Pixel level data-dependent triangulation with its applications

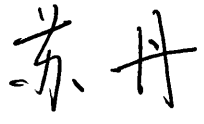
Su, Dan

Award date:
2003

Awarding institution:
University of Bath

PIXEL LEVEL DATA-DEPENDENT TRIANGULATION WITH ITS APPLICATIONS

Submitted by Dan Su
for the degree of Doctor of Philosophy
of the University of Bath
2003



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

Digital images are of great importance nowadays to life and technology. A good model to represent digital images is essential to digital image processing and other image-related applications. Many image models have been proposed from researching the underlying statistical and spatial features of digital image arrays. However most results are either complex or not adequate.

We propose a pixel level data-dependent triangulation image model, for a broad range of applications, using a triangulation mesh to represent images. The triangles are chosen so that image edges align with edges of the triangles; in particular the hypotenuse is selected to follow oblique edges. The strength of this model is that it represents the orientations of edges therefore keeps the most visually important feature of images. It is a generic model and is applicable to all types of images. It is very simple and efficient.

This has led to several important applications such as arbitrary resolution enhancement, arbitrary rotation, demosaicing of digital colour images and other applications of still images in continuous space. Moreover, the simplicity and efficiency of this model make it applicable in hardware which means real-time high quality image reconstruction and manipulations can be achieved.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor, Professor Philip Willis, for his encouragement, support, help and his useful insights into research. I especially appreciate his patience in guiding me into research.

I want to thank all the members in the Media Technology Research Centre at the University of Bath for their advice and help. My special thanks to John Collomosse, Dr. David Duke, Dr. Peter Hall, Dr. Joy Lu and Dr. Man Qi.

I would like to thank my family and my friends for their support. In particular, I would like to thank my parents for their love and support, my uncles for their encouragement and support, and my girlfriend Wenjia Liu for her love and support.

Declaration

The research presented in this thesis was conducted by the author.

All views expressed in this thesis are those of the author, and do not reflect those of the University of Bath.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Images are produced by a variety of physical devices, including still and video cameras, x-ray devices, electron microscopes and radar, and used for a variety of purposes, including scientific research entertainment, medical, business, industry, military, civil and security. The goal in each application is for an observer, human or machine, to extract useful information about the scene being imaged.

There are two principle types of images: continuous and discrete. A scene in the natural world is continuous. Image acquisition devices use a sampling process to digitise the image and thus convert the continuous image into a discrete digital form. This can be stored in a computer or on some form of storage media. The term "digital image" is commonly used for this data. It has intensity values only at a set of discrete points with intensities drawn from a set of discrete values. The discrete locations are often called "pixels" and the intensity of each location are often called "pixel value".

There is almost no area of technical endeavour that is not affected in some way by digital images. Their various applications raise the need to better represent, store, analyse and manipulate vast amounts of visual data. Most digital computers, as their names suggest, can only deal with discrete images. What we perceive in the real word are continuous images, which are always one step away from the data on which the computer operates. If we want a discrete image to be seen by human, we must display it continuously. This leads to a fundamental problem: reconstructing continuous images from discrete ones. In particular, how to return

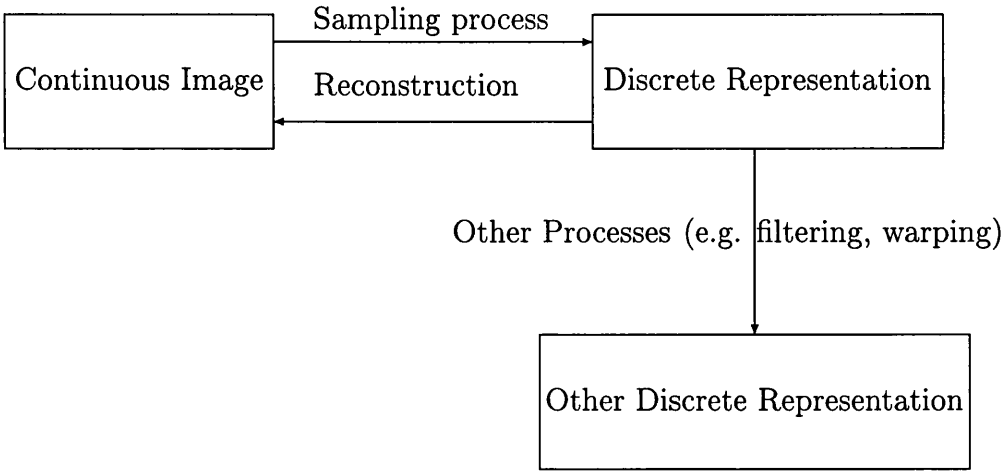


Figure 1.1: Digital image processing

from the discrete sampling data to a continuous space, thus escaping from the limitations of pixels and the usual discrete sampling.

We illustrate this process in Figure 1.1. Digital images have been taken from continuous images by the sampling process. Given the representation, we are able to perform other processes on it: e.g. filtering, warping and segmentation. Finally we can reconstruct continuous images from the discrete representation.

Developing an adequate tool to represent digital images therefore plays a critical role in image processing and other image-related applications. The representation of digital images clearly affects other image processing applications and the reconstruction of continuous images.

However this remains a challenging problem in image processing research. An image is not a direct measurement of the properties of physical objects being imaged. It is rather a complex interaction among several physical processes: the intensity and distribution of illumination, the physics of the interaction of illumination with the matter comprising the scene, the geometry of projection of the reflected or transmitted illumination and the electronic characteristics of the sensor. Moreover, intensities are recorded to finite precision which always leads to errors in the reconstructed continuous intensity. Representation of digital images should also help in reducing the reconstruction errors.

A lot of researchers have studied the field of finding a suitable image model which provides an abstraction of large amounts of image data and provides an analytical

representation for explaining the image's intensity distribution.

The simplest and most popular image model can be represented as a two-dimensional light-intensity function, denoted by $f(x, y)$, where the value or amplitude of f at spatial coordinates (x, y) gives the intensity of the image at that point. This model is simple; however it ignores almost all the interactions among physical processes and the geometry features of the images.

Spatial features, especially edges, are particular important in image processing. Edges contain very important information about the image. When we perceive an object in our environment, the edges of an object play an important role in stimulating our vision system and conveying visual information to our mind. It is edges that help us recognise objects and reveal their structure. Preserving edges is important in almost all image processing tasks but it is difficult to find a good model that preserves the characteristics of edges effectively and efficiently.

Among the characteristics of edges, edge orientation is the most important one. However, edge orientation is an underlying parameter embedded in the 2D digital image data array. It is not straightforward to reveal edge orientations by dealing with image data directly. Moreover, edges have other two important spatial features: the intensities *across* the edge vary significantly but they are almost the same *along* the edge. These features makes edge play a significant role in digital image representations.

It is well known that humans are better at judgement and machines are better at measurement. Although it is easy for humans to distinguish the edges, edge detection remains a difficult problem for image processing. Edge detection has been an active research area in the past decades, however most existing algorithm either do not detect edges effectively or employ very complex models to describe edges. Moreover, the aim is not simply the detection of edges in the image, but the representation of the intensity distribution and the preservation of the underlying edge features.

We propose an image model using a data dependent triangulation (DDT) [27]: the triangulation mesh is dependent on both the geometry and the intensity distribution of the image data. The drawback of normal DDT schemes is that they are too complex and one can always find another triangulation of the same

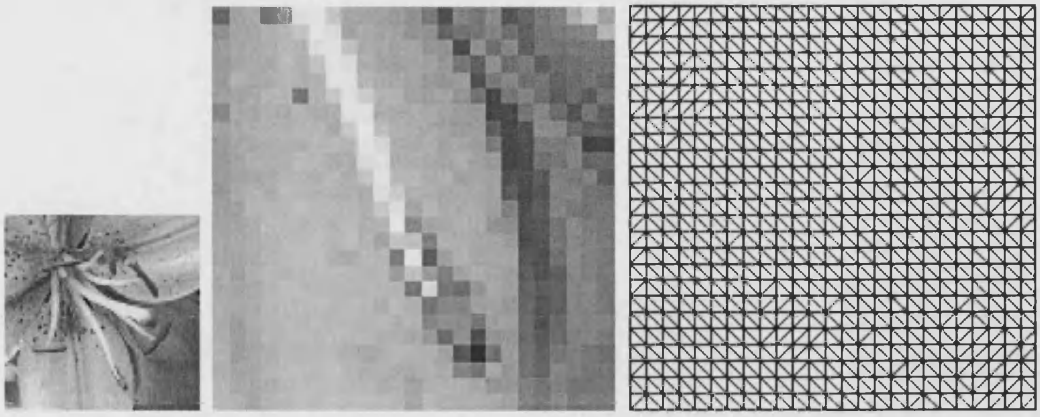


Figure 1.2: A sample image mesh DDT. Left: a flower image. Middle: a magnified view of the bottom stamen. Right: the pixel level data-dependent triangulation of the stamen

image and claim this triangulation is better. Another shortcoming of the normal DDT is that it cannot handle very small features of the image because it cannot triangulate at that level.

We suppose there is an edge passing through every square of four pixels in the image. Then we triangulate the four-pixel square and represent the edge by the diagonal of this square. A triangulation mesh is built by triangulating every four-pixel square in the image and the edges of the triangles are chosen to correspond to the edges in the image (Figure 1.2). We call this a *pixel level deta-dependent triangulation.*

This model allows us to perform other applications on this representation and to recover continuous intensities from discrete image data samples. Various applications can use this representation: such as arbitrary resolution enhancement, arbitrary rotation and other applications of still images in continuous space. We have studied its application in these image processing tasks and developed several novel algorithms for them.

1.1 Contributions

This thesis gives a solution to the image modelling problem. We provide a data dependent triangulation mesh to represent images. Unlike former approaches,

we do not assume knowledge of the low-pass filtering kernel, or attempt to find a statistical rule about the local geometry, or explicitly extract edge-orientation information: we model the image as a triangulation mesh with the edges of triangles corresponding to the edges of the images. It is a universal model for all images and provides universal solutions to many image processing problems. It is very simple to implement and efficient as well.

We have employed the model in several image related applications, i.e. image interpolation and demosaicing of colour images. Our model is very simple which makes it easy to adapt to various applications.

In particular, this model provides an image interpolation scheme which provides higher visual quality than traditional interpolation schemes. A statistical assessment also shows that this approach produces good overall image quality. The complexity of the new method is similar to bilinear interpolation and better than most existing methods. A hardware implementation shows that high-quality image interpolation can be done in real-time. The model is also extended and applied to the image demosaicing problem. By avoiding interpolation across edges, the new algorithm successfully solves the problem of colour artifacts around the edges. It provides a reasonable solution to the colour image demosaicing problem because it produces good reconstruction efficiently.

This thesis also studies the texture synthesis problem. We present a detailed survey of texture synthesis and introduce a new texture synthesis method using particle swarm optimisation for patch-based texture synthesis. We extend this method to texture transfer, constrained multi-sample texture synthesis and perspective texture synthesis.

1.2 Organisation

The rest of this thesis is organised as follows:

- **Chapter 2: Background** We will briefly review the history and the scientific background of image modelling problem. We also study the background

of other applications we will cover in this thesis, e.g. image interpolation, demosaicing of colour images and texture synthesis.

- **Chapter 3: Image Modelling** We start by describing the traditional and recent image modelling algorithms, especially those edge-directed schemes and the triangulation approaches. We introduce the motivation behind our work and importance of edges in image modelling. Then we introduce triangulation and present a novel pixel level data dependent triangulation approach for image modelling.

  Image modelling is concerned with developing an empirical model to represent the global features from the discrete sampling that has produced the image. In this thesis, the edge behaviour is a focus and so we use a data-dependent triangulation that aligns well with edges to represent digital images. Other applications of digital images can be developed from this model.

- **Chapter 4: Image Interpolation** We then present how our image model can be applied to image interpolation and how the results are improved by interpolation along the edge orientation. We will study one important and difficult application: the magnification of still images and some other applications such as rotation, the perspective transform and a non-uniform example, the lens effect. We assess our experimental results and simulate the potential of our algorithm to be used in hardware.

  Image interpolation is a particular instance of modelling where we create an empirical model that reconstructs the image values at any arbitrary position in a continuous space. Interpolation is performed according to the image model which represents the features of the underlying image, especially the edges. Image interpolation focuses on reconstructing continuous images from the digital samples while image modelling focuses on representing the features of the digital images.

- **Chapter 5: Demosaicing of Colour Images** In this chapter we will adapt our model and use it in the demosaicing of colour images generated by current single-chip digital cameras. We will demonstrate that our model is effective compared to traditional methods, when applied to the commonly-used Bayer Colour Filter Array pattern. We demonstrate that the proposed method gives superior reconstruction quality, with smaller visual defects than other methods.

Demosaicing is a particular application of interpolation where interpolation is performed on the demosaiced discrete data samples.

- **Chapter 6: Texture Synthesis** We first give a survey of traditional and contemporary texture synthesis methods. We will introduce a texture synthesis method based on patch-based sampling texture synthesis method. It uses Particle Swarm Optimisation for searching process thus accelerate synthesis while keeps high quality. It is simple and easy to implement compared to other acceleration schemes. Then we extended the basic method to texture transfer and constrained texture synthesis. It is effective for these applications. Then we extend the method to synthesising perspective textures from an input sample image. The method synthesises the texture directly on the surface, rather than synthesising a texture image and then mapping it to the surface.

- **Chapter 7: Conclusions and Future Work** We finally make our concluding remarks and provide an overview of future research orientation.

Chapter 2

Background

In this chapter we give the background to the image modelling problem. We also study the some other image models and briefly introduce our image model and solutions.

2.1 Image Modelling

Image modelling refers to the analytical representation of discrete image data and provides an explanation of the image's intensity distribution. Clearly an effective image model is important for applications based on this model. However, although digital image processing has been an active research field for over thirty years, image modelling is still very much an unsolved problem, in particular, effective and efficient modelling preserving the edge features of the image. Some solutions have been shown, however they are either very complex or are not effective for all images.

Previous image models can be classified into four categories.

- The *probabilistic models* treat the image as a statistical representation of numerical data taken from the image source according to various statistical distributions. One popular tool is the Discrete Markov Random Field

(MRF)[97] which models contextual information of digital images and specifies local characteristics of an image by conditional probability models. Several improvements based on MRF such as the doubly stochastic process [99] and the dual lattice process [31] have been proposed, aiming at better capture of the global statistics and nonstationarities of the image source.

- The *deterministic models* treat a digital image as a two-dimensional data matrix of discrete samples and seek global geometry features of the image. Deterministic 2D sinusoidal models [50], polynomial models [15] and the recently-proposed computed AM-FM models [35] all try to catch the global features of images. These models are appropriate for a specific subset of images (e.g. highly structured images) but not for images containing complex structures (e.g. textures with lots of edges). The alternative local models can be thought of as a 3D extension of time-series models for images (e.g. 3D Casual [55], NSHP [98]). Another example is the PDE-based model [64] which is used in nonlinear diffusion and also finds promising applications in image enhancement and restoration [7, 52].

- The *wavelet-based models* have an energy compaction property, in both the space and frequency domain. They facilitate the task of statistical modelling of the image source. Current wavelet based models including a classification strategy [51, 105, 39] to distinguish coefficients around edges from those in smooth regions. A nonlinear approximation [4] is superior to linear approximation. The statistical inference approach [48] determines the image's characteristics and estimates edge orientation by a Least-Square estimation strategy.

- The *triangulation models* [25, 106] model the image as a data *independent* triangulation or a data *dependent* triangulation (DDT) to represent the image source. The data independent triangulation depends only on the distribution of the data points while the latter depends on the data values as well.

We introduced our new image model - the *image mesh DDT*. We represent the image as a data dependent triangulation mesh with the diagonals of the mesh corresponding to the edges in the image. In particular, we divide each four pixel square into two triangles by the diagonal: the diagonal either goes to the NE-SW or NW-SE direction. The direction of the diagonal is chosen to correspond to the

edge in the image. We also extend our model by considering the local intensity instead of only each four-pixel square. Our triangulation mesh is very simple and completely regular. We avoid the complexity of a full DDT method while keeping the advantage of DDT that improves the reconstruction quality.

2.2 Image Interpolation

Image interpolation is a link between the discrete world and the continuous one by recovering the continuous intensity surface from discrete image data samples. It is a well-studied area in computer graphics and image processing but it remains a challenging problem.

Previous image interpolation methods can be classified into three categories:

- The *classic* methods including nearest-neighbour, linear interpolation (bilinear in 2D or trilinear in 3D) [81], bicubic [59] and cubic B-spline [84]. These methods are widely used in computer software. Typically they are implemented through convolution of the image samples with a single kernel. They suffer from edge blurring or artifacts along the edges.

- The *advanced* methods have been proposed to improve the interpolation quality. There are many directions of these approaches. PDE-based approaches [9, 57] apply a nonlinear diffusion process controlled by the local gradient. POCS (Projection-Onto-Convex-Set) schemes[70] formulate the interpolation as an ill-posed inverse problem and solve it by regularised iterative projection. Orthogonal transform methods focus on the use of the discrete cosine transform (DCT) [56, 77]. Directional methods [12, 37] examine an image's local structure around edge areas to direct the interpolation. Variational methods formulate the interpolation as the constrained minimisation of a functional [41, 75]. Adaptive interpolations [5, 45, 19] spatially adapt the interpolation to better match the local structure around edge area.

- The recently developed *edge-directed* methods improve interpolated image quality by taking edge information into account [48, 6, 11, 57, 58]. The edge directed interpolation (EDI) [6] generates a high resolution edge map

and uses it to direct high-resolution interpolation. The New Edge Directed Interpolation (NEDI) attempted to estimate local covariance characteristics at low resolution and used them to direct interpolation at high resolution. Battiato et al. [11] proposed a method taking into account information about discontinuities or sharp luminance variations while doing the interpolation. Morse et al. [57, 58] presented a scheme that uses existing interpolation techniques as an initial approximation and then iteratively reconstructs the isophotes using constrained smoothing.

We have used our model for the image interpolation problem. In particular, we are interested in still image magnification. We used our model to generate a triangulation mesh to represent this image. Then higher resolution images are interpolated from this mesh. In particular, a higher resolution pixel will be interpolated from a triangle rather than from a four pixel square (as bilinear interpolation does) or from an even bigger window as (bicubic interpolation does). Because the triangulation mesh is generated corresponding to the edges of the image, the interpolator will always interpolate along the edge but not across it. The new method will keep the edge sharp while retaining the smoothness along the edge. We will compare our results with traditional methods and assess them both visually and statistically. Our algorithm provides good image quality with almost the same computational efficiency as bilinear interpolation.

2.3 Colour Image Demosaicing

One very important industry problem is the reconstruction of a full-resolution colour image from the *mosaiced* sample taken by current single-chip colour digital cameras. The digital cameras acquire images through a colour filter array which leads to mosaiced images: only one primary colour (R, G or B) in one pixel. The so-called "demosaicing" process is thus needed to interpolate full colour images.

Previous demosaicing approaches can be classified into two categories:

- The *colour correlation methods* address the problem introduced by traditional methods such as bilinear interpolation. This induces relatively large

errors in the edge regions and the eye is especially sensitive to edge quality [73]. Various methods [43, 2, 61] use colour correlation instead of original colour space. They produce better results because there is a high correlation between the red, green and blue channels. However, they ignore the edge orientation in the images.

- The *edge-directed* approaches attempt to maintain edge details or limit hue transitions. Adams' edge oriented method [1] interpolates the missing colour elements according to the edge orientation of the image but it only detects the vertical and horizontal edges. Ramanath [68] used an adaptive interpolation, achieving edge orientation adaptation. Cok [20] proposed a method using a constant hue-based interpolation to make sure there are no sudden jumps in hue, especially over edges. The median-based interpolation [29] proposed by Freeman first does a linear interpolation and then applies a median filter of the colour differences (red-minus-green and blue-minus-green channels). Laroche and Prescott [44] proposed a method called *gradient based interpolation* and it is used in the Kodak DCS 200 digital camera system. Hamilton and Adams [32] used an adaptive colour plane interpolation which is a modification of the method by Laroche and Prescott [44]. However, all these methods are complicated and computationally slow.

We adjusted our model and applied it to the demosaicing problem. Moreover, we used colour-difference space [61] instead of original space as the former better explains the correlation between different colour channels. We demonstrated that our model is effective compared to traditional methods, when applied to the commonly-used Bayer Colour Filter Array pattern [1]. Our experimental results show that the proposed method gives superior reconstruction quality, with smaller visual defects than other methods. Furthermore, the complexity and efficiency of the proposed method is very close to simple bilinear interpolation, making it easy to implement and fast to run.

2.4 Texture Synthesis

Texture is a particular problem. Texture has an excessive number of edges which makes it difficult for normal triangulation approaches. Texture also has some

particular applications other than magnification and rotation. Normally in computer graphics and image processing, texture is mapped on the object surface to make it more realistic. These features make texture a special situation. An alternative way to represent textures in continuous space is to synthesise a matching texture, using relatively few parameters without much expense of computation time. The synthesised texture should have the same spatial feature as the sample texture. It is thus possible to use the synthesised texture, with arbitrary size and orientation, to map the object surface.

Previous texture synthesis methods can be classified into several categories:

- The *procedural* texture synthesis is the use of a function or set of functions applied to a set of points in order to generate a texture [110]. Solid texture [90, 60, 47] textures the surface by 'placing' the object in the field, and obtaining a texture from the intersection of the surface of the object and the field. Hypertexture [63] used a density function that describes how the object should behave in the area where it transitions between the outside and inside of the object. Cellular Texture [100] used a new basis function to produce textured surfaces resembling flagstone-like tiled areas. The Reaction-Diffusion [82] approach is based on a process in which two or more chemicals diffuse over a surface and react with one another to produce stable patterns.

- The *Feature Matching* texture synthesis uses models such as pyramids and wavelets to catch the features of the texture and then generates a new image by matching the model. Examples are the steerable pyramid introduced by Heeger and Bergen [36] and the multi-resolution filter-based approach by De Bonet [16]. They use the pyramid to analysis input texture and to catch spatial features of textures. Wavelet approaches [65] are used to model textures by decomposing the texture image to complex wavelets and synthesising new textures by matching the joint statistics of these wavelets.

- The *Markov Random Field (MRF)* approaches assume that a texture is "local" and "stationary". They estimate the local conditional probability density function (PDF) and synthesise pixels incrementally. The non-parametric sampling scheme by Efros [26] models each pixel of the image as a square window around that pixel and then synthesises the next pixel

by searching for the best-fit (least error) pixel through matching the neighbouring window. Wei and Levoy [93] improved Efros's method by using a multiresolution image pyramid. These methods synthesised one pixel at a time. Xu et al. [102] proposed a texture synthesis algorithm based on random patch pasting. This idea is developed and modified by other researchers, for example, the patch-based sampling method [49] and the image quilting approach [27]. They synthesise new texture one patch at a time thus are much faster.

- The *surface texture synthesis* directly synthesises textures on 3D surfaces. Wei [94] presented a method to synthesise general textures over arbitrary manifold surfaces by generalising the definition of searching neighbourhoods. Turk [83] developed an algorithm by sorting a hierarchy of points from low to high density over a given surface and searching for the best-fit pixel from the set.

Liang et al. [49] produced a real-time synthesis process by patch-based sampling. It searches all patches from the sample texture and picks a best match patch to generate new texture. It avoids mismatching features across patch boundaries by sampling texture patches according to the local conditional MRF density. Liang's method can re-synthesise high-quality texture images in real-time. It remains effective when pixel-based sampling algorithms fail to produce good results. It uses feathering blending in the boundary zones, thus providing a smooth transition between adjacent texture patches.

Yan Zhang proposed a texture synthesis method [108] based on patch-based sampling method. It uses Particle Swarm Optimisation for searching the best match patches; this accelerates the synthesis process and keeps the synthesis quality. The PSO algorithm either gives a best match or an approximate best match which is good for texture synthesis because texture synthesis needs some randomness and the synthesised texture should 'look like' the original one. Zhang's method is simple, easy to implement and more efficient compared to the techniques used in Liang's method.

Xiaogang Xu [103] did some research on texture transfer based on Ashikhmin's method [8]. However his method is synthesising textures one pixel at a time thus very computationally expensive. We extend the PSO based method to texture

transfer and visual inspection shows that our method produces better results and is much more efficient.

Constrained texture synthesis is also studied by other researchers [8]. However most of them only take one sample texture and synthesise textures on constrained areas from that sample texture. We have extended PSO based texture synthesis to multi-sample constrained texture synthesis. We take several sample textures and one picture as input and then synthesise textures from different sample textures into different areas of the picture.

Most previous methods synthesise textures on 2D plane or 3D surfaces. Consider there is a 2D plane in perspective view (a 2D surface viewed in 3D), synthesis methods on 2D plane cannot synthesise textures on this surface. Those methods [94, 83, 104] which synthesise texture on 3D surfaces are too complex for this situation because they involve 3D mesh generation and calculation. We present a new method which is derived from the PSO texture synthesis. We adjusted his method to be used in perspective projection. This application leads to some new problems such as aliasing, which we also address. It produces textures with similar quality and speed to the 2D counterpart (Efros's image quilting algorithm). The algorithm is effective and efficient, producing high quality synthesised images very rapidly.

Chapter 3

Image Modelling

3.1 Introduction

Digital images now routinely convey information in most branches of science and technology. There is almost no area of technical endeavour that is not impacted in some way by digital images. A digital image is composed of a finite number of elements, each of which has a particular location and value. The most obvious feature of digital images, considered as numerical data, is their very large size. A natural image with normal resolution and size consists of millions of pixels and each pixel takes one of hundreds of different values. Another key feature of image data is its spatial structure. The interplay between stochastic digital data and spatial variations is one of the most interesting aspects of digital image processing. To construct a model of digital images to represent the features from such a huge database can be a considerable challenge.

Image modelling, as defined in this thesis, is a general phrase which can be applied to all of the following aspects:

- It provides an abstraction of large amounts of data contained in images.
- It refers to analytical representation for explaining the image's intensity distribution.

- It facilitates the development of systematic algorithms to accomplish specific tasks.

However, the representation of an image is very much an unsolved problem in image processing. Finding a model that is able to accurately characterise and represent images is very difficult because of the huge number of images and their countless spatial structures. Over the past two decades, a lot of research has been done within this field and image models have gone through several phases of improvements.

Traditional image modelling techniques can be put into two categories: probabilistic models and deterministic models.

The probabilistic approaches treat the image as a statistical representation of numerical data taken from the image source. Probabilistic image models suppose the image is spatially stationary, which makes implementation simple: effectively the same statistical rules and computations are performed repeatedly at different locations in the image [48]. The multivariate Gaussian model is one of the earliest approaches to represent the global statistics of an image source. Another popular tool is the Discrete Markov Random Field (MRF)[97]. It models contextual information of digital images and specifies local characteristics of an image by conditional probability models. Several improvements based on MRF such as the doubly stochastic process [99] and the dual lattice process [31] have been proposed, aiming at better capture of the global statistics and nonstationarities of the image source. MRF theory has been applied to many applications in image segmentation [24] and restoration [99].

The deterministic methods treat an image as a two-dimensional data matrix of discrete samples taken from a two-dimensional continuous space. Instead of trying to find the statistical rules from the image source, deterministic methods seek the global geometry features of the image. Deterministic 2D sinusoidal models [50], polynomial models [15] and the recently-proposed computed AM-FM models [35] all try to catch the global features of images. These models are appropriate for a specific subset of images (e.g. highly structured images) but not for images containing complex structures (e.g. textures with lots of edges). The alternative local models can be thought of as a 3D extension of time-series models for images (e.g. 3D Casual [55], NSHP [98]). Another example is the PDE-based model

[64] which is used in nonlinear diffusion and also finds promising applications in image enhancement and restoration [7, 52].

Recent development of image modelling is mainly due to the discovery of bases suitable to represent the characteristics of images. One class of models known as fractals [10] is good for describing natural scenes with a large amount of self-similarity. Fractal models have found many successful applications in image synthesis, compression and analysis. Another popular class of models is the wavelet-based representation [53, 88] of images. These approaches model the image features by wavelet transforms in both the space and the frequency domain. Current statistical modelling techniques in the wavelet domain mostly use a classification strategy [51, 105, 39] to distinguish coefficients around edges from those in smooth regions. Deterministic models in the wavelet domain mainly focus on using functions in Besov space to describe the behaviour of wavelet coefficients. Nonlinear approximation [4, 21] has also been found to be superior to linear approximation. The statistical inference approach [48] determines the image's characteristics and estimate edge orientation by a Least-Square estimation strategy. Another important class of models uses a triangulation mesh. Given a set of data points and corresponding data values, the data *independent* triangulation methods [13, 78] build a triangulation mesh to represent the image source depending only on the distribution of the data points. In the data *dependent* triangulation approaches [25, 106], the triangulating of the image depends on the data values as well.

A lot of effort has been put into the area of image modelling but it remains a difficult and unsolved problem. Effective and efficient image models should capture fundamental features of digital images and should be fairly easy to implement compared to the traditional and widely used bilinear interpolation. In particular, they should cope with edges of images well because, as we will explain later in this section, edges are fundamental features in the image source and contain lots of information. Although several approaches based on geometric or statistical analysis have been proposed, these edge models are either very complex or can only handle vertical or horizontal edges. In this chapter, we will present a novel approach to representing images which achieves edge orientation adaptation by modelling the image as a data dependent triangulation.

3.2 Pixel Level Data-Dependent Triangulation

3.2.1 Introduction

The need for triangulation arises in a wide variety of applications ranging from physics to meteorology, from mathematics to computer graphics. It is used in robotics to plan the motion of a robot. Triangulation is also used in computer vision to present stereo data. It is useful in rendering images because current graphics cards use triangles as primitives and can draw triangles rather efficiently. It is widely used to model surface geometry. Significant theoretical advances in using triangulations for geometric modelling have been made. It is a well-studied problem in computer graphics [89].

Before we move on, we define some notation.

P represents a set of data points in the xy plane, V represents a set of data values in which each element is the data value of a point in set P. T means a triangulation which is a set of triangles with each of them consisting of three points from P.

The triangulation shall satisfy the following conditions:

- P is the set of all vertexes of triangles in T. That means every triangle vertex is an element of P and vice versa.
- Every edge of a triangle in T contains only two points from P
- The union of all triangles in T is the convex hull of P.
- The intersection of any two different triangles in T is either empty, or is a shared edge or vertex.

Triangulation can be classified into two categories: data independent triangulation and data dependent triangulation according to whether or not its topology is determined by V.

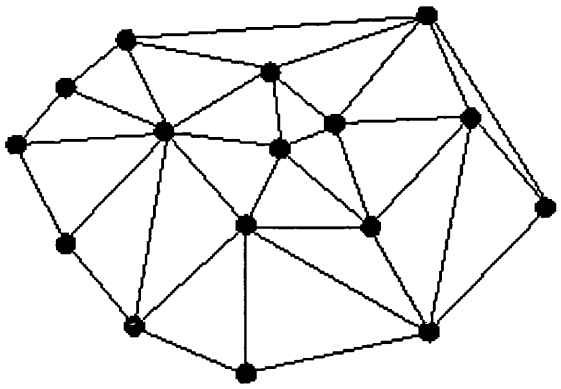


Figure 3.1: A Delaunay triangulation

The most popular data independent triangulation method, a method which only considers the data points' domain positions P, is the *Delaunay Triangulation* [72, 74]. A Delaunay triangulation T is such that for each triangle t , there is no vertex of T in the interior of $t's$ circumcircle (Figure 3.1). An optimal Delaunay triangulation means the triangles have good aspect ratios and the triangles should be as equiangular as possible, hence avoiding 'sliver' (very thin) triangles. Sliver triangles are undesirable for many applications such as Finite Element Modelling and graphical rendering since their shape can cause numerical inaccuracies in FEM calculations and visual discontinuities in smoothly shaded surfaces. In many applications, the Delaunay triangulation is useful. However, Delaunay triangulations do not necessarily produce the triangulation which is the best approximation to a given surface, for two reasons:

1. sliver triangles are necessary to give a good approximation to some surfaces (Figure 3.2) [72];

2. the swapping of edges which the Delaunay criterion invokes can cause artificial break lines where none exist in the original terrain (Figure 3.3) [74].

As its name suggests, data dependent triangulation also depends on the underlying data set V as well as the positions P. Data dependent triangulation will provide a better approximation to the underlying surface V [25, 106]. Data dependent triangulations sometimes produce sliver triangles, with their long side in the direction of small curvature. Although regarded as 'bad' for interpolation, sliver triangles are *good* for approximating a preferred direction, as we have seen in Figure 3.2. Many data dependent triangulation approaches have been shown

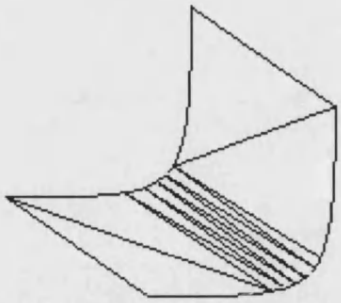


Figure 3.2: Silver triangles are required to represent this surface

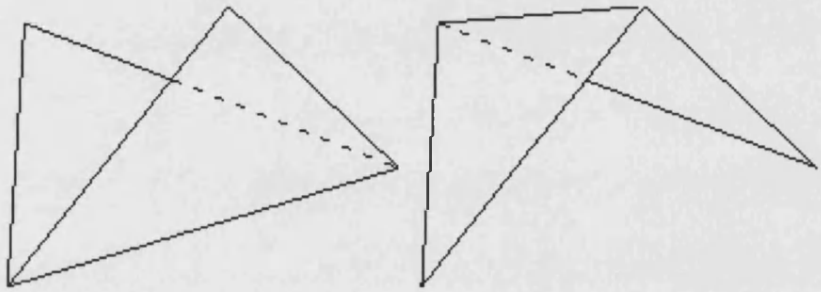


Figure 3.3: An edge swap in a quadrilateral. The representation of the geometry will be affected by such an edge swap

in the last two decades [71, 17, 76, 67] in a variety of applications. It is intuitively clear, and supported by their work that the interpolation over such a data dependent triangulation will provide a better approximation to the underlying surface V. Thus, the problem of adapting the shapes of the triangles to the behaviour of the underlying data set is important and proved difficult. Those triangulation approaches normally use optimisation procedures (e.g. Lawson's local optimisation [25]) in order to produce a better approximation of the function V. A good data dependent triangulation of a given function V over a given set P will optimise some quality, which can be referred to as "smoothness". The smoothest triangulation will usually differ for two different underlying functions over the same set of points P while the data *independent* triangulation will always produce the same triangulation over the same set P. Different optimal algorithms have also been proposed in order to yield a better (smoother) triangulation mesh [25, 106]. However they are all complex and no perfect optimal algorithm has been found for all functions.

For an image, if we let P correspond to the set of pixels and $V(x, y)$ be the

Figure 3.4: An edge image

intensity of the pixel (x, y), we can then model the image as a triangulation mesh. It is easy to conclude that data dependent triangulation is desired because data dependent triangulation is a better representation of given function (image intensity) over a given set (image pixels). It can be seen that the data dependent triangulation will better represent the intensity distribution of the image. This gives us a hint that data dependent triangulation might be a good solution for the image modelling as it represent the intensity distributions well. In particular, the edges of the images can be well preserves by the data-dependent triangulation as we can swap the triangles according the edges to force the triangles align well with edges.

In the following section, we will study the geometry of the edges and present how to model edges by a data-dependent triangulation mesh.

3.2.2 Edge Modelling

In this section we look at the concept of digital edges and their geometry features a little closer. Intuitively, an edge is a set of connected pixels that lie on the boundary between two regions. An edge is a purely 'local' concept whereas a region boundary is a more global variant of the same idea. In a digital image, an edge manifests itself as a spatially coherent discontinuity in image intensity. A reasonable definition of "edge" requires the ability to measure grey-level transitions in a meaningful way. We start by modelling an edge intuitively. Figure 3.4 shows an image contains an edge and the triangulation outcome is shown in Figure 3.5.

Although triangulation is popular in geometric modelling, it isn't widely used in image processing, e.g. image reconstruction. Triangulation methods are com-

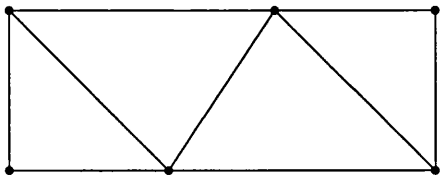


Figure 3.5: Triangulation of the edge image in Figure 3.4

plex to implement compared to bilinear reconstruction which is also simpler to compute. (The bilinear reconstruction is a well-known approach that uses the intensity of the four nearest neighbours to predict the desired pixel. Different coordinates (weights) are given to each neighbour to interpolate the unknown pixel. It is quite straightforward and is widely used in image processing because of its simplicity.)

In the following section, we will present a pixel level data dependent triangulation which is as simple as bilinear interpolation while keeping the advantages of normal data dependent triangulation.

3.2.3 A Generic Problem

We address the problem of modelling an arbitrarily-oriented edge in a triangulation mesh. Sampling gives information about the intensity distribution of the image. In image processing, when the sampling includes an edge, the triangulation mesh should fully exploit the edge information provided by the samples. Intuitively, the edges of the triangulation mesh should correspond to the edges in the images.

We start with a generic problem of image reconstruction. Let's look at Figure 3.6. Assuming P is the unknown pixel, our main problem is how to estimate P from its local neighbourhood pixels N. For example, in Figure 3.6, N contains all available local neighbours of P and we predict P from N.

Traditional methods such as bilinear reconstruction or bicubic reconstruction take all the pixels of N and predict the unknown pixel P by *weighting*, a terminology

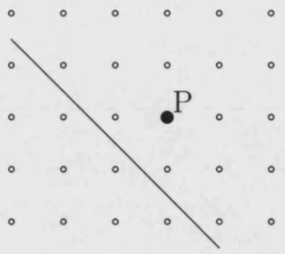


Figure 3.6: A generic problem

used to indicate that pixels within the set N are given different coefficients when calculating P, thus giving them different importance (weight). The weighting coefficients are chosen from a weighting function which is normally decided by the distance from P.

However, those methods simply ignore the edge information of N. If there is an edge across the neighbour set N, as we see in Figure 3.6, the prediction of P needs to be altered. We've already noted in the last section that the intensity field will have a significant change across the edge and will be almost homogeneous along the edge. That is to say, the P will be more like the pixels on the same side of the edge and should be predicted from the pixels on the upper side of the edge. Any unknown pixel falling in one subset should be predicted only using that subset and avoiding the subset on the other side. The spatial features of the edges, sharpness across edges and smoothness along the edge, can be kept by doing this.

We can model this by triangulating N such that the edge of the triangles corresponds to the edge of N. So the prediction of P will be done within the triangle in which it falls. This is just normal data dependent triangulation. However, it is very complex to triangulate the whole image and get an optimal triangulation mesh.

3.2.4 Pixel Level Data Dependent Triangulation

With these considerations in mind, we develop a new data-dependent triangulation approach at pixel level. We only consider a four pixel square. Firstly, we suppose that there is an edge passing within a square of four pixels. If this edge

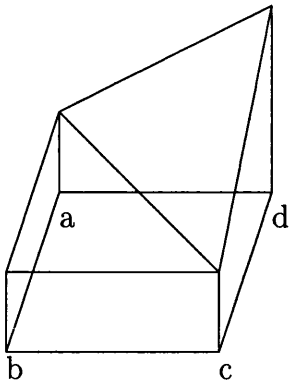


Figure 3.7: Triangulation in a four-pixel square

cuts off one corner, one pixel will have a value substantially different to (it could be bigger or smaller) the other three. Call this pixel the *outlier*. Imagine that we represent the intensity of the pixel as the height of a terrain. In effect, the three similar pixels define a plateau, relatively flat, while the outlier value is at the bottom of the cliff (if smaller) or the top of a peak (if higher) (Figure 3.7). This gives us a hint that if we want to predict a pixel within the relatively flat region we should not use the outlier. Classical interpolation methods like bilinear reconstruction suffer from edge blurring because they use all four pixels.

So we can use the diagonal of the square to correspond to the edge in the image. The diagonal should be the one which does *not* connect to the outlying pixel value, the one most different to the other three.

Obviously, using the diagonal to triangulate the four pixels cannot correspond to arbitrary angle edges. The diagonal can only roughly represent the orientation of the edge. We would have to use sub-pixel triangulation to represent arbitrary angles, but that adds more complexity to the algorithm. Our aim is to keep the algorithm as simple as possible. We will demonstrate in this thesis that triangulation by diagonal is enough in most situations and can provide excellent results. It is the direction-selection method that is key.

For a grey-scale image, we represent the brightness of the pixel as height. Suppose pixels a, b and c are the same height while d is higher than these three (Figure 3.7). Obviously a, b and c define a flat region while d is the most different pixel to the other three. Thus we connect diagonal ac and get the triangles ABC and ADC. In general, if b or d is the most different pixel, the edge should be ac, otherwise bd will be the edge.

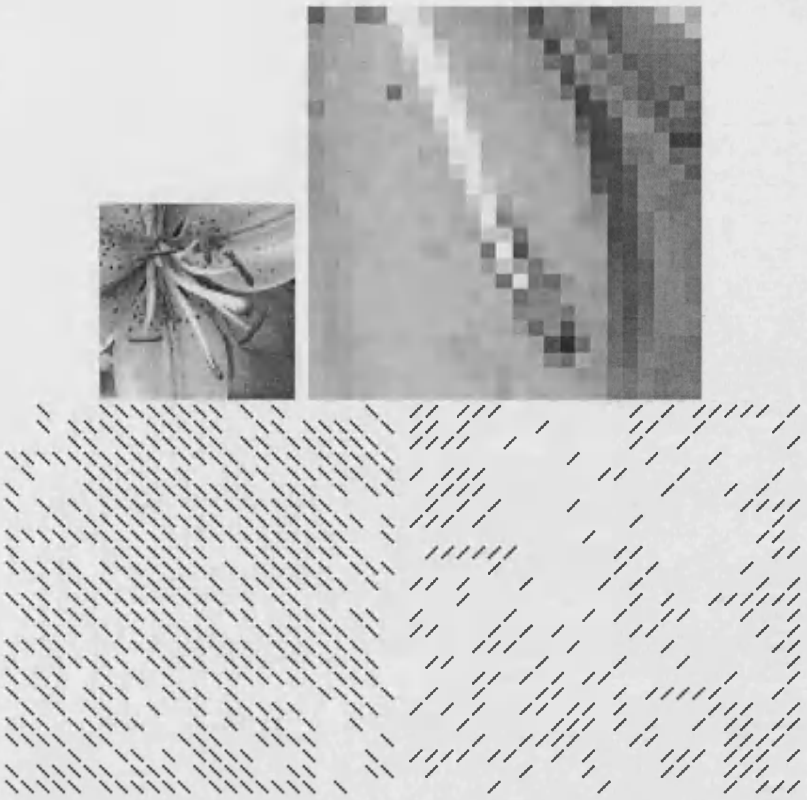


Figure 3.8: Top left: a part of a flower image. Top right: a magnified view of the bottom stamen. Bottom left: the pixel level data dependent triangulation of the stamen (NW-SE direction) Bottom right: NE-SW direction

There are other situations if a and d are very different to b and c; or a and b are very different to c and d. In these cases it makes little difference which diagonal is chosen. The edge is roughly either horizontal (ad are different to bc) or vertical (ab are different to cd) and the triangle will always cross the edge. It is similar to bilinear interpolation in these cases.

Thus, we can match the edge by the diagonals. In Figure 3.7, when predicting the pixel falling in triangle ABC, we won't use the value of d which is very different to this plateau. For two pixels falling in different triangles, the height of the pixels will be quite different and thus the sharpness of edge is kept. It is easy to see that in very smooth regions, the interpolating is able to keep its smoothness as well, even across triangle boundaries because the terrain is relatively flat and there is no strong edge in the local area.

Our method is thus to fit the finest triangular mesh to the source pixels. This mesh is completely regular except that the diagonals are locally selected to run

in the same general directions as any visible edge. It makes the model easy to implement and to use in other applications.

So any P_n within the square N_n will be predicted from the subset M_n, which consists of the three vertices of the triangle in which it falls. Figure 3.8 shows a magnified view of the stamen of a flower and its pixel level data dependent triangulation. (We only show the diagonals of the triangles for a clearer view.) We divide the triangulation into two meshes, each one only containing a specific direction. The stamen and a black edge near the stamen both roughly have NW-SE orientation. It is clear to see that the corresponding triangles also cluster in the NW-SE direction, which matches the edges of the image. In particular, note the absence of NE-SW diagonals near these linear features.

3.2.5 Optimisation

The algorithm should recognise either the highest or the lowest pixel as the most different one in order to find the outlier pixel. There is an efficient way to choose the direction of the edge. Instead of finding the outlier, we simply compare the difference $|a - c|$ with $|b - d|$ and connect the pair with smaller difference as the diagonal. The proof that this is equivalent to finding the outlier pixel is in Appendix A. This saves computing time, needing only two subtractions and a comparison instead of sorting four pixels and then comparing between the highest or the lowest pixel and the average value to determine the most different one.

3.2.6 Extended Model

Some problems still remain in our basic model. For example, close study of the triangulation of the stamen reveals a problem. The actual local edge, goes in the NW-SE direction while some diagonals in the lowest stamen areas give the NE-SW direction. This contradicts the local edges leading to some deterioration of edge quality. The reason that some diagonals contradict the local edge orientation is because our basic method only considers the four pixel square, ignores the local intensity and thus it is unable to catch the local geometry. To correct this problem we need to apply our extended model and consider information in the local area.

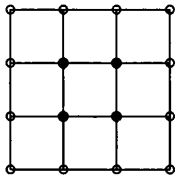


Figure 3.9: 3 × 3 square neighbour window

We assume the image is locally stationary. That is to say the intensity of a pixel is dependent on its spatial neighbourhood while independent of the rest of the image. Instead of a normal least-square adaptive edge prediction scheme, we simply consider the neighbour window's edge direction. To predict the edge direction in a four pixel square, we consider the eight squares around the target square (Figure 3.9) and adjust each square's edge direction if needed. In particular, if the most of the eight surrounding squares are in one direction while the middle square is the other, we will adjust the middle square's diagonal direction according to the majority of the eight surrounding squares.

Actually the extended model can be considered as a two-pass process. In the first pass, we apply the basic model: go through every four pixel square and calculate its direction. Then, in the second pass, we adjust the direction in a four pixel square by considering the surrounding eight squares. We use a threshold (which is set to 6 in all experiments in this thesis) to determine whether there is a strong edge orientation within a local area. If the total of the eight surrounding directions exceeds the threshold then most edges in these squares go in one direction. This means there is a strong edge orientation in this window. Then we adjust the edge in the central target square to that direction. If there is no strong local direction, then we accept the direction given by the four pixels.

Obviously our extended model increases complexity, but not much. It is a trade off of complexity and quality. It is worth noting that this additional complexity is only in preparing the mesh, not in using it to generate an image. We will show in this thesis that in most situations our basic model is effective.

Figure 3.10 is the comparison of our basic model and the extended model which considers the local intensity. We can make the observation that most triangles remain the same as in the basic model but the extended model is better able to catch the local geometry and match the orientation of the stamen and the black

edge. Figure 3.10 is a triangulation of 625 triangles, our basic model generates 418 diagonals in NW-SE direction and 207 diagonals in NE-SW direction while our extended model produces 438 and 187 respectively. They differ only on 20 diagonals, mainly along the stamen and the black edge. The extended model better preserves the local geometry.

3.2.7 Algorithm Complexity

We analyse the complexity of the basic model and the extended model in this section. Suppose the image I has width and height m, so the number of pixels is $n = m^2$. The number of triangles in the triangulation is then $(m-1) \times (m-1) \times 2$. In our implementation, we use a table to record the orientation of the diagonal in each square. As there are only two directions of each diagonal so we can use one bit to store this information. Thus, the total memory requirement for the triangulation is $(m-1)^2 \approx n$ bits. For a normal image with size 1024×1024 the memory requirement is about $128k$ bytes. Compared to the standard $128M$ memory in current PCs, it is very small. Moreover, the memory requirement n is linear with image size n.

In our basic model, each triangle needs two subtractions and one comparison, so the total computation is $(m-1) \times (m-1) \times 2 \times 3 \approx 6n$. Thus, the basic model has a time complexity of $O(n)$.

Our extended model is a two pass process. In the first pass, we calculate just like the basic model and set each triangle's diagonal direction. In the second pass, each triangle needs a sum of eight surrounding squares and a comparison to decide if there is overriding edge orientation in local area. Thus, the computation for each triangle needs two extra computations, and the whole image needs $10n$ computations which is still linear to image size n. The time complexity of the extended model is also $O(n)$.

In conclusion, our model is efficient in both memory and time, and is suitable for handling large images with a linear dependency on the image size. We note a favourable feature: the mesh rendering routine is independent of the method used to determine the diagonals.

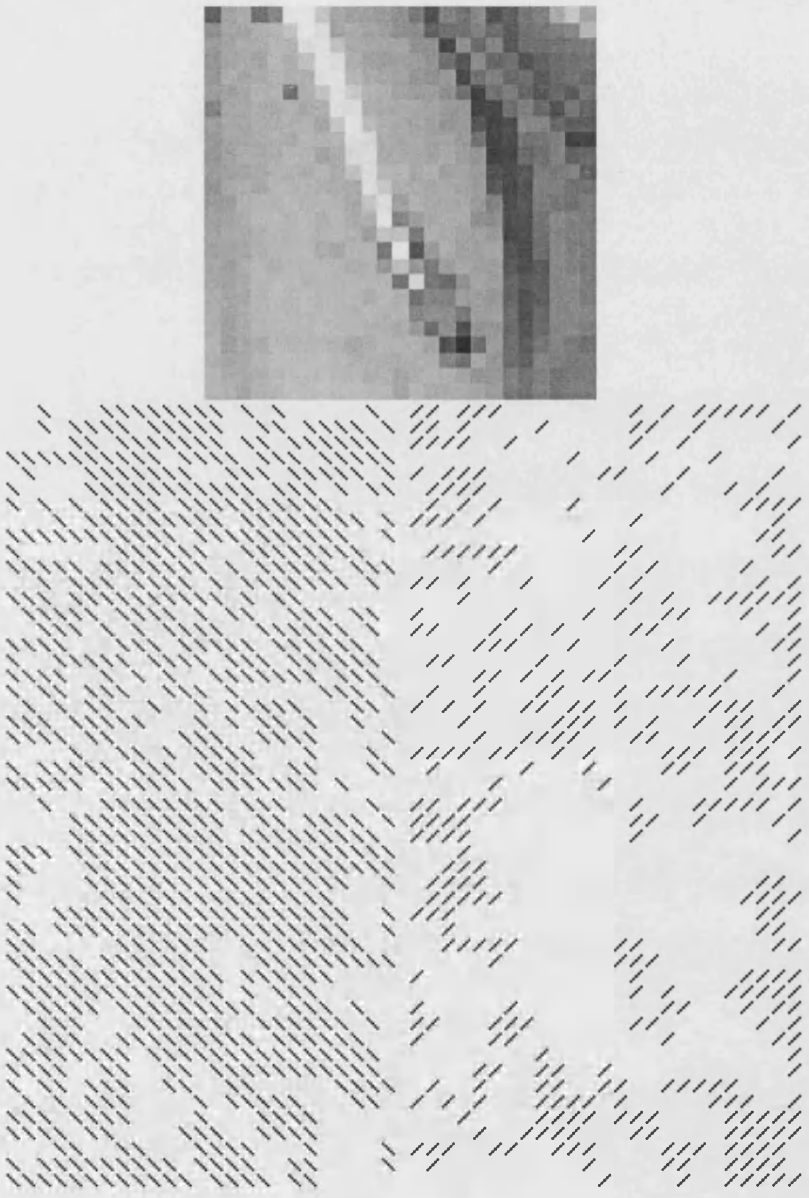


Figure 3.10: Top: a magnified view of the stamen. Middle: the basic model. Bottom: the extended model.

3.3 Concluding Remarks

Pessimistically speaking, image modelling is still very much an unsolved problem. In this thesis, we present a possible solution to this problem. We model the image trying to satisfy some deterministic spatial features related to edge orientation, where edges exist. As edges contain the visually important information of an image, such an approach provides the opportunity of challenging many existing image processing algorithms and developing new algorithms with better performance. We will demonstrate in later chapters of this thesis that our data dependent triangulation model reaches many areas of image processing and produces superior applications in these areas.

The contribution of our algorithm is that it provides a universal model for all images and a universal solution to many image processing problems. It challenges many other algorithms and generates better results. Moreover, it is very simple which makes it easy to implement various applications and it is efficient and easy to use.

Chapter 4

Image Interpolation

In this chapter we will present how our image model can be applied to image interpolation and how the results are improved by interpolation along the edge orientation. We will study one important and difficult application: the magnification of still images (including colour images) and some other applications such as rotation, the perspective transform and a non-uniform example, the lens effect.

4.1 Introduction

Digital image interpolation refers to the recovery of a continuous intensity surface from discrete image data samples. It is a link between the discrete world and the continuous one. There are three important hypotheses for interpolation [81]:

- The underlying data is continuously defined.
- Given data samples, it is possible to compute a data value of the underlying continuous function at any abscissa.
- The evaluation of the underlying continuous function at the sampling points yields the same value as the data themselves.

In general, almost every geometric transformation requires interpolation to be performed on an image, e.g. translating, rotating, scaling, warping or other applications. Such operations are basic to any commercial digital image processing software. Obviously, the quality of the interpolator determines the quality of the desired image.

There are several issues which affect the perceived quality of the interpolated images: sharpness of edges, freedom from artifacts, reconstruction of high frequency details. We also seek computational efficiency, both in time and in memory requirements. Classical interpolation techniques, such as pixel replication, bilinear or bicubic interpolation have the problems of blurring edges or of artifacts around edges. Although these methods preserve the low frequency content of the sample image, they are not able to recover the high frequencies which provide a picture with visual sharpness.

Standard interpolation methods are often based on attempts to generate continuous data from a set of discrete data samples through an interpolation function. These methods attempt to improve the ultimate appearance of re-sampled images and minimise the visual defects arising from the inevitable resampling error.

It has been recognised that taking edge information into account will improve the interpolated image's quality [48, 6, 11, 57, 58] and it is known that the human visual system makes significant use of edges [86]. Instead of approaching interpolation as simply fitting the interpolation function, these methods consider also the geometry of the image. Li [48] asserts that the quality of an interpolated image mainly depends on *the sharpness across the edge* and *the smoothness along the edge.*

Li et al.[48] attempted to estimate local covariance characteristics at low resolution and used them to direct interpolation at high resolution (NEDI - New Edge Directed Interpolation) while Allebach et al. [6] generated a high resolution edge map and used it to direct high-resolution interpolation (EDI - Edge Directed Interpolation). Battiato et al. [11] proposed a method by taking into account information about discontinuities or sharp luminance variations while doing the interpolation. Morse et al. [57, 58] presented a scheme that uses existing interpolation techniques as an initial approximation and then iteratively reconstructs the isophotes using constrained smoothing. They emphasise the importance of

the "smoothness" quality, if the isophotes are not to be visually intrusive. As will shortly become clear, we too accept this need to fit the visual geometry.

The above schemes demonstrate improved visual quality (in terms of sharpening edges or suppressing artifacts) by using a model to preserve the edges of the image and to tune the interpolation to fit the source model. However they are complex compared to traditional methods and thus computationally expensive.

Another approach is triangulation modelling. Triangulation has been an active research topic during the past decade. It is popular in geometric modelling. However, image reconstruction using triangles isn't widely used, probably because of the complexity of the triangulation method. Yu et al.[106] modelled images as data dependent triangulation meshes and reconstructed images from the triangulation mesh. Their approach adapted traditional data-dependent triangulation with their new cost functions and optimisations. The data dependent triangulation thus matches the edges in the image and improves the reconstructed image. However their methods are relatively complex.

We have discussed in the last chapter that our pixel level data dependent triangulation model is able to preserve the edge orientation of the image. It is clear that this model can be applied to the image interpolation problem. Our scheme is thus an edge-directed interpolation but differs from those previously published [106, 6, 11, 57, 58]. We do not assume knowledge of the low-pass filtering kernel or attempt to find a statistical rule about the local geometry. Our approach is related to that of Yu but is simpler and faster because it does not involve any cost function or repeating optimisation process. Our mesh is very simple and completely regular. We avoid the complexity of a full DDT method while keeping the feature of DDT that improves the reconstruction quality. In the following sections, we will demonstrate our algorithm used in arbitrary magnification of still images and other applications.

4.2 Principle of the Algorithm

An image can be represented as a pixel level data dependent triangulation mesh, with the edges of triangles correspond to edges in the image. Thus, given a

sample image, we get a triangulation mesh M produced by our image model.

An interpolation technique is then used to render the image from mesh M. The mesh M is taken as input to the interpolation process and a pixel image I will be output, where each pixel is sampled from the mesh. The image I is thus the desired image reconstructed from the sample image.

The sample is calculated by triangle interpolation (we will discuss this shortly) of the three values from the mesh triangle surrounding the sample point. We can choose our output grid of pixels to be any resolution and any orientation. This allows arbitrary magnification, rotation or other applications. We can also vary the sample spacing, to produce other effects, such as warping.

This process is quite straightforward. It takes the triangulation mesh from our model as input and applies simple triangle interpolation to get the interpolated image. The complexity of the algorithm is thus very low: it depends on the triangle interpolation which is similar to bilinear interpolation.

4.3 Image Magnification

4.3.1 Background

Image magnification is the term given to the image processing operation which achieves a higher resolution image than the one afforded by the physical sensor. Image magnification has been used in obtaining high quality images and is found in areas such as surveillance and automatic target recognition. Image magnification is also called super-resolution, zooming, resolution enhancement, enlargement, etc. However these all refer to the same operation.

Traditionally, magnification is accomplished through convolution of the image samples with a single kernel - typically bilinear, bicubic [59], or cubic B-spline [84]. Many recent algorithms have been proposed to improve the magnification results. PDE-based approaches [9, 57] apply a nonlinear diffusion process controlled by the local gradient. POCS (Projection-Onto-Convex-Set) schemes[70]

formulate the interpolation as an ill-posed inverse problem and solve it by regularised iterative projection. Orthogonal transform methods focus on the use of the discrete cosine transform (DCT) [56, 77]. Directional methods [12, 37] examine an image's local structure around edge areas to direct the interpolation. Variational methods formulate the interpolation as the constrained minimisation of a functional [41, 75]. Those interpolation schemes we have mentioned in the first section can all be used in magnification.

We will first apply our image model to image magnification. In particular, we are interested in magnifying still images, both in grey scale and colour. We will justify our algorithm by evaluating the results both subjectively and objectively. The understanding of perceived image quality is still very limited and we still have to rely on subjective evaluation. However, there are some statistical tools which allows us to assess the performance objectively although these tools are not perfect.

4.3.2 Image Interpolation by Pixel Level Data Dependent Triangulation

As we mentioned earlier, spatial features of edges play an important role in natural images. An ideal interpolation scheme should therefore always adapt to edge orientation. Edge sharpness across the edge and smoothness along the edge should be well kept by doing this.

We apply the pixel level data dependent triangulation to the image and build a triangulation mesh where the edges of triangles correspond to edges in the image. So each four pixel square is divided into two triangles. The diagonal either goes in the NE-SW or NW-SE direction depending on the local spatial geometry. Thus if we want to interpolate a higher resolution pixel falling in one of the triangles, we will use only the three vertices to do interpolation. This will not blur the edge and will preserve the smoothness along the edge. Classical interpolation methods like bilinear interpolation suffer from edge blurring because they interpolate from all four pixels.

To illustrate the approach, Figure 4.1 shows a triangulation in a four-pixel square.

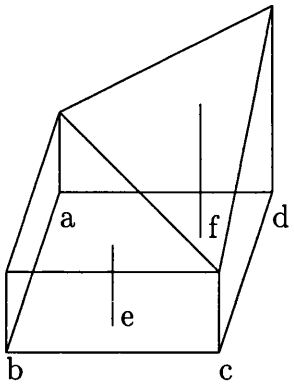


Figure 4.1: Interpolation in two triangles

Suppose e and f are two super resolution pixels falling in different triangles. We will interpolate e within triangle abc and interpolate f within triangle acd. The height of e and f correspond to their interpolated value. It is easy to see that e and f are significantly different and thus the edge will remain sharp in magnified images. Clearly if the areas are relatively smooth, the two triangles should have similar height and the interpolator is able to keep its smoothness as well, even across triangle boundaries.

Suppose the low-resolution source image is X and the high-resolution image to be generated is Y. We first scan the sample image X to initialise a lookup table which records the edge direction of all four-pixel squares. For any super-resolution pixel we can distinguish in which triangle of X the pixel falls. The high-resolution image Y is then produced by interpolation. For each y_{ij} we do an inverse mapping to the sample image X and determine the surrounding four pixel square. We use the lookup table to select the right triangle, then interpolate within the triangle to get y_{ij}.

We use inverse mapping because it has a number of benefits. First it can be used at arbitrary resolution. We are not constrained in any way by the resolution of the source data. Second, there is no requirement to align the target grid parallel to the source grid, so arbitrary rotation is possible at no additional cost. Third, sampling can be irregular to provide warps, although the sampling rate must not be too low because this would cause break-up.

We use linear interpolation within the triangles. However there is some confusion of terminology in the literature, which we need to clarify before proceeding. "Bi-

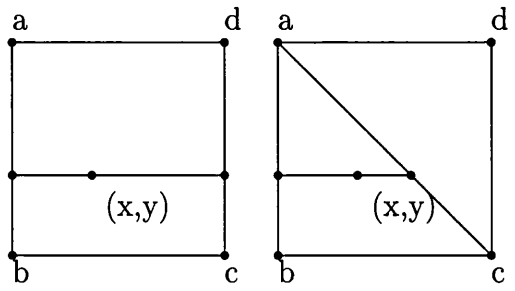


Figure 4.2: Left: bilinear interpolation. Right: triangle interpolation

linear interpolation" strictly refers to interpolating four points and we will use the term only in that sense. In the graphics community, three-value interpolation, as used in Gouraud shading, is also called bilinear interpolation, although it is only a degenerate case. We will distinguish this by calling it "triangle interpolation". (We are grateful to Professor Ken Brodlie, at the University of Leeds, UK, for drawing our attention to this.)

Figure 4.2 illustrates how bilinear interpolation and our triangle interpolation are performed in a unit square. We give the mathematical formula of these two interpolations.

$$B(x,y) = I_b + (-I_b + I_c)x + (I_a - I_b)y + (-I_a + I_b - I_c + I_d)xy$$
$$T(x,y) = I_b + (-I_b + I_c)x + (I_a - I_b)y$$

$B(x,y)$ is the bilinear interpolation and $T(x,y)$ is the triangle interpolation. For $0 \leq x, y \leq 1$, (x,y) is the position of the point being interpolated relative to the four corners. I_a is the pixel value of pixel a and so on. It is clear from the above formula that bilinear interpolation differs from triangle interpolation in the xy term: the bilinear interpolation is controlled by all the four pixels and the triangle interpolation is a piecewise linear interpolation.

For simplicity, we first consider that $I_a = I_b = 0$, $I_c = I_d = 1$ which means there is a vertical edge across the square. In this case, the bilinear interpolation becomes $B(x,y) = x$ and triangle interpolation becomes $T(x,y) = x$ which is identical to bilinear interpolation. It is easy to see that if $I_a = I_d = 0$, $I_b = I_c = 1$ (there is a horizontal edge), bilinear interpolation will again give the same result as triangle interpolation. However, if we set $I_a = I_b = I_c = 0$, $I_d = 1$, then I_d is different to the other pixels. In this case, we triangulate the square as Figure 4.2 right shows. Bilinear interpolation becomes $B(x,y) = xy$ and triangle interpolation

$T(x, y) = 0$. It is clear that triangle interpolation is better in this case because P is in a triangle with three vertices being zero.

It is clear from the above analysis that if the square is flat or the edge is roughly horizontal or vertical, the triangulation approach will produce almost identical results as bilinear interpolation. This feature will keep the smoothness along the edge and in the smooth area in images. However if there is a clear edge defined (not horizontal or vertical), i.e. one pixel is quite different to the other three, our method is superior to bilinear interpolation and keeps the edge sharp.

We mentioned in the last chapter that our basic model has some limits in that it only considers the four-pixel square, ignoring the surrounding values. We have seen in figure 3.10 that the diagonals of the lowest of the stamen contradict the local edge orientation. This will leads to some deterioration of edge reconstruction quality. It only catches the micro-geometry (pixel-level), not the local geometry due to edges passing through several pixels. To correct this we need to apply our extended model, as discussed in 3.2.6.

Our extended model considers a local neighbouring window by arranging 16 pixels as 3×3 squares. To predict the edge direction in a four pixel square, we will first set directions in each square and then, in a second pass, we consider the eight squares around the target square and see whether there is a strong edge orientation in this window. If most edges in these squares go in one direction then we adjust the edge in the target square to that direction. In our case we do this if at least 6 of the 9 squares have the same direction. If there is no strong edge direction in the local area then we only consider the target square when choosing the edge. All decisions are made on the original data so that changes do not influence nearby decisions taken later.

It is worth noting that the interpolation process of the extended method remains exactly the same, but the input triangulation mesh is now generated by our extended model.

4.3.3 Algorithm Analysis and Comparison

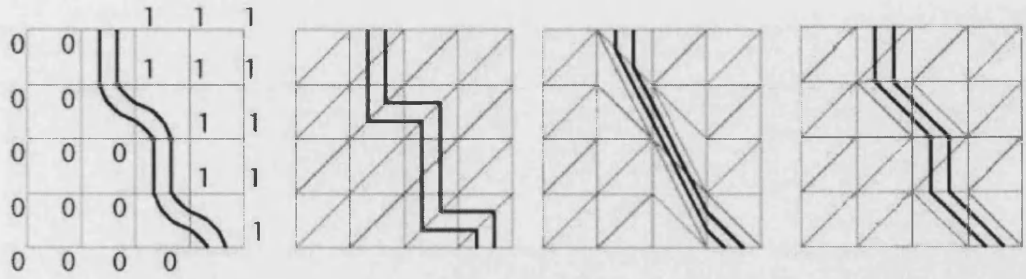
It is easy to see that the triangle interpolation has the same complexity as bilinear interpolation which is linear with image size n. We discussed the complexity of our models in 3.2.7 and explained that both our basic and extended model have time complexity $O(n)$. Thus, combining with the interpolation, both the basic and extended method have time complexity of $O(n)$. Our method is thus efficient in both memory and time, and is suitable for handling large images with a linear dependency on the image size.

Yu et al. [106] propose an image reconstruction method using data dependent triangulation. They use a new cost function and an improved optimisation algorithm to generate an optimised triangulation mesh. Their method is able to model an image effectively. It is complex to implement and is computationally slow. It takes several iterations to get an optimised triangulation and each iteration takes "between 0.5 and 5 seconds" even for a small image (80×80) on a consumer-grade PC. Another limitation of the method is it cannot catch single-pixel and small features.

Figure 4.3 shows some contours resulting from different methods. Figure 4.3a is the contour from bilinear interpolation of a simple image of 25 pixels whose intensities are zero or one and the vertexes of the square are pixel centres. No triangles are involved in this bilinear interpolation. Figure 4.3b shows the contours from a Delaunay triangulation of the image, Figure 4.3c shows the contours from Yu's DDT method [106] and Figure 4.3d is the contour from our method.

It is clear that Yu's DDT method and our methods generated straighter contours than the bilinear interpolation and Delaunay triangulation. Smoother contours tend to produce the least offensive artifacts in interpolation. Yu's DDT method produces even straighter contours than our method because it can model edges at any arbitrary angle while ours are modelling 45 degree angles. Our method can be thought of as a simplified data dependent triangulation and it actually produces the same contour as the widely used Lawson's local optimisation algorithm in the data-dependent triangulation method [106]. This Yu's method used an iterative look-ahead edge swap optimisation which produced smoother contours.

Our method generates the triangulation mesh simply by inserting diagonals. This



a: bilinear b: Delaunay c: DDT[106] d: Our method

Figure 4.3: Contours from different methods

leads to some degradation in quality since normal DDT methods can model the edge at arbitrary angles. However our method provides a notable trade-off between quality and speed. Although the DDT method can in principle give higher quality, ours is very easy to implement and much faster. Also our method needs only a small byte array to store the triangulation mesh while a full DDT requires a more complicated structure and more storage space. Another advantage of our extended method is it is able to catch small and local features.

Other researchers [67] also use DDT for data interpolation, aiming at a better optimisation of DDT according to their cost functions and optimisation processes. Our method avoids this. We will now demonstrate that the method is effective and that it does provide high-quality reconstructed images compared to conventional methods.

4.3.4 Implementations

We implemented several interpolation methods and applied them to a test image to show the results. (Figure 4.3, 4.4) The image should have well-defined edges (to test edge sharpness), thin linear features and small details (to ensure they are retained) and smoothly varying areas (to reveal any discontinuity). The flower image we have used has these features. We compare our method with bilinear interpolation and bicubic interpolation which were produced from Matlab 5 built in functions. We also compare to New Edge Directed Interpolation (NEDI) [48] as to our best knowledge it is a good interpolation method providing high interpolation quality. The NEDI is implemented by a Matlab program provided by

its author. We use a C++ program and the giga image library [96] to implement our methods.

Greyscale images were processed exactly as already described. When selecting edge direction in colour images, we convert the RGB components of each pixel into luminance using the following formula [106] where L stands for luminance:

$$L = 0.21267R + 0.71516G + 0.07217B$$

The edge direction is decided by the luminance values. Interpolation is performed in the R,G,B planes independently. This method generates good results because colour does not contribute as significantly as intensity to the information content of images, as Van Essen et al.[86] say. Figures 4.6, 4.7 and 4.8 show three examples of magnification of colour images, one with edges, one with some textures and one with some fine details. Our subjective visual experiments suggest that out method generated good results on all the images.

4.4 Experimental Assessment

4.4.1 Visual Assessment

We performed preliminary tests both to check the implementations and to permit a visual assessment of the methods.

Figures 4.3 and 4.4 show the comparison results. All the images in Figures 4.3 and 4.4 are magnified from the original flower image on left top of Figure 4.3 by a factor of 4. Figure 4.5 shows a close-up view of the stamen using our basic and extended method. This illustrates that the basic method has some artifacts along the stamen which are reduced in the extended method. Figure 4.9 shows the various methods used to magnify the colour flowers image by a factor of 3.5.

From visual inspection our method produces better images than bilinear and bicubic interpolation, and the NEDI method is good as well (Figures 4.3 and

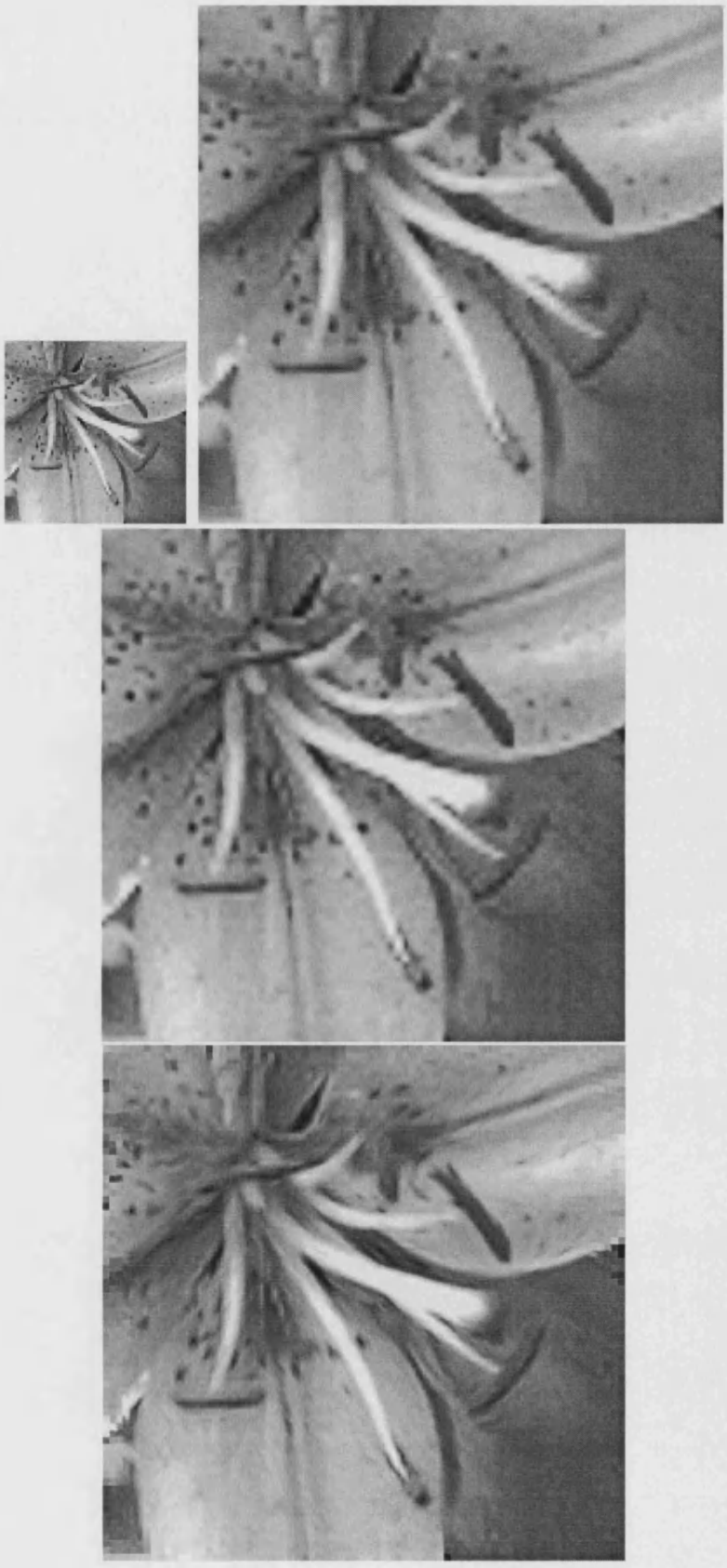


Figure 4.4: Detail flower image magnifying by 4. Top: bilinear interpolation. Middle: bicubic interpolation. Bottom: the NEDI method.

Figure 4.5: Comparison of our basic method and extended method. Top: interpolation using basic method. Bottom: interpolation using extended method.

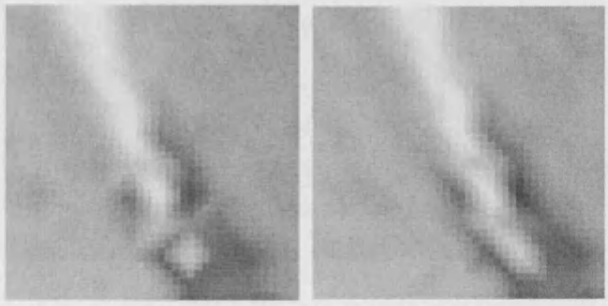


Figure 4.6: Magnified view of the stamen. Left: selecting edge only by four-pixel squares. Right: selecting edge by a 3×3 square neighbour window.

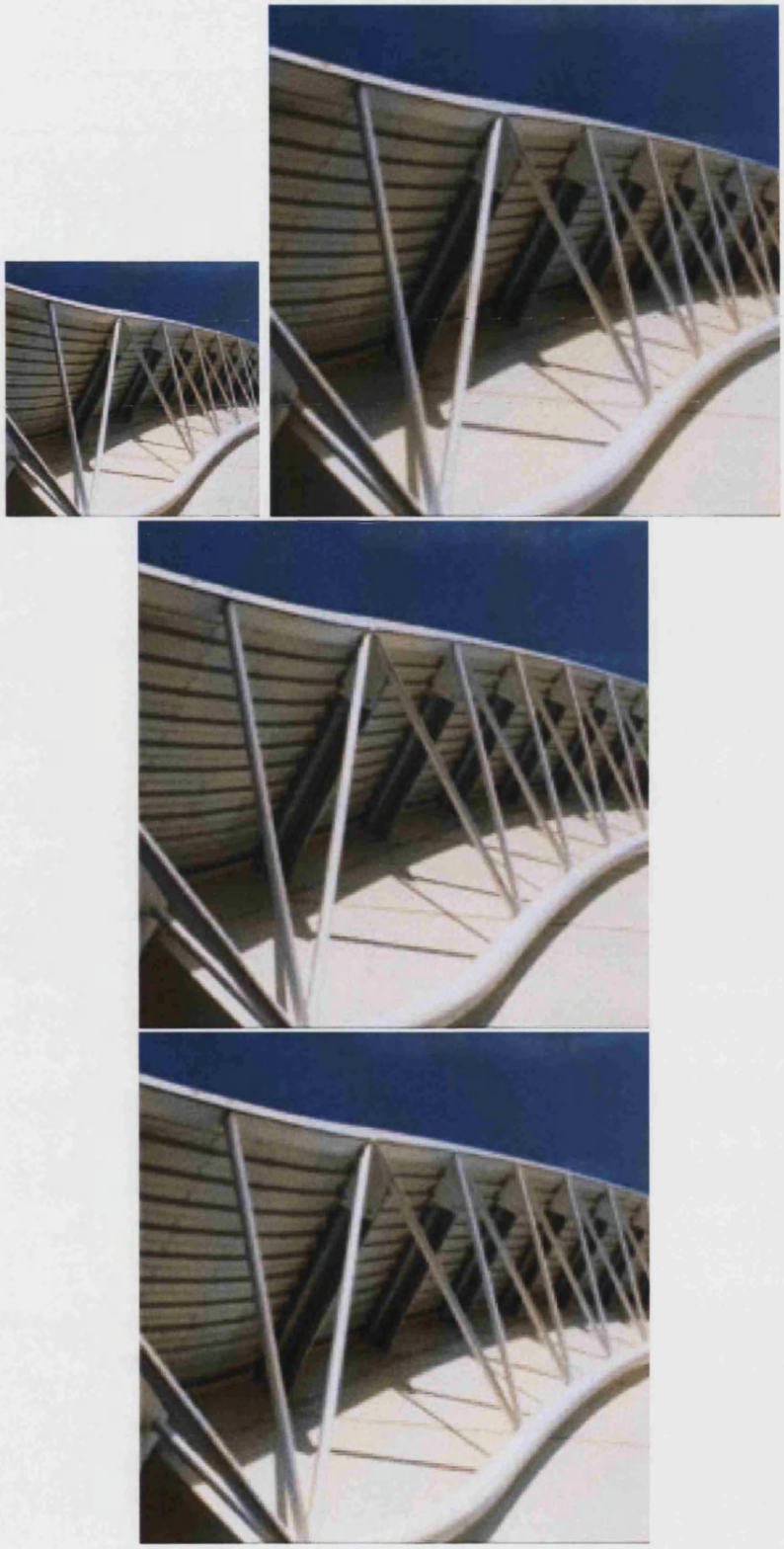


Figure 4.7: Roof image magnified by a factor of two. Top left: original image. Top right: bilinear interpolation. Middle: bicubic interpolation. Bottom: our extended method

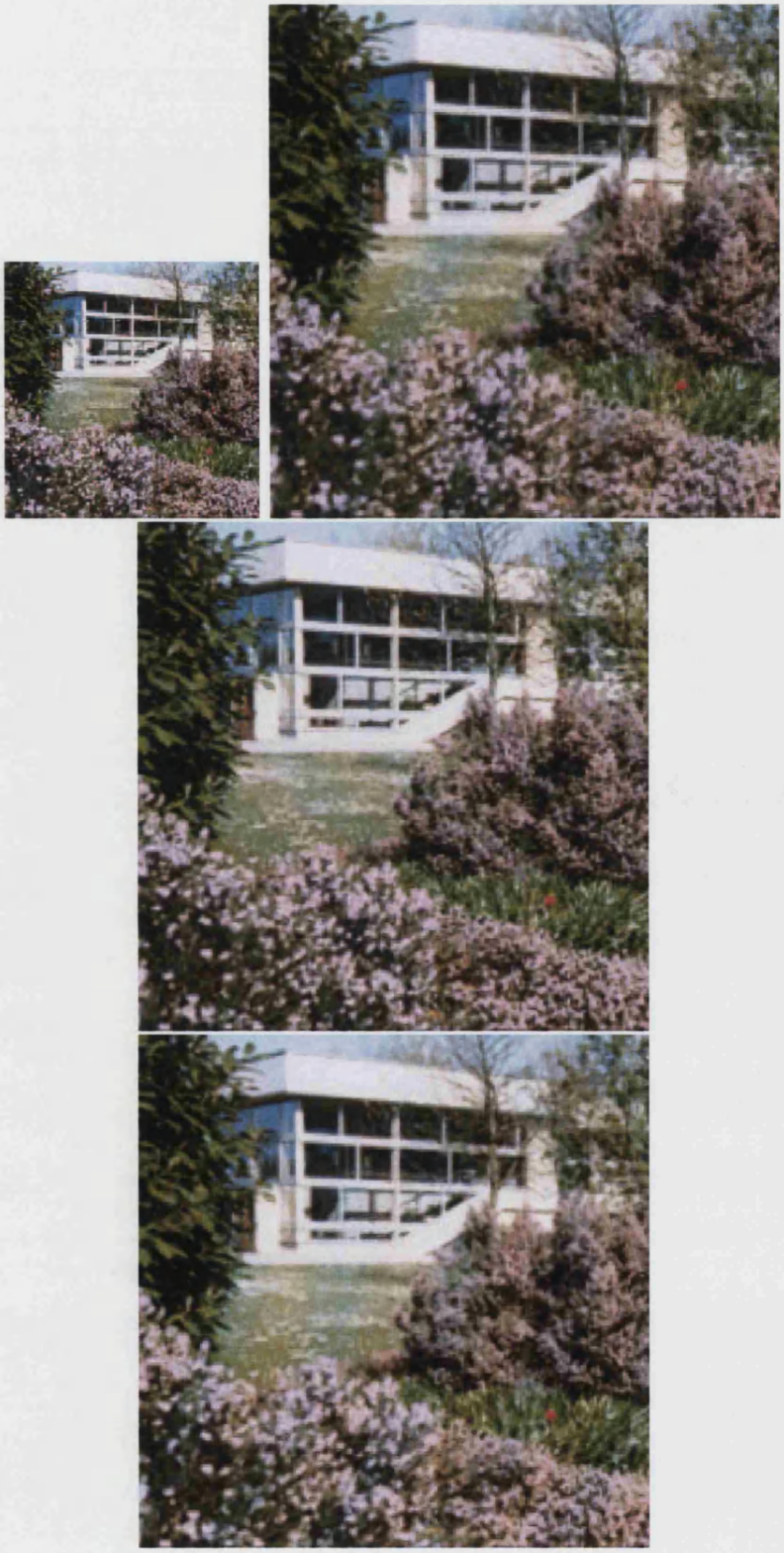


Figure 4.8: Flowers image magnified by a factor of two. Top left: original image. Top right: bilinear interpolation. Middle: bicubic interpolation. Bottom: our extended method

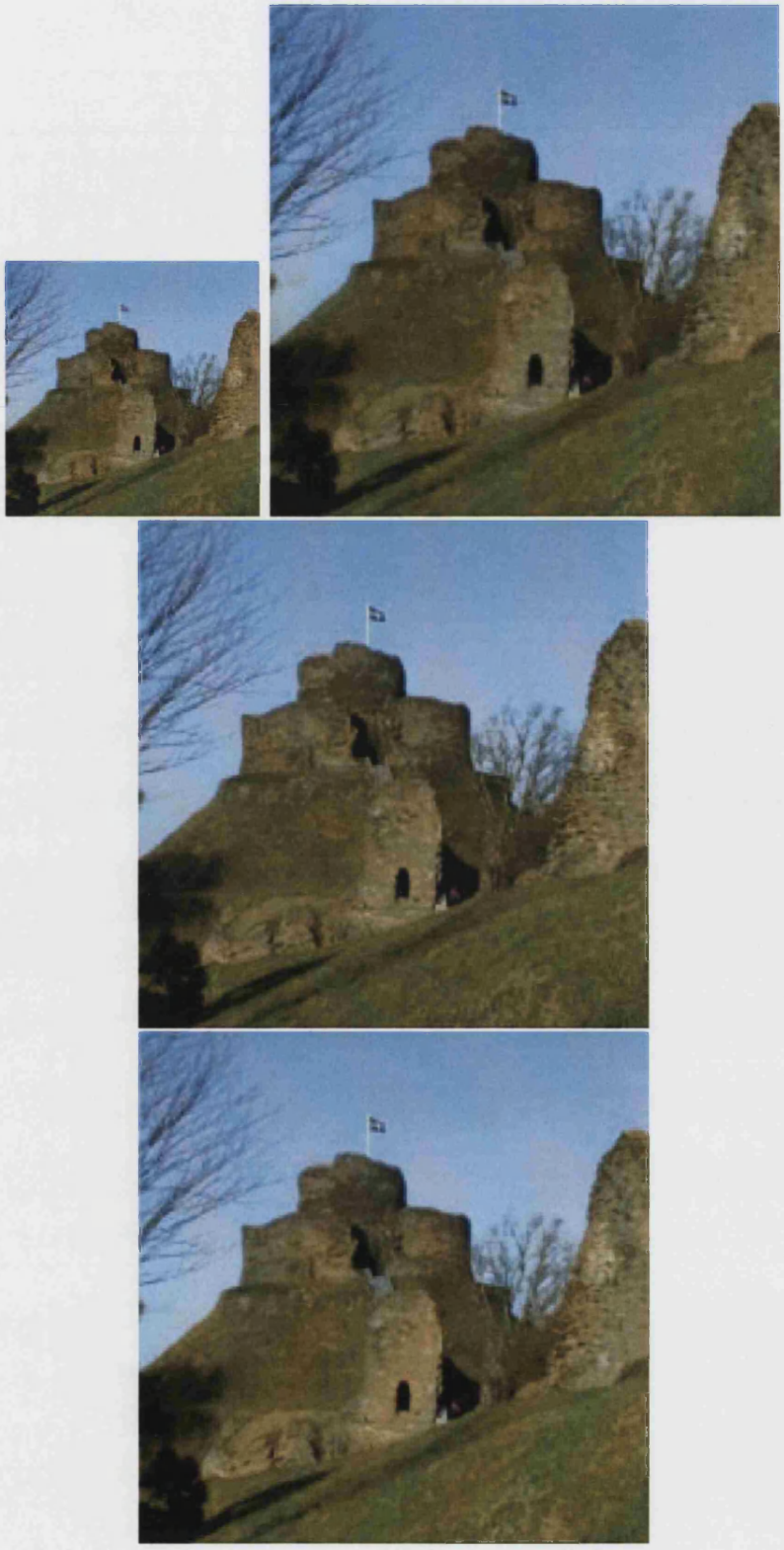


Figure 4.9: Launceston image magnified by a factor of two. Top left: original image. Top right: bilinear interpolation. Middle: bicubic interpolation. Bottom: our extended method

4.4). However, it seems NEDI's weighting algorithm changes the contrast of the image. The bilinear interpolation suffers from blurring of the edges. The bicubic method introduces sharper edges but more artifacts.

This visual assessment is however very subjective, depending on the viewer and the images used. Our visual assessment shows that our method produced good results on different kinds of images, i.e. textures, fine details and edges. It is as good as other methods in smooth areas but improved in edges. Due to the printing process, it is hard to detect the differences with some images. Some high quality images are provided in the attached CD.

Our method is subjectively good by our visual experiments. In order to obtain objective results, we next performed analytical testing.

4.4.2 Quality Assessment

To perform analytical assessment of the interpolated images, we need a quality measure. The degradation based method [87] is not able to report the "jagged" artifacts related to the orientation of edges. Daly's visible differences predictor [23] produces an error image which characterises the regions in the test image that are visually different from the original image. It is however difficult to use error images to compare different methods. Therefore we used mean-square error (MSE) as our assessment tool. The MSE is the cumulative squared error between the reconstructed and the original image. It is widely used in image processing to evaluate reconstructed image fidelity. The formula for calculating MSE is as follows:

$$MSE = \frac{1}{IJ}\sum_{i,j}(x_{ij} - y_{ij})^2$$

where I, J are the width and height of the image, x_{ij} is the value of pixel ij in original image and y_{ij} is the value of pixel ij in reconstructed image.

Our method aims at improving edge quality on magnified images and retaining a good overall quality as well. Thus we produced one sample image set of five

Figure 4.10: A portion of the flower image magnified by a factor of 3.5 using: Top right: bilinear interpolation. Middle: bicubic interpolation. Bottom: our extended method.

'edge' images with size 200 × 200 (Figure 4.10). This set is used to assess the edge reconstruction quality. We used twenty 768 × 512 nature images as another more general test set to assess the overall reconstruction quality.

In theory, there is no perfect way to judge the magnification quality. Because the image we have got is of fixed resolution, we don't know what the 'correct' image is if it is magnified. In order to analyse error, we need to know or simulate it. So we start with an original image, generate a lower resolution version, then use different methods to magnify it. Then we compare the magnified images with the original image. This is not perfect but it provides a reasonable way to analyse the reconstruction quality.

The down-sampled images could be obtained by averaging down or sub-sampling. However, edge blurring and ringing effects are introduced by averaging, while sub-sampling breaks down the geometry and introduces artifacts. We chose a Gaussian filter as the point-spread function with its standard deviation representing the radius of the point-spread function. Each pixel at the target image (down-sampled image) is considered as a point-spread function represented by a Gaussian distribution. It is down-sampled from some part of the source image, represented by another point-spread function. In this case the radius of the point-spread in the source image is double that of the radius in the target image. Thus, we calculate the standard deviation of the target Gaussian distribution, then double this to get that of the source image. This is then used to down-sample, by convolution.

We used pixel replication, bilinear interpolation, bicubic interpolation, NEDI, our basic method and our extended method to obtain the reconstructed images. All reconstructed images are magnified by a factor of two. Then we compared the original images and the reconstructed images in the test set and calculated the averaged MSE results.

4.4.3 Quality of Edges

Our first test was to check the quality of well-defined edges. For the test set we generated five samples with a single edge of varying angle (30, 45, 60, 0 and 90 degrees). Each edge is black one side and white the other side (Figure

Figure 4.11: Image set of five images with different edges. The angles are 0, 30, 45, 60 and 90 degrees

4.10). Table 4.1 shows the corresponding MSE results. We put 0° and 90° in the same column because they give the same results for all methods. Our basic and extended methods have the same results in all these situations because our basic method is able to preserve the geometry well in these simple cases.

The MSE results report that our method gets the best (lowest) score in every case except at 0° and 90°. In these two cases pixel replication gets the best score, which it is trivially able to do. (In principle it should achieve zero MSE but the Gaussian sampling introduces some grey edge pixels.) Bicubic beats us here because its interpolation more sharply models these high-contrast edges. Our method is the equal of bilinear interpolation as we expect. Although our triangulation gives edges of 45°, it also performs well on 30° and 60°. Bicubic and bilinear interpolation are slightly worse because they suffer from artifacts or blurring on the edge. Pixel replication does not generally catch the geometry very well and NEDI suffers from the effects of its weighting algorithm.

	30°	45°	60°	0°, 90°
Our methods	28.8	28.9	28.8	26.0
Bicubic	29.7	31.5	29.3	22.2
Bilinear	34.0	38.4	34.0	26.0
Replication	41.8	45.4	41.5	9.2
NEDI	43.3	47.6	43.4	27.6

Table 4.1: MSE results of edge images

4.4.4 Quality of Real Images

In order to test the method on "smoother" and more typical images, we used twenty 24-bit 768 × 512 colour nature images as another test set. The values, averaged over the test set, are reported in Table 4.2.

	R	G	B
Bicubic	109.4	119.4	123.8
Extended	117.6	127.8	132.7
Bilinear	118.2	128.4	133.1
Basic	118.6	128.8	133.7
Replication	126.1	134.8	138.7
NEDI	198.6	197.9	187.4

Table 4.2: MSE results of real images

There is a clear consistency of each channel's performance and there is also a clear consistency of each method's performance. Bicubic interpolation gets the best score (least error). Our methods rank close to the bilinear method. Our basic method is slightly worse than the bilinear method because it sometimes gives the wrong edge direction. Our extended method is slighter better than bilinear interpolation because our approach is better in edge areas and is almost the same in smooth areas.

Pixel replication gets a low score as we expect. NEDI surprisingly gets the lowest score although it has good visual reconstruction quality. We presume this is because the contrast of the image has been changed by NEDI's weighting algorithm and thus it produces numerically the wrong image, albeit a pleasing one. This emphasises the need to moderate any analysis with visual inspection.

We can thus conclude that bicubic interpolation produces the lowest overall mean squared error. Our extended method is quite close to this. Visual inspection of our method shows that it produces good results, which we believe is due to its better edge performance. We will now show that our method is much quicker than bicubic interpolation and comparable in speed to inferior methods.

4.4.5 Quality of Other Images

Because our model is a generic one and is applicable to all kinds of images. We have shown that it is effective for natural images and Figure 4.11, 4.12 show that our method is also good for medical and satellite images (Used with permission of the National Geographic Society, Image is by Robert Stacey, WorldSat International Inc.). It is especially good for magnification of those images which have a lot of small details because our method is also capable of catching small features

of the images.

4.4.6 Performance Assessment

We implemented bilinear interpolation, bicubic interpolation, our basic method and our extended method by C++ code and compared their computational performance. We used the real natural colour images test set again. We down-sampled every image to 384 × 256 pixels (using the method described earlier). Then we magnified the down-sampled images by a factor of 2 and also by a factor of 3.5. We used the Keys approach for the bicubic interpolation. Table 3.3 shows the performance comparison on a machine with an Intel Pentium4 3G processor and 1G DDR system memory. Our extended method uses the 3 × 3 square window. All figures are in seconds.

	Bilinear	*Basic*	*Extended*	*Bicubic*
magnify 2	0.359	0.406	0.412	3.621
magnify 3.5	1.105	1.162	1.170	10.914

Table 4.3: Performance comparison

We can see from the table that our method is only slightly slower than bilinear interpolation. Importantly, bicubic is an order of magnitude slower than the other methods. The averaged times for calculating the triangle mesh are included in the above figures. For our basic and extended method these are 0.041 and 0.049 seconds respectively. Factoring these out reveals that our methods are linear with the number of pixels generated.

In conclusion, our extended method is comparable in speed to bilinear interpolation while providing better reconstruction results both visually and statistically. In comparison to bicubic interpolation, our extended method is much faster and visually better, especially in edge reconstruction. These two methods are statistically similar. Our method is fast, simple and modest in memory needs.

Figure 4.12: The X-ray head image on the top left is magnified by a factor of 4 using: Top right: bilinear interpolation. Middle: bicubic interpolation. Bottom: our extended method.

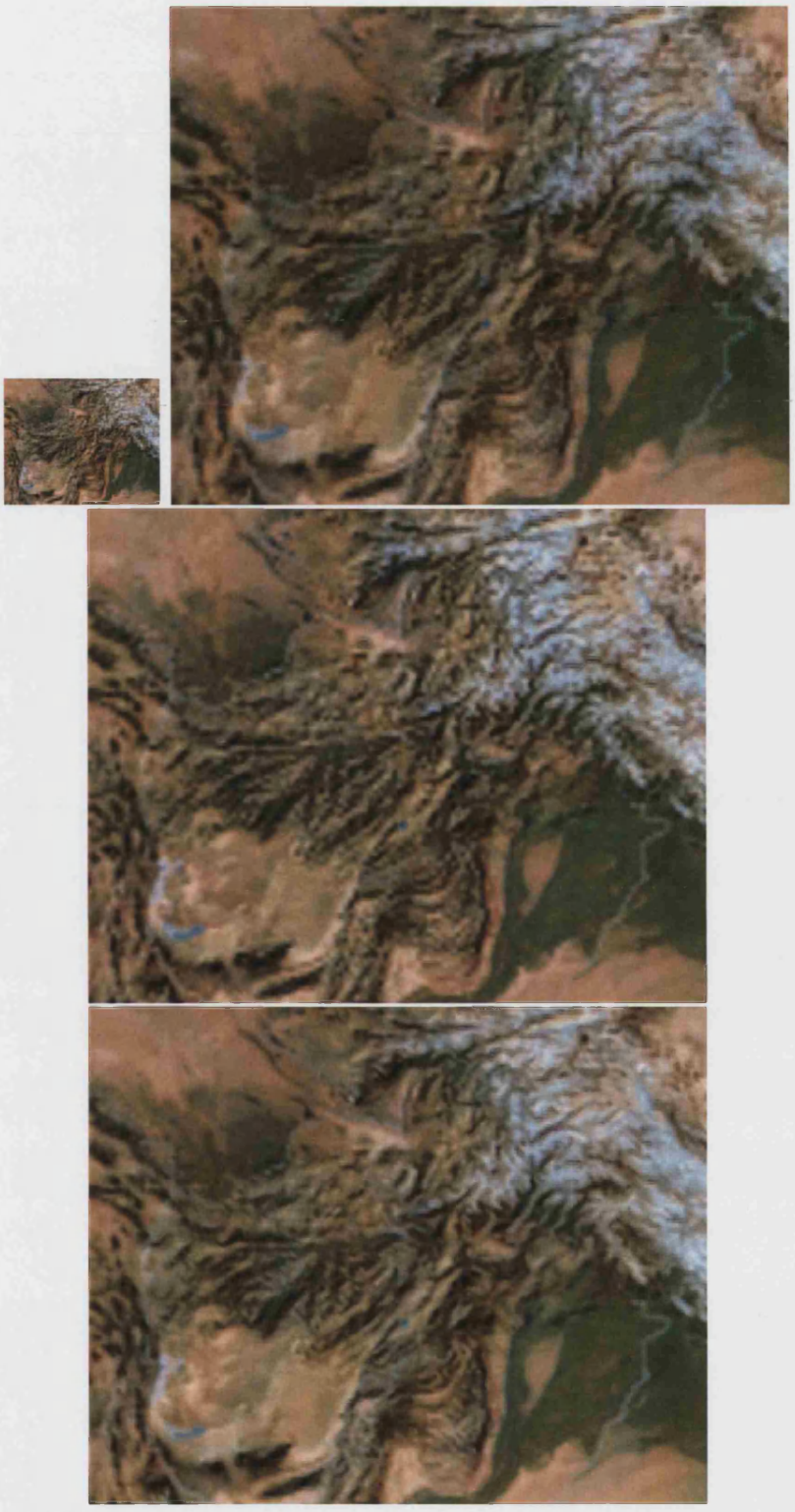


Figure 4.13: The satellite image on the top left is magnified by a factor of 4 using: Top right: bilinear interpolation. Middle: bicubic interpolation. Bottom: our extended method.

4.4.7 Hardware Implementation

More and more complex graphics operations have moved to the graphics co-processor or accelerator, including shading, texturing, anti-aliasing and bilinear interpolation. These features of graphics cards make it possible to create extremely realistic games and simulations.

However the only interpolation algorithms currently available on graphics cards are triangular and bilinear interpolation: the others are too complex. High quality image reconstruction in real-time still remains a difficult and unsolved problem. Our pixel level data dependent triangulation makes a step in this direction.

A graphics card can handle tens of millions of triangles per second and it can interpolate within triangles. This suggests that we convert any image to a triangle mesh and then pass the mesh to the graphics card. The card will deal with the mesh in real-time.

We have used OpenGL to explore the potential of our method in hardware implementation. We first generated a triangle mesh using our basic or extended model. Then we used OpenGL to pass the mesh to the graphics card so that it could manipulate the mesh, such as by scaling and rotating. These manipulations can be in 3D, at no extra cost. Our experimental results showed that high quality reconstructed images can be generated in real-time.

We used the OpenGL GL-TRIANGLE-STRIP to build the triangle mesh. This routine needs all of the triangles to have the same orientation. Thus we started a new GL-TRIANGLE-STRIP whenever the diagonal direction changes. All of these strips were saved in a display list which was then used to render the image.

The program flow of the OpenGL process is as follows:

1. Build a byte array to record the diagonals of the triangles.
2. Set up all the GL-TRIANGLE-STRIP and save them in a display list.
3. Render the image and call an OpenGL loop, waiting for keyboard response and doing manipulation corresponding to the key pressed.

Figure 4.14: Screenshot of the openGL implementation of using our method to manipulate images. The parrot image is magnified in perspective view.

We have tested several images with size 768×512 pixels, in the same machine: an Intel Pentium 4 3G processor and an NVidia GeForce 4 graphics card with 128M memory. Using our extended method, the time for preparing the mesh for an image with 768×512 pixels was under 0.2 seconds. Once the triangle mesh was loaded, the graphics card did all further manipulation. We used key presses for scaling or rotation, causing the appropriate updates to the transformation matrix.

The GeForce 4 graphics card specification claims a rendering speed of 136 million vertices per second. This equates to about 45M triangles per second. This latter *rate* could increase with triangle strips (due to vertex sharing), though of course the *number* of triangles which can be rendered at full speed is limited by the card memory. With our test image meshes having less than 1M triangles, the graphics card easily gives real-time zooms, translations and rotations. Figure 4.13 gives an illustration of using openGL to manipulate the parrot image with size 768×512. The image has been magnified and rotated along x and y axis in perspective view. The image can be scaled and rotated in real-time.

4.5 Other Applications

Due to the simplicity of our algorithm, it is easy to apply to many other image manipulations. For example, we can rotate the image by any angle (Figure 4.14a). We scan the target image and inverse rotate each pixel back to the sample image and interpolate the value. We can get a perspective transform of an image. On any given y scan line, we calculate the pixel at (x, y) by sampling the source image at (sx, ty) where s, t are scale factors which vary linearly with height (We are assuming the y axis is the centre of the screen). Figure 4.14b shows the result. We can also produce a magnifying lens effect (Figure 4.14c). If the lens has radius R, then its disc is filled with the image from a small disc with radius r at the same centre. For any pixel inside R, we scale down to r, evaluate the original value at r and output it in R.

These are variants on the same general technique: to evaluate the target pixel p, we evaluate pixel $F(p)$ where F is a simple inverse mapping to the original image. Then we interpolate in the triangle where it falls. This generality is a strength of our method.

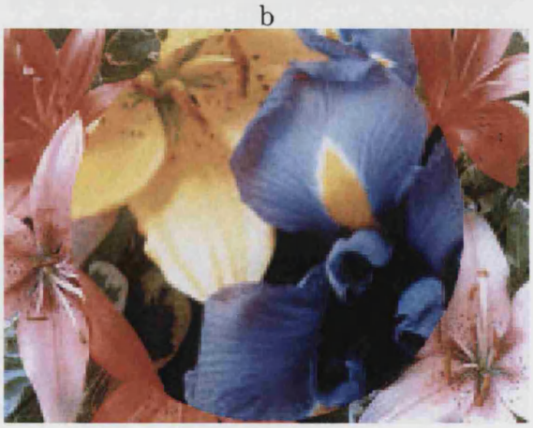
4.6 Concluding Remarks

In this chapter we have presented a new method of image interpolation. We represent an image as a data-dependent triangulation mesh. Every four-pixel square is divided into two triangles with the diagonal corresponding to the local edge of the image. The desired pixel can then be interpolated from the triangle in which it falls, determined by inverse mapping.

Other variants of the diagonal choice procedure can also be tried. For example, a pair of suitable digital filters might be better at distinguishing the local edge direction; or the threshold could be different to the one we chose. Other variants of the sampling procedure can be used, the interpolation providing some security against sampling defects. These two procedures are independent and neatly corresponding to the image modelling and image rendering phases.

a

b



c

Figure 4.15: a: Flower image rotated by 27 degrees. b: a perspective view of the flower image. c: a lens effect of the flower image

The new interpolation approach generates images with better visual quality than traditional interpolation schemes. The statistical assessment also shows that our scheme produces good overall image accuracy, second only to bicubic interpolation. The complexity of the new method is similar to bilinear interpolation and much lower than the bicubic method. We avoid the time-consuming optimisations that others use but still produce good results very quickly.

Bilinear and bicubic interpolation are widely used in commercial software package due to their simplicity. For example, Photoshop uses three interpolation engines: pixel replication, bilinear and bicubic interpolation. Our method produces better results with almost the same computation cost as bilinear interpolation. There is a potential for our method to be used in gaming and the image processing industry.

Furthermore, our method has several advantages. It is used without iteration. It achieves arbitrary factor magnification, rotation, perspective transform and warp through a single mechanism. Our scheme is very simple to implement and computationally fast. It requires little data structure overhead to generate the mesh image. Moreover, our meshes can be rendered on a graphics card which makes real-time image reconstruction possible. There is a potential for our method to be used in gaming and image manipulation generally. This simple data dependent triangulation model can also be used in other applications, such as demosaicing of colour images. We also studied its use in 4-colour separation for printing. Above all, we have demonstrated that a simple approach, sensibly used, can rapidly generate excellent results.

Chapter 5

Colour Image Demosaicing

In this chapter we will present a new method for the demosaicing of colour images generated by current single-chip digital cameras. We will demonstrate that our model is effective compared to traditional methods, when applied to the commonly-used Bayer Colour Filter Array pattern. Results show that the proposed method gives superior reconstruction quality, with smaller visual defects than other methods. Furthermore, the complexity and efficiency of the proposed method is very close to simple bilinear interpolation, making it easy to implement and fast to run.

5.1 Introduction

Colour digital cameras have become widely available consumer products in recent years. In order to reduce cost, these digital cameras use a single Charge-Coupled Device (CCD) sensor with an overlayed colour filter array (CFA) to acquire colour images, thus avoiding the need for three separate arrays (one for each primary colour) and the associated complex optical system to split the light path.

There are various filter patterns but the Kodak Bayer CFA pattern is the filter pattern most frequently used and we will concentrate on that pattern. Figure 5.1 shows this filter pattern, where R is red, G is green and B is blue. Each pixel of the CCD thus sees only one primary colour, determined by which filter overlays

G	B	G	B	G	B
R	G	R	G	R	G
G	B	G	B	G	B
R	G	R	G	R	G
G	B	G	B	G	B
R	G	R	G	R	G

Figure 5.1: Bayer Colour Filter Array Pattern (U.S. Patent 3,971,065, issued 1976)

it. This give us a *mosaic* of samples. More green filters are used because of the visual importance of this central area of the spectrum: the eye is more sensitive to green and this area is more significant to the perceived luminance. The pattern shown thus provides a higher spatial frequency sampling of green, in comparison with blue or red. There are as many green pixels as red and blue combined.

Since there is only one colour primary at each position, we can reconstruct the image at the spatial resolution of the CCD only if we interpolate the two missing primary values at each pixel. That is, at a green pixel we have to generate red and blue values by interpolating nearby red and blue values. A corresponding process is required at red (to get green and blue values) and at blue pixels (to get green and red values). This interpolation process is called CFA interpolation or *demosaicing*. The demosaicing process clearly has a significant influence and is thus the key factor in the production of high quality images. Given the limited computing resource of a digital camera and its in-built computer, the computation efficiency should also be considered.

The obvious place to start is with traditional image interpolation methods, such as nearest neighbour, bilinear interpolation and cubic convolution. Bilinear interpolation is often used due to its simplicity and efficiency[73]. However, it induces relatively large errors in the edge regions and the eye is especially sensitive to edge quality. To address this, other authors have proposed techniques which are sensitive to the data. Examples are Adams' edge oriented method [1] and various colour correlation methods [43, 2, 61]. Adams' method interpolates the missing colour elements according to the edge orientation of the image but it only detects the vertical and horizontal edges. Interpolation methods using colour correlation produce better results because there is a high correlation between the red, green and blue channels. However they ignore the edge orientation in the images.

Some more complicated methods have been proposed and they attempted to maintain edge details or limit hue transitions. Ramanath [68] used an adaptive interpolation, achieving edge orientation adaptation. Cok [20] proposed a method using a constant hue-based interpolation to make sure there are no sudden jumps in hue, especially over edges. The median-based interpolation [29] proposed by Freeman is a two pass process: the first one is a linear interpolation and the second one is a median filter of the colour differences (red-minus-green and blue-minus-green channels). Laroche and Prescott [44] proposed a method called *gradient based interpolation* and it is used in the Kodak DCS 200 digital camera system. This method is a three pass process with the first one being linear interpolation of the luminance channel (green) and the others being interpolation of colour difference channels. Hamilton and Adams [32] used an adaptive colour plane interpolation which is a modification of the method by Laroche and Prescott [44]. According to Ramanath's survey [69], Freeman's median based interpolation method[29] is the best overall method among these. However, all these methods are complicated and thus computationally slow.

We have shown that our pixel level data-dependent triangulation model can be applied to image interpolation which both matches the edge orientation of the images and correlates the red, green and blue channels. Our scheme generally produces superior reconstruction quality and is rapid. The model was applied to full-information images (that is, red, green and blue values for every pixel) with the aim of magnification or other image manipulations. In this chapter we show how our model can be adapted to supply the missing primary values of a CFA image – demosaicing – and the advantages it has in this application. Our method produces good results while remaining simple and efficient.

We will justify our algorithm by evaluating the results both subjectively and objectively. Our experiments show that images produced by our method have better visual quality than classical linear interpolation. Our approach is almost as simple as bilinear interpolation. Other methods are more complex.

5.2 The Demosaicing Algorithm

We have already considered data-dependent triangulation as a method for calculating super-resolution image values; that is, values "in between" the pixel positions. This is useful in changing the resolution of an image, distorting it in various ways, rotating it etc. In all these applications however, the original data is complete: there is a known (R, G, B) value at every source pixel. For demosaicing, we have to adjust the method to generate those primary values which are missing from the Bayer CFA pattern.

5.2.1 Principle of the Algorithm

The Bayer CFA pattern alternates red and green filters on one row, then green and blue filters on the next row. This pattern repeats on subsequent pairs of rows. This means that a blue sample has red samples diagonally adjacent and green samples orthogonally adjacent (Figure 5.2). A red sample has blue samples diagonally adjacent and green samples orthogonally adjacent.

Our task is to interpolate the missing primaries in order to get a complete (R, G, B) triple at each position. What Figure 5.2 illustrates is the equivalence of blue and red; while Figure 5.3 emphasises that the green samples are differently disposed. In fact, the green samples can be considered to be arranged on a grid at 45° relative to the other values. Moreover their spacing differs to that of the other values. The attraction of our DDT method is that it is independent of both the spacing and the orientation of the source data. It permits us to predict values at any spacing (regular or irregular) and orientation, wherever we need them.

If we consider just the red values, it can be seen that they form a regular grid of columns and rows. The same is true of blue values. It is easy to see that our DDT model can be applied here. We can triangulate each of these as already described, choosing the diagonals in the NW-SE or NE-SW direction, to favour the image edge directions. The green values can be thought of as forming a regular grid tilted at 45° (Figure 5.3). Triangulating this will produce diagonals which are in fact disposed either vertically or horizontally.

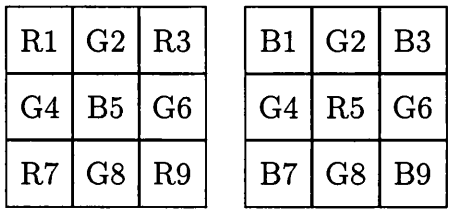


Figure 5.2: Left: Red square. Right: Blue square

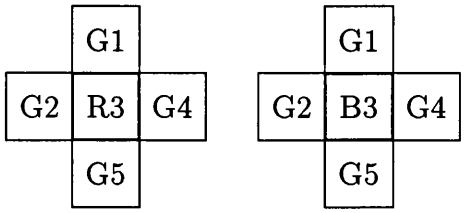


Figure 5.3: Green crosses

Thus, given a sample mosaiced image, we apply our image model and produce three meshes, one for each primary, with the green mesh being spaced and oriented differently to the other two.

Our input is thus three meshes, configured according to our image model. Because each pixel of the input image only has one primary colour, the other two primary colours will be sampled from the corresponding two meshes. These samples are calculated by triangle interpolation of the three values from the mesh triangle surrounding the sample point.

5.2.2 Original Colour Space

We will first use our method in original colour space. In an implementation, there is no need to produce three meshes explicitly. Suppose the sample image is X and the output image to be generated is Y. We first scan the sample image X to initialise three lookup tables, one for each primary. Each table has one bit to record the edge direction at every 2×2 'square' of pixels of that primary colour. To reconstruct an image pixel, we first determine which two primaries need to be recovered. We then use the corresponding lookup tables to establish in which triangle the image pixel sits in each mesh. This establishes three values to be interpolated for each of the two missing primaries.

In fact, only two values are needed. Suppose we are interpolating for red values of blue or green pixels. For demosaicing, the target pixel will always fall on the boundary of the triangle. Hence the interpolation is always the average of two vertex values.

Thus we get the following formulae for the Red and Blue squares (Figure 5.2):

$$R_{B5} = (R1 + R9)/2 \qquad or \qquad R_{B5} = (R3 + R7)/2$$
$$R_{G4} = (R1 + R7)/2$$
$$R_{G8} = (R7 + R9)/2$$

$$B_{R5} = (B1 + B9)/2 \qquad or \qquad B_{R5} = (B3 + B7)/2$$
$$B_{G4} = (R1 + R7)/2$$
$$B_{G8} = (R7 + R9)/2$$

Similarly the following are the formulae for the Green crosses (Figure 5.3):

$$G_{R3} = (G1 + G5)/2 \qquad or \qquad G_{R3} = (G2 + G4)/2$$
$$G_{B3} = (G1 + G5)/2 \qquad or \qquad G_{R3} = (G2 + G4)/2$$

In all cases therefore, the value is reconstructed as the average of two source values, those values being chosen by our DDT method. This simplifies the interpolation and avoids the need for inverse mapping.

5.2.3 Colour Difference Space

Treating R, G and B planes independently ignores the correlation among the colour planes and produces colour mis-registration. Recent research [2, 61, 68] has shown that interpolation performance can be significantly improved by exploiting the correlation among the colour planes. These methods are based on the assumption that the red and blue values are perfectly correlated to the green value over the extent of the interpolation neighbourhood. They define the colour differences $KR = G - R$ and $KB = G - B$ and interpolate in this colour dif-

ference space. In other words, these methods transform the operation into the KR or KB domain instead of performing the interpolation in the G channel. We calculate the KR and KB values and interpolation using these values, then we reconstruct the original colour values: $R = G - KR$, $B = G - KB$ and $G = R + KR$ or $G = B + KB$.

The formula for interpolation in colour difference space is therefore different to that for original colour space. First we need the formulae for calculating the colour difference value in R, B and G pixels. For every Green Cross (Figure 5.3) there is either a Red or a Blue pixel, thus

$$KR_{R3} = (G1 + G2 + G4 + G5)/4 - R3$$
$$KR_{B3} = (G1 + G2 + G4 + G5)/4 - B3$$

and for Green pixels, there are always two Red and two Blue pixels surrounded, therefore the KR and KB value of G pixels are always G minus the average of two Red or Blue values, for KR and KB calculation, respectively.

Thus, we get the following formulae for the Red and Blue squares, according to the triangulations we have generated for each colour channel (choices between values reflect the choices of diagonals in the triangulations):

$$KR_{B5} = (KR_{R1} + KR_{R9})/2 \quad or \quad KR_{B5} = (KR_{R3} + KR_{R7})/2$$
$$G_{B5} = B5 + (KB_{G2} + KB_{G8})/2 \quad or \quad G_{B5} = B5 + (KB_{G4} + KB_{G6})/2$$
$$R_{B5} = G_{B5} - KR_{B5}$$
$$R_{G4} = G4 - (KR_{R1} + KR_{R7})/2$$
$$R_{G8} = G8 - (KR_{R7} + KR_{R9})/2$$

$$KB_{R5} = (KB_{B1} + KB_{B9})/2 \quad or \quad KB_{R5} = (KB_{B3} + KB_{B7})/2$$
$$G_{R5} = R5 + (KB_{G2} + KB_{G8})/2 \quad or \quad G_{R5} = B5 + (KB_{G4} + KB_{G6})/2$$
$$B_{R5} = G_{R5} - KB_{R5}$$
$$B_{G4} = G4 - (KB_{B1} + KB_{B7})/2$$
$$B_{G8} = G8 - (KB_{B7} + KB_{B9})/2$$

and the following formulae for Green crosses:

$$G_{R3} = R3 + (KR_{G1} + KR_{G5})/2 \quad or \quad G_{R3} = R3 + (KR_{G2} + KR_{G4})/2$$
$$G_{B3} = B3 + (KB_{G1} + KB_{G5})/2 \quad or \quad G_{B3} = B3 + (KB_{G2} + KB_{G4})/2$$

In the image boundary areas, there is no enough information for the interpolation we have described. So we modify the interpolation in image boundaries. In particular, we get the G value for Red and Blue pixels by averaging the two Green pixels on the same boundary line. We get the R/B value for Green pixels by averaging two Red/Blue pixels on the same boundary line and copy the nearest Blue/Red pixel to get B/R value. Then the R value for Blue pixels and B value for R pixels can be calculated by averaging the R or B values from the two Green pixels on the same boundary line.

This method is based on the assumption that colour difference is relatively flat over small regions. This assumption is valid within smooth areas of the image but is not valid around the edges in the image. Colour mis-registration would still exist around the edges if bilinear interpolation was applied. Our method effectively solves the problem by interpolation *along* the edges in colour difference space, as Figure 5.4 shows. It avoids colour mis-registration by not interpolating *across* the edges in the colour difference space.

5.3 Experimental Results

5.3.1 Quality Assessment

We have performed various tests on two images, one of a boat (Figures 4.4 and 4.5) and one of a macaw (Figures 4.6 and 4.7). In each case, the top left image is the original 24 bit image of size 768×512. From this we prepared a mosaic image by, at each pixel, discarding the two primaries indicated by the CFA pattern. This mosaic image was then used to perform the various reconstructions shown, again at 768×512.

We applied several different demosaicing methods to the test images: bilinear

Figure 5.4: Portions of: a: original boat image. b: median based interpolation. c: bilinear interpolation in the original colour space. d: bilinear interpolation in the colour difference space. e: our method in the original colour space. f: our method in the colour difference space

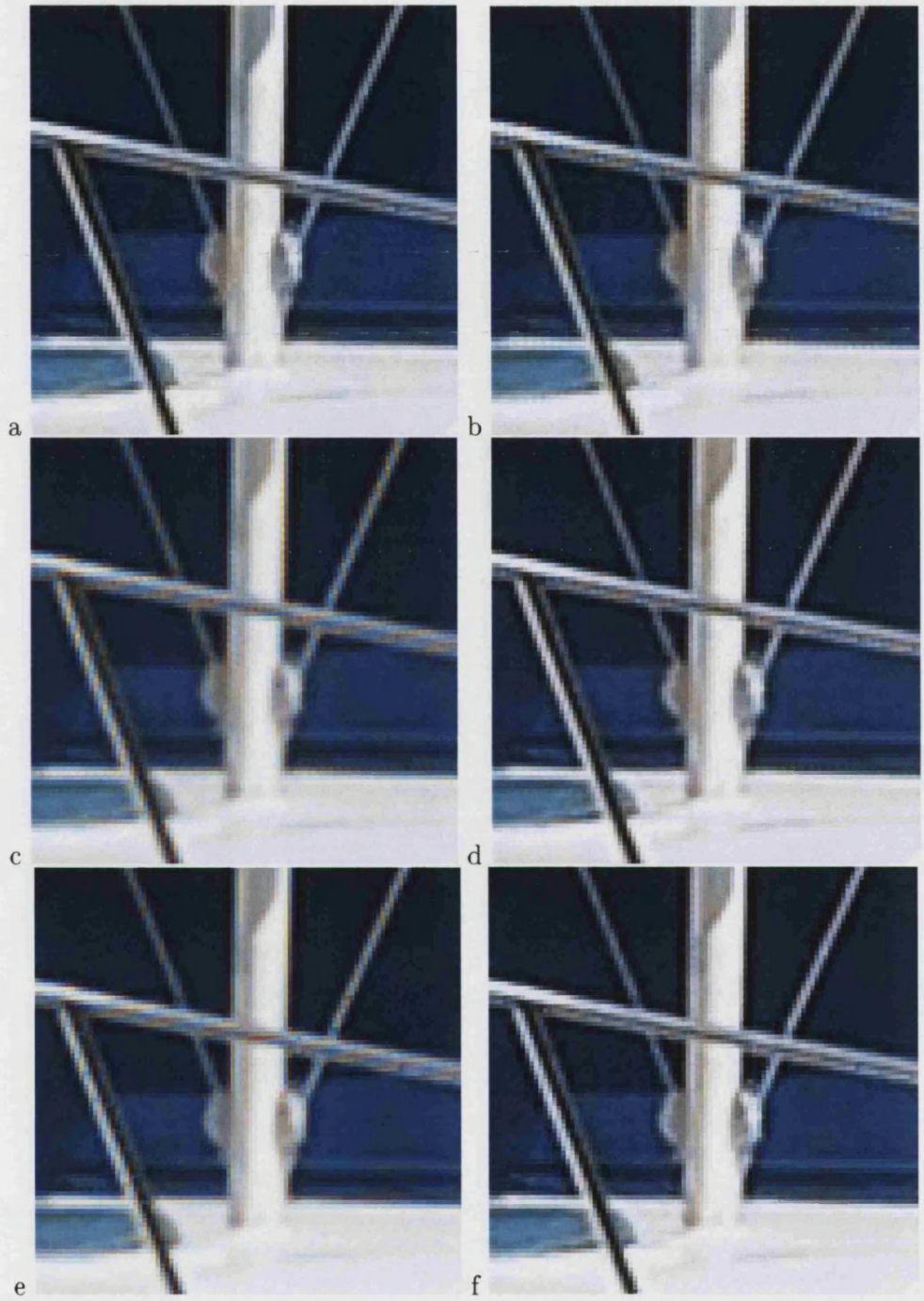


Figure 5.5: Close-up comparison of: a: original boat image. b: median based interpolation. c: bilinear interpolation in the original colour space. d: bilinear interpolation in the colour difference space. e: our method in the original colour space. f: our method in the colour difference space

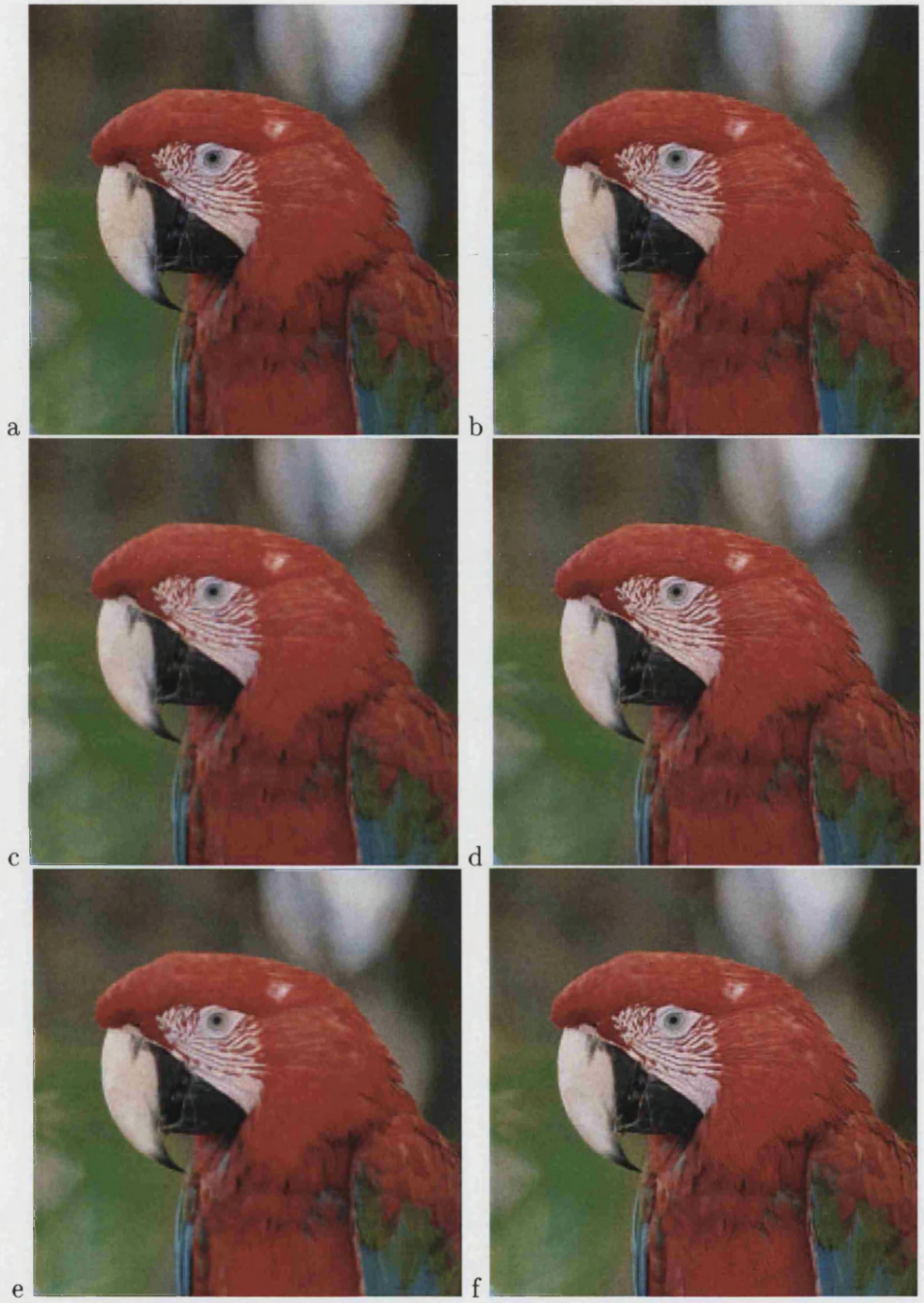


Figure 5.6: Portions of: a: original macaw image. b: median based interpolation. c: bilinear interpolation in the original colour space. d: bilinear interpolation in the colour difference space. e: our method in the original colour space. f: our method in the colour difference space

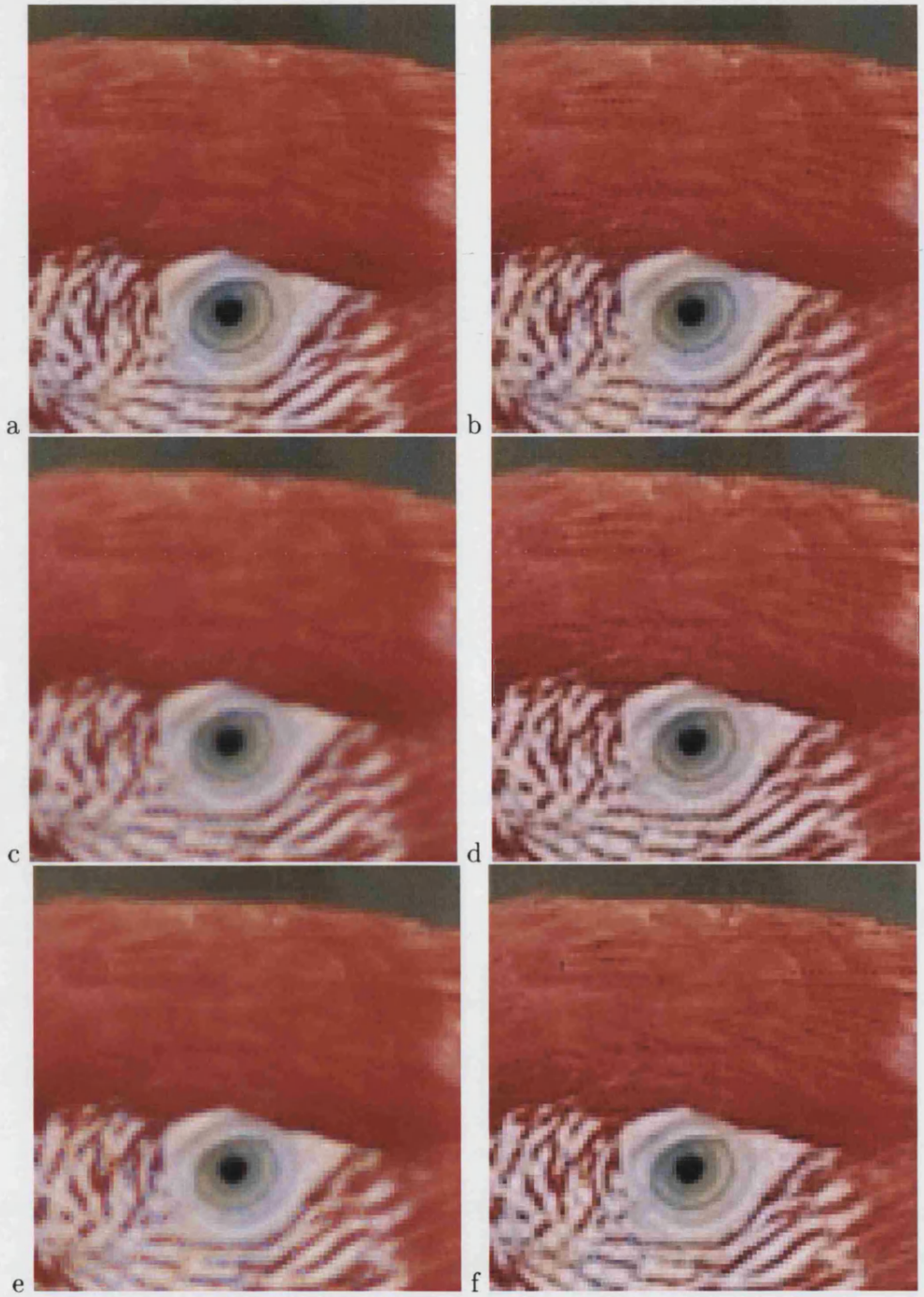


Figure 5.7: Close-up comparison of: a: original macaw image. b: median based interpolation. c: bilinear interpolation in the original colour space. d: bilinear interpolation in the colour difference space. e: our method in the original colour space. f: our method in the colour difference space

interpolation in both original colour space and colour difference space, our data-dependent triangulation method in both original colour space and colour difference space and Freeman's median based method [29].

If we compare bilinear interpolation and our method in original colour space and in colour difference space, it can be seen that interpolation in the colour difference domain has better reconstruction quality than interpolation in the original colour space. When using original colour space, colour mis-registration is clearly visible near the thin lines in the boat picture and around the top of the macaw where there is a sharp colour transition. Interpolation in colour difference space reduces most of these errors, however some colour mis-registration and artifacts are still clearly visible in bilinear interpolation and median based interpolation. There are also noticeable dotted artifacts around those edges. Our method has the least colour mis-registration error in both images. It avoids both of these problems because it better preserves the geometric regularity and interpolates along the edge orientations of the image.

Direct visual inspection indicates that our method produces good reconstruction quality. However, we wanted to explore a more analytical assessment of the visual quality of the interpolated images, though this is not straightforward to define, let alone measure. We used the Peak Signal-to-Noise Ratio (PSNR) which is commonly used as a measure of image quality in digital image compression and reconstruction as our assessment tool.

The PSNR is based on *Mean-Squared Error* (MSE). The Mathematical formulae for the two are:

$$MSE = \frac{1}{IJ}\sum_{i,j}(x_{ij} - y_{ij})^2$$

$$PSNR = 20\log_{10}\frac{S}{\sqrt{MSE}}$$

We used twenty 24-bit 768×512 colour nature images as our test set. The PSNR values are calculated for the three colour channels independently and averaged over the test set. Table 5.1 shows the corresponding PSNR results where BL means bilinear interpolation in original space, BLD means bilinear interpola-

	BL	*BLD*	*DDT*	*DDTD*	*MEDIAN*
R	31.47	35.07	31.22	33.96	35.70
G	35.35	39.38	35.10	37.99	40.02
B	31.01	34.21	30.74	32.31	34.97

Table 5.1: PSNR results of different methods

tion in colour difference space, *DDT* means data-dependent triangulation (our method), *DDTD* means our method in colour difference space and *MEDIAN* is median-based interpolation. The values are averaged over the test set.

Table 5.2 shows the PSNR results of three colour channels for the different methods. The results are averaged over the twenty images.

The PSNR results show a clear benefit from the use of colour difference space, for both bilinear interpolation and our method (Median based interpolation also uses colour difference space). These results confirm earlier work supporting colour difference space [43, 2, 61]. When comparing the two methods, the PSNR results show that bilinear interpolation is only marginally better than ours. As we discussed, our method is designed for solving the problem of colour mis-registration in edge areas. So for images which mainly consist of smooth areas, bilinear interpolation will give a better statistical result because it uses more information for interpolation. Median based interpolation gets the best score which means it produces the best overall reconstructed image.

However, informal observation confirms that our method gives improved edge quality. It looks better because human eyes are more sensitive to edges and our method is better at retaining edges. Our overall result is very close to bilinear interpolation which means our method produces good overall reconstruction images.

5.3.2 Performance Assessment

Table 5.3 shows the performance comparison on a Pentium 4 machine which 3G CPU and 1G DDR system memory. We used the 20 images again and timed the four methods. All the methods are implemented in C++ code and all the figures in the table are seconds.

BL	*BLD*	*DDT*	*DDTD*	*MEDIAN*
0.311	1.086	0.691	0.961	1.555

Table 5.2: Performance comparison of different methods

As we expected, median based interpolation is the slowest because it is a two pass process which adds extra computation time. Bilinear interpolation and our method using the colour difference space require more computation than the method in original colour space. Of the two methods using colour difference space, our method is faster. This is true even including the overhead of initialising the triangulations in three colour channels (about 0.23 seconds in this case). Our method is significantly faster because it only requires two pixels to interpolate while bilinear interpolation requires four pixels. We have already shown that it has good overall quality and visually better edges. We suggest that these features make it a better choice for demosaicing colour images.

5.4 Conclusion

In this chapter we have presented a new method for demoisaicing of colour images. The new method is based on our data-dependent triangulation model. The mosaiced image is represented as three primary colour triangulation meshes. The interpolation is done within these triangulations, which match the edge orientation of the images. By avoiding interpolation across edges, the new algorithm successfully solves the problem of colour artifacts around the edges. We also applied the scheme in colour difference space which helps to reduce the artifacts caused by colour mis-registration.

We have applied our method to the Bayer CFA pattern and our method offers simplicity and efficiency. The PSNR results also demonstrate that our method is very close to the best comparator in producing the 'right' data, while visual inspection shows that the data is more effectively deployed to produce sharp edges. It is also much faster.

Given the limited computing resource of a digital camera and its computer, we believe our method provides a reasonable solution to the colour image demosaicing problem because it produces good reconstruction efficiently.

Chapter 6

Texture Synthesis

In this chapter, we will present a survey of traditional and contemporary texture synthesis methods. From this we introduce a texture synthesis method proposed by Yan Zhang at Jilin University, China [108]. It derives from a patch-based sampling method [49] and uses particle swarm optimisation to search for the best match patches, thus accelerating the synthesis process while ensuring image quality. We will present our extension of this method to texture transfer and constrained multi-sample texture synthesis.[1]

We also develop a new method based on patch-based sampling for synthesising textures on perspective surfaces from an input sample image. The method synthesises the texture directly on the surface, rather than synthesising a texture image and then mapping it to the surface. The synthesised textures have the same qualitative visual appearance as the example texture, and cover the surface without distortion, repetition or aliasing artifacts.

[1]The texture synthesis using PSO is the work of Yan Zhang at Jilin University, China (The author provided the original implementation for patch-based sampling texture synthesis). The texture transfer and constrained multi-sample texture synthesis using PSO were done during the author's research visit at Jilin University working together with Yan Zhang in August 2003.

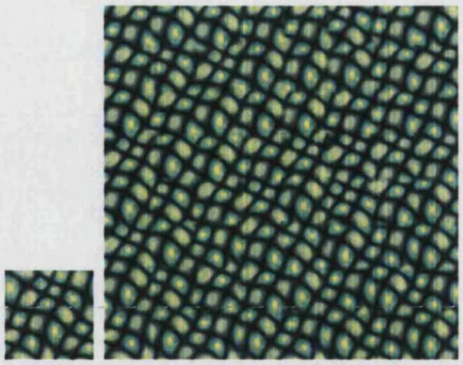


Figure 6.1: Left: sample image. Right: the synthesised image with arbitrary size and similar visual appearance to the sample.

6.1 Introduction

Normally, pictures generated by a computer do not appear as realistic as photographs or video images. This problem of lack of realism arises because the level of detail in a real picture is greater than the level that could be generated by the techniques to date. People are looking for higher realism images without much expense of computation time.

Adding shadows and texture mapping became highly developed methods to enhance Phong shaded scenes so that they were more visually interesting and looked more realistic or esoteric. Texture mapping was first introduced by Blinn [14] as a technique for adding the appearance of surface detail by projecting or wrapping a texture image onto an object surface. It can enhance the visual interest of a scene without adding too much processing cost.

While texture mapping itself is straightforward, acquiring the images to use for textures is not always easy. One way to generate texture images is texture synthesis. The texture synthesis approach can be stated as follows: Given a texture sample, synthesise a new texture with arbitrary size that is sufficiently different from the given sample texture, yet appears to be generated by the same stochastic process when perceived by a human observer (Figure 6.1).

Potential benefits of texture synthesis include the ability to create large and/or tiled texture from a small sample. With this ability, texture synthesis can be used for image repair (for example, filling a hole in a texture), and for textures in games (i.e. the walls and grounds in "Tomb Raider"). It can be also used to improve

the apparent realism of other rendered images (texture maps, environment maps etc).

Its power comes from its generality: any image can be used as a texture. Hence real images, hand-drawn images and synthetic images are all possible sources. Textures can be used one-off (i.e. to add the appearance of a desktop to a computer screen) or repeated (brickwork, sand, grass etc). They can also be blended with other information. Hence textures are supported by graphics cards and are widely used in rendering.

Simple texture mapping, as used on games cards, is a fast but trivial solution. This reveals some of the problems. It only copes with regularly repeated patterns. The patterns have to be chosen so they produce a plausible repeat and, where the tiling is required to be less visible, the values of the boundary pixels have to be manually set to hide the boundary.

In the general case, texture synthesis attempts to produce texture which is not identical to the sample but which is statistically similar to it. This is useful because it makes rendering more realistic by avoiding simple repetition.

In this chapter we will focus on texture strictly in the form of a design or a pattern whose intention is to deceive the viewer as to the regularity of the surface [18]. Textures are spatially homogeneous and consist of repeated elements, often subject to some randomisation in their location, size, colour, orientation, etc [65]. They can describe a wide variety of surface characteristics such as those of wall, fur and skin. There are two kinds of texture: regular (these can be characterised by a set of primitives and a placement rule, such as bricks in a wall) and stochastic (these do not have easily identifiable primitives, such as grass). Many texture images lie between these two categories. The texture of an image has a very important character, which is: given a proper size, any part of the texture image looks similar to any other part.

Texture synthesis has been an active research topic in computer graphics for many years and there are many ways to generate new textures. In order to evaluate an algorithm, we have to set the criteria for successful texture synthesis.

- Quality: A good algorithm should generate a new image that looks like the

sample image. There should be no obvious blur, repetition, or mismatching features in the output image. It should appear to show the same stochastic process as the input image. If we can analyse the sample image and the synthesised image by a model such as the steerable pyramid [66], the sample image and result image should be very similar. Additionally if the sample were taken from scanned photographs, the synthesised texture should be photo-realistic.

- Speed: The major motivation of texture synthesis is a method which aims to enhance realism without much computation time. Most older algorithms are extremely slow, even small images (256×256) need several hours on a mid-level PC. A good approach should synthesise in real-time or reasonably rapidly.

6.1.1 Texture Synthesis Tasks

Texture synthesis proves difficult because it is always hard to discover the stochastic process from a given texture sample.

The major challenges of texture synthesis are:

- Recognition: how to capture the stochastic process and the texture scale from the given sample image. A successful recognition model is very important because the visual quality of the output image is dependent on the accuracy of the model. It is difficult to characterise texture images, using either deterministic or statistical models. There are many algorithms and models to analysis the texture images. They belong to two kinds of approaches, one is trying to catch texture globally and other is treating texture as local. None of the existing techniques can produce a completely satisfactory solution for all kinds of textures.

- Generation: how to develop an efficient sampling procedure to generate new textures from a given sample texture and the analysed model. This is essential for successful texture generation because it will determine the quality and the speed of the synthesis process. The operation of the sampling procedure involved in assigning a value to a pixel depends on the

rendering algorithm used and we can only ever calculate the value of an image function at these points [91].

6.2 Previous Work

Over the past three decades, texture synthesis has been investigated in computer graphics and many algorithms have been developed.

6.2.1 Traditional Texture Synthesis Methods

Most traditional texture synthesis approaches are based on procedural texture synthesis.

Procedural texture synthesis is the use of a function or set of functions applied to a set of points in order to generate a texture. Procedural methods can be very fast. It is easy to introduce a time-varying variable to the model thus creating animation. The method requires little memory because it synthesises on the fly [85]. These methods can produce textures directly on 3D meshes so the texture mapping distortion problem is avoided. However existing procedural methods are only specialised emulators of the generative processes of certain types of texture. Different textures are usually generated by different models so these methods are applicable to only limited classes of textures.

Solid Texture: Solid texturing is a powerful way to add detail to the surface of rendered objects. A solid texture is a three-dimensional procedural texture field. The surface is textured by 'placing' the object in the field, and obtaining a texture from the intersection of the surface of the object and the field. This is done by evaluating the procedural texture at the surface points. Solid texture can increase the aliasing artifacts because a pixel may project into a region of texture that contains many variations over the projected area. An approach to anti-aliasing is to filter the three-dimensional field over a small volume of texture space that contains the surface element, just as we filtered over a small area of two-dimensional texture space. The advantage of solid texture is that objects of arbitrary shape can be textured [90, 60, 47].

Hypertexture: In 1989 Perlin [63] extended the solid texture technique and developed a method that he called 'hypertexture'. One of the main distinctions between solid texture and hypertexture is that hypertexture objects have no well-defined boundaries. Instead they have a density function that describes how the object should behave in the area where it transitions between the outside and inside of the object. Perlin uses this approach to produce such effects as hair, fur, fire and erosion effects. This method is both a modelling and a texture technique.

Perlin also defined a noise function that takes a three-dimensional position as its input and returns a single scalar value. It can simulate turbulence and produce a surprising variety of realistic, natural-looking texture effects [62]. A single piece of noise can be put to use to simulate a remarkable number of effects. By far the most versatile of its applications is the use of the so-called turbulence function, as defined by Perlin [62], which takes a position x and returns a turbulent scalar value.

Cellular Texture: Worley [100] present a new basis function that complements Perlin noise, based on a partitioning of space into a random array of cells. He used this new basis function to produce textured surfaces resembling flagstone-like tiled areas, organic crusty skin, crumpled paper, ice, rock, mountain ranges, and craters. The new basis function can be computed efficiently without the need for precalculation or table storage.

Reaction-Diffusion: Turk [82] used a reaction-diffusion approach to do texture synthesis. Reaction-diffusion is a process in which two or more chemicals diffuse over a surface and react with one another to produce stable patterns. Reaction-diffusion can produce a variety of spot and stripe patterns, much like those found on many animals. Developmental biologists think that some of the patterns found in nature may be the result of reaction-diffusion processes. So a computer model can be textured by simulating a reaction-diffusion process on the surface of the model.

6.2.2 Contemporary Methods

Most recent work on texture synthesis can be put into three categories:

(i) Feature Matching

These kinds of methods use models such as pyramids and wavelets to catch the features of the texture and then generate a new image by matching the model. They try to do the recognition process by defining a statistical model to compute global statistics in feature space and sample images from the texture model directly.

Heeger and Bergen [36] used a steerable pyramid to analyse an input image and to catch a set of features in terms of histograms of filter responses. A new image can be synthesised by matching the histograms of these features. However they failed to capture relationships across scales and orientations and their method cannot get good results with more structured images.

De Bonet [16] improved Heeger's method by using a multi-resolution filter-based approach. He extended the use of steerable pyramids to consider interactions between different levels in the pyramids. It works better than [36] but will produce boundary artifacts if the input texture is not tileable.

Portilla and Simoncelli [65] use a more advanced synthesis procedure by decomposing the texture image to complex wavelets and synthesising a new image by matching the joint statistics of these wavelets. Their method is better than [36] and can catch global structures very well but fails with local patterns and some highly structured patterns.

All these approaches are very complex and need extra work on building the appropriate model and analysing the sample image. They always are good at some kinds of textures that have been specified in advance. However, it is hard to find a generic feature set that can describe all textures. These methods try to compute global statistics, which is difficult, so none of the feature matching algorithms can provide a completely satisfactory solution.

(ii) Markov Random Field (MRF)

Markov Random Field methods assume that a texture is "local" and "stationary". That means each pixel of a texture image is determined by a small set of neighbouring pixels and is independent of the rest of the image. This character

is the same for all the pixels and thus the image appears similar all over. The method works by estimating the local conditional probability density function (PDF) and synthesising pixels incrementally.

There are two approaches to MRF-based synthesis.

a. Pixel-based

The texture of an image can be expressed as interrelationships between pixels in that image. The synthesis process is to analyse and reproduce interactions between individual pixels. Pixel-based methods define a function to estimate the local conditional probability density of each pixel and generate a new image pixel by pixel. They are based on best-fit searching. Each pixel is selected by searching the input image for the patch of pixels that is most similar to the nearby pixels already synthesised in the output image.

Efros and Leung [26] developed a very good method by growing texture using non-parametric sampling. The neighbourhood of each pixel of the image is modelled as a square window around that pixel. The size of the window should correspond to the texture's stochastic feature. Put another way, the size of the window has to cover an area big enough to represent the texture's pattern. The next best-fit pixel, given its neighbours synthesised so far, is found by searching the sample image and finding all similar neighbourhoods. The difference between two windows is calculated by a normalised sum of squared differences. Efros's method produces good results for most kinds of texture images (Figure 6.1) but it is very slow. For some textures, it has a tendency to "slip" into a wrong part of the search space and start to grow garbage.

Wei and Levoy [93] improved Efros's method by using a multiresolution image pyramid based on a hierarchical statistical method. To generate each pixel in the output pyramid, a patch in the input pyramid similar to surrounding pixels in the current layer and the layers above is searched for. They also accelerated Efros's method by using tree-structured vector quantisation (TSVQ). Wei's approach also produces good result (as good as Efros's) and is much faster than Efros's. It has the same problem with growing garbage, even worse in some textures. (Figure 6.1).

Some related work has been done by Harrison [34] and Zhu et al. [109]. They are also based on MRF and generate pixels one by one.

b. Patch-based

More recent papers based on Efros and Leung [27] have much better performance. They all define a patch with size depending on the features of the texture. They try to find the best match of the whole patch, rather than of one pixel. The algorithms search for the best patch by comparing the overlap between patches synthesised so far and the new one. The new image is then generated patch-by-patch.

Xu et al. [102] proposed a texture synthesis algorithm based on random patch pasting. Their technique is a combination of traditional procedural methods and statistical sampling methods. The results of their algorithm are not as good as Efros's because their method does no statistical modelling or analysis and pastes blocks by randomly choosing from input images. It also has a problem with mismatching features across patch boundaries. However their method provides a new idea by generating texture patch-by-patch, and is thus much faster.

This idea is developed and modified by other researchers to get better results. Liang et al. [49] developed Xu's method and produced a real-time synthesis process by patch-based sampling. It searches all patches from the sample texture and picks a best match patch to generate new texture. It avoids mismatching features across patch boundaries by sampling texture patches according to the local conditional MRF density. Liang's method can re-synthesise high-quality texture images in real-time. It remains effective when pixel-based sampling algorithms fail to produce good results. It uses feathering blending in the boundary zones, thus providing a smooth transition between adjacent texture patches.

Efros and Freeman [27] improved their paper [26] and got a similar method to Liang's approach [49]. Efros's algorithm searches for the best-match patch like Liang's method but it reduces the mismatching feature across patch boundaries by making the minimum error boundary cut between two overlapping blocks.

Figure 6.2 shows some samples of different methods. They were implemented in C++ code on a Linux workstation with a Pentium 400 processor. The first

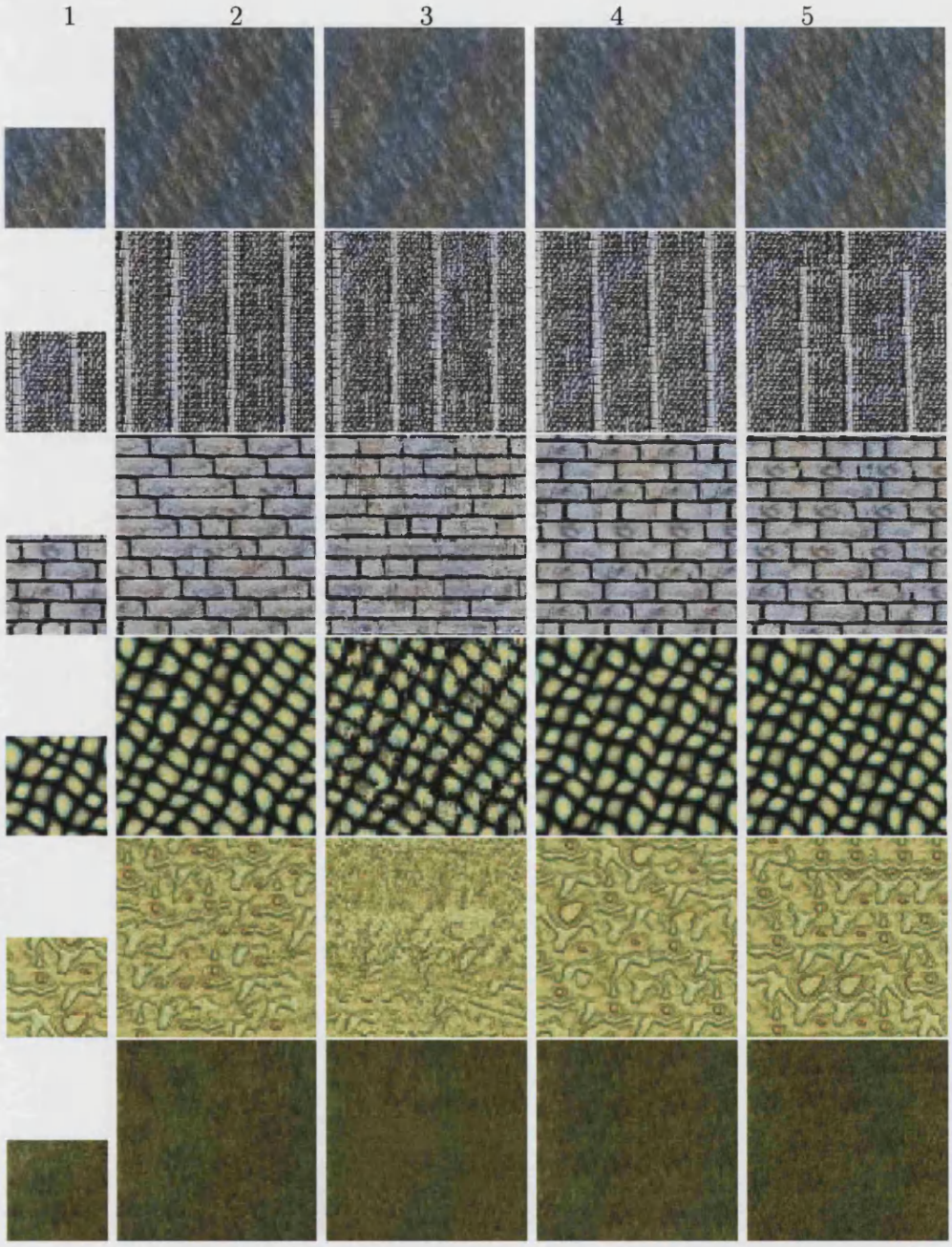


Figure 6.2: Some examples of texture synthesis, Column 1: sample texture, Column 2: Efros's non-parametric sampling. Column 3: Wei's pyramid. Column 4: Liang's patch-based. Column 5: Efros's image quilting.

column contains sample images. The second column are results synthesised by Efros's non-parametric sampling with weight window 23: computing times range from 33,195 secs to 35,196 secs. The third column are results synthesised by Wei's method using a 4-level pyramid with the biggest weight window 11: computing times range from 9,541 secs to 11,756 secs. The fourth column are results synthesised by Liang's method with block size 40, computing times are about 2 secs. The fifth column are results synthesised by Efros's image quilting algorithm with patch size 40: computing times are about 6 secs.

(iii) Texture Synthesis on Surfaces

Some papers appeared recently showed successful approaches to texture synthesis on 3-D surfaces. Wei [94] presented a method to synthesise general textures over arbitrary manifold surfaces. He extended his texture synthesis algorithm [93] by generalising the definition of searching neighbourhoods. For each mesh vertex, the method establishes a local parameterisation surrounding the vertex, uses this parameterisation to create a small rectangular neighbourhood with the vertex at its centre, and searches a sample texture for similar neighbourhoods. The solution is robust and is applicable to a wide range of textures.

Turk [83] independently developed an algorithm for texture synthesis on surfaces. A hierarchy of points from low to high density over a given surface is created and these points are connected to form a hierarchy of meshes. Then the user specifies a vector field over the surface that indicates the orientation of the texture. The mesh vertexes on the surface are then sorted such that visiting the points in order will follow the vector field and will sweep across the surface from one end to the other. Each point is then visited in turn to determine its colour. The colour of a particular point is found by examining the colour of neighbouring points and finding the best match to a similar pixel neighbourhood in the given texture sample. The colour assignment is done in a coarse-to-fine manner using the mesh hierarchy. His method fits the surface naturally and seamlessly.

Ying [104] described two synthesis methods, based on the work of Wei [94] and Ashikhmin [8]; the results are similar to these two, but directly on surface. The synthesised textures have the same qualitative visual appearance as the example texture, and cover the surfaces without the distortion or seams of conventional texture mapping.

6.3 Texture Synthesis and Texture Transfer using Particle Swarm Optimisation

We will now introduce a texture synthesis method proposed by Yan Zhang at Jilin University, China [108]. This method is based on patch-based sampling texture synthesis [49]. It uses particle swarm optimisation to search for the best match patches thus accelerating the synthesis process. It keeps the synthesis quality and is easy to extend to other applications such as texture transfer and constrained texture synthesis.

6.3.1 Texture Synthesis by Patch-Based Sampling

As we stated in section 6.2, the patch-based sampling method is a Markov Random Field based method and it synthesises new texture one patch at a time. Define the unit of synthesis B_k to be one of the square patches from the set S_B of all such patches in the input texture image I. The patch size $W_B \times W_B$ of the B_k will be decided by the user. It must be big enough to capture the relevant structure in the texture but small enough so that the interaction between these structures is left to the algorithm. To synthesise a new texture image, first randomly take a patch B_0 from S_B and paste it onto the left bottom of output image O. We now wish to find another patch to paste adjacent to it. Search S_B for patches that by some measure agree with their neighbours along the region of overlap in O. Figure 6.3 shows how to match a new patch and an already synthesised area. The dark area is already synthesised, the blue patch is the new patch to be synthesised and the area in dotted lines is the boundary zone. The boundary areas of the already synthesised area and the new patch should overlap and should be similar within some error tolerance.

The searching process is important to texture synthesis because it determines the new patch to be pasted onto the output image. In order to keep stochastic features, the algorithm forms a set P of patches so that the error in the overlap is within some error tolerance. Then we randomly pick one patch from set P and paste it onto the output image and repeat this process until all the image has been filled. After each patch has been chosen, we blend its boundary area to

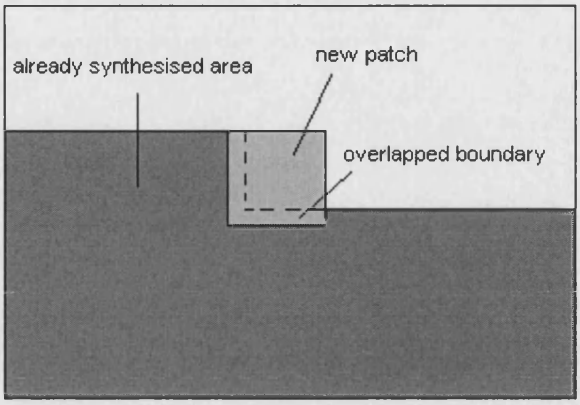


Figure 6.3: Patch-based texture synthesis

improve the synthesis quality.

The whole process is as follows,

1. Randomly pick a $W_B \times W_B$ patch B_0 from input texture I and paste B_0 on the lower left corner of output texture O.
2. Search I to get a set P of patches such that each patch's boundary matches the already synthesised area within some error tolerance.
3. Randomly choose one patch from P, paste it onto the O and blend its boundary area.
4. Repeat until O is filled.

The feathering blending [80] is used to provide a smooth transition between adjacent texture patches.

It is clear that forming the set P is the main computation load of this method. We need to search the set S_B of all $W_B \times W_B$ patches from I for patches whose boundary matches the already synthesised area. As Liang [49] mentioned, this search is essentially a k nearest search problem in the high dimensional space, which is computationally demanding. It is hard to find an algorithm better than brute-force search if we insist on getting the exact nearest neighbours. However, due to the special features of texture synthesis, it is acceptable that we can use

the approximate nearest neighbours which leads to many optimisation algorithms. Liang optimised his method with an optimised KD-tree, a quadtree pyramid and principal components analysis. This accelerated the method substantially but is very complex to implement.

6.3.2 Texture Synthesis using Particle Swarm Optimisation

(i) Particle Swarm Optimisation

The original idea of PSO was proposed by J.Kennedy and R.C.Eberhart [42]. It was discovered through simulation of a simplified social model. Numerous variations of the basic algorithm has been developed and are applied to many applications.

The basic algorithm of PSO involved forming a set of particles over the search space, each with an individual, initially random, location and velocity vector. The particles travel over the search space, remembering the best fit location experienced. During each iteration, each particle adjusts its velocity vector based on its momentum and the influence of its best location and the best location of its neighbours. Then it computes a new point to examine. Each particle tends to a local, non-optimal extrema. However, by each particle considering both its own memory and that of its neighbours, the entire swarm tends to converge on the global extrema.

The particle uses the following formulae to update its velocity and location:

$$V(t+1) = V(t) + rand() \times c1 \times (pBest(t) - present(t)) + rand() \times c2 \times (gBest(t) - present(t)))$$
$$present(t+1) = present(t) + V(t+1)$$

Here, t stands for time t, V(t) is the velocity vector, present(t) is the location factor. pBest(t) is the location vector for the best fitness the individual particle has yet encountered. gBest(t) is the global best fitness already encountered, which is the minimum of the pBest(t) of all particles. rand() is the random function

and c1, c2 are the *cognitive and social learning rates*, respectively. These two rates control the relative influence of the memory of the neighbourhood to the memory of the individual. Normally they are set to 2 [42].

(ii)Texture Synthesis using PSO

As mentioned we are not aiming at always finding the best match due to texture synthesis's special feature: synthesised texture should *look like* the sample texture and keep the randomness of textures. The PSO algorithm will either give us the best location or an approximate best location. Thus it is suitable for the texture synthesis search process as the approximate best location is good enough. Now we apply the PSO algorithm to texture synthesis.

Randomly set a number of positions in input image I and treat these points as virtual particles. These particles can be thought as virtual points of the image and each particle determines a patch by setting the left upper corner of the patch as the position of this particle. When the particles travel through the image we compare the patches they determine with the synthesised area and find the best-match patch (the best fit position of the particles). We will now explain in detail how to apply PSO searching in texture synthesis.

- *Fitness Function* Each particle will travel through I according to its location and velocity function. At each location we need to calculate the fitness of the current location. The fitness is then the distance between the patch A determined by particle and patch B which is the patch just synthesised (Figure 6.4). The formula for calculating fitness between two patches A and B is as follows:

$$d(A,B) = sqrt\{\sum_{i=1}^{k}\sum_{j=1}^{l}[(R(p_A^{ij})-R(p_B^{ij}))^2+(G(p_A^{ij})-G(p_B^{ij}))^2+(B(p_A^{ij})-B(p_B^{ij}))^2]\}$$

  where R(), G(), B() are the RGB value of the pixel. p_A^{ij} means the pixel at location ij in patch A and k, l means the width and height of the boundary area of the patch.

- *Attributes of the Particles* Texture synthesis is performed in 2D space, so we

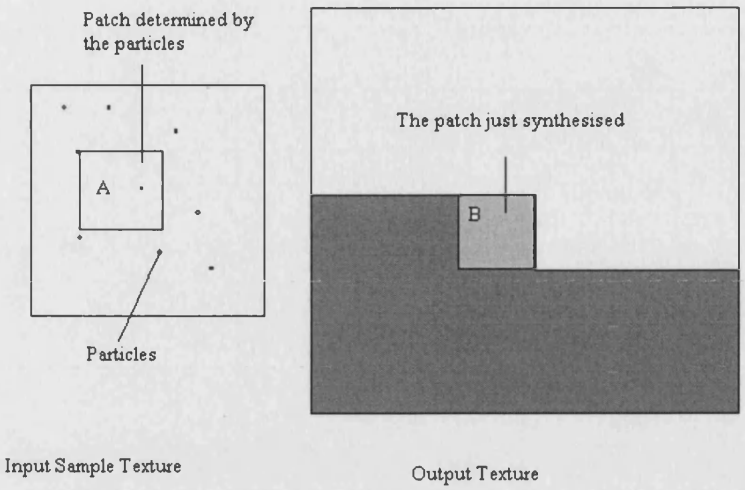


Figure 6.4: PSO based texture synthesis

update the position and velocity of the particles using the formula described in 6.3.2(i). In particular, we define n particles on the sample texture and record the best fitness location of each particle together with the global fitness which is the best fitness among all particles.

Each particle will be given an initial velocity vector $V_i = (V_{ix}, V_{iy})$. We define another two vectors $L_i = (L_{ix}, L_{iy})$ and $G_i = (G_{ix}, G_{iy})$

$$L_{ix} = LBestX_i - PresentX_i \qquad L_{iy} = LBestY_i - PresentY_i$$

$$G_{ix} = GBestX_i - PresentX_i \qquad G_{iy} = GBestY_i - PresentY_i$$

Now we update the particles using the formulae described in the last section.

The PSO algorithm is an iteration process. We will terminate the iteration when it exceeds the maximum iteration number defined by the user or the program finds the best location. In the former case it gives an appropriate location. A function *dmin* is used to determine the best location. If the difference between a patch and the synthesised area is less than *dmin* then we get the best location and stop the iteration.

$$dmin = \lambda sqrt(\sum_{k=1}^{A}(R(p_k)^2 + G(p_k)^2 + B(p_k)^2))$$

where A is the number of pixels in the boundary area and p_k represents the

value of the *kth* pixel in the boundary zone of the just synthesised patch.

Once we terminate the iteration we will then use the patch decided by the best global fitness location and paste the patch onto the output texture. λ is the error tolerance and it is set to 0.2 in our algorithm because, as Liang mentions, this error threshold is most suitable for keeping the randomness while ensuring synthesis quality and avoiding repetition. [49]

The algorithm can be stated as follows:

1. Randomly pick a $W_B \times W_B$ texture B_0 from input texture I and paste B_0 on the lower left corner of output texture O.
2. If the PSO algorithm found a best location then we paste the best location patch to O. Otherwise we terminate the iteration when it exceeds the maximum iteration number and we have found the best solution so far. Blend the corresponding patch and paste it to O.
3. Repeat until O is filled.

6.3.3 Synthesis Results and Algorithm Analysis

(i) Synthesis Results

Figure 6.5 gives some examples of the synthesis results using the PSO algorithm and Liang's patch-based sampling method [49]. These results are all generated by using 20 particles and an upper limit of 100 iterations. From visual inspection, our method is effective for both structure and stochastic textures. It is as good as the original patch-based sampling synthesis method.

(ii) Analysis of the Number of Particles and the Number of Iterations

It is understood that the number of particles in the input texture will affect the synthesis quality. If we have more particles on the searching space, it is

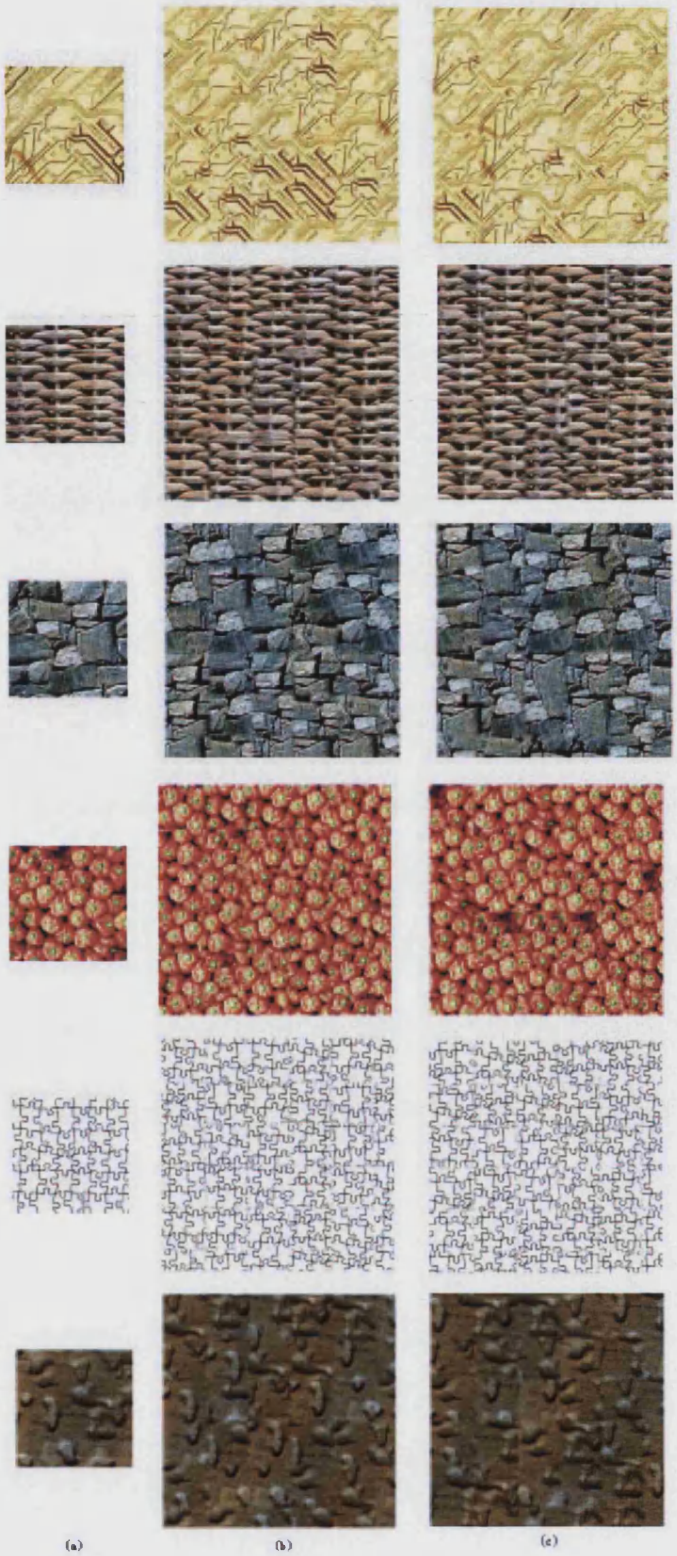


Figure 6.5: PSO based texture synthesis. Column (a) are input samples with size 100×100, column (b) are synthesised by patch-based sampling method and column (c) are our results using 20 particles and the 100 iterations. They both have size 200×200.

more likely the algorithm can find the best location. However, more particles also means longer computing time. It is a trade off between quality and time. We are aiming to find the number of particles which provides high quality while remaining efficient. We have tried different numbers of particles: 10, 20 and 30 particles on a 100×100 sample texture. Figure 6.6 shows the results.

The stochastic features of the synthesised texture are increased when the number of the particles increases. However the computing time is also increased. It takes 3, 8 and 13 seconds for using 10, 20 and 40 particles, respectively. There is no numerial or objective measurement tool to assess the texture quality yet. However from our subjective visual experiments from Figure 6.6 that using 20 particles is good enough for these sample textures with size 100×100. Obviously we have to use more particles if the input texture is big, and decrease the number of particles if input texture is small. And the particles are highly depend on the features of the texture, i.e. we have to use different number of particles for a structured texture such as wall and a stochastic texture such as grass.

When we are doing the synthesis, we either find the best solution or stop the iteration when it reaches the iteration limit. The number of iterations clearly affects the synthesis quality and the computing time. We tried 500, 350 and 100 iterations for 100×100 sample textures. Figure 6.7 shows different synthesis results with different iterations. They all use 20 particles. The computing time for 500, 350 and 100 iterations are 10, 7 and 3 seconds, respectively.

The number of iterations marginally changes the synthesis quality. However it significantly affects the synthesis speed. As we can see from Figure 6.5 and 6.7 with 20 particles for 100×100 sample textures, 100 iterations can generate good results. The number of iterations is also linear with the size of sample textures.

(iii) Performance Analysis

The PSO based texture synthesis is simple and fairly easy to implement. It doesn't require any analysis time in contrast to the optimised kd-tree (KD Tree), quadtree pyramid (QTP) and principal components analysis (PCA) in Liang's method. We use sample textures with size 100×100 and synthesise new textures with size 200×200. We have assessed the computing time and table 6.1 shows

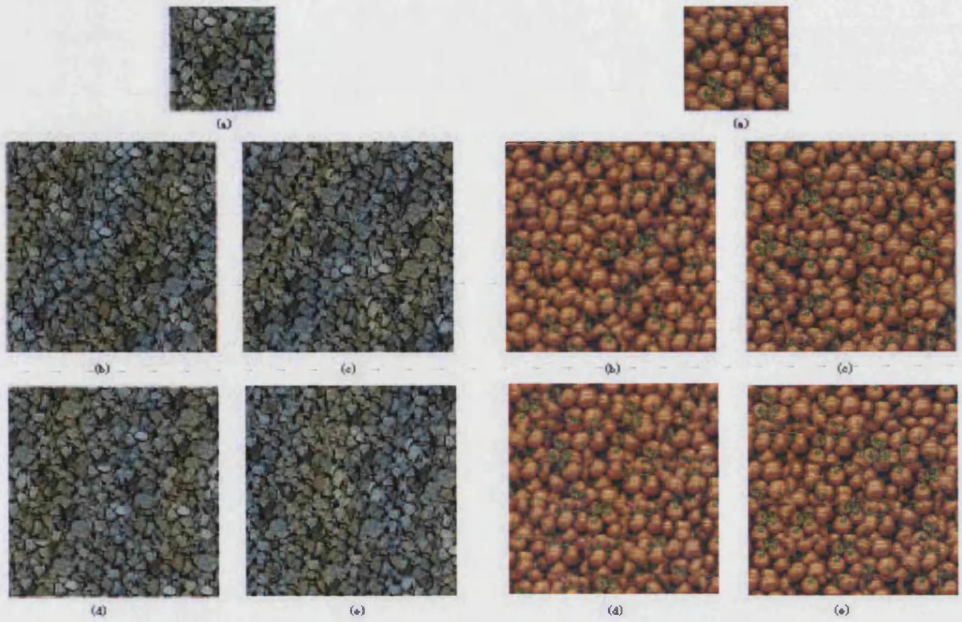


Figure 6.6: PSO based texture synthesis with different particles. (a) are the sample textures. (b) are the results generated by [49]. (c) uses 10 particles, (d) uses 20 particles and (e) uses 40 particles

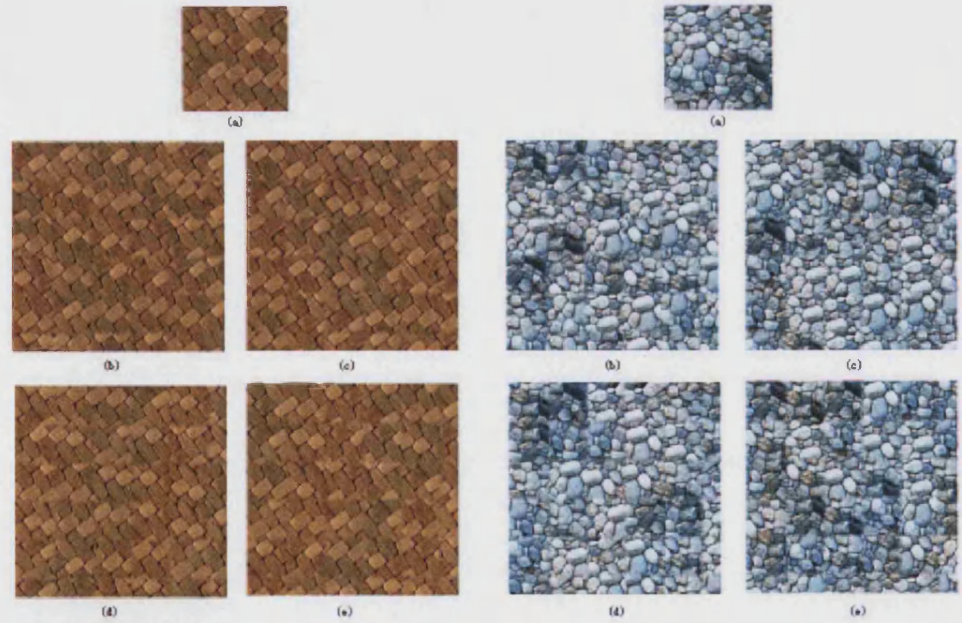


Figure 6.7: PSO based texture synthesis with different iterations. (a) are the sample textures. (b) are the results generated by [49]. (c) uses 500 iterations, (d) uses 230 iterations and (e) uses 100 iterations

Method	*Analysis Time*	*Synthesis Time*
Original Patch-based	0.00	13.1
Patch-based+KDTree+QTP+PCA	5.24	0.16
PSO based method	0.00	3.26

Table 6.1: Performance comparison

the performance comparison on a Dell 4100 Pentium 3 machine with 1G CPU. All the methods are implemented in C++ code and all the figures in the table are seconds. The PSO based method uses 20 particles and 100 iterations.

It is clear that our method using 100 iterations and 20 particles is simpler and faster than the patch-based method using QTP, KDTree and PCA.

6.4 PSO Based Texture Transfer

Texture transfer takes a sample texture and a picture as input and transfers the features of the texture to the picture so that the picture shows texture features. It can also be thought of as transferring the features of the picture to texture. In his paper [103], Xiaogang Xu did texture transfer based on Ashikhmin's method [8]. However his method synthesises textures one pixel at a time and thus is very computationally expensive. We now extend our PSO based synthesis method to do texture transfer.

We use a YIQ system [38]. The Y channel of the YIQ system represents the brightness information of the image and the IQ channel keeps the colour information. Our basic idea is blending the Y channels of texture and picture, then use the IQ information from the input picture (if we want to transfer texture to picture); or use IQ information from the input texture (if we want to transfer picture to texture). Once we get the new YIQ values, we can convert back to RGB and write to the output picture.

The YIQ and RGB system can be calculated from the following formula.

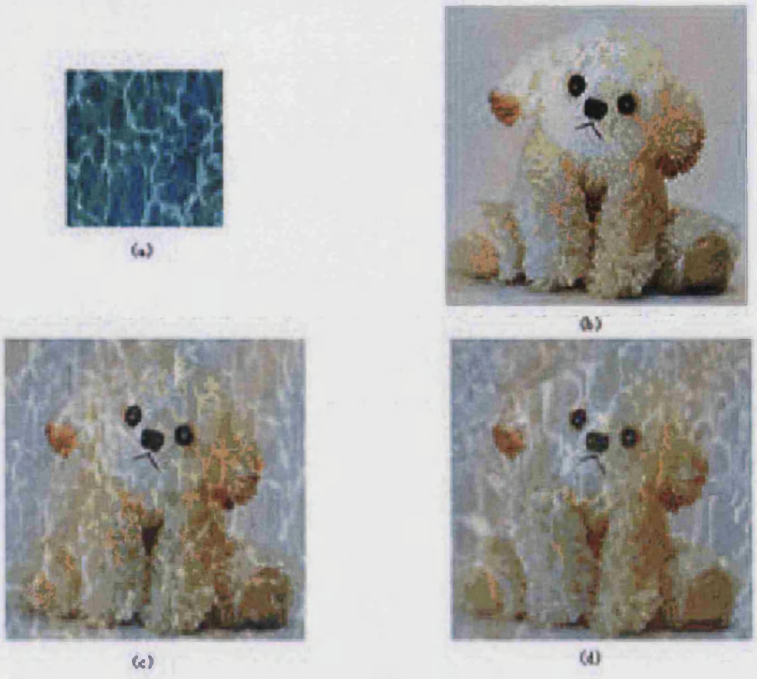


Figure 6.8: PSO based texture transfer: transferring texture to picture. (a) is sample texture, (b) is input picture (c) is our result and (d) is the result from [103]

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Firstly we synthesise a temporary texture T from the sample texture I. The temporary texture has the same size as the input picture. Then we convert the input picture P and temporary texture T to YIQ format. We can now mix the Y value of T and P, adjusting the Y value of T so that the picture is 'transfered' to the texture. We use the following formula to get the new Y value.

$$OutY = \lambda \times Y_T + (1 - \lambda) \times Y_P$$

where λ is a floating value between 0 and 1. Y_T means the Y value of the

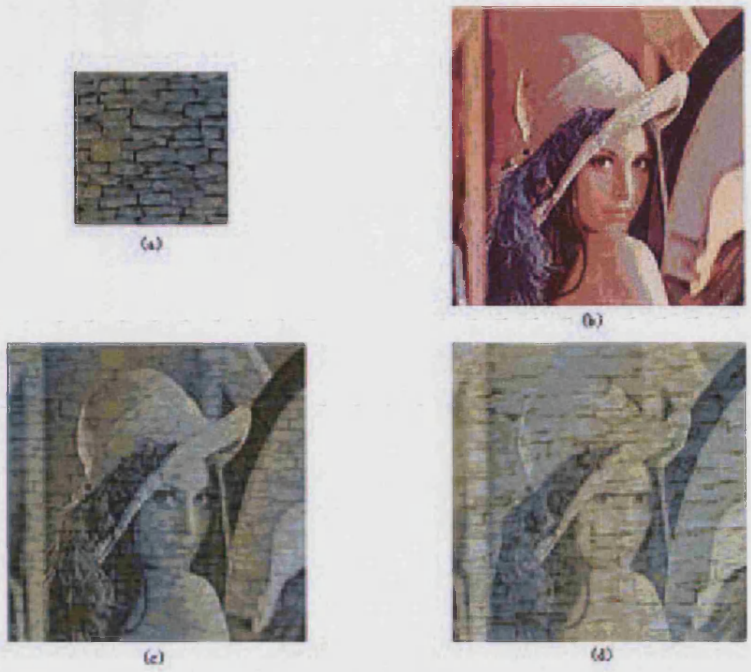


Figure 6.9: PSO based texture transfer: transferring picture to texture. (a) is sample texture, (b) is input picture (c) is our result and (d) is the result from [103]

temporary texture T and Y_P means the Y value of the input picture P.

We can adjust the λ value to control the output texture, making it look more like texture or picture. Once we get the mixed Y value, we can use the IQ value from either input picture P or temporary texture T, depending on whether we want to transfer texture to picture or transfer picture to texture.

The texture transfer process can be stated as follows:

1. Synthesise a temporary texture T from sample texture. T is the same size as the input picture P.
2. Convert P and T to YIQ format.
3. For each point of T, calculate the mixed Y value using the Y values from P and T.
4. Get the IQ value from either T or P together with the mixed Y value, calculate the RGB value and output to the output picture O.

Figure 6.8 and 6.9 shows two results. Figure 6.8 is transferring texture to picture which uses the IQ value of input picture and $\lambda = 0.4$. Figure 6.9 is transferring picture to texture which uses the IQ value of the temporary texture and $\lambda = 0.5$ Figure 6.8 and 6.9 (c) are our results and (d) are the results from [103]. It is clear that our results generate better transfer results due to our higher texture synthesis quality compared to their pixel based synthesis method.

6.5 Constrained Texture Synthesis

A normal texture synthesis technique synthesises textures in an image or a surface. The constrained texture synthesises textures on a specific area which can be used in many ways, e.g. photo repair, designing and image filling. Ashikhmin [8] proposed a constrained texture synthesis method using a single texture sample. However sometimes we need two or more textures or pictures as input. We have extended the PSO based texture synthesis method to be used in single and multi sample constrained texture synthesis.

We take $n + 1$ images as input: an input picture P and several texture samples $t_1, t_2, ..t_n$. Then we synthesise texture on different areas from different sample textures.

This is easy to do by using texture transfer. Basically we first synthesise n temporary textures of suitable size from the sample textures $t_1, ...t_n$. Then we transfer one of the temporary textures to a specific area according to the user's need.

We will first examine two-sample constrained texture synthesis. We take three images as input: one is an input picture P and the others are two texture samples $t1$ and $t2$. P has a clearly defined background and foreground so that they are filled by textures synthesised from $t1$ and $t2$. We first generate two temporary texture $T1$ and $T2$ from the two sample textures. They both have the same size as P. Then we take any point from P and set it as background. Call the Y value of this point y. For each pixel of P, we check whether the Y value of this pixel is within some tolerance of y. If it is, then we transfer texture $T1$ to this pixel, otherwise we transfer $T2$ to this pixel. We keep doing this until all the image

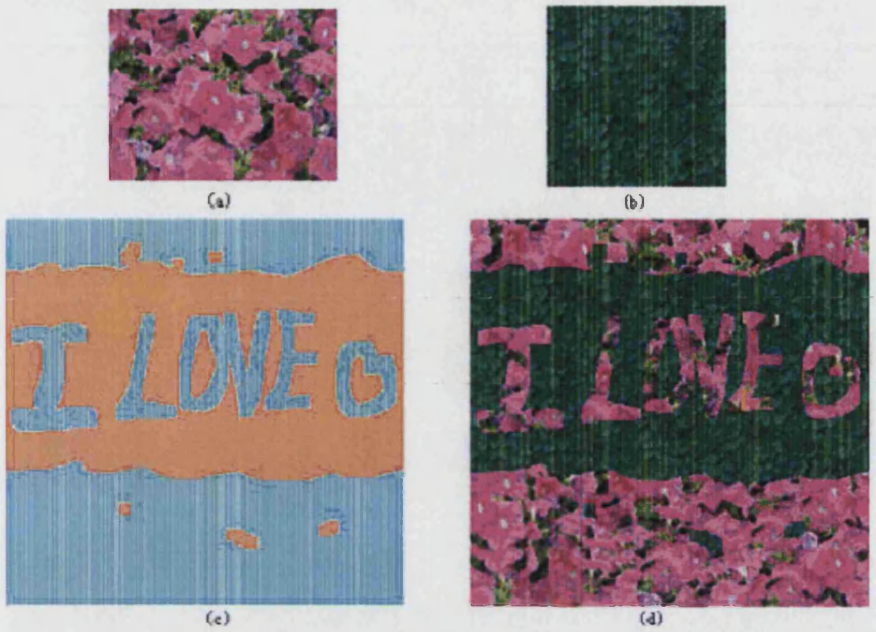


Figure 6.10: Constrained texture synthesis. (a),(b) are two sample textures, (c) is the target picture and (d) is the synthesised result.

has been filled. Figure 6.10, 6.11 and 6.12 shows three examples of constrained texture synthesis.

For multi sample images, the user shall define which areas need to be textured and which samples need to be used. While in two sample case we can define a background and foreground, multi sample cases require some technique to distinguish each special area specified by the user.

6.6 Perspective Texture Synthesis

All the methods presented generate results on planar surfaces or directly onto 3-D surfaces. Sometimes we need to synthesise texture on a planar surface that has been rotated for some angle from a perspective view. Texture synthesis on a 2D surface cannot achieve this while texture synthesis approaches on 3D surfaces are too complex for this task. We will present a method for synthesising perspective texture - a planar surface rotated and viewed in 3D space.

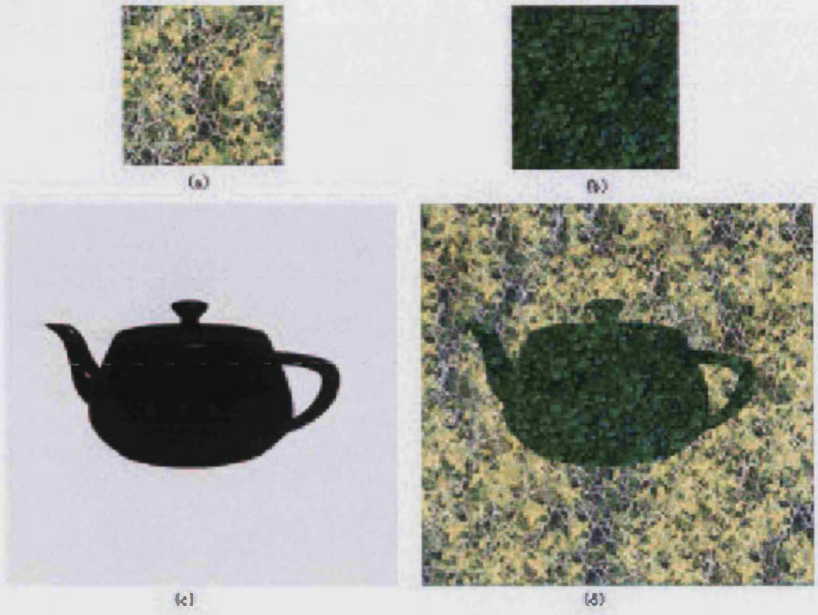


Figure 6.11: Constrained texture synthesis. (a),(b) are two sample textures, (c) is the target picture and (d) is the synthesised result.

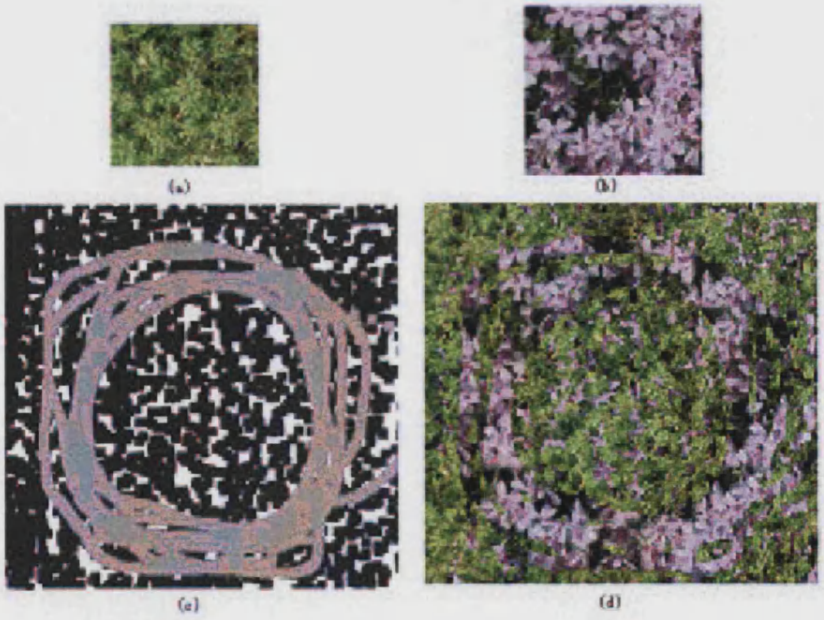


Figure 6.12: Constrained texture synthesis. (a),(b) are two sample textures, (c) is the target picture and (d) is the synthesised result.

The 2D surface in perspective is of particular interest because it is commonly found in film and game applications. For example, when you play the game "Tomb Raider", the walls and the ground are all 2D planes viewed in perspective. Traditional texture mapping is less realistic and introduces texture repetition. Normal texture synthesis in 2D cannot handle this situation. One possible solution is to synthesise a 2D plane texture and map it onto the rotated surface. However this is highly expensive in computing time and memory. Texture synthesis on a 3D surface is not suitable because this is a special case of a 3D surface which should offer a simpler solution. Our texture synthesis method is designed for this case. It avoids the complexity of synthesis method on 3D surface, it is effective and relatively fast. This method is also derived from the patch-based sampling method.

6.6.1 Extension to Perspective Texture

The patch-based sampling approach synthesises new texture only on 2D surfaces. Now if we consider there is a 2D plane in perspective view - a 2D surface viewed in 3D, the patch-based method cannot synthesise textures directly on this surface. Those methods [94, 83, 104] which synthesise texture on 3D surfaces are too complex for this situation because they involve 3D mesh generation and calculation.

We adjust Liang's patch-based method to be used in perspective projection. In principle, our method take a small sample texture as input and directly synthesises texture onto a 2D plane rotated in some angle in 3D space. The 2D plane can be of arbitrary size, rotated with arbitrary angle and viewed in perspective in 3D.

The user defines the rotation angles of the 2D plane in 3D space: α, β, γ; and the patch size w. We define the input texture image to be I, the output image O. The output image O shows the 2D plane rotated by the above angles in 3D space. We will also need to refer to the unrotated plane in 2D space: call this virtual plane V. Finally we need a temporary image T. T is without any rotation and only contains two rows of patches: one row of patches are already synthesised and the other row has the patches to be synthesised. When the latter is full, we swap the roles of these two rows.

The screen is defined at integer coordinate points but mapping each new patch onto the plane O produces image points that are not at integer coordinates. If the nearest displayable point is chosen, and this is repeated for all pixels in an area which the patch covers, the resulting image may have unset pixels in it. Moreover, if we generate perspective texture images, there is a many-to-one mapping, which leads to further loss of information.

Resampling [33] can solve this problem. Each patch after rotation will fall in a specific area on the output image. We call this area the target frame. Each pixel in the target frame is calculated by first checking whether it is rotated by the source patch and, if so, determining the source point from which it was rotated. Here inverse rotation working back from the target to source is used. The source point will not always have integer coordinates and therefore will fall in an area delimited by four pixels. A practical way to do resampling is to note the maximum and minimum x and y source coordinates of the patch. These four points are then rotated to give four target points, of which the maximum and minimum x and y coordinates determine a target rectangle. This rectangle encloses the rotated patch. The reverse rotation and interpolation process is repeated for each target pixel in the rectangle. The intensity of each target pixel is calculated by local interpolation. Bilinear interpolation is used here. It proceeds by forming a weighted average of the intensities of the four nearest neighbour pixels.

The complete algorithm is as follows:

1. We start by randomly picking one patch from I and pasting it onto T.
2. Go through the temporary image T in raster scan order in steps of one patch. As before, search the sample texture I for a set of patches that match, in their overlap with the patch in T (above and left) within some error tolerance (Figure 6.3). Randomly pick one such patch, blend the boundary area and paste it onto T.
3. Resampling. Find the axis-aligned bounding rectangle in O of the new patch. For all pixels within the bounding rectangle we perform an inverse mappping to V. If the pixel being inverse rotated is within the current virtual patch area of V, then we do a bilinear interpolation to get a value

from T and use this value to colour O. If it is outside, then this O pixel is not within the new patch, so we ignore it.

4. After one patch has been pasted, go to 2, repeat until the output image has been filled.

6.6.2 Experimental Results

The results of the synthesis process are shown in Figure 6.13 and 6.14. The top left is the original texture sample, top right is the synthesised image rotated along the Z axis by 30 degrees, the middle is the synthesised image rotated along the x axis by 45 degrees and the bottom is the synthesised image rotated along both the x axis and y axis by 30 degrees. The performance is effective for structured texture, and it is quite good for statistical textures as well. Figure 6.14 shows a highly statistical texture with the synthesised perspective textures. Consider the criteria we have set for successful texture synthesis progress. The qualities of the textures are determined by Liang's patch-based algorithm, which has been proved very good at a wide range of the textures. The new image looks similar to the sample image but with no excessive repetition. There are a few slightly mismatched or distorted boundaries, but the results are acceptable. From our observation, the quality of the texture is as good as most existing 2D algorithms, or even better.

The algorithm is also very fast. Unoptimised C++ code is used to generate these results on a Linux workstation with Pentium 3G processor. The top right image in figure 6.13 (192×192 for unrotated image) took 2.3 seconds and other images in figure 6.13 (320×320 for unrotated image) took 6.0 seconds each.

6.7 Conclusion

In this chapter we have made a survey of texture synthesis. We introduced some traditional and contemporary texture synthesis techniques and we implemented some of them for comparison. Most of these methods can be categorised into procedural texture or statistical sampling. Statistical methods are better than

Figure 6.13: Top left: original structured texture. Top right: synthesised image rotated along Z axis by 30 degrees. Middle: synthesised image rotated along X axis by 45 degrees. Bottom: synthesised image rotated along both X and Y axis by 30 degrees

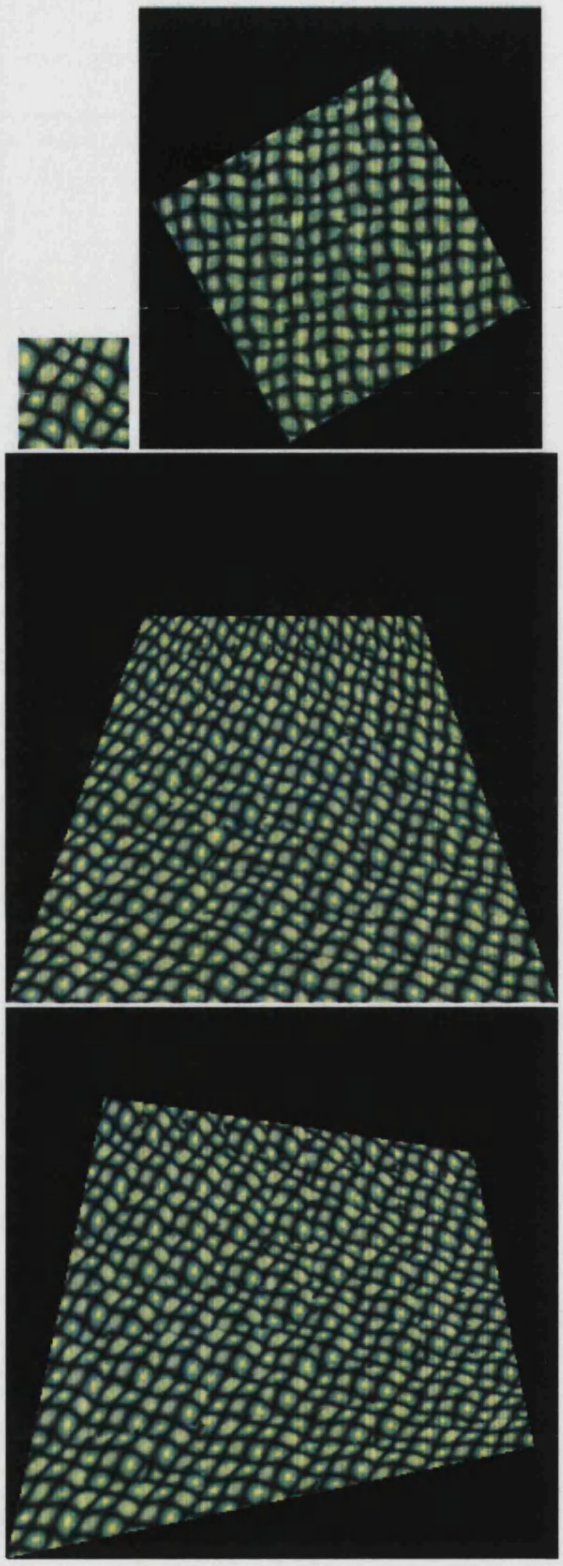


Figure 6.14: Top left: original statistical texture. Top right: synthesised image rotated along Z axis by 30 degrees. Middle: synthesised image rotated along X axis by 45 degrees. Bottom: synthesised image rotated along both X and Y axis by 30 degrees

procedural methods because the former produce higher quality results for general textures while the latter are only applicable to limited classes of texture. Statistical methods can be further categorised into feature matching or Markov Random Field. Methods based on the Markov Random Field (MRF) have been demonstrated to be effective for a broad range of textures. They use pixels directly from input images and paste pixels onto the new image under relationships among pixels. MRF directly makes use of two of the most important aspects of texture, pixels and their interrelationships.

We presented an efficient texture synthesis method. It is based on the patch-based sampling texture synthesis method. However we use a particle swarm optimisation for the searching process thus accelerating searching. It is simple, easy to implement, more efficient compared to other acceleration schemes and it is faster. Visual inspection shows that it generates results as good as the original method.

Moreover we have extended the algorithm to texture transfer. It transfers texture features to pictures or inserts a picture into a texture. We have used the PSO based texture synthesis and the YIQ system and they produced better results than previous methods.

We have implemented multi-sample constrained texture synthesis. We take several sample textures and one picture as input and then synthesise textures from different sample textures into different areas of the picture. The algorithm is very effective.

Additionally, we have presented an efficient method for synthesising a perspective texture from a 2D example texture. The method produces textures with similar quality and speed to their 2D counterpart (Liang's patch-based sampling). This means that those textures that work well with Liang's algorithm also work well with our algorithm. The algorithm produces high quality synthesised images very rapidly.

The sample images we used here are all square images. For some samples from scanned photographs, it is very hard to get square samples. The samples would be rotated by an angle and we can only see the perspective pictures. For future work, we are interested in extending the methods presented here for perspective

texture synthesis from rotated perspective samples.

The user has to specify the number of particles and the maximum iteration number according to the size of the input texture sample in the PSO based texture synthesis method. More research needs to be done to expose the relationship between the sample size and the particle number and the maximum iteration number.

Our future work focuses on extending the algorithm into 3D texture synthesis and dynamic texture synthesis which we see great as a potential of this algorithm.

Chapter 7

Conclusions and Future Work

This thesis addresses a fundamental problem of digital image processing: to find an appropriate representation of digital images which provides a link between continuous images and digital ones. It is the foundation of all other image-related applications. However it still remains a challenging problem to researchers and this is the motivation of our image model. The intuition behind our image representation model is the importance of the roles edges play in images. The contribution of this thesis is that it provides an appropriate tool to exploit fundamental properties of edges and to represent digital images well so that efficient and effective practical applications can be drawn from this model. We also study the texture synthesis problem. We present a survey of texture synthesis and introduce a novel texture synthesis method using particle swarm optimisation for patch-based texture synthesis. We extend this method to texture transfer, constrained multi-sample texture synthesis and perspective texture synthesis.

7.1 Contribution

We propose a pixel level data-dependent triangulation image model. The image is triangulated by a triangulation mesh and the edges are represented by the diagonals of the triangles. The main strength of this model is it represents the orientations of edges and thus keeps the most visual important feature of images. The main advantages of this model are its simplicity and efficiency. It is a generic

model and is effective for all images.

This model allows various applications to use this representation and to recover continuous intensities from discrete image data samples. We have examined several important applications such as arbitrary resolution enhancement, arbitrary rotation, demosaicing of digital colour images and other applications of still images in continuous space. The simplicity of the underlying model leads to simple, effective and efficient applications in those different areas.

Conventional approaches to represent image edges try to detect edge orientations using various techniques. The drawbacks of these approaches are they are often complex and are not robust for all images. Moreover, it is difficult to employ their models in different applications. This thesis provides another direction to image representation and image analysis. Our model does not try to detect a long-range edge or attempt to find a statistical rule for the local geometry. It simply triangulates each four-pixel square according to the intensity of the four pixels or a bigger neighbour window. By doing this in every local square, the global edges of the images are well presented.

We have presented a basic model and an extended model by considering the local neighbouring information to represent local edges. Algorithm analysis shows both have $O(n)$ time complexities which are linear with the image size and similar to bilinear interpolation.

At heart, out model recognises that simple interpolation suffices, provided it covers a sufficiently small area. Any errors in the interpolation is strictly limited by the tiny triangles and the choice of diagonal.

We have used this model in the following applications.

7.1.1 Image Interpolation

The model is applied to image interpolation and in particular: arbitrary magnification, arbitrary rotation and other manipulations on still images. The images are first triangulated by our basic or extended model, then we interpolate from the triangulation mesh. The triangulation mesh generated corresponds to the edges

of the image so that the algorithm will always interpolate along the edge but not across it. Thus the interpolated image will keep the edge sharp while retaining smoothness along the edge. Visual inspection suggests that it generates better results than the traditional methods, e.g. bilinear and bicubic interpolation. It is as simple and efficient as bilinear interpolation.

We also examined the interpolated image quality by using mean square error (MSE). MSE is a fidelity measure between the interpolated image and the original image. The objective MSE results confirm this model is effective.

This model can be implemented in hardware. Thanks to the rapidly-improving technology, a graphics card can now handle tens of millions of triangles per second and interpolate within the triangles. This means our triangulation mesh can be stored and manipulated by graphics card in real-time. We have demonstrated a hardware implementation by using OpenGL and we are pleased to see high-quality real-time image interpolations. This leads to great industrial potential of this model.

7.1.2 Demosaicing of Colour Images

Mosaic images have only one primary colour (R, G or B) in each pixel. Digital cameras use such a moscaic and so the "demosaicing" process is essential to get full colour photographs.

We modified our basic model to use a colour-difference space because it explains the correlation between different colour channels. Because our model can tune the interpolator along the edges, it avoids the colour mis-registration in edge areas which traditional linear interpolation suffers. Visual and MSE inspection show our model gives superior reconstruction quality. It is also very fast.

7.1.3 Texture Synthesis

This research started by studying texture and texture representation and later widened to image representation. The texture work represented here is therefore

in the discrete domain. Even here however, we use interpolation in small areas, in keeping with our general approach. Texture has an excessive number of edges and it has some special applications such as mapping and synthesis rather than interpolation.

We have made a detailed survey of texture synthesis and compared different methods. The patch-based Markov Random Field methods are so far the best ones. We extended the patch-based sampling method into perspective which means texture synthesis can be done in two and a half dimensional space.

Another experiment done together with Yan Zhang at Jilin University, China is extending their PSO based texture synthesis method into texture transfer and constrained texture synthesis. The PSO based texture synthesis is derived from the patch-based sampling method and it accelerates the searching process which reduces the computing time. We have used the PSO based texture synthesis for texture transfer (which transfer a picture to a texture) and multi-sample constrained texture synthesis (which synthesises texture on different constrained areas from different samples).

7.2 Future Work

There are a number of directions in which the work of this thesis can be continued.

- **Interpolation in other colour spaces** Our method performs very well in full RGB colour space, however it would have been interesting to do some of the interpolation in colour difference space, as we introduced in the demosaicing problem in chapter 5. Colour difference space considers the correlation between colour channels which might improve the interpolation quality.

  We are also considering performing interpolation in the YIQ colour space which we introduced in chapter 6. In principle our method triangulates the image according to the luminance of the image. The Y channel of the YIQ colour space represents the brightness and IQ channel keeps colour information.

- **CMYK Colour Printing** The CMYK (cyan, magenta, yellow, black) colour space is commonly used in colour printers. Cyan, magenta, and yellow are the complements of red, green, and blue, respectively. Mixing cyan, yellow, and magenta produces black. To maintain black colour purity, a separate black ink is used rather than printing three colour inks to generate it. However, the density of CMYK inks cannot be varied continuously across an image, so a range is produced by halftones. In halftones, translucent CMYK ink dots of variable size are printed in overlapping grids. Each grid is placed at a different angle for each ink colour. Smaller halftone dots mean more reflected light from the white paper and thus lighter appearance in that colour. By printing different sizes of halftone dots of different colours and mixing them together, a full colour image can be produced by printers.

  Our image model can be applied to a more accurate CMYK colour printing. The size of the halftone dots depends on how much colour it needs in that particular location. Because halftone grids are placed with different angles, each halftone dot will be in floating point coordinates in the sample image. Therefore it is possible to use our model to interpolate the value of the halftone dots and get the colour amounts. As our model avoids interpolating across edges, the interpolated halftone dots might reducing colour blurring across the edge thus improving printing quality compared to the traditional bilinear interpolation.

- **Pixel Ordering** Many image related algorithms work on a linearised version of the digital images, assuming a strong coherence among nearby pixels. The scan-line order, where the pixels are traversed horizontally line by line, is the most common one. However, the spatial coherence among the nearby pixels is typically anisotropic rather than directional along the horizontal lines. Other scan methods such as the Peano-Hilbert space filling curve have been proposed to take advantage of the local similarities inherent in images. The sequence of pixels visited along the scan order is called pixel ordering. A good choice increases the autocorrelation of the resulting pixel sequence and thus increases the lossless image compression rates.

  Our image model might be applied to the pixel ordering problem. As we mentioned, our extended model considers the local geometry information and triangulates the four-pixel square. Careful study of the triangulation mesh reveals that the triangles tend to cluster together representing areas

along the edge in the image. Thus it is possible to find the underlying local similarities by following some rules defined by our image model. It is clear that pixels along edges have more similarities than pixels across edges. If a scan order along the edge can be applied according to the triangulation mesh, it might increase the lossless image compression rates.

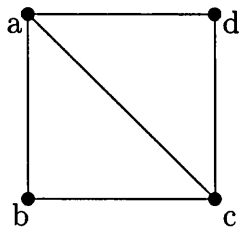
7.3 Conclusion

The author intended to tackle the fundamental problem of digital image processing and tried to find a good representation of digital images which is generic for all kinds of images and is effective for other digital image applications. This leads to the research in this thesis which presents an effective image model to exploit fundamental properties of edges in the images using pixel level data-dependent triangulation. It is not only a better representation than other approaches but also much more efficient and is generic for different kinds of images.

Appendix A

Proof

Consider a four pixel square *abcd*. We will first prove that, if pair *ac* has smaller difference than *bd*, then b or d is the outlier pixel and we should connect *ac*. That is to say, if $|a-c| < |b-d|$ then b or d is either the biggest or the smallest pixel.



Suppose $|a-c| < |b-d|$, and suppose $a \geq c$, then $a - c < |b-d|$.

1. Suppose $b > d$. Then $a - c < b - d$ $(b > d, a \geq c)$, hence $a - b < c - d$ $(b > d, a \geq c)$.

   We suppose $a > b$ and $c < d$, then $a - b > 0$ and $c - d < 0$, so we get $a - b > c - d$. However, we have the formula $a - b < c - d$ before which means our assumption that $a > b$ and $c < d$ is wrong.

   Because $a > b$ and $c < d$ is wrong, either $a < b$ or $c > d$ or $a < b, c > d$ with the condition $(b > d, a \geq c)$. In these cases, either b is the biggest pixel $(b > a, b > c, b > d)$ or d is the smallest pixel $(d < c, d < a, d < b)$.

2. Suppose $b < d$, then $a - c < d - b$ $(b < d, a \geq c)$, hence $a - d < c - b$ $(b < d, a \geq c)$.

We suppose $a > d$ and $c < b$. Then $a - d > 0$ and $c - b < 0$, so we get $a - d > c - b$. However, we have the formula $a - d < c - b$ before which means our assumption that $a > d$ and $c < b$ is wrong.

Because $a > d$ and $c < b$ is wrong, either $a < d$ or $c > b$ or $a < d, c > b$ with the condition $(b < d, a \geq c)$. In these cases, either b is the smallest pixel $(b < c, b < a, b < d)$ or d is the biggest pixel $(d > b, d > a, d > c)$.

We have proved that if pair ac has the smaller difference $(|a - c| < |b - d|)$, there are two situations. One is that either b is the biggest pixel or d is the smallest pixel. The second is that either b is the smallest pixel or d is the biggest pixel. In either case the outlier is either b or d and ac should be the edge. Using the same method we can prove that if pair bd has the smaller difference $(|b - d| < |a - c|)$, the outlier is either a or c and bd should be the edge.

So we can conclude that drawing the edge between the least-different diagonal pair gives the same result as drawing the edge which isolates the outlier.

References

[1] J.E.Adams, Jr., "Interactions between color plane interpolation and other image processing functions in electronic photography" *Proc. of SPIE*, Vol. 2416, pp. 144-151, 1995

[2] J.E.Adams, Jr. "Design of practical colour filter array interpolation algorithms for digitacameras" *Proc. of SPIE*, Vol. 3028, pp. 117-125, 1997

[3] A.Aldroubi and M.Unser, "Sampling procedures in function spaces and asymptotic equivalence with Shannon's sampling theorem", *Numerical Function Analysis and Optimization*, Vol. 15, pp. 1-21, 1994

[4] J.P.D'Ales and A.Cohen, "Non-linear approximation of random functions", *SIAM Journal of Applied Math.*, pp. 518-540, 1997

[5] V.R.Algazi, G.E.Ford and R.Potharlanka, "Directional interpolation of images based on visual properties and rank order filtering", *Proceeding of International Conference on Acoust, Speech Signal Processing*, pp. 3005-3008, 1991

[6] J.Allebach and P.W.Wong, "Edge-directed interpolation", *ICIP' 96*, Vol. 3, pp. 707-710, 1996

[7] L.Alvarez, P.L.Lions and J.M.Morel, "Image selective smoothing and edge detection by nonlinear diffusion. II", *SIAM J. Numerial analysis*, Vol. 29, No. 3, pp. 845-866, 1992.

[8] M.Ashikhmin, "Synthesizing natural textures", *2001 ACM Symposium on Interactive 3D Graphics*, pp. 217-226, 2001

[9] B.Ayazifar and J.S.Lim, "Pel-adaptive model-based interpolation of spatially sub-sampled images", *Proceeding of Intl. Conf. on Acoust. Speech and Signal Processing*, Vol. 3653, pp. 181-184, 1992

[10] M.Barnsley, "Fractals everywhere", Academic Press, New York, 1998

[11] S.Battiato, G.Gallo, F.Stanco, "A locally-adaptive zooming algorithm for digital images", *Image and vision Computing*, Vol. 20, No. 11, pp. 805-812, September 2002

[12] S.D.Bayrakeri and R.M.Mersereau, "A new method for directional image interpolation", *proc. Int. Conf. Acoustics, Speech, Sig. Process*, Vol. 4, pp. 2383-2386, 1995

[13] M.Bern and D.Eppstein, "Mesh generation and optimal triangulation", *Computing in Euclidean Geometry*, Lecture Notes Series on Computing, Vol 1, pp. 23-90, World Scientific, Singapore, 1992

[14] J.Blinn, and M.Newell, "Texture and reflection in computer generated images", *Communications of the ACM 19, (1976)*, pp. 542-547, 1976

[15] R.M.Bolle and D.B.Cooper, "Bayesian recognition by approximating image intensity functions with quadratic polynomials", *IEEE Trans. Patt. Anal. Mach. Intell.*, Vol. 6, pp. 418-429, 1984

[16] J.S.D Bonet, "Multiresolution sampling procedure for analysis and synthesis of texture images" *Computer Graphics (1997) ACM SIGGRAPH* , pp. 361-368, 1997

[17] J.Brown, "Vertex based data dependent triangulations", *Computer Aided Geometric Design*, Vol. 8, No. 3, pp. 239-251, 1991

[18] P.Burger and D.Gillies, "Interactive Computer Graphics - Functional Procedural and Device-level Methods", *Addison-Wesley*, pp. 411, 1989

[19] S.Carrato, G.Ramponi and S.Marsi, "A simple edge-sensitive image interpolation filter", *Proceeding of International Conference on Image Processing*, pp. 711-714, 1996

[20] D.R.Cok, "Signal processing method and apparatus for producing interpolated chrominance values in a sampled colour image signal", U.S. Patent No. 4, 642, 678 (1987)

[21] A.Cohen et al., "On the importance of combining wavelet based nonlinear approximation with coding strategies", Preprint 2000.

[22] F.Crow, "Summed-area tables for texture mapping", *Computer Graphics (1984)* ACM. SIGGRAPH, Vol. 18, pp. 207-212., 1984

[23] S.Daly, "The visible differences predictor: An algorithm for the assessment of image fidelity." *Digital Images and Human Vision*, A.Watson, Ed. Cambridge, MA:MIT Press, 1993

[24] H.Derin and H.Elliot, "Modelling and segmentation of noisy and textured images using Gibbs random field", *IEEE TRans. Pattern Anal. Mach. Intell.*, Vol. 9, pp. 39-55, 1987

[25] N.Dyn, D.Levin, and S.Rippa, "Data dependent triangulations for piecewise linear tnterpolation", *IMAJ. Numerical Analysis*, Vol. 10, pp. 127-154, 1990.

[26] A.Efros and T.Leung., "Texture synthesis by non-parametric sampling", *International Conference on Computer Vision*, volume 2, pp. 1033-8, 1999.

[27] A.A.Efros, William T. Freeman, "Image quilting for texture synthesis and transfer", *Computer Graphics (2001) ACM SIGGRAPH*, pp. 341-346, 2001

[28] A.Fournier, D.Fussel, and L.Carpenter, "Computer rendering of stochastic models", *Communications of the ACM 25*, pp. 25-39, 1982

[29] W.T.Freeman, "Median filter for reconstructing missing colour samples", U.S. Patent No. 4, 724, 395 (1988)

[30] A.F.Gamasutra "Run time mip-map filtering" *Game Developer Magazine*, vol. 2, Issue 48, CMP Media LLC, 1998

[31] S.Geman and D.Geman, "Stochastic relaxation, Gibbs distributions and Bayesian restoration of images", *IEEE Trans. Patt. Anal. Mach. Intell.*, Vol. 6, pp. 721-741, 1984

[32] J.F.Hamilton and J.E.Adams, "Adaptive color plane interpolation in single sensor colour electronic camera", U.S. Patent No. 5, 629, 734, (1997)

[33] J.B.Hanson and P.J.Willis, " A method of rotating areas on a raster scan graphic display", *Displays*, Butterworth & Co (Publishers) Ltd, 1982

[34] P.Harrison., "A non-hierarchical procedure for re-synthesis of complex textures", *WSCG 2001 Conference proceedings*, pp. 190-197, 2001.

[35] J.P.Havlicek et al. "Skewed 3D Hilbert transformations and computed AM-FM models", *Proceeding of International Conference on Image Processing*, Chicago, Oct. 1998

[36] D.J.Heeger and J.R.Bergen, "Pyramid-based texture analysis/synthesis", *Computer Graphics (1995), ACM SIGGRAPH*, pp. 229-238, 1995

[37] K.Jensen and D.Anastassiou, "Subpixel edge localisation and the interpolation of still images", *IEEE Trans. Image Process*, Vol. 4, No. 3, pp. 285-295, 1995

[38] Y.D.Jia, "Machine vision (in Chinese)", *Technology Press*, pp. 154-155, China, 2000

[39] R.L.Joshi, et. al. "Comparison of different methods of classification in subband coding of images", *IEEE Trans. Image Processing*, Vol. 6, pp. 1473-1486, Nov. 1997

[40] B.Julesz, "Visual pattern discrimination", *IRE Trans Info Theory*, IT-8(2): 84-92, 1962.

[41] N.B.Karayiannis and A.N.Venetsanopoulos, "Image interpolation based on variational principles", *Signal Process*, Vol. 25, pp. 259-288, 1991

[42] J.Kennedy and R.C.Eberhart, "Particle swarm optimization", *Proc. IEEE int'l conf. on neural networks*, Vol. IV, pp. 1942-1948, IEEE, 1995

[43] R.Kimmel, "Demosaicing: Image reconstruction from colour CCD samples", *IEEE Trans. Image Processing*, Vol. 8, pp. 1221-1228, Sept. 1999

[44] C.A.Laroche and M.A.Prescott, "Apparatus and method for adaptively interpolating a full colour image utilising chrominance gradients", U.S. Patent No. 5, 373, 322 (1994)

[45] S.W.Lee and J.K.Paik, "Image interpolation using adaptive fast B-spline filtering", *Proceeding of Interpolation Conference on Acoust. Speech Signal Processing*, pp. 177-180, 1993

[46] T.M.Lehmann, C.Gonner, K.Spitzer, "Survey: Interpolation methods in medical image processing", *IEEE Transactions on Medical Imaging*, Vol. 18, No. 11, pp. 1049-1075, November 1999

[47] J.Lewis, "Algorithms for solid noise synthesis", *Computer Graphics (1989), ACM SIGGRAPH*, vol 23, pp. 263-270, 1989

[48] X.Li, "Edge directed statistical inference with applications to image processing", *Ph.D Thesis*, Princeton University, May 2000

[49] L.Liang, C.Liu, Y.Xu, B.Guo and H-Y.Shum, "Real-time texture synthesi by patch-based sampling", *Technical Report MSR-TR-2001-40, Microsoft Research*, 2001

[50] M.S.Longuet-Higgins, "Statistical properties of an isotropic random surface", *Phil. Trans. Royal Soc.*, London, A, 250, pp. 151-171, 1957

[51] S.LoPresto, K.Ramchandran and M.Orchard, "Image coding based on mixture modelling of wavelet coefficients and a fast Estimation-Quantisation framework", *Proceeding of Data Compression Conference*, pp. 221-230, March 1997

[52] R.Malladi and J.A.Sethian, "A unified approach to noise removal, image enhancement and shape recovery", *IEEE Trans. on Image Processing*, Vol. 5, No. 11, pp. 1554-1568, Nov. 1996

[53] S.Mallat, "A wavelet tour of signal processing", Academic Press, 1998

[54] B.Mandelbrot, "The fractal geometry of nature", *W.H.Freeman*, SanFrancisco, 1982

[55] P.A.Maragos, R.W.Shafer and R.M.Mersereau, "Two dimensional linear prediction and its applications to adaptive predictive coding of images", *IEEE Trans. Acoust. Speech and Signal Processing*, Vol. 32, pp. 1213-1229, 1984

[56] S.A.Martucci, "Image resizing in the discrete Cosine transform domain", *Proc. Int. Conf. Image Processing*, Vol. 2, pp. 244-247, 1995

[57] B.S.Morse and D.Schwartzwald, "Isophote-based interpolation", *Proc. IEEE Int. Conf. Image Processing*, Vol. 3, pp. 227-231, 1998

[58] B.S.Morse and D.Schwartzwald, "Level-Set image reconstruction", *Proc. Computer Vision and Pattern Recognition 2001 (CVPR'01)*, pp. 333-340, IEEE 2001

[59] A.N.Netravali and B.G.Hasskell, "Digital pictures: representation, compression and standards", 2nd Ed., New York:Plenum Press, 1995

[60] D.R. Peachey. "Solid texturing of complex surfaces", *Computer Graphics (1985) ACM of SIGGRAPH*, vol 19, pp. 279-86,, 1985

[61] S.C.Pei and I.K.Tam, "Effective colour interpolation in CCD colour filter array using signal correlation" *Proc. IEEE Int. Conf. Image Processing*, Vol. 3, 2000, pp. 488-491

[62] K.Perlin, "An image synthesizer", *Computer Graphics(1985) ACM SIGGRAPH*, vol 19, pp. 287-296, 1985

[63] K.Perlin, "Hypertexture", *Computer Graphics (1989) ACM SIGGRAPH*, vol23, pp. 253-62, 1989

[64] P.Perona and J.Malik, "Scale-space and edge detection using anisotropic diffusion", *ITTT TRans. on Patt. Anal. and Mach. Intell.*, Vol. 12, No. 7, pp. 629-639, July 1990

[65] J.Portilla and E.P.Simoncelli, "Texture representation and synthesis using correlation of complex wavelet coefficient magnitudes", *TR 54, CSIC*, vol 40, pp. 49-71, Madrid, 1999

[66] J.Portilla and E.Simoncelli, "A parametric texture model based joint statistics of complex wavelet coeficients", *International Journal of Computer Vision*, vol 40, issue 1, pp. 49-71, 2000

[67] E.Quak and L.Schumaker, "Cubic spline fitting using data dependent triangulations", *Computer Aided Geometric Design*, Vol. 7, pp. 293-302, 1990

[68] R.Ramanath, "Interpolation methods for the bayer colour array", *MS thesis*, North Carolina State University, Raleigh, NC (2000)

[69] R.Ramanath, et. al. "Demosaicking methods for Bayer color arrays", *Journal of Electronic Imaging*, Vol. 11, No. 3, pp. 306-315, July 2002.

[70] K.Ratakonda and N.Ahuja, "POCS based adaptive image magnification", *Proceeding of Intl. Conf. on Image Processing*, Vol. 3, pp. 203-207, 1998

[71] L.Rila and A.G.Constantinides, "Image coding using data dependent triangulation", *Proc. 1997 13th Int'l Conf. Digital Signal Processing*, Part2, pp. 531-534, IEEE Press, Piscataway, N.J., 1997

[72] S.Rippa, "Long and thin triangles can be good for linear interpolation", *SIAM Journal on Numerical Analysis*, 29(1):257-270, February 1992.

[73] T.Sakamoto, C.Nakanishi and T.Hase, "Software pixel interpolation for digital still cameras suitable for a 32-Bit MCU" *IEEE Trans. Comsumer Electronics*, Vol. 44, No. 4, pp. 1342-1352, Nov. 1998

[74] L.L.Scarlatos, "A refined triangulation hierarchy for multiple levels of terrain detail", *In IMAGE V Conference*, pp. 114-122. Image Society Inc, June 1990.

[75] R.R.Schultz and R.L.Stevenson, "A Bayesian approach to image expansion for improved definition", *IEEE Trans. Image Process*, Vol. 3, No.3, pp. 233-242, 1994

[76] L.Schumaker, "Computing optimal triangulations using simulated annealing", *Computer Aided Geometric Design*, Vol. 10, pp. 329-245, 1993

[77] E.Shinbori and M.Takagi, "High quality image magnification applying the gerchberg-Papoulis iterative algorithm with DCT", *Systems and Computers in Japan*, Vol. 25, No. 6, pp. 80-90, 1994

[78] P.Su, L.Robert and S.Drysdale, "A comparison of sequential Delaunay triangulation algorithms", *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, Association for Computing Machinery, pp. 61-70, June 1995

[79] H.Sun and W.Kwok, "Concealment of damaged block transform coded images using projection onto convex set", *IEEE Trans. on Image Processing*, Vol. 4, pp. 470-477, April 1995

[80] R.Szeliski and H.-Y. Shum., " Creating full view panoramic mosaics and environment maps", *Computer Graphics (1997) ACM SIGGRAPH*, pp. 251-258, 2001

[81] P.Thevenaz, T.Blu and M.Unser, "Image interpolation and resampling", *Handbook of Medical Imaging, Processing and Analysis*, I.N. Bankman, Ed., Academic Press, San Diego CA, USA, pp. 393-420, 2000

[82] G.Turk, "Generating textures on arbitrary surfaces using Reaction-Diffusion", *Computer Graphics(1991), ACM SIGGRAPH*, vol 25, pp. 289-298, 1991

[83] G.Turk, "Texture synthesis on surfaces", *Computer Graphics (2001) ACM SIGGRAPH*, pp. 347-354, 2001

[84] M.Unser, A.Aldroubi and M.Eden, "Fast B-Spline transforms for continuous image representation and interpolation", *IEEE TRans. Pattern Anal. Mach. Int.*, Vol. 13, No. 3, pp. 277-285, 1991

[85] S.Upstill. "The RenderMan Companion", *Addison Wesley*, 1989

[86] D.C.Van Essen et al, "Information processing in the primate visual system: An integrated system perspective", *Science*, Vol. 255, No. 5043, 1992, pp. 419-423

[87] N.Damera-Venkata, T.D.Kite, W.S.Geisler, B.L.Evans and A.C.Bovik, "Image quality assessment based on a degradation model." *IEEE Transactions on Image Processing*, Vol. 9, pp. 636-650, April 2000

[88] M.Vetterli and J.Kovacevic, "Wavelets and subband coding", Prentice-Hall, 1995

[89] A.Watt, F.Policarpo. "The Computer Image", Addison-Wesley, 1998

[90] A.Watt, M.Watt, "Advanced Animation and Rendering Techniques", *Addison-Wesley*, 1992

[91] A.Watt, "3D Computer Graphics", Third Edition, *Addison-Wesley*, 2000

[92] A.Watt and F.Policarpo "3D Games Real-time Rendering and Software Technology" *Addison-Welsey*, 2001

[93] L.Y.Wei and M.Levoy, "Fast texture synthesis using tree-structured vector quantization" *Computer Graphics (2000) ACM SIGGRAPH* , pp. 479-488, 2000

[94] L.Y.Wei, M.Levoy, "Texture synthesis over arbitrary manifold surfaces", *Computer Graphics (2001) ACM SIGGRAPH* , pp 355-360, 2001

[95] M.Weinberger, G.Seroussi and G.Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardisation into $JPEG_LS$", Technical report HPL-98-193, Nov. 1998

[96] P.J.Willis, "GigaLib, A pixel-based graphics library", University of Bath, July, 2000

[97] J.W.Woods, "Two-dimensional discrete Markovian fields", *IEEE Trans. on Information Theory*, Vol. 18, pp. 232-240, 1972

[98] J.W.Woods, "Two-dimensional Kalman filters", *Two-dimensional Digital Signal Processing*, 42, Topics in Applied Physics, Springer-Verlag, NewYork, pp. 155-205, 1981

[99] J.W.Woods, "Image estimation using doubly stochastic Gaussian random field models", *IEEE Trans. Patt. Anal. Mach. Intell.*, Vol. 9, pp. 245-253, 1987

[100] S.Worley, "A cellular texture basis function", *Compute Graphics (1996), ACM SIGGRAPH*, pp. 291-294, 1996

[101] X.Wu, "An algorithmic study on lossless image compression", *Proceeding of Data Compression Conference*, pp. 150-159, Snowbird, Mar. 1996

[102] Y.Xu, B.Guo, and H.Y.Shum. "Chaos mosaic: Fast and memory efficient texture synthesis", *Technical Report MSR-TR-2000-32, Microsoft Research*, 2000.

[103] X.Xu and L.Ma, "Texture mixing and texture transfer", *Computer Aided Design and Computer Graphics (in Chinese)*, China, Jan 2003

[104] L.Ying, A.Hertzmann, H.Biermann, D.Zorin, " Texture and shape synthesis on surfaces", *Proc. 12th Eurographics Workshop on Rendering*, pp. 301-312., 2001

[105] Y.Yoo, A.Ortega and Bin Y, "Image subband coding using context-based classification and adaptive quantisation", *IEEE Trans. On Image Processing*, Vol. 8, pp. 1702-1715, Dec. 1999

[106] X.Yu, B.Morse and T.W.Sederberg, "Image reconstruction using data-dependent triangulation", *IEEE Trans. on Computer Graphics and Applications*, Vol. 21 No. 3, pp 62-68, May/June 2001

[107] W.Zeng and B.Liu, "Geometric-structure-based directional filtering for error concealment in image/video transmission", *Proceeding of SPIE Wireless Data Transmission*, Vol. 2601, Photonics East, Philadelphia, Oct. 1995.

[108] Y.Zhang, "Texture synthesis using Particle Swarm Optimization", *M.Sc thesis*, Jilin University, 2003

[109] S.C.Zhu, Y.N.Wu, and D.Mumford., " Filters, random fields and maximum entropy (frame)", *International Journal of Computer Vision*, 27(2): 1-20, 1998.

Publications by the Author

The following is a list of publications made by the author:

- Dan Su and Philip Willis, "Image Interpolation by Pixel Level Data-Dependent Triangulation", *submitted to Computer Graphics Forum*, October 2002
- Dan Su and Philip Willis, "Demosaicing of Colour Images Using Pixel Level Data-Dependent Triangulation", *Proc. Theory and Practice of Computer Graphics (TPCG 2003)*, pp. 16-23, 2003, IEEE

Volume xx (200y), Number z, pp. 1–13

Image Interpolation by Pixel Level Data-Dependent Triangulation

Dan Su, Philip Willis

Department of Computer Science, University of Bath, Bath, BA2 7AY, U.K.
mapds, P.J.Willis@bath.ac.uk

Abstract

We present a novel image interpolation algorithm. The algorithm can be used in arbitrary resolution enhancement, arbitrary rotation and other applications of still images in continuous space. High resolution images are interpolated from the pixel level data-dependent triangulation of lower resolution images. It is simpler than other methods and is adaptable to a variety of image manipulations. Experimental results show that the new "mesh image" algorithm is as fast as the bilinear interpolation method. We assess the interpolated images' quality visually and also by the MSE measure which shows our method generates results comparable in quality to slower established methods. We also implement our method in graphics card hardware using OpenGL which leads to real-time high-quality image reconstruction. These features give it the potential to be used in gaming and image processing applications.

1. Introduction

Digital image interpolation is the recovery of a continuous intensity surface from discrete image data samples. It is a link between the discrete world and the continuous one. In general, almost every geometric transformation requires interpolation to be performed on an image, e.g. translating, rotating, scaling, warping or other applications. Such operations are basic to any commercial digital image processing software.

There are several issues which affect the perceived quality of the interpolated images: sharpness of edges, freedom from artifacts and reconstruction of high frequency details. We also seek computational efficiency, both in time and in memory. Classical techniques, such as pixel replication, bilinear or bicubic interpolation have the problem of blurred edges or artifacts around edges. Although these methods preserve the low frequency content of the sample image, they are not able to recover the high frequencies which provide a picture with visual sharpness.

Standard interpolation methods are often based on attempts to generate continuous data from a set of discrete data samples through an interpolation function. These methods attempt to improve the ultimate appearance of re-sampled images and minimise the visual defects arising from the inevitable resampling error.

Traditionally, interpolation is accomplished through convolution of the image samples with a single kernel – typically a bilinear, bicubic[1], or cubic B-spline[2]. A number of algorithms have been proposed to improve the magnification results. PDE-based approaches[3] apply a nonlinear diffusion process controlled by the local gradient. POCS (Projection-Onto-Convex-Set) schemes[4] formulate the interpolation as an ill-posed inverse problem and solve it by regularised iterative projection. Orthogonal transform methods focus on the use of the discrete cosine transform (DCT)[5, 6]. Directional methods[7, 8] examine an image's local structure around edge areas to direct the interpolation. Variational methods formulate the interpolation as the constrained minimisation of a functional[9, 10].

It has been recognised that taking edge information into account will improve the interpolated image's quality [11, 12, 13, 14] and it is known that the human visual system makes significant use of edges[18]. Instead of approaching interpolation as simply fitting the interpolation function, these methods consider also the geometry of the image. Li[11] asserts that the quality of an interpolated image mainly de-

pends on *the sharpness across the edge* and *the smoothness along the edge.*

Li et al.[11] attempted to estimate local covariance characteristics at low resolution and used them to direct interpolation at high resolution (NEDI - New Edge Directed Interpolation) while Allebach et al.[12] generated a high resolution edge map and used it to direct high-resolution interpolation (EDI - Edge Directed Interpolation). Battiato et al.[13] proposed a method by taking into account information about discontinuities or sharp luminance variations while doing the interpolation. Morse et al.[14, 15] presented a scheme that uses existing interpolation techniques as an initial approximation and then iteratively reconstructs the isophotes using constrained smoothing. They emphasise the importance of the "smoothness" quality, if the isophotes are not to be visually intrusive. As will shortly become clear, we too accept this need to fit the visual geometry.

The above schemes demonstrate improved visual quality (in terms of sharpening edges or suppressing artifacts) by using a model to preserve the edges of the image and to tune the interpolation to fit the source model. However they are complex compared to traditional methods and thus computationally expensive.

Another approach is triangulation modelling. Triangulation has been an active research topic during the past decade. It is popular in geometric modelling. However, image reconstruction using triangles isn't widely used, probably because of the large number of triangles needed. Yu et al.[16] modelled images as data dependent triangulation meshes and reconstructed images from the triangulation mesh. Their approach adapted traditional data-dependent triangulation[17] (DDT) with their new cost functions and optimisations. The data dependent triangulation thus matches the edges in the image and improves the reconstructed image. Their method is relatively complex and computationally expensive.

We develop a new edge-directed method for image interpolation. We call this an *image mesh* DDT. We do not assume knowledge of the low-pass filtering kernel or attempt to find a statistical rule about the local geometry. Our approach is related to that of Yu but is simpler and faster because it does not involve any cost function or repeating optimisation process. Our mesh is very simple and completely regular. We avoid the complexity of a full DDT method while keeping the feature of DDT that improves the reconstruction quality. We will demonstrate our algorithm used in arbitrary magnification of still images and other applications.

2. Image Mesh Data-Dependent Triangulation

2.1. Principle of the Algorithm

We first consider the case that there is an edge passing between a square of four pixels. If this edge cuts off one corner, one pixel will have a value substantially different to the other three. We call this pixel the *outlier*. Imagine that we represent the brightness of the pixel as the height of a terrain. In effect, the three similar pixels define a plateau, relatively flat, while the outlier value is at the bottom of the cliff (if smaller) or the top of a peak (if higher) (Figure 1). This gives us a hint that if we want to interpolate a high resolution pixel within the relatively flat region we should not use the outlier. Classical interpolation methods like bilinear interpolation suffer from edge blurring because they use all four pixels to do interpolation. We only use three.

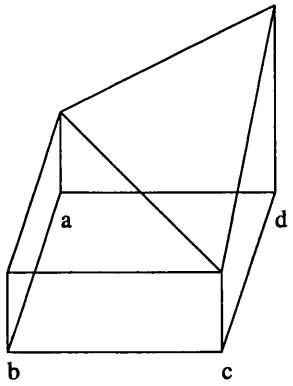


Figure 1: *Triangulation in a four-pixel square*

The strength of employing triangles in this way is that we model edges in the image. In effect we tune the interpolator to match edges. In Figure 1, when interpolating the high-resolution pixel falling in triangle *abc*, the interpolator won't use the value of *d* which is very different to this plateau. For two pixels falling in different triangles, the height of the vertices will be quite different and thus the sharpness of the edge is kept. It is easy to see that in very smooth regions, the interpolator keeps smoothness as well, even across triangle boundaries.

This simple geometry suggest a way to guide the interpolation so that smoothness within the regions and sharpness between the flat region and cliff region can both be kept. If the diagonal is to correspond to the edge in the image, the diagonal should be the one which does *not* connect to the outlying pixel value, the one most different to the other three.

Suppose pixels *a, b* and *c* are the same height while *d* is higher than these three. Obviously *a, b* and *c* define a flat region while *d* is the most different pixel to the other three. Thus we connect diagonal *ac* and get the triangles *abc* and *adc*. In general, if *b* or *d* is the most different pixel, the edge should be *ac*, otherwise *bd* will be the edge. There are other situations if *a* and *d* are very different to *b* and *c*; or *a* and *b* are very different to *c* and *d*. In these cases it makes little difference which diagonal is chosen. The edge is roughly either horizontal (*ad* are different to *bc*) or vertical (*ab* are

different to cd) and the triangle will always cross the edge. It is similar to bilinear interpolation in these cases.

Obviously, using the diagonal to triangulate the four-pixel square cannot correspond to edges of arbitrary angle. The diagonal can only roughly represent the orientation of the edge. We could use sub-pixel triangulation to represent arbitrary angles, but that would add more complexity to the algorithm. Our aim is to keep the algorithm as simple as possible. We will demonstrate in this paper that triangulation by diagonal is enough in most situations and can provide excellent results. It is the direction-selection method that is the key.

Our method thus fits the finest triangular mesh to the source pixels. This "image mesh" is completely regular except that the diagonals are locally selected to run in the same general direction as any visible edge. To generate a new image, possibly at higher resolution, the target pixels are located in the source mesh. We then evaluate each target pixel from the triangle in which it sits. It is interpolated using only the information from the three triangle vertices. In edge areas, the interpolator won't interpolate any two pixels that fall in different triangles. In other words, the new high-resolution image has the edges sharp and the smooth areas smooth.

2.2. Implementation and Optimisation

Suppose the low-resolution image is X and the high-resolution image to be generated is Y. Our algorithm can be expressed as two steps. We first scan the sample image X to initialise a 2D array which records the edge direction of all four-pixel squares. In the second step we scan Y. For each y_{ij} we inverse map to the sample image X and use the array to identify the triangle in which the point falls. Then we interpolate within that triangle to get the value of y_{ij}.

In the first step, the algorithm has to determine the outlier pixel. This has to be done repeatedly, so speed is important. Instead of finding the outlier directly, we compare the difference $|a-c|$ with the difference $|b-d|$ and connect the pair with smaller difference. The proof that this is equivalent to finding the outlier pixel is in Appendix A. This saves computing time, needing only two subtractions and a comparison. Doing it directly would require sorting four pixels and then comparing the highest and lowest pixels with the average value.

We use inverse mapping in the interpolation step because it has a number of benefits. First it can be used at arbitrary resolution. We are not constrained in any way by the resolution of the source data. Second, there is no requirement to align the target grid parallel to the source grid, so arbitrary rotation is possible at no additional cost. Third, sampling can be irregular to provide warps, although the sampling rate must not be too low because this would cause break-up. Finally it is a single-step method.

We use linear interpolation within the triangles. However there is some confusion of terminology in the literature, which we need to clarify before proceeding. "Bilinear interpolation" strictly refers to interpolating between four values and we will use the term only in that sense. In the graphics community, three-value interpolation, as used in Gouraud shading, is also called bilinear interpolation, although it is only a degenerate case. We will distinguish this by calling it "triangle interpolation".

Figure 2 shows a flower image with the magnified view of the tip of the lowest stamen and the pixel level data dependent triangulation mesh of that stamen. (We only show the diagonals of the triangles for a clearer view.) We represent the triangulation in two diagrams, each one only containing a specific direction. The stamen and a black edge near the stamen both roughly have NW-SE orientation. It is clear to see that the corresponding triangles also cluster in the NW-SE direction, which matches the edges of the image. In particular, note the absence of NE-SW diagonals near these linear features.

2.3. Extended Method

Some problems still remain in our basic model. For example, close study of the triangulation of the stamen (Figure 2) reveals a problem. The actual local edge goes in the NW-SE direction while a few diagonals in the lowest stamen areas give the NE-SW direction. This leads to some small deterioration of edge quality. These diagonals contradict the local edge orientation because our basic method only considers the four-pixel square, ignoring the surrounding values. This only catches the micro-geometry (pixel level), not the local geometry due to edges passing through several pixels. To correct this we have developed an extended model where we consider this extra information.

We assume the image is locally stationary. That is to say, the intensity of a pixel is dependent on its spatial neighbourhood while independent of the rest of the image. The neighbourhood of a pixel can be modelled as a window around this pixel. Instead of a normal least-square adaptive edge prediction scheme, we simply consider the neighbourhood window's edge direction. Our basic method considers four pixels arranged in a square. Our extended method considers 16 pixels arranged as 3×3 squares. To predict the edge direction in the central square, we consider all of them (Figure 3). If most of these squares have their diagonals in one particular direction, then we impose that direction on the central square. In our case we do this if at least 6 of the 9 squares have the same direction. All decisions are made on the original data so that changes do not influence nearby decisions taken later.

Obviously our extended model increases complexity, but very marginally. It is worth noting that this additional complexity is only in preparing the diagonals, not in using the mesh to interpolate an image.

Figure 4 shows the diagonals resulting from our extended

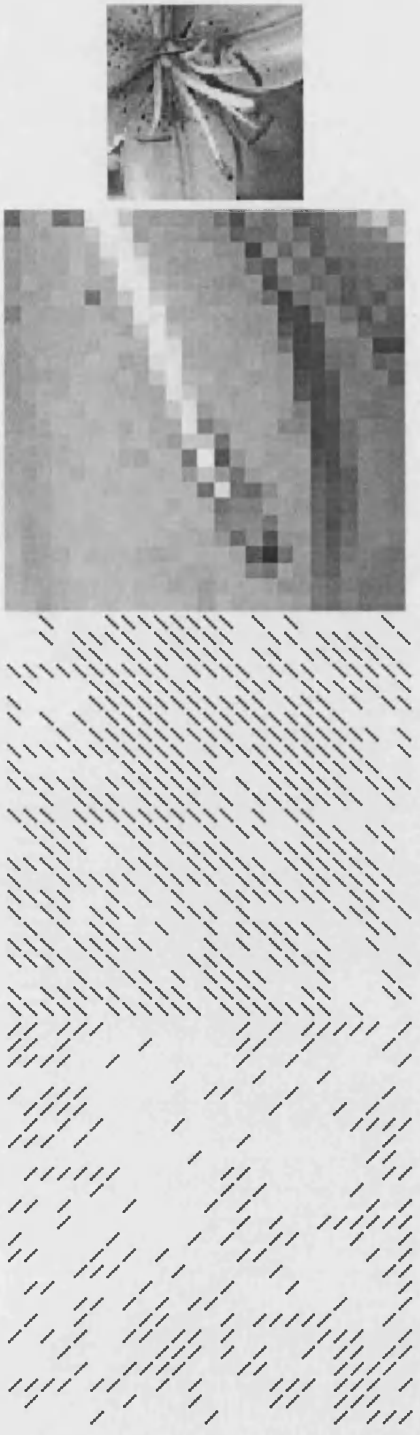


Figure 2: *Top: a part of a flower image. Second: a magnified view of the bottom stamen. Third: the pixel level data dependent triangulation of the stamen (NW-SE direction) Bottom: NE-SW direction*

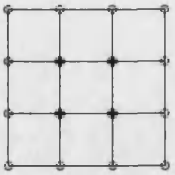


Figure 3: *Neighbourhood of 3 × 3 squares*

Figure 4: *The triangulation mesh of the extended method. Top: NW-SE direction. Bottom: NE-SW direction*

method. The stamen of Figure 2 has 625 diagonals. Our basic method generates 418 diagonals in the NW-SE direction and 207 diagonals in the NE-SW direction while our extended method produces 438 and 187 diagonals respectively. They differ only in 20 diagonals, mainly along the stamen and the black edge: the extended method better preserves the local geometry.

2.4. Algorithm Analysis

We analyse the complexity of the basic method and the extended method in this section. Suppose the image I has width and height m, so the number of pixels is $n = m^2$. The number of triangles in the triangulation is then $(m-1) \times (m-1) \times 2$. In our implementation, we use a table to record the orientation of the diagonal in each square. As there are only two diagonal directions we use one bit to store this information. Thus, the total memory requirement for the triangulation mesh is $2(m-1)^2 \approx 2n$ bits. For a normal image with size 1024×1024 the memory requirement is $256KB$. Compared to the standard $256MB$ memory in current PCs, this is very small. Moreover, the memory requirement $2n$ is linear with the number of pixels n.

In our basic method, each triangle needs two subtractions and one comparison, so the total computation is $(m-1) \times (m-1) \times 2 \times 3 \approx 6n$.

Our extended method has two steps in preparing the mesh. In the first step, we calculate just like the basic method and set each triangle's diagonal direction. In the second step, each triangle needs a sum of eight surrounding squares and a comparison to decide if there is an overriding edge orientation in local area. Thus, the computation for each triangle needs two extra computations, and the whole image needs $10n$ computation which is still linear with image size n.

Then follows the interpolation step. It is easy to see that the triangle interpolation has the same complexity as bilinear interpolation which is linear with n. Thus, both the basic method and the extended method have a time complexity of $O(n)$.

Our method is thus efficient in both memory and time, and is suitable for handling large images with a linear dependency on the image size.

2.5. Algorithm Comparison

Yu et al.[16] propose an image reconstruction method using data dependent triangulation. They use a new cost function and an improved optimisation algorithm to generate an optimised triangulation mesh. Their method is able to model an image effectively. It is complex to implement and is computationally slow. It takes several iterations to get an optimised triangulation and each iteration takes "between 0.5 and 5 seconds" even for a small image (80×80) on a consumer-grade PC. Another limitation of the method is it cannot catch single-pixel and small features.

Our method can be thought of as a simplified data dependent triangulation (DDT). It generates the triangulation mesh simply by inserting diagonals. This leads to some degradation in quality since a normal DDT can model the edge at arbitrary angles. However our method provides a notable trade-off between quality and speed. Although the DDT method can in principle give higher quality, ours is very easy to implement and much faster. Also our method needs only a small byte array to store the triangulation mesh while a full DDT requires a more complicated structure and more storage space. An advantage of our extended method is it is able to catch small and local features.

Other researchers[19] also use DDT for data interpolation, aiming at a better optimisation of DDT according to their cost functions and optimisation processes. Our method avoids this. We will now demonstrate that the method is effective and that it does provide high-quality reconstructed images compared to conventional methods.

3. Experimental Assessment

3.1. Implementations

We implemented several interpolation methods. The images from bilinear interpolation and bicubic interpolation were produced from Matlab 5 built-in functions. The NEDI method was tested from a Matlab program kindly provided by its originator. We used a C++ program and our own graphics library to implement our methods.

Greyscale images were processed exactly as already described. When selecting edge directions in colour images, we converted the RGB components of each pixel into luminance using the following formula[16] where L stands for luminance:

$$L = 0.21267R + 0.71516G + 0.07217B$$

The edge direction was determined by these luminance values. Interpolation was performed in the R,G,B planes independently.

3.2. Visual Assessment

We performed preliminary tests both to check the implementations and to permit a visual assessment of the methods. We wanted to use an image in both in greyscale and in colour. The flower image we have used has well-defined edges (to test edge sharpness), thin linear features and small details (to ensure they are retained) and smoothly varying areas (to reveal any discontinuity).

Figures 5 and 6 show the comparison results. All the images in Figures 5 and 6 are magnified from the flower image of Figure 2 by a factor of 4.

Figure 7 shows a close-up view of the stamen using our basic and extended method. This illustrates that the basic

Figure 5: *Detail of image magnified by 4. Top: bilinear interpolation. Middle: bicubic interpolation. Bottom: NEDI*

Figure 6: *Detail of image magnified by 4. Top: our basic method. Bottom: our extended method*

Figure 7: *Magnified view of the stamen. Left: our basic method. Right: our extended method*

method has some artifacts along the stamen which are reduced in the extended method.

Figure 8 shows the various methods used to magnify the colour flower image by a factor of 3.5.

From visual inspection our method produces better images than bilinear and bicubic interpolation, while the NEDI method is better still (Figures 5 and 6). However, it seems NEDI's weighting algorithm changes the contrast of the image. The bilinear interpolation suffers from blurring of the edges. The bicubic method introduces sharper edges but more artifacts.

We next performed analytical testing.

3.3. Quality Assessment

To perform analytical assessment of the the interpolated images, we need a quality measure. The degradation based method[20] is not able to report the "jagged" artifacts related to the orientation of edges. Daly's visible differences predictor[21] produces an error image which characterises the regions in the test image that are visually different from the original image. It is however difficult to use error images for reconstructed image quality ranking as Daly mentioned in his paper. Therefore we used mean-square error (MSE) as our assessment tool. The MSE is the cumulative squared error between the reconstructed and the original image. It is widely used in image processing to evaluate reconstructed image fidelity.

Our method aims at improving edge quality on magnified images and retaining a good overall quality as well. Thus we produced one sample image set of five "edge" images with size 200×200 (Figure 9) and used twenty 768×512 real nature images as a more general test set.

In theory, there is no perfect way to judge the magnification quality. Because the image we have is of fixed resolution, we don't know what the 'correct' magnified image is. In order to analyse error, we need to know or simulate this image. So we start with an original image, generate a lower resolution version, then use different methods to magnify it. Then we compare the magnified image with the original image. This is not perfect but it provides a reasonable reference against which to measure the reconstruction quality.

The down-sampled images could be obtained by averaging or sub-sampling. However, edge blurring and ringing are introduced by averaging, while sub-sampling breaks down the geometry and introduces artifacts. We chose a Gaussian filter as the point-spread function with its standard deviation representing the radius of the point-spread function. Each pixel at the target image (the down-sampled image) is considered as a point-spread function represented by a Gaussian distribution. It is down-sampled from some part of the source image, represented by another point-spread function.

Figure 8: *A flower image magnified by a factor of 3.5 using: Top: bilinear interpolation. Middle: bicubic interpolation. Bottom: our extended method.*

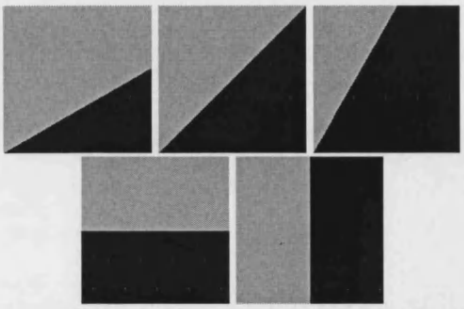


Figure 9: *Set of five edge images. The angles are 30, 45, 60, 0 and 90 degrees*

	30°	45°	60°	0°, 90°
Our methods	28.8	28.9	28.8	26.0
Bicubic	29.7	31.5	29.3	22.2
Bilinear	34.0	38.4	34.0	26.0
Replication	41.8	45.4	41.5	9.2
NEDI	43.3	47.6	43.4	27.6

Table 1: *MSE results of edge images*

In this case the radius of the point-spread in the source image is double that of the radius in the target image. Thus, we calculate the standard deviation of the target Gaussian distribution, then double this to get that of the source image. This is then used to down-sample, by convolution.

We used pixel replication, bilinear interpolation, bicubic interpolation, NEDI, our basic method and our extended method to obtain the reconstructed images. All reconstructed images are magnified by a factor of two and then compared to the original image.

3.3.1. Quality of edges

Our first test was to check the quality of well-defined edges. For the test set we generated five samples with a single edge of varying angle (30, 45, 60, 0 and 90 degrees). Each edge is black one side and white the other side (Figure 9). The down-sampled edge images are magnified by a factor of two and compared to the original edge images to get the MSE results which is reported in Table 1. The MSE is performed on 0 255 range for the grey-scale edge images. We put 0° and 90° in the same column because they give the same results for all methods. Our basic and extended methods have the same results in all these situations because our basic method is able to preserve the geometry well in these simple cases.

The MSE results report that our method gets the best (lowest) score in every case except at 0° and 90°. In these two cases pixel replication gets the best score, which it is trivially able to do. (In principle it should achieve zero MSE but the Gaussian sampling introduces some grey edge pixels.) Bicubic beats us here because its interpolation more sharply models these high-contrast edges. Our method is the equal of bilinear interpolation as we expect. Although our triangulation gives edges of 45°, it also performs well on 30° and 60°. Bicubic and bilinear interpolation are slightly worse because they suffer from artifacts or blurring on the edge. Pixel replication does not generally catch the geometry very well and NEDI suffers from the effects of its weighting algorithm.

3.3.2. Quality of real images

In order to test the method on "smoother" and more typical images, we used twenty 24-bit 768 × 512 colour nature images as another test set. These images are down-sampled, magnified by different methods by a factor of two and compared to the original images. We perform the MSE comparison on R,G,B channels independently and Table 2 reported the averaged MSE values / stdandard deviations over 20 images from the test set. BC is the bicubic interpolation, EXT is our extended method, BL is bilinear interpolation, DDT is our basic method and PR is pixel replication.

	R	*G*	*B*
BC	109.4 / 85.1	119.4 / 106.7	123.8 / 121.2
EXT	117.6 / 92.8	127.8 / 115.2	132.7 / 131.3
BL	118.2 / 92.9	128.4 / 115.2	133.1 / 130.9
DDT	118.6 / 93.6	128.8 / 116.1	133.7 / 132.3
PR	126.1 / 99.7	134.8 / 120.8	138.7 / 137.2
NEDI	198.6 / 180.1	197.9 / 159.9	187.4 / 154.5

Table 2: *MSE results of real images*

There is a clear consistency of each channel's performance and there is also a clear consistency of each method's performance. Bicubic interpolation gets the best score (least error). Our methods rank close to the bilinear method. Our basic method is slightly worse than the bilinear method because it sometimes gives the wrong edge direction. Our extended method is slighter better than bilinear interpolation because our approach is better in edge areas and is almost the same in smooth areas.

Pixel replication gets a low score as we expect. NEDI surprisingly gets the lowest score although it has good visual reconstruction quality. We presume this is because the contrast of the image has been changed by NEDI's weighting algorithm and thus it produces numerically the wrong image,

albeit a pleasing one. This emphasises the need to moderate any analysis with visual inspection.

We also did a statistical t-test over the MSE results of the different methods and the results shows no significant difference between our method and the bilinear and bicubic interpolation. The NEDI method performs the weakest here but the results do not show that is worse than ours at the 95% confidence level.

We can conclude that bicubic interpolation produces the lowest overall mean squared error. Our extended method is quite close to this and is statistically indistinguishable from other methods (except significantly better than NEDI). Visual inspection of our method shows that it produces good results, which we believe are due to its better edge performance. We will now show that our method is much quicker than bicubic interpolation and comparable in speed to inferior methods.

3.4. Performance Assessment

We implemented bilinear interpolation, bicubic interpolation, our basic method and our extended method by C++ code and compared their computational performance. We used the real natural colour images test set again. We down-sampled every image to 384 × 256 pixels (using the method described earlier). Then we magnified the down-sampled images by a factor of 2 and also by a factor of 3.5. We used the bicubic interpolation proposed by Keys [22]. Table 3 shows the performance comparison on a machine with an Intel Pentium4 3G processor and 1G DDR system memory. Our extended method uses the 3 × 3 square window. All figures are in seconds.

	Bilinear	*Basic*	*Extended*	*Bicubic*
magnify 2	0.359	0.406	0.412	3.621
magnify 3.5	1.105	1.162	1.170	10.914

Table 3: *Performance comparison*

We can see from the table that our method is only slightly slower than bilinear interpolation. Importantly, bicubic is an order of magnitude slower than the other methods. The averaged times for calculating the triangle mesh are included in the above figures. For our basic and extended method these are 0.041 and 0.049 seconds respectively. Factoring these out reveals that our methods are linear with the number of pixels generated.

In conclusion, our extended method is comparable in speed to bilinear interpolation while providing better reconstruction results visually. In comparison to bicubic interpolation, our extended method is much faster and visually better, especially in edge reconstruction. These two methods have almost identical MSE values. Our method is fast, simple and modest in memory needs.

3.5. Hardware Implementation

More and more complex graphics operations have moved to the graphics co-processor or accelerator, including shading, texturing, anti-aliasing and bilinear interpolation. These features of graphics cards make it possible to create extremely realistic games and simulations.

However the only interpolation algorithms currently available on graphics cards are triangular and bilinear interpolation: the others are too complex. High quality image reconstruction in real-time still remains a difficult and unsolved problem. Our pixel level data dependent triangulation makes a step in this direction.

A graphics card can handle tens of millions of triangles per second and it can interpolate within triangles. This suggests that we convert any image to a triangle mesh and then pass the mesh to the graphics card. The card will deal with the mesh in real-time.

We have used OpenGL to explore the potential of our method in hardware implementation. We first generated a triangle mesh using our basic or extended model. Then we used OpenGL to pass the mesh to the graphics card so that it could manipulate the mesh, such as by scaling and rotating. These manipulations can be in 3D, at no extra cost. Our experimental results showed that high quality reconstructed images can be generated in real-time.

We used the OpenGL GL-TRIANGLE-STRIP to build the triangle mesh. This routine needs all of the triangles to have the same orientation. Thus we started a new GL-TRIANGLE-STRIP whenever the diagonal direction changes. All of these strips were saved in a display list which was then used to render the image.

The program flow of the OpenGL process is as follows:

1. Build a byte array to record the diagonals of the triangles.
2. Set up all the GL-TRIANGLE-STRIP and save them in a display list.
3. Render the image and call an OpenGL loop, waiting for keyboard response and doing manipulation corresponding to the key pressed.

We have tested several images with size 768 × 512 pixels, in the same machine: an Intel Pentium 4 3G processor and an NVidia GeForce 4 graphics card with 128M memory. Using our extended method, the time for preparing the mesh for an image with 768 × 512 pixels was under 0.2 seconds. Once the triangle mesh was loaded, the graphics card did all further manipulation. We used key presses for scaling or rotation, causing the appropriate updates to the transformation matrix.

The GeForce 4 graphics card specification claims a rendering speed of 136 million vertices per second. This equates

to about 45M independent triangles per second. This latter *rate* could increase with triangle strips (due to vertex sharing), though of course the *number* of triangles which can be rendered at full speed is limited by the card memory. With our test image meshes having less than 1M triangles, the graphics card easily gives real-time zooms, translations and rotations.

4. Other Applications

Figure 10 shows our extended method applied to three colour images chosen to include edge, texture and smooth features, magnified by a factor of 2.

Due to the simplicity of our algorithm, it is easy to apply in other ways. For example, we can rotate the image by any angle (Figure 11 – top). We inverse rotate each target pixel back to the sample image and interpolate the value. We can also generate a perspective transform of an image. On any given y scan line, we calculate the pixel at (x, y) by sampling the source image at (sx, ty) where s, t are scale factors which vary linearly with height (We are assuming the y axis is the centre of the screen). Figure 11 (middle) shows the result. We can produce a magnifying lens effect (Figure 11 – bottom). If the lens has radius R, then its disc is filled with the image from a smaller disc with radius r at the same centre. For any pixel inside R, we scale down to r, evaluate the original value at r and apply it at radius R.

In general, these are variants on the same technique: to evaluate the target pixel p, we evaluate pixel $F(p)$ where F is a simple inverse mapping to the original image. Then we interpolate in the triangle where it falls.

5. Discussion

In this paper we have presented a new method of image interpolation. We represent an image as a data-dependent triangulation mesh. Every four-pixel square is divided into two triangles with the diagonal corresponding to the local edge of the image. The desired pixel can then be interpolated from the triangle in which it falls, determined by inverse mapping.

Other variants of the diagonal choice procedure can also be tried. For example, a pair of suitable digital filters might be better at distinguishing the local edge direction; or the threshold could be different to the one we chose. Other variants of the sampling procedure can be used, the interpolation providing some security against sampling defects. These two procedures are independent and neatly correspond to the image modelling and image rendering phases.

The new interpolation approach generates images with better visual quality than traditional interpolation schemes. The error assessment also shows that our scheme produces good overall image accuracy. The complexity of the new method is similar to bilinear interpolation and much lower

Figure 11: *Top:Flower image rotated by 27 degrees. Middle: a perspective view of the flower image. Bottom: a lens effect of the flower image*

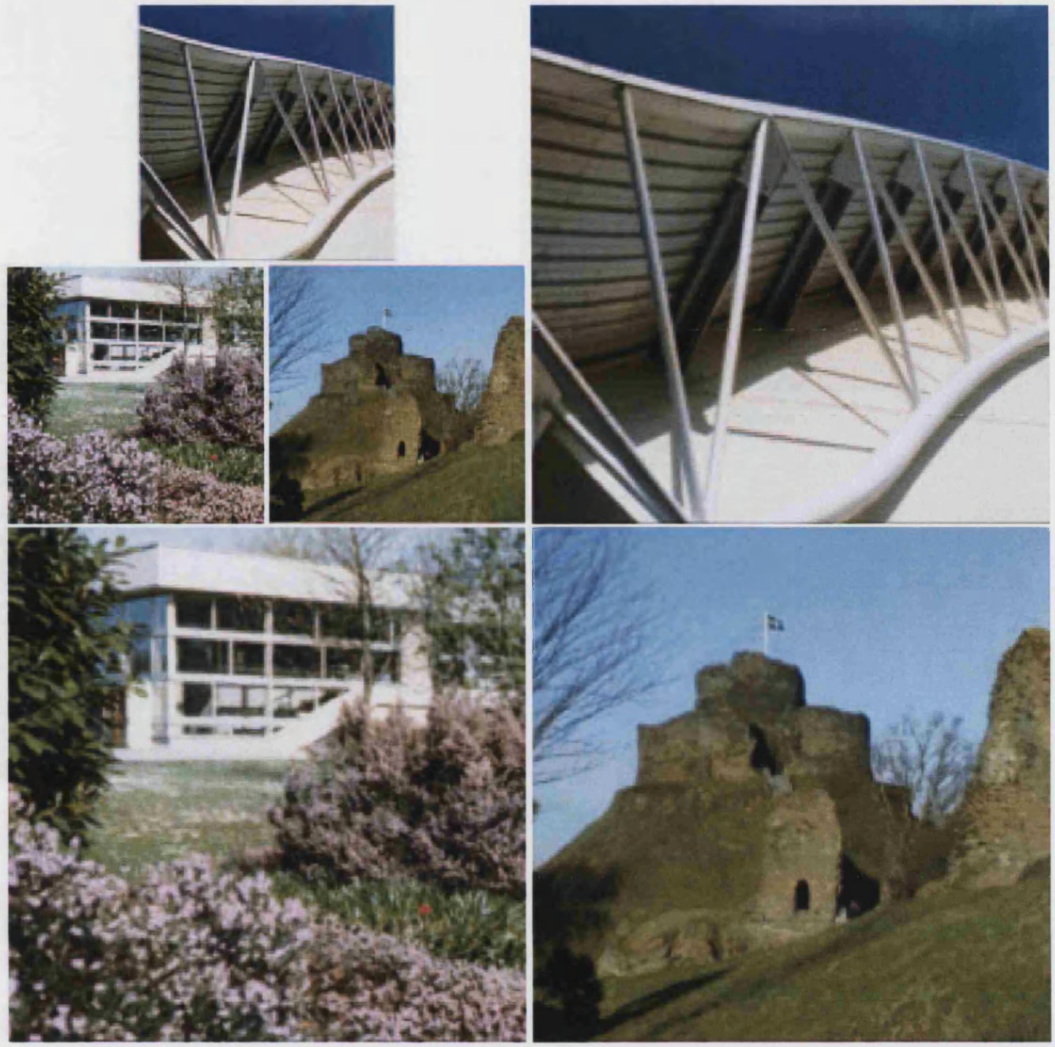


Figure 10: *Three sample images of* 192 × 192 *shown upper left. The corresponding larger images are magnified of a factor of 2, using our extended method.*

than the bicubic method. We avoid the time-consuming optimisations that others use but still produce good results very quickly.

Our method has several advantages. It requires no iteration. It achieves arbitrary factor magnification, rotation, perspective transform and warp through a single mechanism. Our scheme is very simple to implement and computationally fast. It requires little data structure overhead to generate the mesh image. Moreover, our meshes can be rendered on a graphics card which makes real-time image reconstruction possible. There is a potential for our method to be used in gaming and image manipulation generally. We have also extended our model to an important commercial application: demosaicing of colour images (the reconstruction of a full-resolution colour image from the *mosaiced* sample generated by current single-chip digital cameras)[23]. We are investigating its use in 4-colour separation for printing. Above all, we have demonstrated that a simple approach, sensibly used, can rapidly generate excellent results.

Acknowledgements

The authors would like to thank Dr. Xin Li at Sharp Labs of America at Camas, WA, USA for kindly providing his code

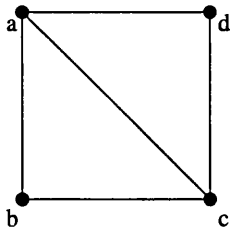
for comparison tests and Dr. P.W. Wong at IDzap, USA for providing the flower image shown in this paper. We would also like to thank our colleagues Dr Peter Hall and Dr Man Qi for their suggestions and discussion. The anonymous referees gave excellent advice which has greatly improved the paper. We are grateful to Professor Ken Brodlie, at the University of Leeds UK, for long ago pointing out to one of us the distinction between true bilinear interpolation and what we here call triangular interpolation. We also thank Dr. Steven Ruzinsky for discussions and suggestions about interpolation. The work is funded under EPSRC project *Quasi-3D Compositing and Rendering* and a UK-funded *Overseas Research Scholarship* funded by Universities UK.

References

1. A.N. Netravali and B.G. Hasskell, "Digital Pictures: Representation, Compression and Standards", 2nd Ed., New York:Plenum Press, 1995 1
2. M. Unser, A. Aldroubi and M. Eden, "Fast B-Spline Transforms for Continuous Image Representationa and Interpolation", *IEEE TRans. Pattern Anal. Mach. Int.*, Vol. 13, No. 3, pp. 277-285, 1991 1
3. B. Ayazifar and J.S. Lim, "Pel-adaptive model-based interpolation of spatially subsampled images", *Proc. of Intl. Conf. on Acoust. Speech and Signal Processing*, Vol. 3653, pp. 181-184, 1992 1
4. K. Ratakonda and N. Ahuja, "POCS based adaptive image magnification", *Proc. of Intl. Conf. on Image Processing*, Vol.3, pp. 203-207, 1998 1
5. S.A. Martucci, "Image Resizing in the Discrete Cosine Transform Domain", *Proc. Int. Conf. Image Processing*, Vol.2, pp. 244-247, 1995 1
6. E. Shinbori and M. Takagi, "High Quality Image Magnification Applying the Gerchberg-Papoulis Iterative Algorithm with DCT", *Systems and Computers in Japan*, Vol. 25, No. 6, pp. 80-90, 1994 1
7. S.D. Bayrakeri and R.M. Mersereau, "A New Method for directional Image Interpolation", *Proc. Int. Conf. Acoustics, Speech, Sig. Process*, Vol.4, pp. 2383-2386, 1995 1
8. K. Jensen and D. Anastassiou, "Subpixel Edge Localization and the Interpolation of Still Images", *IEEE Trans. Image Process*, Vol. 4, No. 3, pp. 285-295, 1995 1
9. N.B. Karayiannis and A.N. Venetsanopoulos, "Image Interpolation Based on Variational Principles", *Signal Process*, Vol. 25, pp. 259-288, 1991 1
10. R.R. Schultz and R.L. Stevenson, "A Bayesian Approach to Image Expansion for Improved Definition", *IEEE Trans. Image Process*, Vol.3, No.3, pp. 233-242, 1994 1
11. X. Li and M. Orchard, "New Edge Directed Interpolation", *Proc. IEEE Int. Conf. Image Processing*, Vol. 2, pp. 311-314, 2000 1, 2
12. J. Allebach and P.W. Wong, "Edge-directed interpolation", *Proc. IEEE Int. Conf. Image Processing*, Vol. 3, pp. 707-710, 1996 1, 2
13. S. Battiato, G. Gallo, F. Stanco, "A locally-adaptive zooming algorithm for digital images", *Image and Vision Computing*, Vol. 20, No. 11, pp. 805-812, September 2002 1, 2
14. B.S. Morse and D. Schwartzwald, "Isophote-based interpolation", *Proc. IEEE Int. Conf. Image Processing*, Vol. 3, pp. 227-231, 1998 1, 2
15. B.S. Morse and D. Schwartzwald, "Level-Set Image Reconstruction", *Proc. Computer Vision and Pattern Recognition 2001 (CVPR'01)*, pp. 333-340, IEEE 2001 2
16. X. Yu, B. Morse, T.W. Sederberg, "Image Reconstruction Using Data-Dependent Triangulation", *IEEE Computer Graphics and Applications*, Vol. 21 No. 3, pp. 62-68, May/June 2001 2, 5
17. N. Dyn, D. Levin, S. Rippa, "Data Dependent Triangulations for Piecewise Linear Interpolation", *IMA Journal of Numerical Analysis*, Vol. 10, pp. 127-154, Institute of Mathematics and its Applications, 1990 2
18. Van Essen DC, Anderson CH, and Felleman DJ, "Information Processing in the Primate Visual System: An Integrated System perspective", *Science*, Vol. 255, No. 5043, pp. 419-423, 1992 1
19. E. Quak and L.L. Schumaker, "Cubic spline interpolation using data dependent triangulations", *Comput. Aided Geom. Design*, Vol. 7, pp. 293-301, 1990 5
20. N. Damera-Venkata, T.D. Kite, W.S. Geisler, B.L. Evans and A.C. Bovik, "Image Quality assessment based on a degradation model." *IEEE Transactions on Image Processing*, Vol. 9, pp. 636-650, 2000 7
21. S. Daly, "The visible differences predictor: An algorithm for the assessment of image fidelity." *Digital Images and Human Vision*, A.Watson, Ed. Cambridge, MA:MIT Press, 1993 7
22. Robert Keys, "Cubic Convolution Interpolation for Digital Image Processing", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 29, No. 6, pp.1153-1160, 1981 9
23. D. Su and P.J. Willis "Demosaicing of Colour Images Using Pixel Level Data-Dependent Triangulation" *Proc. Theory and Practice of Computer Graphics (TPCG 2003)*, pp. 16-23, 2003, IEEE. 11

Appendix A: Proof

Consider a four pixel square *abcd*. We will first prove that, if pair *ac* has smaller difference than *bd*, then *b* or *d* is the outlier pixel and we should connect *ac*. That is to say, if $|a-c| < |b-d|$ then *b* or *d* is either the biggest or the smallest pixel.



Suppose $|a-c| < |b-d|$, and suppose $a \geq c$, then $a-c < |b-d|$.

1. Suppose $b > d$. Then $a-c < b-d$ $(b > d, a \geq c)$, hence $a-b < c-d$ $(b > d, a \geq c)$.
   We suppose $a > b$ and $c < d$, then $a-b > 0$ and $c-d < 0$, so we get $a-b > c-d$. However, we have the formula $a-b < c-d$ before which means our assumption that $a > b$ and $c < d$ is wrong.
   Because $a > b$ and $c < d$ is wrong, either $a < b$ or $c > d$ or $a < b, c > d$ with the condition $(b > d, a \geq c)$. In these cases, either *b* is the biggest pixel $(b > a, b > c, b > d)$ or *d* is the smallest pixel $(d < c, d < a, d < b)$.
2. Suppose $b < d$, then $a-c < d-b$ $(b < d, a \geq c)$, hence $a-d < c-b$ $(b < d, a \geq c)$.
   We suppose $a > d$ and $c < b$. Then $a-d > 0$ and $c-b < 0$, so we get $a-d > c-b$. However, we have the formula $a-d < c-b$ before which means our assumption that $a > d$ and $c < b$ is wrong.
   Because $a > d$ and $c < b$ is wrong, either $a < d$ or $c > b$ or $a < d, c > b$ with the condition $(b < d, a \geq c)$. In these cases, either *b* is the smallest pixel $(b < c, b < a, b < d)$ or *d* is the biggest pixel $(d > b, d > a, d > c)$.

We have proved that if pair *ac* has the smaller difference $(|a-c| < |b-d|)$, there are two situations. One is that either *b* is the biggest pixel or *d* is the smallest pixel. The second is that either *b* is the smallest pixel or *d* is the biggest pixel. In either case the outlier is either *b* or *d* and *ac* should be the edge. Using the same method we can prove that if pair *bd* has the smaller difference $(|b-d| < |a-c|)$, the outlier is either *a* or *c* and *bd* should be the edge.

So we can conclude that drawing the edge between the least-different diagonal pair gives the same result as drawing the edge which isolates the outlier.

Demosaicing of Colour Images Using Pixel Level Data-Dependent Triangulation

Dan Su, Philip Willis
Department of Computer Science
University of Bath
Bath, BA2 7AY, U.K.
mapds, P.J.Willis@bath.ac.uk

Abstract

Single-chip digital cameras use an array of broad-spectrum Charge-Coupled Devices (CCD) overlayed with a colour filter array. The filter layer consists of transparent patches of red, green and blue, such that each CCD pixel captures one of these primary colours. To reconstruct a colour image at the full CCD resolution, the 'missing' primary values have to be interpolated from nearby samples. We present an effective colour interpolation using a simple pixel level data-dependent triangulation. This interpolation technique is applied to the commonly-used Bayer Colour Filter Array pattern. Results show that the proposed method gives superior reconstruction quality, with smaller visual defects than other methods. Furthermore, the complexity and efficiency of the proposed method is very close to simple bilinear interpolation, making it easy to implement and fast to run.

1 Introduction

Colour digital cameras have become widely available consumer products in recent years. In order to reduce cost, these digital cameras use a single Charge-Coupled Device (CCD) sensor with an overlayed colour filter array (CFA) to acquire colour images, thus avoiding the need for three separate arrays (one for each primary colour) and the associated complex optical system to split the light path.

The Kodak Bayer CFA pattern is the filter pattern most frequently used. Figure 1 shows this filter pattern, where R is red, G is green and B is blue. Each pixel of the CCD thus sees only one primary colour, determined by which filter overlays it. More green filters are used because of the visual importance of this central area of the spectrum: the eye is more sensitive to green and this area is more significant to the perceived luminance. The pattern shown thus provides a higher spatial frequency sampling of green, in comparison with blue or red. There are as many green pixels as red and blue combined.

G	B	G	B	G	B
R	G	R	G	R	G
G	B	G	B	G	B
R	G	R	G	R	G
G	B	G	B	G	B
R	G	R	G	R	G

Figure 1. Bayer Colour Filter Array Pattern (U.S. Patent 3,971065, issued 1976)

Since there is only one colour primary at each position, we can reconstruct the image at the spatial resolution of the CCD only if we interpolate the two missing primary values at each pixel. That is, at a green pixel we have to generate red and blue values by interpolating nearby red and blue values. A corresponding process is required at red (to get green and blue values) and at blue pixels (to get green and red values). This interpolation process is called CFA interpolation or *demosaicing*. The demosaicing process clearly has a significant influence and is thus the key factor in the production of high quality images.

The obvious place to start is with traditional image interpolation methods, such as nearest neighbour, bilinear interpolation and cubic convolution. Bilinear interpolation is often used due to its simplicity and efficiency[1]. However, it induces relatively large errors in the edge regions and the eye is especially sensitive to edge quality. To address this, other authors have proposed techniques which are sensitive to the data. Examples are Adams' edge oriented method [2] and various colour correlation methods [3, 4, 6]. Adams' method interpolates the missing colour elements according to the edge orientation of the image but it only detects the vertical and horizontal edges. Interpolation methods using colour correlation produce better results because there is a

high correlation between the red, green and blue channels. However they ignore the edge orientation in the images.

We have earlier proposed an effective interpolation algorithm using a simple pixel level data-dependent triangulation[5] which both matches the edge orientation of the images and correlates the red, green and blue channels. Our scheme generally produces superior reconstruction quality and is rapid. The method was applied to full-information images (that is, red, green and blue values for every pixel) with the aim of magnifying or rotating them. In this paper we show how our method can be adapted to supply the missing primary values of a CFA image – demosaicing – and the advantages it has in this application. First we need to summarize the principles of our earlier-reported method.

2 Our Data-Dependent Method

We consider an image to be an array of triangles, with an apex on each pixel value. In other words, we produce the finest mesh that the data directly supports. Our motivation is that processing images in this form is simpler than in pixel form. Significantly, the mesh permits us to think of the image as a continuous object rather than a discrete one: we can readily calculate the colour value at any position within the image, using interpolation across the triangle containing the point being sought.

There is some confusion of terminology in the literature, which we need to clarify before proceeding. "Bilinear interpolation" strictly refers to interpolating four points and we will use the term only in that sense. In the graphics community, three-value interpolation, as used in Gouraud shading, is also called bilinear interpolation, although it is only a degenerate case. We will distinguish this by calling it "triangle interpolation". (We are grateful to Professor Ken Brodlie, at the University of Leeds UK, for drawing our attention to this.)

Our scheme is based on the technique of data-dependent triangulation (DDT)[7] but we use it at pixel level. The algorithm attempts to minimize the visibility of the reconstruction errors, thus producing visually pleasing results.

Each four pixel square is divided into two triangles by a diagonal. The diagonal either goes in the NE-SW or the NW-SE direction, so we are free to choose which direction to use at each square. We choose the direction which more closely approximates the edge in that small area of the image. It is this which allows our method to be sensitive to edge direction.

We first consider the case that there is an edge passing between a square of four pixels. If this edge cuts off one corner, that corner's pixel will have a value substantially different (it could be bigger or smaller) to the other three. We call this pixel the *outlier*. If the luminance of the pixel

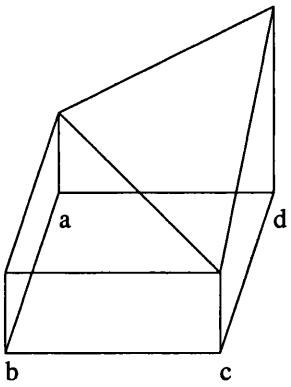


Figure 2. Triangulation in a four-pixel square

is the height of a terrain, then the three similar pixels are a plateau, relatively flat, while the outlier value is at the bottom of the cliff (if smaller) or the top of a peak (if higher) (Figure 2). So, if we want to interpolate a value within the relatively flat region we do not use the outlier. Choosing a diagonal the ends of which are not the outlier, we ensure that it runs in much the same direction as the actual edge. We then use triangle interpolation, avoiding the blurring produced by four-pixel bilinear interpolation.

Our mesh is thus regular in that there is a triangle apex at every pixel: it is a complete, regular grid and an $m \times n$ picture will always have $2 \times m \times n$ triangles. However, close examination will reveal that the diagonals are locally chosen to match the image edges. This completes our consideration of edge direction.

We now consider the choice of colour space. Recent demosaicing methods[3, 4, 6] have shown that taking the strong dependency among the colour planes into account can significantly improve the interpolation performance. In particular, Adams[4] and Pei[6] proposed interpolation in colour difference space instead of in the original colour space. (We will explain colour difference space later, in section 3.1.2) However, their methods do not directly take into account the edges of the image and so generate some colour misregistration. Our new method combined their use of colour difference space with our edge-sensitive mesh.

We use inverse mapping to do the interpolations[5]. In general, to evaluate the target pixel p, we evaluate pixel $F(p)$ where F is a simple inverse mapping to the original image. Then we interpolate in the triangle where it falls.

Experiments showed that images reconstructed by our scheme have better visual quality than those produced by bilinear interpolation. They lack the artifacts of colour misregistration of the edges thus improved subjective quality because human visual system are more sensitive to edges. We used two statistical tools to evaluate the reconstruction quality and the results showed that our method gets very close scores to bilinear interpolation in colour difference

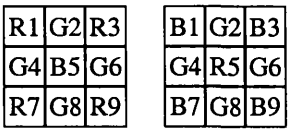


Figure 3. Left: Red square. Right: Blue square

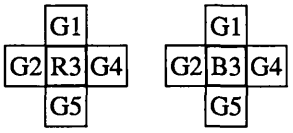


Figure 4. Green crosses

space which means good overall reconstruction quality.

The rest of the paper is organized as follows. In the next section we describe the new demosaicing method, justify it and describe the implementation. Then we briefly suggest some other applications. Next we show some experimental results. Finally we make some concluding remarks.

3 The Demosaicing Algorithm

So far we have considered data-dependent triangulation as a method for calculating super-resolution image values; that is, values "in between" the pixel positions. This is useful in changing the resolution of an image, distorting it in various ways, rotating it etc. In all these applications however, the original data is complete: there is a known (R, G, B) value at every source pixel. For demosaicing, we have to adjust it to generate those primary values which are missing from the Bayer CFA pattern.

3.1 Implementation

3.1.1 Original Colour Space

The Bayer CFA pattern alternates red and green filters on one row, then green and blue filters on the next row. This pattern repeats on subsequent pairs of rows. This means that a blue sample has red samples diagonally adjacent and green samples orthogonally adjacent (Figure 3). A red sample has blue samples diagonally adjacent and green samples orthogonally adjacent.

Our task is to interpolate the missing primaries in order to get a complete (R, G, B) triple at each position. What Figure 3 illustrates is the equivalence of blue and red; while Figure 4 emphasizes that the green samples are differently disposed. In fact, the green samples can be considered to be arranged on a grid at 45° relative to the other values. Moreover their spacing differs to that of the other values. The attraction of our DDT method is that it is independent of both the spacing and the orientation of the source data. It permits us to predict values at any spacing (regular or irregular) and orientation, wherever we need them.

If we consider just the red values, it can be seen that they form a regular grid of columns and rows. The same is true of blue values. We can triangulate each of these as already described, choosing the diagonals in the NW-SE or NE-SW direction, to favour the image edge directions. The green values can be thought of as forming a regular grid tilted at 45° (Figure 4). Triangulating this will produce diagonals which are in fact disposed either vertically or horizontally.

We therefore need to produce three meshes, one for each primary, with the green mesh being spaced and oriented differently to the other two. However, there is no need to produce these meshes explicitly. Suppose the sample image is X and the output image to be generated is Y. We first scan the sample image X to initialize three lookup tables, one for each primary. Each table has one bit to record the edge direction at every 2×2 'square' of pixels of that primary colour. To reconstruct an image pixel, we first determine which two primaries need to be recovered. We then use the corresponding lookup tables to establish in which triangle the image pixel sits in each mesh. This establishes three values to be interpolated for each of the two missing primaries.

In fact, only two values are needed. Suppose we are interpolating for red values of blue or green pixels. For demosaicing, the target pixel will alway fall on the boundary of the triangle. Hence the interpolation is always the average of two vertex values. For example, in Figure 3 left, the red value in $B5$ is actually the average of $R1$ and $R9$ or the average of $R3$ and $R7$ depending on the direction of the diagonal. The red value in $G4$ is the average of $R1$ and $R7$ and in $G8$ it is the average of $R7$ and $R9$.

The interpolation of blue values for red or green pixels can be done in exactly the same way as for red values. To interpolate green values of red or blue pixels, we simply locate the surrounding green cross and then interpolate either vertically or horizontally. For example, in Figure 4 left, the green value for $R3$ is either the average of $G1$ and $G5$ or the average of $G2$ and $G4$. In all cases therefore, the value is reconstructed as the average of two source values, those values being chosen by our DDT method. This improves on our earlier method, both in simplifying the interpolation and in avoiding the need for inverse mapping.

3.1.2 Colour Difference Space

Treating R, G and B planes independently ignores the correlation among the colour planes and produces colour mis-

registration. Recent research [4, 6] has shown that interpolation performance can be significantly improved by exploiting the correlation among the colour planes. Those methods are based on the assumption that the red and blue values are perfectly correlated to the green value over the extent of the interpolation neighborhood. They define the colour differences $K_R = G - R$ and $K_B = G - B$ and interpolate in this colour difference space.

In other words, these methods transform the operation into the K_R or K_B domain instead of performing the interpolation in the G channel. For example, using our method to interpolate the missing green element at a red pixel, say $R3$ in Figure 4 left, would use either $G1$ and $G5$ or $G2$ and $G4$. In fact we use the colour difference approach, so instead interpolate K_R from K_R1 and K_R5; or K_R2 and K_R4. The green value of this red pixel is thus recovered by $G = R - K_R$. The missing green element of blue pixels can be interpolated in the same way and the interpolation of red and blue elements uses the same approach.

This method is based on the assumption that colour difference are relatively flat over small regions. This assumption is valid within smooth areas of the image but is not valid around the edges in the image. Colour misregistration would still exist around the edges if bilinear interpolation was applied. Our method effectively solves the problem by interpolation *along* the edges in colour difference space, as Figure 5 shows. It avoids colour misregistration by not interpolating *across* the edges in the colour difference space.

3.2 Other Applications

As already mentioned, pixel level data-dependent triangulation can be used in arbitrary resolution enhancement, arbitrary rotation and other applications of still images in continuous space. This remains true for demosaicing: we are not obliged to reconstruct at the CCD resolution or orientation. However, there is less information available than in a full colour image, so we cannot expect the quality to be as high. What we do claim is that there is less introduced visible error; what there is, is visually acceptable.

4 Experimental Results

4.1 Quality Assessment

We have performed various tests on two images, one of a boat (Figures 5 and 6) and one of a macaw (Figures 7 and 8). In each case, the top image is the original 24 bit image of size 768 × 512. From this we prepared a mosaic image by, at each pixel, discarding the two primaries indicated by the CFA pattern. This mosaic image was then used to perform the various reconstructions shown, again at 768 × 512.

We applied three different demosaicing methods for the test images: bilinear interpolation in the original colour space, bilinear interpolation in colour difference space and our data-dependent triangulation method in colour difference space.

If we compare Figure 6(b) and Figure 6(c), it can be seen that interpolation in the colour difference domain has better reconstruction quality than interpolation in the original colour space. Colour misregistration is clearly visible near the thin lines in the boat picture and around the top of the macaw where there is a sharp colour transition. There are also noticeable dotted artifacts around those edges. Our method avoids both of these problems because it better preserves the geometric regularity and interpolates along the edge orientations of the image.

Direct visual inspection indicates that our method produces good reconstruction quality. However, we wanted to explore a more analytical assessment of the visual quality of the interpolated images, though this is not straightforward to define, let alone measure. Degradation-based quality measures[8] and the visible differences predictor (VDP)[9] have been proposed. These two vision models are quite complicated and it is difficult to compare different reconstruction methods. Instead we chose two methods to assess the reconstruction. One is *cross-correlation* proposed by Battiao *et al.*[10] and Lehmann *et al.*[11]. They use cross-correlation between the original picture and the reconstructed picture to assess the quality of reconstruction. The other is *Peak Signal-to-Noise Ratio* (PSNR) which is commonly used as a measure of image quality in digital image compression and reconstruction.

The cross-correlation coefficient C between two images X, Y is:

$$C = \left| \frac{(\sum_{i,j} x_{ij} y_{ij} - IJab)}{\sqrt{((\sum_{i,j} x_{ij}^2 - IJa^2)(\sum_{i,j} y_{ij}^2 - IJb^2))}} \right|$$

where a and b denote respectively the average value of image X and Y; and I and J are the image's width and height. The cross-correlation coefficient is between 0 and 1, where a higher score means better reconstruction quality.

The PSNR is based on *Mean-Squared Error* (MSE). The MSE is the cumulative squared error between the reconstructed and the original image. The Mathematical formulae for the two are:

$$MSE = \frac{1}{IJ} \sum_{i,j} (x_{ij} - y_{ij})^2$$

$$PSNR = 20 \log_{10} \frac{S}{\sqrt{MSE}}$$

where S is the maximum pixel value. Logically, a higher value of PSNR is better because it means that the ratio of

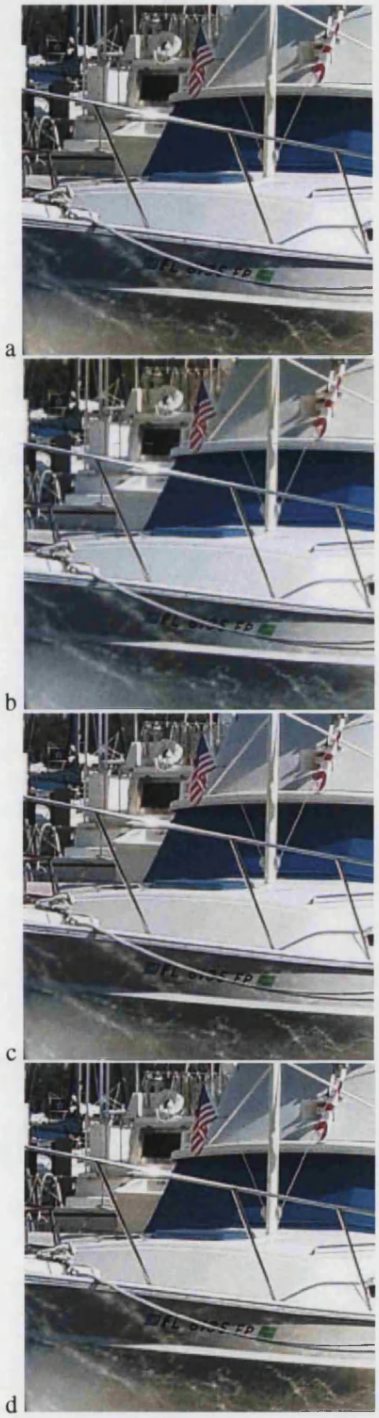


Figure 5. Portions of: a: original boat image. b: bilinear interpolation in the original colour space. c: bilinear interpolation in the colour difference space. d: our method in the colour difference space

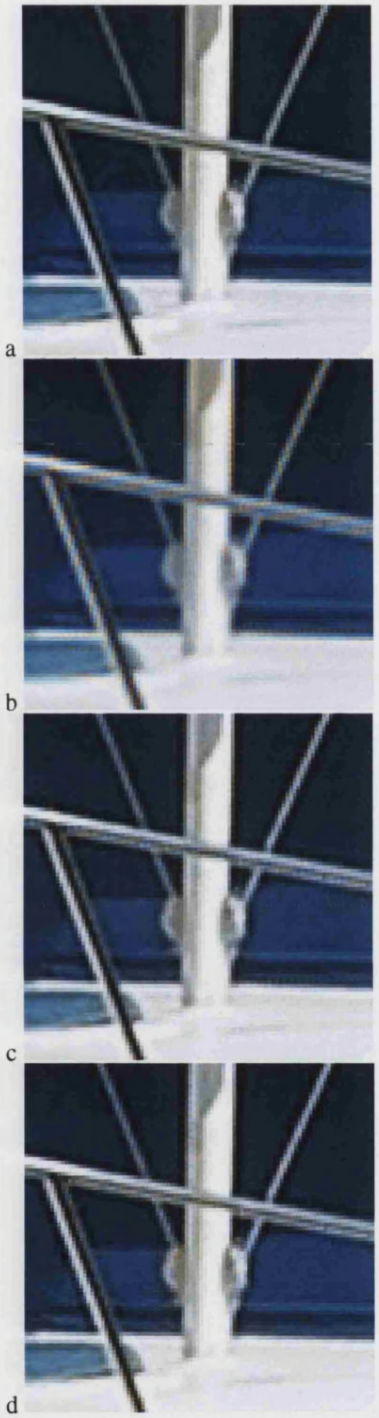


Figure 6. Close-up comparision of: a: original boat image. b: bilinear interpolation in the original colour space. c: bilinear interpolation in the colour difference space. d: our method in the colour difference space

Figure 7. Portions of: a: original macaw image. b: bilinear interpolation in the original colour space. c: bilinear interpolation in the colour difference space. d: our method in the colour difference space

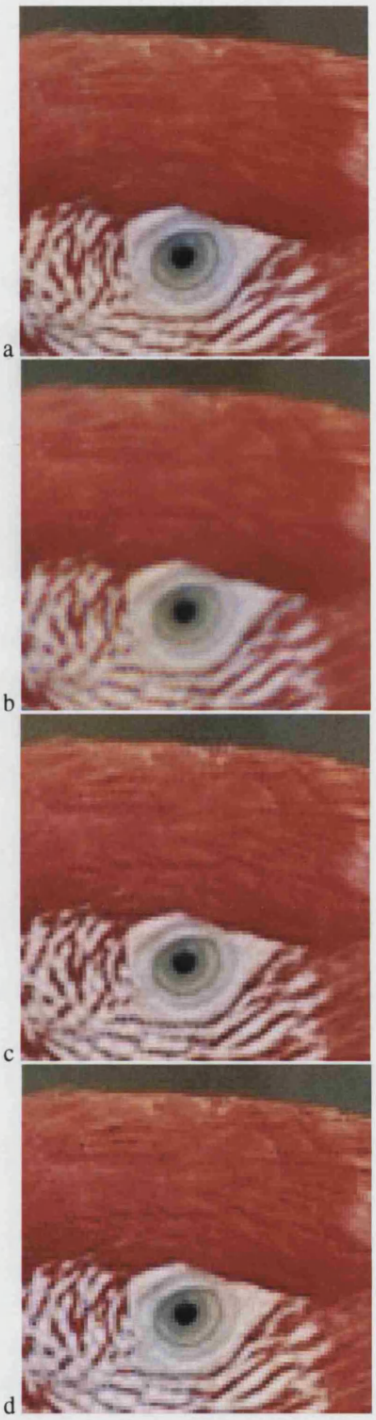


Figure 8. Close-up comparision of: a: original macaw image. b: bilinear interpolation in the original colour space. c: bilinear interpolation in the colour difference space. d: our method in the colour difference space

Signal to Noise is higher. Here, the 'signal' means the original image and the 'noise' is the error in reconstruction.

We used twenty 24-bit 768×512 colour nature images as our test set. Because we use colour images, we compute the cross-correlation coefficients of the R, G, B planes independently and average these three. The PSNR values are calculated for the three colour channels independently.

The table below shows the corresponding cross-correlation results where BL means bilinear interpolation in original space, BLD means bilinear interpolation in colour difference space, DDT means data-dependent triangulation (our method) and $DDTD$ means our method in colour difference space. The values are averaged over the test set.

BL	BLD	DDT	$DDTD$
0.989957	0.995719	0.988859	0.993706

Cross-correlation results show that bilinear interpolation is marginally better than our method, when both use the same colour space. Colour difference space is better than original colour space but only by a very small amount, approximately 0.5%. The table below shows the PSNR results of three colour channels for the different methods. The results are averaged over the twenty images.

	BL	BLD	DDT	$DDTD$
R	31.4685	35.0733	31.2224	33.9629
G	35.3369	39.3796	35.0995	37.9909
B	31.0098	34.2068	30.7429	32.3131

The PSNR results show a clear benefit from the use of colour difference space, for both bilinear interpolation and our method. These results confirm earlier work supporting colour difference space. When comparing the two methods, the PSNR results show that bilinear interpolation is only marginally better than ours. As we discussed, our method is designed for solving the problem of colour misregistration in edge areas. So for images which mainly consist of smooth areas, bilinear interpolation will give a better statistical result because it uses more information for interpolation. Informally observation confirms that our method gives improved edge quality. It looks better because human eyes are more sensitive to edges and our method is better at retaining edges. Our overall result is very close to bilinear interpolation which means our method produces good overall reconstruction images. It is however simpler to implement.

4.2 Performance Assessment

The following table shows the performance comparison on a Pentium II 400 machine with 256M memory. We used the 20 images again and timed the four methods. All the methods are implemented by C++ code and all the figures in the table are seconds.

BL	BLD	DDT	$DDTD$
1.82	5.71	2.32	3.66

As we expected, both methods using the colour difference space require more computation than the method in original colour space. Of the two methods using colour difference space, our method is faster. This is true even including the overhead of initializing the triangulations in three colour channels (About 1.05 seconds in this case). Our method is significant faster because it only requires two pixels to interpolate while bilinear interpolation requires four pixels. We have already shown that it has good overall quality and visually better edges. We suggest that these features make it a better choice for demosaicing colour images.

5 Conclusion

In this paper we have presented a new method for demoisaicing of colour images. The new method is based on data-dependent triangulation but we use it at pixel level. The interpolation is done within the triangulation, which matches the edge orientation of the images. By avoiding interpolation across edges, the new algorithm successfully solves the problem of colour artifacts around the edges. We also applied the scheme in colour difference space which helps to reduce the artifacts caused by colour misregistration.

We have applied our method to the Bayer CFA pattern and our method offers simplicity and efficiency. The cross-correlation and PSNR results also demonstrate that our method is very close the best comparator in producing the 'right' data, while visual inspection shows that the data is more effectively deployed to produce sharp edges. It is also much faster.

Our method can also be used for general image manipulations such as arbitrary scaling, rotation, perspective transform, warp and so on. Thus it can be used for digital zoom and simple effects.

Acknowledgments

The authors would like to thank Dr J E Adams at Eastman Kodak Company in Rochester, NY, USA for providing the macaw image and the Eastman Kodak Company for the boat image. The work is funded under the UK EPSRC project *Quasi-3D Compositing and Rendering* and by an *Overseas Research Scholarship*.

References

[1] T.Sakamoto, C.Nakanishi and T.Hase "Software Pixel Interpolation for Digital Still Cameras Suitable for a 32-Bit MCU" *IEEE Trans. Comsumer Electronics* Vol. 44, No. 4, pp. 1342-1352, Nov. 1998

[2] J.E.Adams, Jr. "Interactions between Color Plane Interpolation and Other Image Processing Functions in Electronic Photography" *Proc. of SPIE* Vol. 2416, pp. 144-151, 1995

[3] R.Kimmel, "Demosaicing:Image reconstruction from colour CCD samples, " *IEEE Trans. Image Processing* Vol. 8, pp. 1221-1228, Sept. 1999

[4] J.E.Adams, Jr. "Design of Practical Colour Filter Array Interpolation Algorithms for Digital Cameras" *Proc. of SPIE* Vol. 3028, pp. 117-125, 1997

[5] D.Su and P.J.Willis, "Image Interpolation by Pixel Level Data-Dependent Triangulation", submitted to *Computer Graphis Forum, October 2002*

[6] S.C.Pei and I.K.Tam, "Effective colour interpolation in CCD colour filter array using signal correlation" *Proc. IEEE Int. Conf. Image Processing* Vol. 3, 2000, pp. 488-491

[7] N.Dyn, D.Levin, and S.Rippa, "Data Dependent Triangulations for Piecewise Linear Interpolation", *IMAJ. Numerical Analysis,* Vol. 10, pp. 127-154, 1990.

[8] N.Damera-Venkata, T.D.Kite, W.S.Geisler, B.L.Evans and A.C.Bovik, "Image Quality assessment based on a degradation model." *IEEE Transactions on Image Processing*, Vol. 9, pp. 636-650, April 2000

[9] S.Daly "The visible differences predictor: An algorithm for the assessment of image fidelity." *Digital Images and Human Vision* A.Watson, Ed. Cambridge, MA:MIT Press, 1993

[10] S.Battiato, G.Gallo, F.Stanco, "A locally-adaptive zooming algorithm for digital images" *Image and Vision Computing* Vol. 20, no. 11, pp. 805-812, September 2002

[11] T.M.Lehmann, C.Gonner, K.Spitzer, "Survey: Interpolation Methods in Medical Image Processing", *IEEE Transactions on Medical Imaging*, Vol. 18, no. 11, November 1999