run()* procedure makes sure that the plug-in is called correctly and that the return parameters are set. It then runs the plug-in. The GIMP plug-in template handles most of this setting up with only minor parameter name changes where necessary.

### 4.1.3 The User Interface

There are three different run modes in the *run()* procedure namely, run interactively where the plug-in is called from a menu item, run non-interactively where it is called from a script or another plug-in, or run with the same values as previously which happens when the Ctrl-F shortcut is used to re-run the previous plug-in. For the purposes of the image completion plug-in, the GIMP_RUN_INTERACTIVE mode is used to call the options dialog for the plug-in. The user interface for this plug, shown in Figure 4.1, was created using the GTK toolkit which provides facilities for the creation of common user interface objects such as buttons, tables and scrollbars, which are known as GTKWidgets. Each widget is simply added to the dialog container and then connected to methods which set the parameter values for the plug-in as in the following code extract to set the action of the 'Apply' button:

```
/*connect widgets to functions*/
g_signal_connect (GTK_OBJECT (button), "clicked",
                  G_CALLBACK (dia-
log_button_update), setlevelbox);
```

The parameters which are adjustable on the user interface are the radius of the Gaussian blur to be applied, the number of pyramid levels to be constructed during the fast approximation phase, the size of the search region around each target pixel and the size of the source fragment to be composited (the neighbourhood region size).

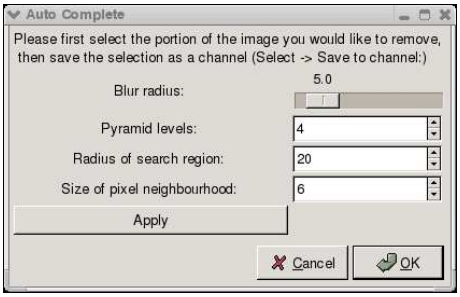



Figure 4.1: User Interface for the Automatic Image Completion Plug-in

## 4.2 Fast Approximation implementation

The first phase of the implementation of the fragment-based image completion algorithm is the fast approximation step. Here, advantage is taken of the GIMP concept of a *drawable*. A drawable is anything in an image that can be drawn on with the GIMP painting tools. These include layers, channels, layer masks and selection masks. Having different layers in an image allows one image to be constructed of several conceptual parts that can be manipulated and viewed independently of other parts of the image. In GIMP, layers form a stack with the bottom of the stack being the background. Each layer can have its own attributes including its level of transparency which determines how much of the layer below it is visible. In order to check the results of each phase of the fast approximation process, each successive step in the image manipulation is added as a new layer to the original image (Figure 4.2). This allows the accuracy of intermediate image manipulations to be checked. The visibility of each layer and channel can be turned on or off to view the stages independently or their cumulative effect.

In order to perform some of the common operations necessary for this step, such as the multiplication or addition of images, separate subroutines are defined that utilise some of the built-in GIMP functions. These routines need to iterate through every pixel in the image in order to perform their operations. To be able to get and set the value of a pixel in GIMP, the *gimp_drawable_get_pixel()* and *gimp_drawable_set_pixel()* functions need to be called from the GIMP function library. Repeatedly making these external calls to the GIMP is computationally expensive however which leads to relatively slow processing time.

In order to optimize these functions, a new data type called a fast_image is created as follows:

```
typedef struct fast_image
```

[H]


Figure 4.2: Channels and Layers dialogs which show how each processing phase can be viewed separately.

```
{
    int bands;
    int width;
    int height;
    guint8* pixels;
}fast_image;
```

Methods are defined to create a fast_image from a gint32 type, which is the type used by GIMP for regular drawables, and to get the pixel value at specified coordinates in the fast_image. In the above structure, 'bands' refers to the number of colour bands or bytes per pixel in the image. 'Width' and 'height' refer to the dimensions of the whole image and 'pixels' is a pointer to the value of each pixel in the image. In the fast approximation method the various layers upon which operations need to be performed are converted into fast images as soon as they are created. During this creation process the *gimp_drawable_get_pixel()* function is called once per pixel and that value is stored in the fast_image structure. This fast_image is then passed as a parameter to the add and multiply methods to avoid having to make the expensive external methods calls to the gimp to get the pixels values within those methods every time they are called.

The process of repeatedly blurring, shrinking, expanding and blurring the image to make the boundary colours converge into the unknown region (Eq. 3.1) is handled by making use of an existing GIMP procedure to scale a layer and another GIMP plug-in to perform a Gaussian blur. In order to use the Gaussian blur plug-in, the *gimp_run_procedure()* procedure is called which

takes the call to the *plug_in_gauss_iir()* plug-in as a parameter along with the parameters of that plug-in as specified in the PDB. The radius of the Gaussian blur that is applied is adjustable on the user interface since it affects both the convergence rate and the dilution of the border colours. The wider the unknown region the greater the blur radius can be. The number of times the blurring and scaling takes place on each iteration is determined by the number of pyramid levels, $l$, (Eq.3.1) which is also determined by the user. This pyramid level affects the density of the blurred region which in turn also affects the rate of convergence.

The test for convergence takes the difference between the value of the image $Y$ at the start of the last scale and blur loop and at the end of that loop. If the difference is sufficiently small then the approximate colours have converged in the centre of the unknown region, completing the fast approximation phase of the algorithm.

## 4.3 Confidence map and Candidate region map implementation

The confidence map is generated by first checking the value of each pixel in the inverse matte which is passed to the confidence map procedure as a parameter. If the pixel is white (corresponding to 1 in Eq. 3.2) then it is a known pixel and the corresponding confidence map value is also white. The unknown regions are black in the inverse matte and so the formula (Eq. 3.2) is applied there to calculate the corresponding confidence map values. The confidence map (Figure 3.2a) is added as a new channel to the image instead of as another layer. This is because it represents more of a selection mask that defines the unknown regions in the other layers that will be manipulated rather than a separate layer which itself forms part of the final result.

The confidence map is not re-generated from scratch at each iteration of the completion loop. Instead the same confidence map drawable is simply updated during the composite phase at the same time as the final image is updated. While this technique does not strictly adhere to the method of the original paper, it cuts down on the processing time by avoiding having to recalculate the value of every pixel in the map and making further computationally expensive eternal calls to the GIMP on each iteration of the completion loop (such as *gimp_drawable_get_pixel()*).

The procedure to calculate the candidate position map takes the confidence map drawable as a parameter. For each pixel in the confidence map, if that pixel is non-white (unknown), the value of the corresponding pixel in the candidate position map is calculated according to Eq. 3.3. This results in a map of all the candidate target positions with the order they are to be searched for determined by their grey value. The candidate pixel with the maximum value (closest to

white) is determined to be the next target pixel and its coordinates are passed as parameters to the procedure which performs the search of the rest of the image looking for an appropriate source region to sample detail from. The resulting candidate map (Figure 3.2b) is also added to the image as a new channel.

## 4.4 Search implementation

The way that the search algorithm (Eq. 3.4) is implemented in the plug-in is a slight simplification from the technique used in the original paper which searched not only over all pixels x and y, but also over five size scales and eight orientations of the image. It also compared luminance values as well as the colour values of $T$ and $S$. In order to minimize the processing time for the image, these refinements have not been implemented in the plug-in code. One of the expected impacts of this simplification is that for images where the same texture pattern appears in different scales (such as a brick wall receding into the background) the search function will not necessarily be able to find the best source match. This will not however affect images which have a frontal perspective. Implementing the extra refinements would also require significant extra processing time to complete the image judging from the amount of time it takes to search through just one version of the image. Adding another five scales and eight orientations of the image could mean this search procedure takes approximately 13 times longer to execute. Another limiting factor is that it is not easy to obtain luminance values for pixels, so the *d_func()* function (corresponding to $d$ in Eq. 3.4) which calculates the difference between pixels only takes into account their RGB values which are easily obtainable by using the *gimp_image_pick_color()* function which returns the RGB colour triplet as opposed to the *gimp_drawable_get_pixel()* function which returns a single value for a pixel.

The search algorithm is implemented as a procedure which takes the target coordinates selected from the candidate map, the image resulting from the fast approximation and the confidence map as parameters. During the search process, the pixel values at the target coordinates need to be compared with the pixel values of all the possible source coordinates within a reasonable radius to find the best possible match. This radius is set on the user interface and will depend on how rich in detail the area surrounding the the target pixel is. The larger the search area the longer it will take to find the best match and the more chance there is that an inappropriate source pixel will be selected. Furthermore, all the pixel values in the neighbourhood (the size of which is also determined by the user) around each potential source pixel are averaged to determine the best possible match with the neighbourhood surrounding the target pixel. The colour differ-

ence between the source and target pixels that is used in this averaging process is determined by the *d-func()* function. This process needs to utilize the *gimp_drawable_get_pixel()* and the *gimp_image_pick_color()* functions for both the target and source, but it is these external function calls which use up so much processing time when they are called from within the *d_func()* function. In a 120x160 pixel image the search procedure needs to be called between 130 and 400 times depending on the size of the gap to be filled. This means that those four external calls happen approximately 14 400 times for each new target pixel (ie. for each search). Processing an image in this manner takes approximately five and a half hours.

To optimize this search function and avoid having to make as many expensive external calls, an initializing loop is run at the beginning of the search function which calls *gimp_drawable_get_pixel()* for each pixel in the image and then places that value in a 2D array called imageArray which is the size of the image. Each time the pixel value is needed from that point onward, the values are read from the array instead of calling the GIMP function again. Also in this initializing loop the colour of each pixel is placed in a colour array (colArray). When the *d_func()* function is called, the appropriate colArray values are passed to it which are already of type GimpRGB so the *gimp_image_pick_color()* function does not need to be called at all in *d_func()*. This means that for a 120x160 pixel image there are approximately 38 400 external calls (initializing the arrays at the beginning) instead of between one million and six million external calls. This reduces the processing time to between 15 and 40 minutes on average, depending on the size of the gap.

After analysing all the pixels, it is the source pixel whose neighbourhood is least different to the neighbourhood of the target pixel that is deemed to be an appropriate match. Figure 4.3(c) shows how matching source regions are chosen from within the search area immediately surrounding the target pixel. The coordinates of this source pixel are then passed to the composite procedure which performs the final phase of the process, that of superimposing the source region onto the target region to approximate the detail.

## 4.5 Composite implementation

The implementation of the composite phase of the algorithm takes advantage of some of the built-in toolbox and editing functions in the GIMP. These functions, select, copy and paste, can be called in the same way as any other function in the GIMP library reference manual. The composite procedure takes the coordinates of the target and sources pixels as input as well as the size of the neighbourhood region around those pixels as determined by the user. The active layer of the image is then set to be the layer showing the result of the fast approximation phase as it

[h]

(a) (b) (c)

Figure 4.3: a) The original image. b) The areas reconstructed by the completion process. c) The completed image showing matching target and source regions are outlined with circles with the red lines pointing to the centre of the target regions.

is onto this layer that the detail is superimposed. The next step is to use *gimp_ellipse_select()* to select the neighbourhood region around the source pixel. This selection is anti-aliased and feathered by one pixel to make the pixels on the edge of the selection more transparent to help with blending. The selection is then copied using *gimp_edit_copy()*. Another selection is then made around the neighbourhood surrounding the target pixel which replaces the previous selection. The *gimp_edit_paste()* function is then called to paste the copied source region into the selected target region. Figure 4.3(c) shows the completed image and outlines two of the source regions that have been composited onto the target regions (the target pixels are indicated by a red line).

Once the final image has been updated the next step is update the confidence map which keeps track of which pixels are still unknown. A similar technique to that used to superimpose detail onto the target regions in the fast approximation layer is used here. Firstly the active layer is set to be the confidence map, then a source region that corresponds exactly to the source region selected in the fast approximation layer is selected. This selection is not feathered however to ensure that the edges of the composited region are not still considered to be unknown which could happen if they were transparent and then pasted over a black region. The source region is then pasted over the target region in exactly the same way as before, but since the active layer is now the confidence map, this has the effect of pasting white regions over the target regions to show that they are no longer unknown (ie. they have a confidence value of one).

## 4.6 How to use the Image Completion Plug-in

To use the image completion plug-in in the GIMP the users first need to select the region of the picture that needs to be removed using any of the available selection tools in the GIMP namely rectangular selections, elliptical selections, hand-drawn selections, contiguous regions selections, selections by colour or selections by shape. The selection must then be saved as a channel by clicking on the 'Select' option on the menu and choosing the 'Save to Channel' option. This creates the mask (the image matte) which is used by the plug-in to replace the desired region. The selection can then be cleared. To run the plug-in, the user must go to the 'Filters' menu, select the 'Misc' option and then choose 'AutoComplete'. A dialog will appear with various parameters that the user can change to suit the type of missing region they are working with. These parameters are the Gaussian blur radius, the number of pyramid levels, the size of the search region and the size of the pixel neighbourhood. No further user input is required.

At present the image completion plug-in only works for rectangular images due to the way the completion algorithm iterates through all the pixels in the image. This is suitable for the majority of images however, so is considered an acceptable constraint.

## 4.7 Summary

Many adaptations to the original algorithm have been made during the implementation process in order to take advantage of the available GIMP functions as well as other GIMP plug-ins. Each phase of the algorithm has been implemented as a separate procedure and several other functions have also been written in order to perform arithmetic operations on the images. The intermediate results from each of the main steps of the algorithm have been added to the image as new layers or channels in order to monitor the progression of the completion process. The fast approximation procedure results in a new layer of the image upon which known regions are later superimposed during the composite step. The confidence map and candidate position map are implemented as new channels which mask off the unknown regions and select target pixels which are then passed as parameters to the search procedure. A plug-in template is used as a base to create the plug-in to make the implementation easier and to adhere to GIMP plug-in standards with regards to code layout. How to use the template is not immediately intuitive but is manageable with help from on-line plug-in tutorials. Certain aspects of the process were overlooked however in these tutorials which the tutorial developed in conjunction with the image completion plug-in intends

to address. The next chapter describes the development of that tutorial and the issues that it covers.

# Chapter 5

# Tutorial Development

Alongside the implementation of the image completion plug-in, an on-line tutorial entitled "Writing a Plug-in for the GIMP in C" (`http://www.cs.ru.ac.za/research/students/g01i0286/TutorialPage/`) has been developed. Making the tutorial available on-line means that it is immediately accessible to users worldwide and is easy to reference at the same time as using the GIMP. Since every plug-in is different, the tutorial takes a very general approach. The intention is to provide a starting point for beginners rather than definitive answers on every aspect of plug-in development. It provides a basic overview of how to get started on building a custom GIMP plug-in and where to look for more help. Areas of plug-in development that are covered include those that proved to be problematic during the development of the image completion plug-in or were discovered to be particularly useful. The tutorial also provides references to where more specific information on plug-ins and the GIMP can be found.

## 5.1 Topics covered

As discussed in Section 2.5, there are many other tutorials on writing GIMP plug-ins available on the Internet [Neary 2004, Turner 1998, Budig 2002] so the new tutorial looks at areas that are not focussed on in the other tutorials. Areas which the other on-line tutorials focus on include detailed explanations of the PDB parameters, dealing with 'pixelregions' and 'tiles' and using scripts such as Scheme for writing plug-ins. Reference has been made to these other tutorials where necessary instead of repeating their content. This tutorial is divided into the following four sections:

1. Home

2. Getting Started
3. Resources and Other GIMP tutorials
4. Hints and Tips

The "Home" page provides a general introduction to the intention of the tutorial. The "Getting Started" section is a series of pages that provide a step-by-step guide for beginners about how to start with plug-in development. Areas that are covered include: how to obtain one of the official plug-in templates from the GIMP developer website and how to use it; accessing the procedural database (PDB) and adding one's plug-in in the GIMP menu; where to modify the template; how to use existing GIMP procedures and plug-ins within one's own custom plug-in; and making a simple interface using GTK (GNU Toolkit). The "Resources and Other GIMP tutorials" section has links to the official GIMP site (`http://www.gimp.org`), the GIMP Developers' Site (`http://developer.gimp.org`) and three subsections: links to other plug-in tutorials, links to GIMP user tutorials and links to important references manuals. These are the GIMP library reference manual, the GTK reference manual and the GIMP User Manual. The "Hints and Tips" section provides a more detailed explanation of certain concepts such as the difference between layers, channels and drawables; potentially problematic areas; and also points back to other topics within the "Getting Started" section such as how to incorporate existing GIMP procedures and plug-ins.

## 5.2 Web page Design

The layout of the Tutorial page is based on the format used by similar on-line plug-in tutorials [Budig 2002, Turner 1998] and is intended to be simple and intuitive. The menu panel is fixed on the left hand side with only the main content pane changing. In order to make the tutorial as easy to follow as possible, there are arrow buttons which guide the user along each step of the "Getting Started" section. The other sections are mainly for reference purposes and so links are displayed in a list format. Standard html together with basic php was used to code this page to ensure that it is rendered correctly in most Internet browsers. Figure 5.1 shows a screen shot of the first page of the tutorial.

For a full view of the tutorial, please refer to Appendix A.

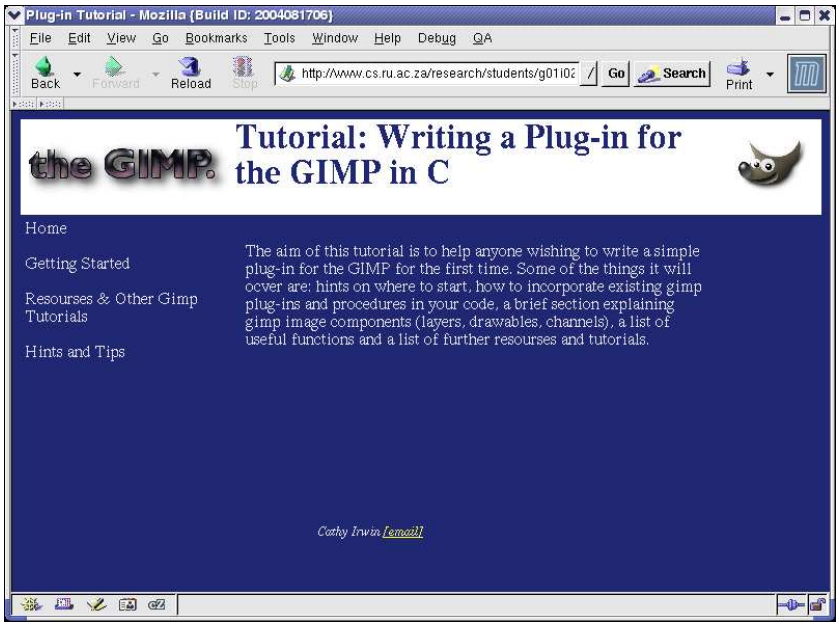



Figure 5.1: Starting page for the on-line plug-in tutorial

## 5.3 Summary

The motivation behind the development of an on-line tutorial on writing plug-ins for the GIMP in C was to highlight the difficulties experienced during the development of the image completion plug-in for the GIMP as well as to point out useful resources in order to make the process of plug-in development easier in the future.

# Chapter 6

# Results

This chapter discusses the results of the image completion plug-in for the GIMP in terms of two aspects of performance: how accurate the image completion algorithm is and how efficient it is as implemented in the GIMP as a plug-in. In terms of the first aspect, since the aim of the plug-in is to reconstruct areas of an image that have been removed in the most realistic way possible, the accuracy of the algorithm is measurable by how accurately it reconstructs the removed pieces. As far as the second performance goal is concerned, the amount of processing time required to complete an image is the determining factor. The following two sections discuss the process of testing the performance of the plug-in in this way. Conclusions are then reached about the type of image reconstruction that the image completion plug-in is most effective for and why the results of this implementation differ in some respects to those achieved by the original authors of the paper upon which this algorithm is based.

## 6.1 Testing Accuracy

In order to test how convincing the results of the automatic image completion process are, a variety of images are used that have different textures and structures to be reconstructed. These are categorised into those images with smooth regions and those with textured regions, and for the textured regions they are further classed into stochastic (random) and structured or geometric textures.

For the sake of consistency, the area around the target pixel to be searched is set to be 40 x 40 pixels for all the tests. The accuracy with which regions are reconstructed partially depends of the radius of the Gaussian blur that is applied during the fast approximation process. This is because the blur radius determines how far the border colour creeps into the unknown region

at each iteration. The bigger the blur radius, the more diluted the creeping colour becomes. Therefore, the smaller the gap, the smaller the blur radius should be to approximate the colours in the unknown region most accurately. For larger missing regions however, a blur radius that is too small means that the colours do not converge to the centre of the unknown region. The size of the neighbourhood region around the pixels also affects the accuracy since it determines the area of the source region to be superimposed onto the unknown target region. A neighbourhood region that is too small may introduce unwanted artifacts into the reconstructed region (Figure 6.2d and e). To reconstruct large, smooth or randomly textured areas, setting a greater neighbourhood region will reduce the processing time without affecting the accuracy of results (Figure 6.2c).

Figure 6.1 shows the results produced by the plug-in for the reconstruction of both geometric and stochastic textures. Since the search algorithm is example-based, using a comparison between the colours and confidence values of the source and target regions, it does not explicitly take geometric structure, such as the curve of the circle (Figure 6.1a), into account. This is because it only copies other areas of the image which are determined to be the most similar to what might fill the missing region, based on the colours surrounding the missing region which converge into the gap during the fast approximation stage. Nowhere else in the image of the circle does that particular curve occur so reconstruction of it is inaccurate at best. By contrast, the geometric lines (Figure 6.1b) are almost perfectly reconstructed because the pattern appears elsewhere in the image.

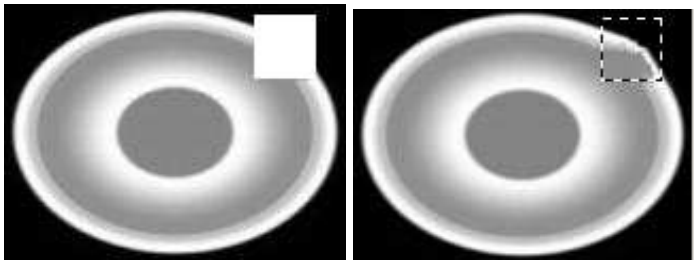
The plug-in works consistently more effectively for the reconstruction of stochastic textures (Figures 6.1c and d). Although the generated patterns are not perfectly realistic, they tile seamlessly with the surrounding areas making the reconstruction hardly noticeable. Figure 6.5 shows the completion results for various photographs. These results confirm the hypothesis that stochastic textures (such as the mountainside surrounding the Hollywood sign and the grass around the dog) are reconstructed most convincingly.

## 6.2 Testing Efficiency

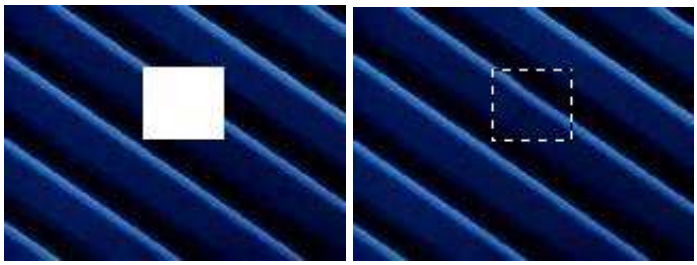
Testing of the efficiency of the automatic image completion algorithm involves determining how long it takes to complete images of a variety of sizes and with a variety of different gap sizes to be reconstructed.

A very simple timing mechanism is used to determine how long the entire process takes. The in-built C *time* class is used as follows: *t1* and *t2* of type *time_t* are declared. In the *run()* procedure of the plug-in code, the system time (*t1*) is taken using the *time()* function just before

[h]



(a)



(b)

(c)

(d)

Figure 6.1: Results for geometric (a-b) and stochastic textures (c-d) showing the source image on the left and the completed image on the right with reconstructed region outlined.

the *fastApprox()* procedure is called and is taken again (*t2*) after ultimate completion has been determined. The difference between t1 and t2 is the number of seconds elapsed since the start of the whole process until the image is completed. The time is taken as follows:

```
time_t t1,t2;
(void) time(&t1);
```

The size of the image has a significant impact on the speed of the completion process since certain functions such as the subtraction, multiplication and search functions iterate through every pixel in the image. The size of the gap to be filled also affects the speed dramatically since the number of unknown pixels determines how many times the image needs to be blurred and scaled to achieve the fast approximation result and how many times the search and composite procedures are called. Factors which affect the speed of the fast approximation procedure specifically are the size of the blur radius (how far the boundary colours creep into the unknown region on each iteration) and the number of pyramid levels to be generated on each iteration. These parameters are adjustable on the user interface to avoid having to recompile the code for every combination. The size of the neighbourhood pixel region (also adjustable on the user interface) directly affects the speed of the search procedure since the greater the neighbourhood region, the more pixels need to be iterated through to look for appropriate matches, but the fewer times the search needs to be called since a larger area of the missing region is filled in on every loop. A larger search region will also increase the processing time.

In order to determine the extent of these effects, images of a variety of sizes with the same size of missing region and images of the same size but with varying size gaps and other parameters are tested. The time taken to complete each image and the number of search iterations required to fill in the missing region is recorded as well as the values of the blur radius, pyramid level and neighbourhood region parameters.

Figure 6.2 shows the completion results for the same image using various parameter settings. To ensure some consistency, the pyramid level was set to four for each image and the search area is set to a 40 x 40 pixel block. The first two rows of images show the completion results for the image when a region of size 30 x 35 pixels has been reconstructed. The third row shows a smaller version of the image, but with the same gap size of 30 x 35 pixels, while the fourth row shows the results for an image with a gap size of half that of the other, 15 x 35 pixels. Table 6.5 summarizes the processing information for results in Figure 6.2. This information indicates that the size of the region to be reconstructed has more of an impact on processing time than the size of the entire image. However, the factor which has the most significant impact on processing time is the size

| *Image* | *Size (pixels)* | *N* | *Blur radius* | *# of Iterations* | *Time (mins)* |
|---|---|---|---|---|---|
| 6.2(a) | 160 x 120 | 4 | 10 | 369 | 46:18 |
| 6.2(b) | 160 x 120 | 4 | 5 | 384 | 56:55 |
| 6.2(c) | 160 x 120 | 8 | 10 | 69 | 12:56 |
| 6.2(d) | 133 x 100 | 4 | 10 | 406 | 37:30 |
| 6.2(e) | 160 x 120 | 4 | 10 | 178 | 22:20 |
| 6.3(b) | 160 x 120 | 6 | 5 | 145 | 22:33 |

Table 6.1: Results for images in Figures 6.2 and 6.3. N indicates the size of the neighbourhood region.

of the neighbourhood region as shown by the results for Figure 6.2(c) where only doubling the neighbourhood size meant that the image took less than a quarter of the time to process than for Figure 6.2(a). The graph in Figure 6.4 illustrates how increasing the neighbourhood size leads to a dramatic decrease in processing time up until a radius of five pixels, then the speedup tapers off until it reaches a plateau at a radius of 11 pixels. The algorithm is unable to complete images where the neighbourhood pixel region has a radius of less than three pixels.

The results also show that a neighbourhood size that is too small compared to the blur radius can lead to unwanted repetition of artifacts such as in Figures 6.2(d) and (e). The smaller the unknown region, the smaller the blur radius and neighbourhood region can be. Figure 6.3(a) shows the original image before areas are removed and compares it to a reconstructed image with a more optimum balance between the blur radius and neighbourhood size parameters (b).

## 6.3 Summary

The results discussed above indicate that the image completion plug-in produces the best results for images where the region to be reconstructed is made up of stochastic texture that occurs often in the surrounding region. The plug-in is effective at making the reconstructed area blend into the surrounding regions and follows the general structure of objects well although any 3-Dimensional structure is not taken into account. Geometrical shapes may not always be completed as expected since the algorithm will complete the image according to the direction the line of the shape was following at the edge of the gap, not according to mathematical principles. The parameters that are adjustable on the user interface determine the accuracy and efficiency with which the image is completed and their values depend of the size of the gap and the kind of texture in the picture. In general, smaller gaps and more structured textures need a smaller blur radius and smaller

(a)

(b) (c)

(d)

(e)

Figure 6.2: Completion results for a variety of parameter settings: First two rows: gap size 30 x 35 pixels (a) Large blur radius, small neighbourhood size. (b) Smaller blur radius. (c) Large blur radius, larger neighbourhood size. Third row: (d) Smaller version of image with gap still 30 x 35 pixels, blur radius and neighbourhood size as in (a). Fourth row: (e) smaller gap size of 15 x 35 pixels.

(a)

Figure 6.3: (a) Original image. (b) Best reconstruction of missing area (outlined by blue square).

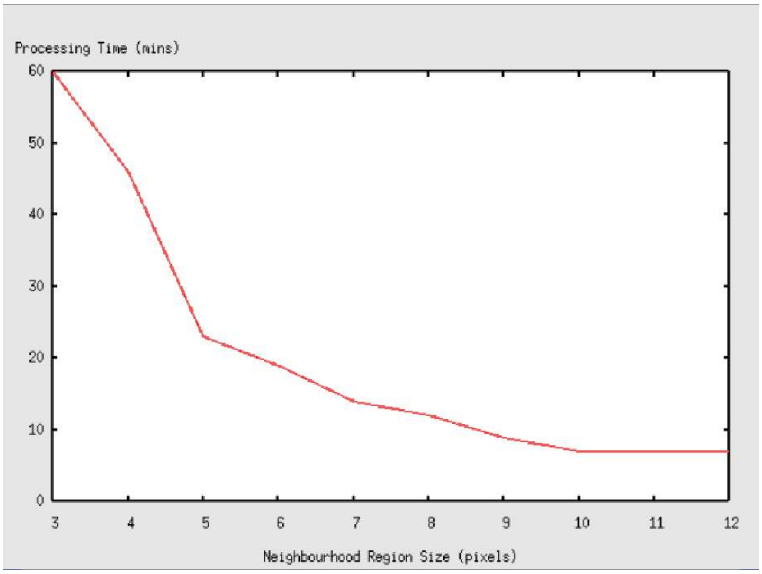



Figure 6.4: Impact of Neighbourhood Region Size on Image Completion Time

[h]

(a)


(b)

(c)

Figure 6.5: Completion results for photographs.

neighbourhood region. Large, smooth or stochastic regions can be accurately completed using a larger neighbourhood region which also decreases processing time.

The results differ slightly from those achieved by the original authors [Drori et al. 2003] because the algorithm has been adapted to suit implementation in the GIMP. The simplifications in the search algorithm in particular mean than the plug-in is less accurate at completing regions where objects are of different scales or rotations. The processing time for the images is also longer than in the original paper because of the different implementation and use of many external GIMP function calls which were used to simplify the implementation.

# Chapter 7

# Conclusion

The fragment-based image completion algorithm [Drori et al. 2003], upon which the image completion plug-in is based, is at the forefront of current image manipulation and restoration techniques. It describes the process of automatically reconstructing the missing portions of images once objects or defects have been removed in a photographically realistic manner. Incorporating the image completion algorithm into the GIMP program makes this innovative technique accessible to the ordinary computer user and easy to use in conjunction with any other image manipulation tools available in the GIMP.

## 7.1 Future Work and Possible Extensions

There is much scope for improving both the efficiency and accuracy of the image-completion plug-in in the future. In terms of design, the search procedure can be extended to search not only over all positions $x$ and $y$ but also over different scales and rotations of the image. This would improve the way the plug-in deals with textures that recede into the background or patterns which repeat at different rotations. Implementation-wise there are still ways to optimize the code to speed up the processing time which is an all important factor in the use of the plug-in, especially as far as using it as part of a image manipulation package such as the GIMP is concerned. In terms of extending the usability of the image completion plug-in in the GIMP, added features such as a preview window for results on the user interface would bring it more in line with the standards of the plug-ins that are packaged in the release versions of the GIMP. Making the plug-in work in a Windows environment is another goal.

The image completion algorithm also has the potential to be extended to other applications such as film editing and special effects. By extrapolating the process of image completion over

a series of frames, even moving objects could be convincingly removed. The possibility of searching over previous or future frames could further enhance the realism of the completion results. Another possible application for this type of algorithm is for video compression to speed up transmission since only partial images would need to be transmitted as long as they could be reconstructed timeously at the destination.

## 7.2 Conclusion

Plug-in development in general for the GIMP is relatively easy and well supported due to the availability of templates, documentation regarding GIMP-specific functions and an active developers' mailing list. While the GIMP function library provides a plethora of useful functions to access the toolbox facilities and manipulate the image components, for the image completion plug-in there are certain common operations such as multiplying and adding images that required custom functions to be coded.

Certain aspects of plug-in development are also not necessarily obvious to the novice developer and for this reason the on-line tutorial, "Writing a Plug-in for the GIMP in C" was developed alongside the actual image completion plug-in. This tutorial is a general guide to getting started and also lists other useful resources.

The results achieved by the image completion plug-in fulfill the aims of this project in that gaps in images can be automatically reconstructed in a realistic and convincing manner with minimal user input required. The user does have the option though of adjusting parameter values that affect the accuracy of results to suit the input image. The processing time taken to complete an image depends on the values of these parameters and the size of the image to be completed. The plug-in is able to reconstruct texture particularly well and also takes some sense of structure into account although it is not suited to the reproduction of geometric shapes.

It is not possible to re-implement the original paper exactly within the GIMP environment due to some of the constraints imposed by using the GIMP functions and the amount of processing time that is required for certain operations in the GIMP. We discuss a plausibly similar implementation however, which takes advantage of some of the built-in GIMP functions that are used to approximate the intentions of the original paper.

# References


[Bertalmio et al. 2000] Bertalmio, Marcelo, G. Sapiro, V. Caselles and C. Ballester (2000). Image inpainting. In *ACM SIGGRAPH* (2000), pp. 417–424.

[Bertalmio et al. 2004] Bertalmio, Marcelo, L. Vese, G. Sapiro and S. Osher (2004). Simultaneous structure and texture image inpainting. In *IEEE Conference on Computer Vision and Pattern Recognition* (2004).

[Brooks and Dodgson 2002] Brooks, Stephen and N. Dodgson (2002). Self-similarty based texture editing. In *ACM Transactions on Graphics* (2002), pp. 653–656.

[Budig 2002] Budig, Simon (2002). Script-fu and plug-ins for the gimp, 2002, <http://www.home.unix-ag.org/simon/gimp/guadec2002/gimp-plugin/html/>.

[Crispell 2003] Crispell, Daniel (2003). Fragment-based image completion. 2003.

[Drori et al. 2003] Drori, Iddo, D. Cohen-Or and H. Yeshurun (2003). Fragment-based image completion. In *ACM SIGGRAPH* (2003), pp. 303–312.

[Efros and Freeman 2001] Efros, Alexei and W. Freeman (2001). Image quilting for texture synthesis and transfer. In *ACM SIGGRAPH* (2001), pp. 341–346.

[Heeger and Bergen 1995] Heeger, David and J. Bergen (1995). Pyramid-based texture analysis/synthesis. In *ACM SIGGRAPH* (1995), pp. 229–338.

[Hertzmann et al. 2001] Hertzmann, Aaron, C. Jacobs, N. Oliver, B. Curless and D. Salesin (2001). Image analogies. In *ACM SIGGRAPH* (2001), pp. 327–340.

[Hirani and Totsuka 1996] Hirani, Anil and T. Totsuka (1996). Combining frequency and spacial domain information for fast interactive image noise removal. In *ACM SIGGRAPH* (1996), pp. 269–276.

[Igehy and Pereira 1997] Igehy, Homan and L. Pereira (1997). Image replacement through texture synthesis. In *IEEE international conference on Image processing* (1997), pp. 186–189.

[Masnou and Morel 1998] Masnou, Simon and J.-M. Morel (1998). Level-lines based disocclusion. In *IEEE International Conference on Image Processing* (1998), pp. 259–263.

[Neary 2004] Neary, David (2004). Writing a gimp plug-in, 2004, <http://dneary.free.fr/guadec-paper/paper.pdf>.

[Oh et al. 2001] Oh, Byong Mok, M. Chen, J. Dorsey and F. Durand (2001). Image based modelling and photo editing. In *ACM SIGGRAPH* (2001), pp. 433–442.

[Sharon et al. 2000] Sharon, Eitan, A. Brandt and R. Basri (2000). Completion energies and scale. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2000), pp. 1117–1131.

[Turner 1998] Turner, Kevin (1998). Writing a gimp plug-in, 1998, <http://www.gimp.org/docs/plug-in/plug-in.html>.

[Wei and Levoy 2000] Wei, Li-Yi and M. Levoy (2000). Fast texture synthesis using tree-structured vector quantization. In *ACM SIGGRAPH* (2000).

[Welsh et al. 2002] Welsh, Tomihisa, M. Ashikhmin and K. Mueller (2002). Transferring color to greyscale images. In *ACM Transactions on Graphics* (2002), pp. 277–280.

[Williams and Jacobs 1997] Williams, Lance and D. Jacobs (1997). Stochastic completion fields: A neural model of illusory contour shape and salience. In *Neural Computation* (1997), pp. 837–858.

# Appendix A

# Tutorial: Writing a plug-in for the GIMP in C


Figure A.1: Getting Started

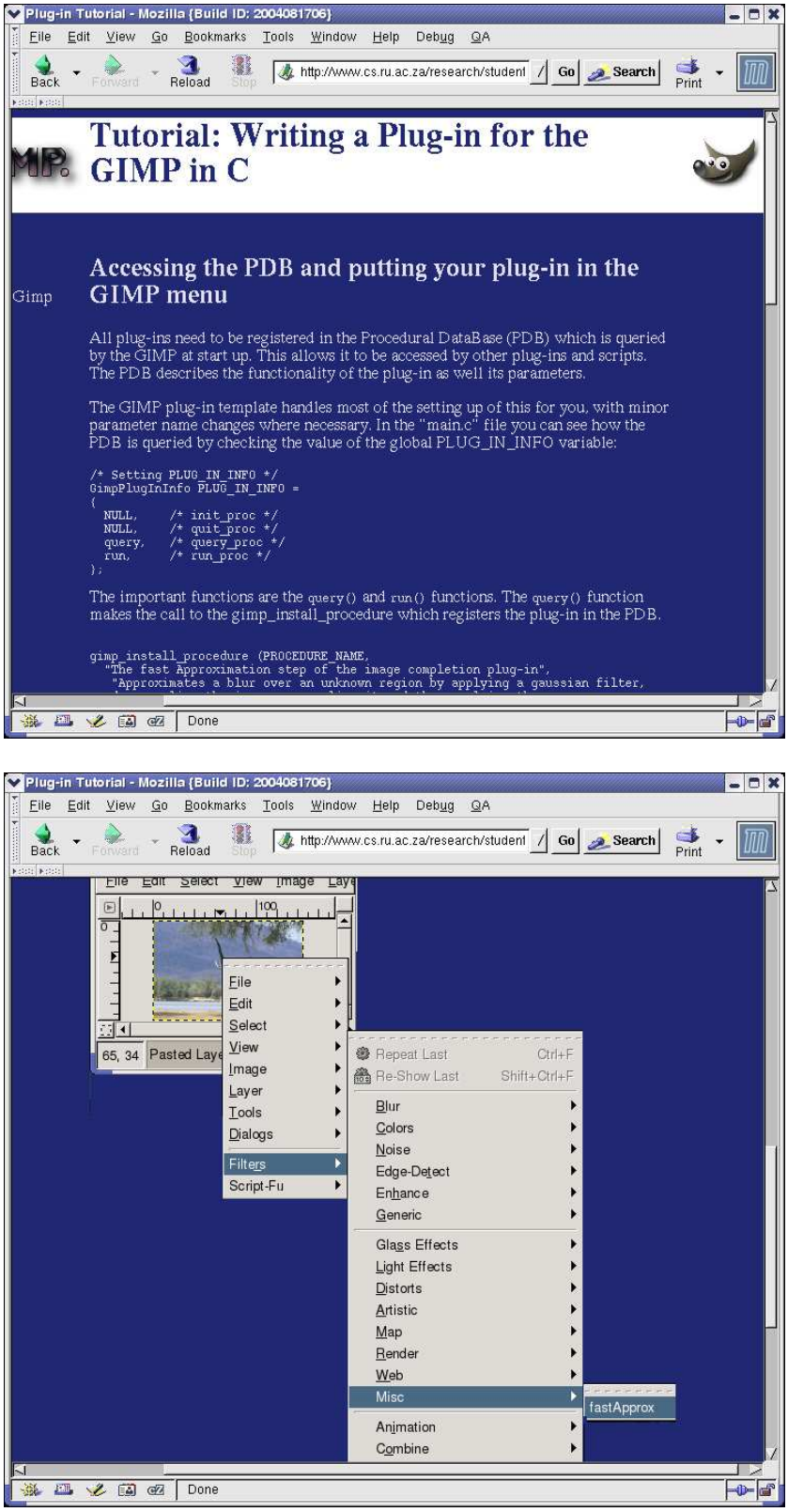
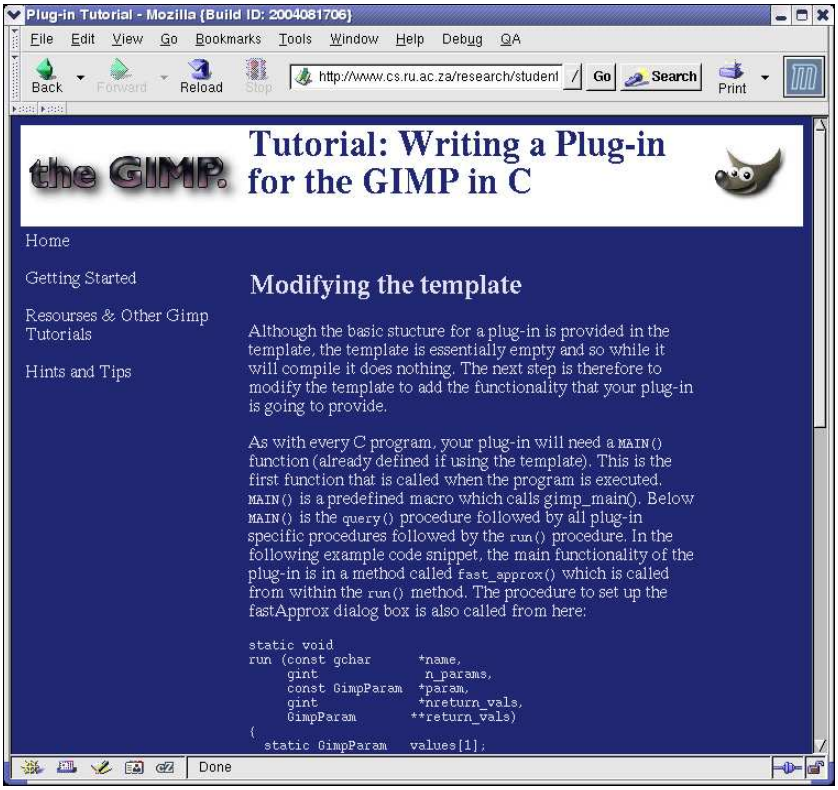



Figure A.2: Accessing the PDB and putting your plug-in in the GIMP menu



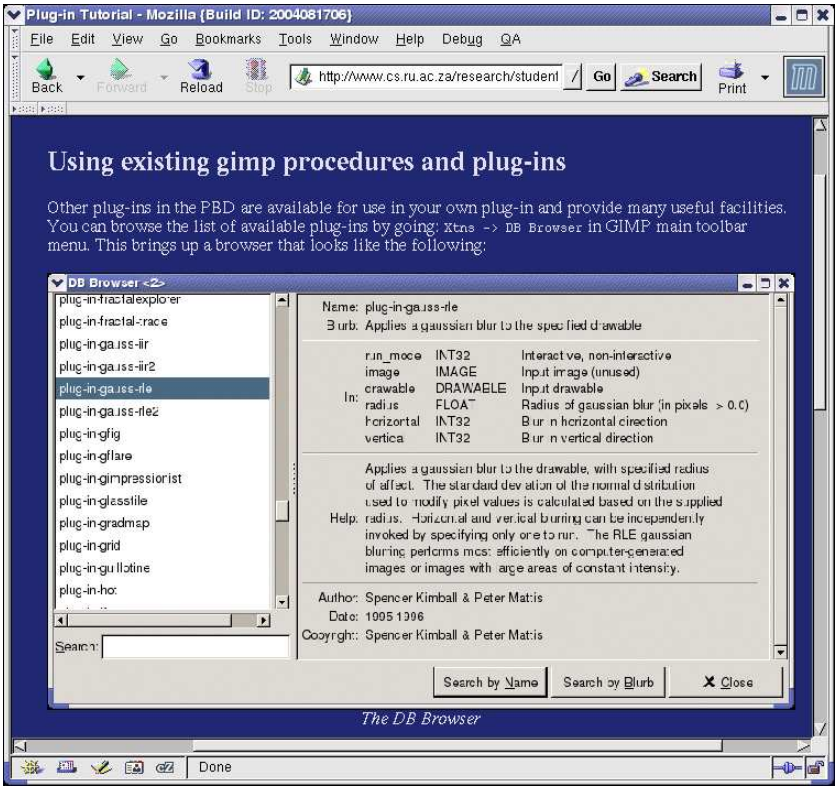




Figure A.3: Modifying the template and Using existing GIMP procedures and plug-ins

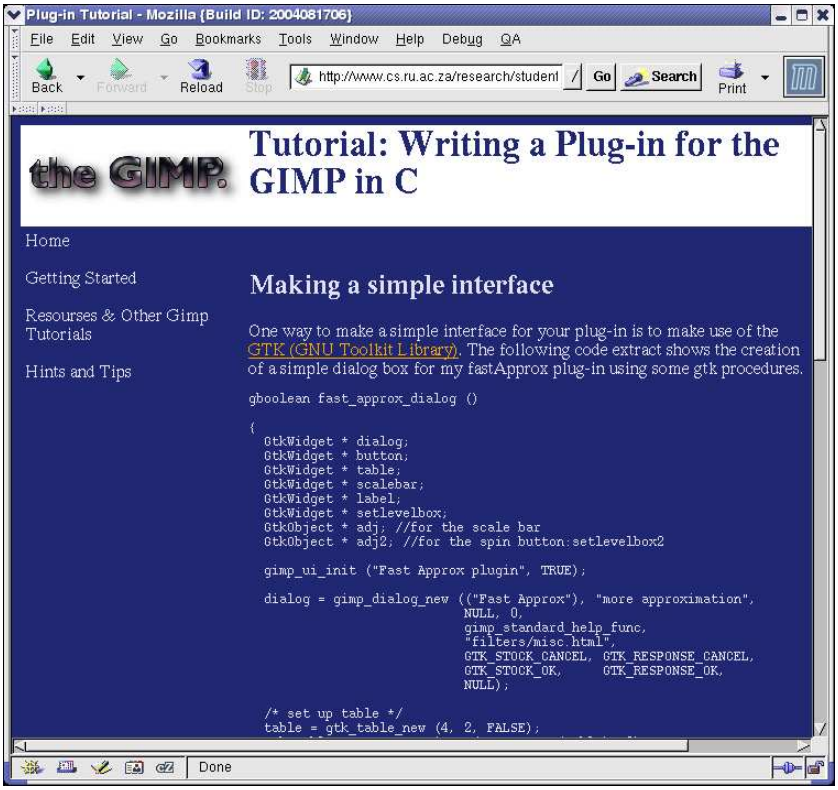



Figure A.4: Making a simple interface and Hints and tips page

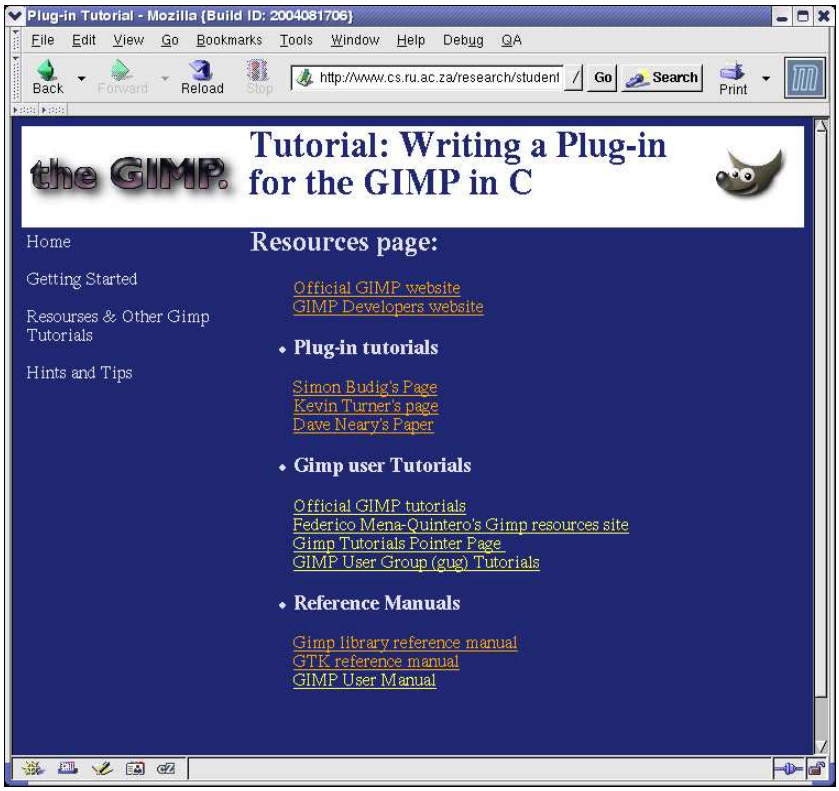



Figure A.5: List of Resources

# Appendix B

# Image Completion Results

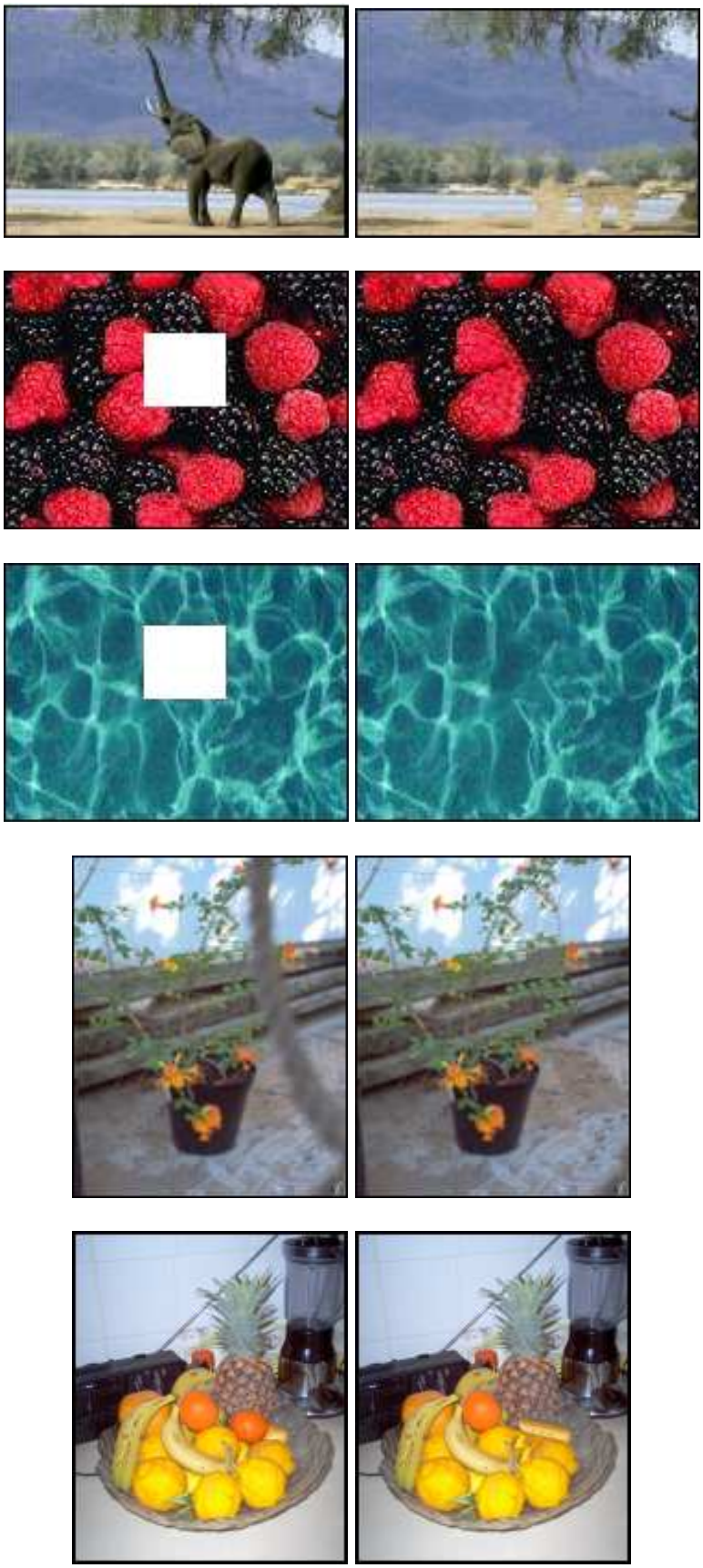


Figure B.1: Further image completion results