University of Southampton
Faculty of Engineering and Applied Science
Department of Electronics and Computer Science

Content Based Image Retrieval Using Scale Space Object Trees

David Paul Dupplaw

A thesis submitted for the degree of
Doctor of Philosophy
September 2002
Supervisor: Dr. P. H. Lewis

University of Southampton

ABSTRACT

Faculty of Engineering and Applied Science

Department of Electronics and Computer Science

A thesis submitted for the degree of
Doctor of Philosophy

Content Based Image Retrieval Using Scale Space Object Trees

by David Paul Dupplaw

Semantic extraction from images is of growing interest in the field of content based retrieval. Classical algorithms are mainly limited to the use of features such as colour or texture estimated globally over the whole image, or the use of shape description for clearly extractable object boundaries. There is little provision for the searching of images based on any semantic notion, other than by the use of matching keywords which are manually associated with the images. We believe that it is the combination of a number of image properties, estimated locally for individual regions rather than globally, that will create the ability to recognise objects, and from there we can work towards bridging the semantic gap.

Using recent advances in scale-space decomposition techniques, it is possible to transform images into trees in scale space, which represent the topology of regions within the image. Using a sieve mechanism we can decompose an image into a discrete scale-space based on the pixel area of regions within the image. Extrema within the image are merged to create new extrema. The merge operations are represented by branches in a tree. The tree then represents the topology of extrema in an image.

We propose that complex objects can be found in the trees using a combination of subgraph isomorphism testing and conventional feature matching and hence provide a vehicle for achieving content-based retrieval and navigation for complex objects in images.

Explanations are given for the subgraph isomorphism techniques, and how they can be supplemented with feature matching to produce scores for feature-topology matches of images.

Finally, we will explain how the scale-trees could be used in combination with a semantic layer to achieve semantic recognition of scenes, by recognition of parts of the topology of a scale-tree as particular objects.

Acknowledgements

The author is grateful to the Engineering and Physical Sciences Research Council (EPSRC) for the support of a research studentship to undertake this work.

Also, I would like to extend my thanks to all the help my supervisor has given me throughout the degree, and also to my parents and my girlfriend for being so supportive.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Computer technology is becoming more pervasive in our everyday lives. With this trend and the strong visual sense of humans, computer interfaces have been moving towards graphical and image manipulation interfaces. Also, with the fast growing capacity of storage devices and network connections and the ease of acquisition of images via digital cameras and cheap desktop photo scanners, the abundance of images available to any one person is becoming overwhelming. Surveys(23) have estimated that world-wide 2,600 new images are created per second (80 billion per year) with an estimated 10 billion of which are available on the internet. Finding the correct image has become an expensive problem.

Because market forces have made image acquisition and storage relatively cheap only within the last few years, there has been limited time for search algorithms to develop. There has been work on the recognition of images, from retrieval of particular images from a large database, to recognising faces from digitised images. But, the main problem with most of the current techniques is that they are very feature oriented. Typical algorithms involve primitive features (colour, texture and shape) estimated globally and though these features may allow fast searching, their usefulness is limited by their simplicity.

Typically a user would not search for something of a particular colour, nor something of a particular shape, but of a particular object, or set of objects. Systems which attempt to solve this problem are few and far between, and, as yet, there are no systems that extract semantics from images in general. The pre-requisite of searching in the object domain is the problem of attempting to decompose an image in a way in which objects, as opposed to features, can be retrieved. This may involve using a hybrid of the primitive features.

The range of image retrieval systems is quite great. At the simplest it can involve text matching which finds images based on keywords removed from the text which surrounds them, such as the Google image search engine (http://images.google.com/). Natural language understanding, where users use phrases to ask for certain images, provides better context to keyword searches. However, relying on the relationships between images and their textual context is unreliable, because it cannot be guaranteed that an image is associated with nearby text due to formatting, or page decoration by the author. Therefore, matching image content is likely to be an improvement on assuming a textual relationship between an image and its context or keywords. Using colour, texture and shape primitives is the first step into content based image retrieval. Systems like QBIC, and Virage implement this type of content-based retrieval. Object retrieval is more complex and could possibly be achieved using combinations of primitive features to move towards object based retrieval; e.g. "find a bus". eVision's Visual Engine (eVe) claims it can achieve object recognition with its unsupervised object segmentation algorithm, and matching, but using only low-level features it is unlikely to be successful in most cases. Scene matching (e.g. "find London") is the simplest form of semantic matching, before moving on to touch (e.g. "find something rough"), and emotion (e.g. "find a happy scene"). The intellectual distance between simple feature based matching or object recognition and full semantic recognition is called the semantic gap and it is currently gaping.

Much of the well documented research is based on the matching of single sets of image-wide features to retrieve similar images. There have been attempts to find more localised similarities within image, and a few have used the adjacency of regions within the image applied to content based retrieval. However, new scale space techniques allow us to decompose the image across scales, and allow us to find regions over various scales. Adjacency between scales becomes important for the topology of the image - within which there may be the topology of one or more objects. From the scale-space domain, trees can be built. Matching these trees, and sub-trees thereof, allows for object based retrieval. Retrieval within a knowledge base would allow for object recognition. All of the recognised objects within an image could be aggregated into a scene graph, which could be matched against semantic concepts in the knowledge base. We have not seen these ideas presented elsewhere, even without the application of content based retrieval.

1.2 Research Objectives

The broad aim of our research is to improve feature representation for content based image retrieval and navigation by extracting, from images, features which are more relevant than those currently in use.

An initial objective was to improve colour based image retrieval with a spatial representation of the colour of an image, and associated matching technique.

As research progressed, a second objective emerged which is to improve general image

retrieval by extracting the topology of an image using scale space techniques and using the topology as a representation for objects in the image and their relationship. It should then be possible to develop schemes for matching topology and sub-topologies in order to locate objects.

1.3 Thesis Structure

This thesis presents how, using trees generated from scale-space decompositions of images, it is possible to search for object topologies, and therefore locate similar objects.

Chapter 2 gives an overview of the current commercial and research content based image retrieval systems, and describes the current state of the content based retrieval community's work on colour, and shape retrieval. The chapter also describes other ways that have been developed to build scale-space trees.

Chapter 3 describes our initial work into improving the content based retrieval techniques currently used, describing some partly new work into spatial colour histogram retrieval. Finally, the chapter describes some related work, on extraction of particular objects (flint hand-axes) from images using well known techniques. The extraction is primarily used for measurements, rather than retrieval, but is new within its field.

Chapter 4 and chapter 5 describes our novel work into providing a new approach to content based retrieval using scale-space trees.

Chapter 4 briefly describes how, with a semantic layer idea, the new image representations can be used to work towards object recognition.

Finally, in chapter 6 we summarise the thesis and bring together conclusions from chapter 5. We also explain further work that could improve the approach we have presented.

Chapter 2

Literature Review

The state of the art in the field of content based retrieval is changing rapidly. From the mid-80s when content based retrieval emerged as a possibility the field has grown to include various different fields of research, from colorimetry, to subgraph isomorphism detection.

In this chapter we will briefly explain some of the content based retrieval products that are on the market, and then some of the techniques that are used to achieve content based retrieval and feature extraction.

2.1 Current Content Based Retrieval Systems

This section describes some of the products which are available on the market to achieve content based retrieval on images, and other multimedia. There are far too many content based techniques and systems to be able to describe them all here so we also point the reader to reviews of the field that can be found in (69; 22; 64; 78). What follows is a brief description of some of the well known and well regarded systems in the field.

2.1.1 QBIC

IBM's QBIC System (25; 54; 55) is probably the most well known content-based retrieval engine for video and images. The image feature extraction engine uses colour, shape, and texture. The use of colour in QBIC was originally limited to the overall colour histogram of an image, or percentages of colour within an image, however, more recent versions of QBIC have included a widget which allows queries based on the spatial colour layout of images to be created (query by sketch). This system is based on a grid mechanism similar to that

described in Section 2.2.3.1. The shape features consist of area, circularity, eccentricity, major axis orientation, and a set of algebraic moment invariants.

2.1.2 eVe

eVe (the eVision Visual Engine) (23), is a commercial product developed by eVision, LLC Technologies. The engine uses automatic segmentation techniques applied to colour, texture, shape, and visual and text meta-tag searching. It attempts automatic segmentation by grouping pixels based on pixel similarity and labels the clusters as objects. Although they profess that this *"brings unsupervised segmentation to the commercial world"* many of their examples contain objects on white background, and those which do not are poorly segmented. The query engine is query-by-example only. This helps with their visual tagging system, where all the objects in the catalogue are pre-clustered based on visual similarity. Users can add metadata to the clusters of images based on certain features (i.e. only the shape, or colour) allowing the searching process to return results quickly based on the clusters.

2.1.3 PicToSeek

PicToSeek (28) is a content-based image search system, designed for use on The Web by the Intelligent Sensory Information Systems research group, at the University of Amsterdam. The system uses a colour model that is colour constant - that is, it is independent of the illumination colour, shadows, and shading cues. PicToSeek, however, is only concerned with the whole image histograms, and does not allow spatially oriented queries.

2.1.4 Virage

Virage (6; 31) is a system produced by Virage Inc. that performs content-based retrieval on video and images, using colour, texture, composition (colour intensity distribution), and structure (shape layout). It also allows for combinations of the above to be used in a single query, unlike QBIC (64). Weights can be varied for each feature type according to the user's needs. The framework was also extended to include domain specific features (such as face detection), as well as the general features (colour, texture, etc.).

2.1.5 RetrievalWare

RetrievalWare is a commercial system developed by Excalibur Technologies Corp. that, among other things, uses colour, shape, texture, brightness, colour layout and image aspect ratio in a query-by-example paradigm to match images. Like Virage, it allows combinations of these features to be used.

2.1.6 VisualSEEk

VisualSEEk (70) is a content based search engine designed at the Center for Image Technology for New Media, at Columbia University, New York. The system uses colour set back-projection to extract regions of colour from images. Colour back-projection is a way of automatically extracting salient regions, by quantizing the image based on the 'colour sets' - which are thresholded histograms. Because colour sets are binary, the histogram matching functions can be reduced which allows efficient indexing. VisualSEEK allows spatial-colour retrieval based on a query built from areas of solid colour, and semantic relations between those areas. The system also includes a wavelet based texture feature. WebSEEk is a web-based version of the VisualSEEk engine that supports queries on both keywords and visual content.

2.1.7 Color-WISE

Color-WISE (67) is an image similarity retrieval system which allows users to search for stored images using matching based on the localized dominant hue and saturation values. It uses a cunning fixed segmentation of overlapping elements to ensure that the matching is slightly fuzzy. The system computes seperate histograms for hue, saturation, and intensity, and reduces their size by finding their area-peak - basically removing noise that is small amounts of isolated colours. Color-WISE uses Microsoft Access to perform the database functions, and uses a similarity metric based on IBM's QBIC system. Querying in Color-WISE is achieved with query-by-image.

2.1.8 Photobook

Photobook from MIT consists of a number of "subbooks" which allow shapes, textures and faces to be extracted from images. Users can query based on the features in each of the subbooks. Later versions allowed human authors to help annotate images, based on "models of society" that reflect the particular domain and set of users.

2.1.9 Blobworld

The Digital Library Project (26) taking place at the University of Berkeley, California, has developed a system called Blobworld which uses low-level grouping techniques to create "blobs of stuff", which can be texture, colour, or symmetry. The blobs can be matched against their content, and their position, and it is possible to use high-level techniques to analyse the semantics of the blobs (such as where they are in relation to other blobs), and conclude what they might represent.

2.1.10 Image-MINER

Image-MINER (5) is an image and video retrieval system developed by the AI group at the University of Bremen. Their colour indexing system for images uses local histograms in a fixed grid geometry. Further grouping of the fixed elements occurs to get 'color-rectangles', which are signatures for their input images. The colour based segmentation module, is part of the larger Image-MINER system which includes video retrieval methods, including shot detection and subsequent 'mosaicing'.

2.1.11 MARS

MARS (Multimedia Analysis and Retrieval System) (63) is a multimedia retrieval system, that is designed to retrieve text, images and video. The image retrieval engine they use is based upon colour and texture. Colour histogram intersection and colour moments are used to match whole images, as well as co-occurance matrix and wavelet based methods of texture analysis.

2.1.12 Other Systems

ART MUSEUM(36) is one of the earliest content based image retrieval systems, developed in 1992. Some other systems include IMatch by mwlabs which uses colour, texture, and shape for content based retrieval, DART by AT&T which also uses colour, texture, shape, as well as locally smooth regions, Netra using colour, texture, shape and spatial location information, with a neural net based automatic image thesaurus construction.

2.1.13 Summary

There are many content based retrieval systems which have been developed within the last decade using relatively simple matching techniques. As described above, most of these systems use colour, and texture, and some have basic spatial location retrieval. However, few of them access the semantics of the image and therefore fail to allow retrieval within the semantic domain. Very recently in the literature, semantic retrieval is becoming an active research topic. For example, Wang et al., in (80), propose a semantic retrieval technique based on region saliency, within the image and of itself, along with texture details. This may be a way to achieve certain semantic-type queries ("find me sky and grass"), but is unlikely to be able to achieve object matching, as objects may consist of a number of regions as a particular topology. The authors in (42) use the topology of outline shape, but fail to use other low level region properties, such as colour and texture.

2.2 Feature Extraction and Retrieval Techniques

Currently, low level feature matching is the underlying method of any content based retrieval system, and it is unlikely to change for the foreseeable future. Therefore it is necessary to have techniques for matching low level features of images, which are robust and, to some extent, useful on their own.

However, for applications which have to identify objects within real world scenes, single low level features alone are inadequate, and it is often usual to combine low level features, and their matching techniques, to present a more complex, and potentially more discriminating, representation of an image. By combining the strengths of a number of features, objects may be able to be identified in the scene where any of the single features used might fail.

Most feature matching techniques will return a similarity, or conversely a distance measure, which represents how similar the two considered *things* are. This alleviates the need to arbitrarily pick some domains into which the *things* can be classified into, allowing for a continuous measurement. However, without selective retrieval by indexing techniques, it requires that all possible known instances be considered against the unknown, or query, *thing*. This will be time consuming for a system with a large database of known instances, so often, during indexing, certain groups of instances are disregarded as being too dissimilar from the query to contribute to the results. This increases the precision of the match, and, in a good indexing system, does not affect recall.

Of the low-level features, colour is the most readily identifiable property, and as such, much of our everyday world is based around colour; for example, in the western world red means stop and green means go, and often if the identity of an object is unknown it is referred to instictively by its colour (e.g. "what is the green thing over there?"). Indeed, many objects are only distinguishable by their colour, which is why colour-blindness is still considered a problem.

Early feature techniques disregarded colour as a valid feature for recognition, maybe due to the earlier expense of colour camera systems. Many also believed that colour, as well as texture, were only secondary roles in primal access, and that the geometric, and volumetric shape of an object were the only efficient access to representations of objects. Swain and Ballard pointed out in their famous paper on colour indexing (73), that *"for routine behavior, ... wherein familiar objects are interacted with repeatedly, color may be a far more efficient indexing feature."*

Having pointed this out, there are many instances in which colour is a misleading feature, in particular when there are coloured lights involved. Even under normal room lighting, captured images tend to take on a yellow hue if not corrected, which can cause problems when searching for similar objects. It may have been the lack of good algorithms to conquer this problem that also led to colour being disregarded early on. However, there has been much work on finding illumination invariant colour features which we will explain later.

Below we describe the colour histogram - a method for storing an image's colour data and for matching images based on colour. We will describe how the histogram can be augmented to find the position of certain colours in the image.

2.2.1 The Colour Histogram

The colour histogram is a representation of the colour content of an image. It counts the frequency of each colour in the image which fall into "bins" in a quantised colour space.

The colour axis of the colour space can be of any type, and most commonly RGB (red, green, blue) histograms are generated due to the ease of acquiring the data from the raw image pixels. Often HSB/V (hue, saturation, brightness/value) histograms are used as they provide better separation of the chromaticity from the luminance while also being relatively easy to compute. Another system which also separates chromaticity and luminance is the Opponent Colour Space, where the axes are $rg = r-g, by = 2*b-r-g, wb = r+g+b$ where the wb axis then represents the luminance. The problem with colour spaces that separate luminance from colour, is that there are no longer single bins representing white and black. For example, when saturation in HSB/V space is 0 (grey scale) hue is undefined. In a $3 \times 3 \times 3$ HSV space, there would be 3 bins for every level of grey, with 9 black bins. In which bin should a black pixel be placed? By distorting the quantisation a little one can overcome the problem by merging all grey bins into single bins, giving an asymmetric histogram.

Directly observing the data in the histogram can allow the simplest form of filtering possible, by returning those images with a certain proportion of colour in them, for example, 75% red.

The simplest form of matching two histograms is to use the sum of the absolute differences of the bins (L_1-distance) or the sum of the squares of the differences of the bins (L_2-distance). So, the most similar image M to the query image Q will be the histogram, H_M most similar to the histogram of the query image, H_Q. Given n bins in the histograms, the most simlar image is that image minimising the L_1-distance (Manhatten or City Block measure):

$$\|H_Q - H_M\| = \sum_{j=1}^{n} \|H_Q[j] - H_M[j]\| \tag{2.1}$$

or the L_2-distance (Euclidean distance measure):

$$\|H_Q - H_M\| = \sum_{j=1}^{n} (H_Q[j] - H_M[j])^2. \tag{2.2}$$

where $H_Q[j]$ is the j^{th} bin of the histogram H_Q.

Matching using the Euclidean or Manhatten metric is not ideal, because the quantisation of the histograms into bins causes aliasing of the histogram data; where two colours may be close in colour space they may be placed into opposite ends of neighbouring bins by the equivalence function. The crude quantisation of the bins could be overcome by using overlapping bins,

or by associating a Gaussian probability with the colour of the bin, so that colours nearer the edges of a bin would be less likely to belong to the bin they are in, and more likely to belong to the neighbouring bin. A technique which achieves this is described in Section 2.2.1.3.

Walcott and Ellis in (79) give an algorithm which searches for objects based on their colour. They use a segmentation to find model colour cues within the image. They segment out the model colours using a region growing algorithm. They match the model colours using a variation on the histogram intersection metric. The technique outperforms many colour based algorithms on their test set, but doesn't take into account the topology of the image and is still based on an illumination dependent technique.

The matching techniques described here work on a simple set of pixels. This can be used to find similar colour images, or with image decomposition, can be used to find similar colour image-parts. Often these images are combined into a histogram on which the matching takes place.

2.2.1.1 Histogram Intersection

Histogram intersection, as defined by Swain and Ballard in 1991(73), finds the number of pixels from the model image that have similar colour pixels in a match image. Given a pair of histograms, I and M, each containing n bins, the normalised histogram intersection is defined as:

$$H(I,M) = \frac{\sum_{j=1}^{n} min(I_j, M_j)}{\sum_{j=1}^{n} M_j} \tag{2.3}$$

where I_j is the j^{th} bin of histogram I.

Swain and Ballard extend the technique to allow for efficient indexing to a large database with their Incremental Intersection. This basically uses the largest value bins in the model and query histograms to find an approximate match. Then it uses the other bins to get better, more accurate results. Because histogram intersection is somewhat insensitive to the number of bins used in the histograms, the computation can give reasonable results back after the initial stage.

Although the histogram intersection calculation is fast, and Swain and Ballard prophesied that it could be used in real-time systems with hardware acceleration, the examples given show that it only works well, when objects are already segmented, and for automated image classification and recognition the results would be less acceptable due to extraneous pixels.

Swain and Ballard went on to explain how the best match in an image can be found using histogram techniques, in particular, their method called Histogram Back-Projection.

2.2.1.2 Histogram Back-projection

Swain and Ballard's paper (73) describes the histogram back-projection technique, which locates the position of the best colour match of pixels, using histogram intersection, by convolving a disc subtending the expected area of the final object with an image which has been

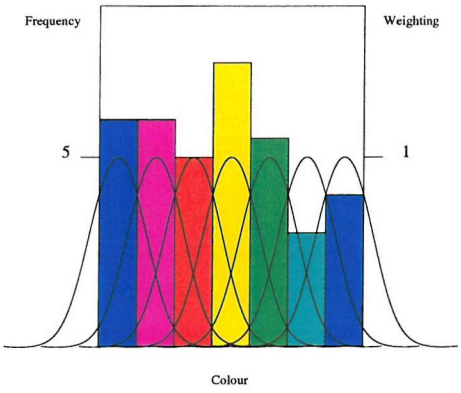


Figure 2.1: Quadratic form histogram matching: each bin is weighted against all other bins to reduce the arbitary selection of the equivalence function.

back-projected with a ratio histogram.

Given histograms I and M, the i^{th} bin of the ratio histogram, R_i, is defined by:

$$R_i = \min\left(\frac{M_i}{I_i}, 1\right) \tag{2.4}$$

This is analogous to the circular Hough transform; the final image contains a match score for each position in the image for the colour of the subtended circle against the colour of the query object. However, convolution is a relatively expensive operation, so Swain carried out the convolution on reduced resolution images. This traded efficiency in the algorithm, against accuracy in the final results. For their intended robotic vision application they felt this was an acceptable compromise.

Histogram back-projection was used to detect flaws in products on a production line with a special colour-profile in (20). The colour profile is built up from a training set of images, and rather than creating a ratio histogram which is back projected, a representative histogram containing those colours which are flaws in the product is created. Back projection of this onto the images of the products produces easily seen areas which are due to flaws.

2.2.1.3 Quadratic Match

The quadratic matching approach was developed by IBM for their QBIC (Query By Image Content) system(54; 25; 55). They found the quadratic match to more closely correspond to the human judgment of colour(29) than Swain and Ballard's. A particular advantage is that using the quadratic match, the bins in the histogram influence each other during the match by an amount dependant on their distance from each other in colour space (see figure 2.1). This means that the problem of similar colours being placed in neighbouring bins due to the equivalence function is partially overcome.

In QBIC the colour features(54) are based on a 256 cell Munsell colour space using the Mathematical Transform to Munsell (MTM), although it can be used with any colour space and quantisation. An inverse table of colour to cell number is produced which allows fast indexing to a histogram bin from the RGB colour space. The histogram is built and normalised so that it sums to unity.

To match two histograms, H and M, the bin by bin difference is calculated into a vector, Z.

$$Z_i = H_i - M_i \tag{2.5}$$

A symmetric similarity matrix, A, is calculated with:

$$a(i,j) = 1 - \frac{d(c_i, c_j)}{d_{max}} \tag{2.6}$$

where c_i and c_j are the i^{th} and j^{th} bin's colour coordinates in the chosen colour space, $d(c_i, c_j)$ is the distance in the colour space between those two colours, and d_{max} is the maximum distance between any two colours. The match value is then given by:

$$\|Z\| = Z^{\mathrm{T}} A Z \tag{2.7}$$

The advantage of this over the histogram intersection is that the weighting between bins is implicit in the matching. The similarity matrix combines the weights between every bin with every other bin, so that the match of a particular bin is affected by those nearby, going some way to alleviating the problem of pixels falling in between two bins, then being placed in one arbitrarily.

The main problem with this method is the calculation cost, which can become prohibitive in a large database, with high resolution histograms. In (29) they describe a method to increase the response which uses a cheap distance measure to filter a large fraction of the database without incurring any misses. Then the more expensive quadratic matches can be used on the remaining database.

Of course, these methods share disadvantages with the Swain and Ballard method. For example, it only works well when objects are already segmented, and would not be accurate for automatic image classification or recognition because the results would be affected by extraneous pixels.

2.2.1.4 Colour Coherence Vectors

Colour coherence vectors (CCVs)(58) classify pixels on whether or not they are coherent - that is, part of a "sizable", approximately homogeneous region. The colours in the image are quantised into their histogram bins, and connected sets determined from the quantised pixels. The pixel count of each connected set is taken, and those sets which are smaller than a predefined threshold, θ, are considered incoherent. The coherence pair (α_j, β_j) for the j^{th} colour

quantised colour represents the number of coherent pixels, α, and the number of incoherent pixels, β. The coherence vector is a set of coherence pairs - one coherence pair per quantised colour: $\langle(\alpha_i,\beta_i),...,(\alpha_j,\beta_j)\rangle$. The vectors for images I and I' are matched by using the sum of differences approach, but for each of the pairs individually:

$$\Delta G = \sum_{j=1}^{n} |(\alpha_j - \alpha'_j) + (\beta_j - \beta'_j)| \tag{2.8}$$

Using successive refinement of the histogram, Pass and Zabih also classify the pixels based on their inclusion in a centre zone - the centre 75% of pixels. Their results suggest that successive refinement of the CCV provides better results than an unrefined CCV. Refining the histogram not only makes finer distinctions between pixels but functions as a statistical filter for successive refinements.

Although CCVs are a useful extension to the histogram technique, they are, in essence, very similar, and suffer from the same problem of aliasing, and lack of spatial information.

2.2.1.5 Colour Correlograms

Correlograms(35) are probability functions which gives the probability distribution of colour from any pixel in the image, by distance from that pixel. So, although they don't explicitly encode spatial locations of objects, they encode the relationship, or correlation, between colours within an image. This provides a local spatial correlation of colours and the global distribution of this spatial correlation. The size of the correlogram is $O(m^2d)$, where m is the number of colours, and d the resolution of the correlogram. Given two colours, c_i and c_j in an $n \times n$ image I and a fixed distance attribute $d \in [n]$, a correlogram is defined as the probability of pixel p_1 of colour c_i being distance d from pixel p_2 of colour c_j:

$$\gamma^{(k)}_{c_i,c_j}(I) = \underset{p_1 \in I_{c_i}, p_2 \in I}{Pr}[p_2 \in I_{c_j} \mid |p_1 - p_2| = k] \tag{2.9}$$

The *autocorrelogram* captures the spatial correlation between identical colours only, and has the size $O(md)$. The autocorrelogram is defined as:

$$\alpha^{(k)}_c(I) = \gamma^{(k)}_{c,c}(I) \tag{2.10}$$

The distance between correlograms is given by the L_1-distance (the sum of absolute value of differences) which is statistically more robust to feature outliers.

$$|I - I'|_{\gamma_i} = \sum_{i,j \in [n], k \in [d]} |\gamma^{(k)}_{c_i,c_j}(I) - \gamma^{(k)}_{c_i,c_j}(I')| \tag{2.11}$$

The Colour Correlogram certainly improves on the basic histogram retrieval performance, but is still, fundamentally, a colour based algorithm, lacking any understanding of the image content.

2.2.2 Illuminant Invariance

There has been a significant amount of research dedicated to illumination-independent image matching. The colour of a pixel in a captured image is dependant on a number of factors: the spectral reflectance of the viewed object, and the spectral distrubution, intensity, and relative position of the illumation. There are two ways to overcome these effects: alter the image pixels, or extract features which are invarient.

The former uses a colour-constancy algorithm (assuming reflected light undergoes no spectral or intensity deformations) to estimate the prevailing lighting conditions of the scene and colour-correct the image (24; 73). However, these seem to be unable to deliver accurate enough estimates for good indexing. Of course, colour-constancy is often not a possible scenario in non-experimental situations where the assumption fails.

The second is to extract colour invariant features directly from an image and use these to index. For example, (81), creates a new chromaticity histogram with scaled colour channels. The features are matched using histogram intersection. Matas, et al. (45) store their models as colour patches, where the colour of the patches have been made invariant to illumination.

Of course, illuminant invariant searching can be counter-intuitive, as in certain circumstances we are often searching for particular lighting conditions; for example, night images, or sunsets. This is not only the downfall of using colour-invariants as a feature, but if the intended object can be any colour, for example, a car, then colour becomes more or less useless.

2.2.3 Matching with Colour Layout

Following Swain and Ballard's early study of colour indexing extensions to colour matching were developed such as including the location of the colour in the match, for example their Histogram Backprojection. In this section we discuss some colour layout matching algorithms for content based retrieval purposes.

2.2.3.1 Grid Matching

Simple histogram matching might be quick, but the lack of any spatial cues in the matching is a downfall. The most obvious way of achieving a spatial matching might be to divide the image into regions and match those separately. Dividing the image into a grid of equal sized area may seem trivial, but it is surprisingly effective at achieving spatial matching (21; 41).

Each grid element consists of a rectangular part of the original image, which itself can be treated as an image, and matched using any of the colour matching techniques explained in the section above.

The image can be matched partially by applying the match to only some of the grid elements. A spatial whole-image match can be achieved by matching each element with elements in the same position in the database image, which is more effective at finding similar images

than simple histogram matching. The smaller areas means searching for an area of a particular colour is more effective than a standard histogram match when a single element is matched against all elements in the database image.

Some problems arise due to the coarseness of the segmentation using the grid method, and a number of techniques have been employed to attempt to overcome this. Sethi et. al (67) use a set of grid elements all which overlap slightly which allows a certain amount of fuzziness to be incorporated into the spatial match. They use a nonlinear function based on the average hue and saturation values in each grid elements to perform the colour match.

An alternative to overcoming problems with histogram matching on grids, is to create features which overcome the problem, using a different segmentation algorithm.

2.2.3.2 Quad-tree Matching

The main shortcoming of using the simple grid method - namely the coarseness of the segmentation causing extraneous background noise to appear in the elements - can be overcome using a segmentation based on the quad-tree (71).

Using an image decomposition method based on the quad-tree (71) (see Section 2.2.6.1) it is possible to achieve spatial colour matching. Our experiments with using quad-trees to increase the accuracy of spatial colour matching is presented in Chapter 3.

The quad-tree method divides an image based on the content of the regions, and so each region is reasonably homogeneous, assuming the correct homogeneity criterion is used. This means that an uneven grid of regions is created, again, each of which can be taken as an image and matched using a colour histogram matching technique. Because the regions are approximately homogeneous, very low resolution histograms can be utilised, to increase response times. However, the matching becomes more interesting as the sizes of the corresponding regions may not be equal. The matching can be achieved using a leaf code identity for each quad-tree element and making some assumptions about the content of the nodes. This technique is described in detail in Chapter 3.

Leung and Ng (41) use a multiresolution quad-tree approach, although they do not merge elements with homogenous content. When matching the histograms, rather than directly matching, or scaling (normalising) the histograms, they use techniques called padding and reduction. Padding increases the size of the query image block to the size of the database image block, by adding to it a number of "desired" pixels - those pixels which will minimise the histogram Euclidean distance. Reduction is the opposite - the "least desired" pixels are removed, thereby keeping the histogram distance to a minimum. They match through the multiresolution tree using the Euclidean distance on the padded or reduced histograms and keep the finest scale images with the smallest distances. They keep proceeding until the distance value at a particular scale is too large to be a good match or the finest scale is reached.

2.2.3.3 Overlapping Grid

The idea in overlapping the grid elements is to avoid the problem of the arbitary grid segmentation which may cause the objects in the grid elements to be divided in such a way that the retrieval will fail. Overlapping the grid elements introduces a certain amount of fuzziness" to be incorporated into the spatial distribution of the colour.

The Color-WISE system (67) uses the mechanism where each grid element is feathered around the edges by 50% thereby giving elements which overlap the 8 surrounding elements. They use the histogram intersection method to provide matching in the elements.

The technique employed in the Artiste project (14), an environment for fine art retrieval, allows the users to find the original painting from a section removed from the painting - something single-scale techniques would not achieve. An advantage is that the the technique can be used with any low level feature matching technique, while also adding the advantages of mutliscale respresentations.

The image is rendered at a number of resolutions. At each resolution a "window" is placed on the image and a CCV, Histogram or other technique's feature is generated from the pixels within the area. The window is a fixed size for the application, and fixed across scales. The windowed area overlaps previous areas by half, to give more accurate feature location. Matching takes place over the whole set of features across the scales and the best match gives the location of the best sub-image by scale and location. In Figure 2.2 the image is converted into a multiscale-CCV which is stored in whatever database is being used. The image part which is to be queried is then converted into a CCV, and CCV matching takes place over the multiscale-CCV representation of the original image. The best matches can be seen at the finest scale at the positions where similar flowers appear. The coarser the scale becomes the worse the matches become.

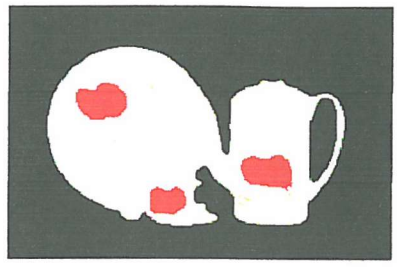
2.2.4 Shape

A major problem with colour is that it cannot distinguish between different objects of the same colour, obviously, and in many applications colour is simply not important. Another of the important aspects of an object is its shape. For example, a settee is very different from a table, but the colour of either object is not of consequence to the type of object it is. The shape of objects has also been used to advantage in waste recycling (72).

There are problems with shape as a feature. In comparison to colour features, the extraction of shape features from an image is a far more difficult task, and there are many techniques which attempt to get shape features from images in noisy circumstances, some with more success than others. Another problem is that objects, other than a sphere, look different from different angles. This means that if you search for an object based on one particular angle, the results are unlikely to include the object from other angles. This is a drawback of using only shape as an index into a database of objects, and why most systems will use other features

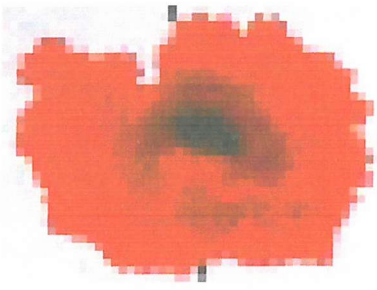
(a)

The database image which we are matching against.



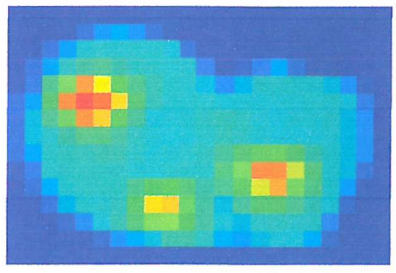
(b)

The coherent histogram colour representation of the image, after the colours in the image are represented by their coherent histogram bin colour.



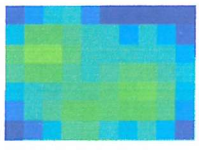
(c)

The image-part we are looking to locate in the database image.



(d)

Fine-scale pseudo-colour representation of the match score (red is a good match, blue is a bad match)



(e)

Medium-scale pseudo-colour image of the match score.

(f)

Low-scale pseudo-colour image of the match score.

Figure 2.2: Matching using the mutiscale overlapping grid mechanism with CCVs as the matching algorithm. Figures 2.2(d), 2.2(e), 2.2(f) show pseudo-colour representations of the match space at a fine, medium, and coarse resolution. Note the best matches (the red colours) are at the points where the flowers occur in the original image.

along with shape. Another way around this is to use a 'semantic layer' and associate different representations of an object with a concept representing an object. MAVIS-2 (19; 18) contains such a device called the Multimedia Thesaurus (74; 75).

2.2.4.1 Segmenting Images To Extract Shape

Extracting shapes from images can be relatively easy for images which contain an object on a single colour background. For 'real-world' images, segmenting objects from the background can be far more difficult.

For objects on a single colour background, segmentation is a case of using an edge detection algorithm and a boundary tracing algorithm to convert the pixels in the edge image into a useful format. We used this technique in a system used for making measurements on ancient flint tools which were going to be digitised in a controlled environment against a dark backdrop. Binary image masks are often used in cases where images are overlaid based on the shape of object, such as in video post-production. The mask can be automatically generated from segmenting the objects from the single colour backdrop. This is the basis of chroma-keying (or blue-screening). Traditionally chroma-keying has been done on blue or green background because the human skin texture does not contain much blue or green - it is mainly red tones. Segmenting in this situation is performed, on the whole, by thresholding. With an object on a black or white background, the thresholding can take place on the grey-level value. For chroma-keying type applications thresholding takes place on the blue or green channel of the RGB signal. Chroma-keying techniques are based on thresholding but have been developed far beyond this.

Segmenting images in noisy "real world" images can be more difficult. Edge detection and boundary tracing is usually only useful on images which have a single colour, high contrast background, like the thresholding example. The Canny edge detector will always give a closed boundary and is useful if a boundary tracing algorithm is to follow the detection.

Region based segmentation has the ability to work in noisy images, because it does not rely on the object explicitly. The image is decomposed until the regions are all homogenous, for example with the split and merge algorithm which repeatedly splits the image into a quadrants until the region inside a quadrant is homogeneous to some homogeneity criterion. The problem is that in richly textured images, or images with alot of aliasing, the homogeneity criterion is very important to avoid the number of regions that are generated becoming large, and the regions becoming pixel-sized.

Hybrid techniques like seeded region growing and watershed algorithm combine region and boundary information but have problems with blurred edges. Finding starting points for the algorithms also becomes a difficult task to automate successfully.

2.2.4.2 Representing and Matching Shapes

Deformable models are used to represent either a class of objects of differing shape, or objects which change shape. Active Contour Models (or Snakes) are deformable models using energy minimisation splines that hone their shape to the contour of an object from an initial estimate of the shape of the object.

The energy minimisation function is a combination of internal and external forces which deform the position of a point on the estimated contour:

$$E_{snake} = \int_0^1 \left(E_{internal}v(s) + E_{image}v(s) + E_{constraint}v(s)\right).ds \tag{2.12}$$

where $E_{internal}$ is the spline energy caused by stretching and bending, E_{image} is the measure of attraction of image features such as contours, $E_{constraint}$ is the external constraints generated by higher level functions, such as prior knowledge, or user-based energies (for example to avoid nearby objects affecting the minimisation procedure). $v(s)$ is the parametric representation of the contour: $v(s) = (x(s), y(s))$, where $x(s)$ and $y(s)$ is the coordinates of the position at distance s along the curve (normalised so the length of the curve is 1, i.e. $0 \leq s \leq 1$).

ACMs can be made multiscale by using a pyramid of images - where the ACM algorithm is run over a range of resolutions from low to high resolution giving a set of ACMs. Of course, the ACM is only a method for extracting shape from an image, a solid representation of the shape for matching needs to be generated.

The simplest measurements which can be taken on a shape are length, and breadth, and an aspect ratio can be calculated. The area of the shape and the length of the perimeter are also easily calculated. Compactness is defined as $\frac{perimeter^2}{area}$. Elongation is a measurements of the ratio of the length of the longest chord of the shape to the longest chord perpendicular to it. The Euler number is defined as $E = C - H$ where C is the number of connected components of a shape, and H is the number of holes. These are all very simple descriptiors but can all be useful in separating simple shapes. Matching is a simple subtraction calculation to give a distance.

Moments (76) have long been the cornerstone of shape matching due to their ease of calculation - assuming one has a binary image to work from! Calculating moments from other shape representations can be more difficult. Moments require the knowledge of which pixels in an image lie within the shape. The method was stumbled upon by a student called Sigfried writing a thesis on character recognition after taking the advice of a lateral thinking methodology book and randomly browsing his library for ideas(76).

The two dimensional moment is defined generally as:

$$M_{jk} = \sum_{l=1}^{n}\sum_{m=1}^{n}(x_l)^j(y_m)^k f(x_l, y_m) \tag{2.13}$$

For simplicity, taking the pixels to be of binary value (black = 0, white = 1), the definition

of moments becomes:

$$M_{jk} = \sum_{A} x^j y^k \tag{2.14}$$

Moments are then combined to provide suitable shape features. M_{00} is the area of the shape, in pixels. The centre of gravity is given by:

$$\bar{x}_1 = \frac{M_{10}}{M_{00}} \tag{2.15}$$

$$\bar{y}_1 = \frac{M_{01}}{M_{00}} \tag{2.16}$$

Furthermore, the skew of the shape is given by:

$$\bar{x}_2 = \frac{M_{20}}{M_{00}} \tag{2.17}$$

$$\bar{y}_2 = \frac{M_{02}}{M_{00}} \tag{2.18}$$

and kurtosis given by the third order descriptor.

Equation 2.19 can be used to make moments invariant to translation and scale.

$$M_{jk} = \sum_{A} \left(\frac{x-\bar{x}}{\sigma_x}\right)^j \left(\frac{y-\bar{y}}{\sigma_y}\right)^k \tag{2.19}$$

Hu defined a set of 7 invariant moments(34) which are sufficient to successfully match shapes, and they are all translation and rotation invariant. They are based on the central moments, which are given by:

$$\mu_{pq} = \frac{1}{NM} \sum_{x=1}^{N} \sum_{y=1}^{M} f(x,y)(x-\bar{x})^p (y-\bar{y})^q \tag{2.20}$$

The moments are defined as such:

$$M_1 = (\mu_{20} + \mu_{02}) \tag{2.21}$$

$$M_2 = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2 \tag{2.22}$$

$$M_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2 \tag{2.23}$$

$$M_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2 \tag{2.24}$$

$$M_5 = (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] + (3\mu_{21} - \mu_{03})(\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2] \tag{2.25}$$

$$M_6 = (\mu_{20} - \mu_{02})[(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03}^2] + 4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03}) \tag{2.26}$$

$$M_7 = (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] - (\mu_{30} - 3\mu_{12})(\mu_{12} + \mu_{03})[(\mu_{30} + \mu_{12})^2 - (\mu_{12} + \mu_{03})^2] \tag{2.27}$$

These can be made scale invariant by normalising them to the size of the blob or edge bounday, using the radius of gyration of a planar pattern: $r = \sqrt{\mu_{02} + \mu_{20}}$. The first scale

invariant moment requires a measurement of camera to observed object, B, and can be ignored for simple shape matching. The moments that are rotation, translation and scale invariant (RST invariant) are given by:

$$M'_1 = rB \tag{2.28}$$
$$M'_2 = \frac{M_2}{r^4} \tag{2.29}$$
$$M'_3 = \frac{M_3}{r^6} \tag{2.30}$$
$$M'_4 = \frac{M_4}{r^6} \tag{2.31}$$
$$M'_5 = \frac{M_5}{r^{12}} \tag{2.32}$$
$$M'_6 = \frac{M_6}{r^8} \tag{2.33}$$
$$M'_7 = \frac{M_7}{r^{12}} \tag{2.34}$$
$$\tag{2.35}$$

Comparing the moments using only Euclidean measures will allow shape differentiation, but the result will be scale dependant. For these features to be comparable, they can be normalised by the mean and standard deviation of the ith moment.

$$m_i \Longrightarrow \frac{m_i - \bar{m}_i}{\sigma_i} \tag{2.36}$$

This is the squared pattern distance measure and it is based on the relative distances of each moment from the mean moment from the set of model shapes. To calculate the distance requires the mean and variance of a dataset to be calculated. This is only possible in a dataset where there is a limited number of possible classifications (for example, in optical character recognition where there are 26 letters, and a known number of extra characters).

Moments of perimeter are computed in a similar way to moments of area but using only pixels which lie on the boundary. Computing the moments of a differentiated (edge detected) image, where pixels represent edge magnitude, we can obtain the moments of the perimeter. They are calculated as:

$$m_{pq} = \int x^p y^q \left[g'(x,y)\right] dxdy \tag{2.37}$$

The more moments that are generated for a shape, the more accurate the representation becomes. However, its very hard to reconstruct and manipulate the shape in this format. This isn't a problem for matching, of course, but the difficulty in understanding what each of the desciptors actually stands for has repercussions on the usability of such a method. Moments are also fairly computationally intensive, and are sensitive to noise.

Fourier coefficients (61; 77; 71; 3; 4) have some similar properties to moments of perimeter, representing the boundary information of a shape rather than the region. They encode the harmonics of a shape-boundary in the same way the Fourier transform encodes the harmonics

of a signal. This means that from the shape descriptors the original shape can be rebuilt to varying levels of resolution. Both moments and Fourier descriptors are lossless representations if an infinite number of descriptors are taken.

There are two methods to calculate the Fourier descriptors. The first, closed-form Fourier descriptors, is to "unroll" a shape into a one-dimensional signal domain, and perform a normal Fourier transform on the resulting signal. The unrolling is achieved by an expansion of the periphery radius as a function of the angle about the centre of gravity of the shape. This technique has been used for archeological artifact recognition in (27), although it can only be used on convex shapes.

The second technique for calculating the Fourier descriptors extracts elliptic Fourier descriptors from a shape boundary(71). Travelling along a curve in a complex plane at a constant speed, a complex function $z(t)$ is obtained, where t is a time variable. If the speed is chosen such that the perimeter of the shape is 2π a periodic complex function is obtained. Given the length of the curve, L, and the curve distance, s, the Fourier descriptors, T_n, are then given by:

$$T_n = \frac{1}{L}\int_0^L z(s)e^{-i(2\pi/L)ns}ds \tag{2.38}$$

The first 15 or so descriptors are enough to describe a shape, such that a rebuilt shape with the first 15 or so harmonics is barely distinguishable from the original. The higher harmonics are important for shapes which are very similar, or have sharp edges (such as a square), and they therefore cannot be ignored. Indeed, to represent a corner, within a continuous signal, with Fourier descriptors requires an infinite number of them.

The resulting descriptors of either Fourier method can be made invariant to rotation, scale, and translation. Ignoring the phase of the descriptors makes them rotation independent, normalising to the size of the second descriptor makes them scale independent, and ignoring the first descriptor makes them translation independent. However, it is important to note that ignoring the phase can result in objects which look significantly different being given a close match score. The phase of the harmonics are more useful if they are relative to each other - thereby preserving the shape, but not the original rotation of the shape.

In our MAVIS-2 shape signature module we use a quadratic histogram matching technique, as described in Section 2.2.1.3 for colour, to match the Fourier descriptors, rather than the L_1- or L_2- distance between equivalent bins. This reduces the effect of arbitary assignment to bins due to noise in the original signal; whether that is a colour signal, or a shape signal. Because we ignore phase to make the descriptors rotation invariant, we are left with a one-dimensional feature space. It is possible to use the quadratic match technique which takes into account frequencies from nearby bins, thereby making the match more fuzzy. This is described in more detail in Chapter 3.

The Fourier shape matching is in effect a multi-resolution function, and the histogram matching matches between and within different levels of scale of the shape. Scale is a very

important variable in matching functions. Many techniques are developed to avoid the issue of scale, and make them invariant to it. However, the relativity of scale can be a useful property.

Matching shapes by using scale-space techniques is a relatively recent advance. A description of scale-space can be found in Section 2.2.5.

Abbasi and Mokhtarian (1; 2; 53) have used the curvature scale space (CSS) image to achieve shape matching. They generate a CSS image (u,σ) by plotting against scale σ, which is the size of the Gaussian smoothing kernel used, the position of the maxima in the normalised boundary u. They match the CSS images by finding the maximum point of a contour in the query CSS image and shift-rotating the it along the u axis so that it matches the tallest contour in the model CSS image. Then they find the nearest maximum in the model CSS image most similar to the second maxmium in the query CSS image, and so on through all the maxima of the query CSS image. The match distance is the summation of the Euclidean distances between all the corresponding maxima. If there are more maxima in the model image then for each unmatched contour the scale σ is added to the score. Before matching they filter out highly dissimilar shapes using eccentricity and circularity features.

In (1) they go on to explain how the CSS mutates with regard to affine deformations in the original shape, and how these mutations don't give a large errors in the matching results - although the use of eccentricity as a pre-filter becomes impossible as, for example, skew alters the eccentricity of the shape.

2.2.5 Scale Space

Scale space is a relatively new concept in signal decomposition. The main idea is that many elements of varying scales make up a single signal and as such they can be decomposed into their component parts - very similar to what the fourier transform does for a signal, but based around morphological constructs and characterised by the extrema.

Andrew Witkin in his early paper 'Scale Space Filtering' (84; 85) explained the principle of signal decomposition using scale-space. He uses a Gaussian kernel convolved with varying standard deviations (σ) over a one-dimensional signal because he believed the Gaussian kernel preserved scale-space causality. As σ is decreased continuously, beginning at a coarse scale, two distinct effects on the extrema are noticed. Firstly, the extrema that are already present are constantly in motion along the signal axis. This is due to the Gaussian blurring effect, and is a problem when using this method for region segmentation in images, as we will explain later. Secondly, at finer scales, new extrema will appear. This second observation is the main principle behind scale-space causality: that "new extrema must not be created in the scale-space representation when the scale parameter is increased." Koenderink, in (39), derived the two-dimensional scale-space based on the assumptions of causality, homogeneity and isotropy, and shows the diffusion equation provides the functionality. The idea in 2D scale-space is that each greyscale pixel at a coarse scale can be traced back to its origins in a finer scale. Diffusion,

being isotropic, means extrema in the image do not wander, but edges in the image do. More recently Lifshitz and Pizer (62) found that the diffusion equation did not, after all, adhere to the scale-space causality principle - it was not "well-behaved" - because, in certain instances, new extrema *were* generated. An example situation would be the image which contains two large extrema connected by a small isthmus. The Gaussian kernel will reduce the isthmus more than the two extrema, and causes a new extrema (minima) to come into existence in between the two.

Another difficulty with using the Gaussian kernel is that the contours (edges) of the regions in the image are distorted by the blurring. Based on anisotropic diffusion, Perona and Malik formulated a scale-space method (59; 82) which encourages intra-regional blurring, and attenuates inter-region blurring, to try to preserve the edges between regions. However, an image that has been diffused to a particular scale will contain regions of many different scales. Also, this approach still uses blurring as a method for removing extrema, and as it has already been seen that blurring can introduce extrema, thereby failing the scale space causality axiom.

A new technique built on the idea of morphology was developed by Bangham et.al. (8; 9; 11; 10; 33). The technique uses connected sets of pixels as regions, and therefore does not affect the contours of the image as the decomposition takes place. Using this technique for content-based retrieval has been introduced in (7) and we explain how we use the technique for content based retrieval with graph matching in Chapter 4.

The reader is pointed to the comparison given in (32) for examples of the differences between these techniques.

2.2.6 Trees for representing images

Trees can be useful for representing images because they explicitly encode the topology based on the decomposition used. Once a tree has been constructed, manipulation of the image is achieved by performing edits to the trees. In this section a description of some of the common types of tree representations will be given.

2.2.6.1 Quad-trees and T-pyramids

T-pyramids and quad-trees are explained in (71). Quad-trees and T-pyramids both decompose an image by splitting the image into quadrants and details of each quadrant is stored. Unlike T-Pyramids, Quad-trees do not store data at every node, only at the leaves. If the four children of a node contain the same value, then there is no need to store them. Stopping construction at homogenous regions makes the representation less expensive, particularly for images such as line art, or logos. Quad-trees may not be balanced whereas a T-pyramid is always balanced. Unfortunately, under very small changes in the image, the quad-tree can change drastically. Also, it is not invariant to rotation, scale, and shifting. The quad-tree can be made RST-invariant by using a normalised shape-of-quad-tree respresentation, in which quad-trees are only created

for the individual objects in the scene. One is then faced with the problem of finding the objects in a scene, which as we have already described is difficult.

2.2.6.2 Containment Trees

Containment trees represent which regions in an image are contained within other regions. This is a crude type of topology tree. The nodes of the tree would contain the contour information of the regions. Such a simple tree method would lead to the same representation being given for many different images. The topology of the tree could be used for preliminary search and the contours at the nodes used for a more precise match, as was achieved in the pictorial retrieval system that used containment trees (38).

2.2.6.3 Shock Trees

Shock trees(68) are a tree representation of a shape based on shock measurements of the boundary. The shock points are singularities in the closed curves, measured by variation of the radius against the medial axis. First order shocks occur at protrusions in the shape, while second order shocks occur at necks in the shape. In either direction from a second order shock are two first order shocks. Third order shocks occur where the sides are parallel either side of the medial axis. A fourth order shock occurs when the medial axis is a point – in the centre of a circle.

2.2.6.4 Watershed and Critical Lake Trees

Critical lake trees are generated by the watershed algorithm. Starting at the minima, the signal is "flooded" to make "lakes". As two lakes merge a node is generated in the critical lake tree with the two minima from which the lakes were generated being children. This visualisation of the algorithm can be understood when the signal is viewed as a topographic relief with the height as the intensity. Nodes can contain information about the pixels as well as depth, area, and volume of lake properties.

2.2.6.5 Min- and Max-trees

Min and Max-trees (65) are connected set operators; the nodes in the tree all contain connected sets. A threshold is used to convert the input signal into a binary output, where 0 is below the threshold, and 1 is above. Starting from a minima, the threshold is increased, and for every pixel, if the binary output of the threshold is 0 the pixel is left, and if the binary output of the threshold is 1 it is moved to a child node. The threshold is then increased, and the process re-applied. In the re-application at some nodes, all pixels may be moved from one node to a new node, and the old node must be removed. The tree that is created is oriented towards the maxima of the image - the maxima are at the leaves - hence the name max-tree. Min-trees are created by starting at the maxima and lowering the threshold so that the tree is oriented towards

the minima of the image. The leaves of the tree will contain information about the connected sets, and the leaves may not necessarily contain connected flat zones. Connected sets can be removed from the images without altering the other regions in the image by moving the data in one node into its parent.

2.2.6.6 Binary Partition Trees

A binary partition tree is a structured representation of the regions that can be obtained from an initial parition of an image. With the correct segmentation algorithm, they can deliver multi-scale representations of the image which are easy to filter and manipulate. Salembier and Garrido in (65) use max-trees to decompose an image into regions then they merge those regions based on a colour homogeneity criterion into a binary tree, until there is only one region left in the image. In their paper, they explain how the trees can be used to aid simple object recognition (e.g. circle detection), a simple coding technique and a filtering algorithm.

2.2.7 Graph Matching for Image Retrieval

Now that graphs and trees are becoming more important in image analysis, due to the more complex and higher level decomposition algorithms that are appearing, visual information retrieval becomes a matter of graph or tree matching. Graph matching is a very powerful technique but its main drawback stems from the fact that it belongs to the class of NP-complete problems. In the worst case the problem of matching one graph to another grows exponentially, and out of computational tractability, for subgraph isomorphisms.

The problem is to match an object represented by a graph to a model graph representing some image or another object. An exact match of graphs is called a graph isomorphism. However, this is often too strict a matching technique to be used in image matching applications, due to noise and variances in the image structure.

A graph isomorphism occurs if, given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ there is a one-to-one and onto mapping, f, between V_1 and V_2 such that an edge E_1 connecting any pair of nodes, $v, v' \in V_1$, there is an edge E_2 connecting $f(v)$ and $f(v')$. Also, for the match to be an isomorphism, the constraint that if $f(v)$ and $f(v')$ are connected in G_2 they must also be connected in G_1.

It is, in fact, not known whether graph isomorphism is NP-complete as there is no proof that it is not, despite there being no algorithm other than NP-complete algorithms to search for graph isomorphism. However, subgraph isomorphism and double subgraph isomorphism (all subgraph isomorphisms of G_1 and G_2) are known to be NP-complete.

Lenaghan et al(40) use association graphs to reduce the problem domain when matching graphs generated from tracing handwriting outlines. Association graphs contain nodes which contain all possible pairings of nodes from the graphs to match. This is still too computationally expensive to be of use for image matching.

Park et al. (57) used a modifiction of the region adjacency graph to create a modified colour adjacency graph (MCAG) which represents the colours in an image by their adjacency. They create a large 2 dimensional matrix representing the fully connected graph with the nodes of the graph representing histogram bins. To avoid having to do graph matching they effectively turn it into a histogram matching using histogram intersection but on the graph representation.

A generalised inexact graph matching technique that, with a little modification, could be used for image matching, is the A* algorithm. The A* algorithm developed by Nillson (56; 48) is a simple tree search graph isomorphism algorithm. From the set of vertices in the input graph the first level of the search tree is generated by compiling a list of pairs of nodes from the model graph with the same labels. Then, the search tree node with the least cost (in the exact case nodes are taken in a random order) is expanded. Again, all the pairs from the remaining vertices with the same vertex labels are compiled into a list and the lowest cost expanded. This continues until the lowest cost isomorphism is found.

Although the above algorithms may work well to increase the efficiency of finding isomorphisms of graphs, their main drawback for an information retrieval role is their lack of indexing: they work with one graph against another. In the literature the prime technique for graph and subgraph isomorphism detection between one graph and a set of graphs was developed by Messmer and Bunke (47; 46; 50; 49; 48; 51) and uses a network of shared subgraphs to achieve less-than intractible (in the best case), while also optimal, results. We use the Network Algorithm for our work, and we have extended the idea to include feature matching and feature score propogation, which can be seen in Chapter 4. More recently Bunke has developed the technique to use decision trees (52; 13).

2.3 Summary

This chapter explained some of the techniques available to match images for content based retrieval that are related to the novel work contained herein. Of course, it is impossible to cover every possible content based retrieval system and technique here, due to the vast amount of research that has taken place in this field. The features which all of these, and most other, algorithms return can be used as label representations in the topology tree which is built from the scale-space decomposition of an image. Although some of the techniques would be less useful than others due to the way in which the image is decomposed — for example, the colour matching at the nodes can be less complex than many of the algorithms as the decomposition is based on the luminance of the colour signal, and therefore the regions at the nodes are approximately homogenous in colour. To begin with, we plan on using the fourier shape, and a simple colour distance measure at the nodes of the topology tree, and when the idea has been evaluated move onto different algorithms as necessary.

Chapter 3

Preliminary Work on Feature Extraction for Image Matching

The content based retrieval research area is growing rapidly and there are many techniques which could be combined in an attempt to achieve the goal of an object recognition system. In investigating the state of the art of content based retrieval we performed some tests and developed some extentions to implementations of some other content based retrieval techniques. This allowed us to identify what their abilities are for matching images and sub-images with the aim of facilitating object recognition. This chapter describes the matching techniques we investigated and the improvements we made. We also indicate why they are, in themselves, not accurate enough for object recognition. In the next section we describe the work we undertook in colour matching to make them more useful for image searching. In the section following, we discuss some work undertaken on the application of shape matching techniques to a real world problem, in this case concerned with archaeological artefacts.

3.1 Colour Matching

Colour is certainly the most commonly used method of image retrieval by content due to its simplicity and effectiveness. To test how effective it is, we began by investigating colour histogram matching.

A colour histogram stores the frequency of colours within an image. Usually the number of colours in the image is reduced, because a natural photographic image may have many thousands, even millions of coloured pixels. Matching is faster the smaller the number of histogram bins - there are less calculations to do - but due to the smaller number of colours, the

less discriminating the histograms are, and the less accuracy the results will have. The number of bins a histogram contains is a compromise between result accuracy and speed of matching.

3.1.1 Manhatten and Euclidean Histogram Matching

Simple colour histogram matching consists of taking the value of the frequency in bin j of the query histogram, H_Q, and comparing it with the frequency in bin j of the match histogram, H_M. The most common method for doing this calculation is the sum of the absolute differences, L_1- or Manhatten (city block) distance, defined as:

$$\|H_Q - H_M\| = \sum_{j=1}^{n} \|H_Q[j] - H_M[j]\| \tag{3.1}$$

or the square root of the sum of the squares of the differences, L_2- or Euclidean distance, defined as:

$$\|H_Q - H_M\| = \sqrt{\sum_{j=1}^{n} (H_Q[j] - H_M[j])^2} \tag{3.2}$$

where n is the number of bins in the histograms H_Q and H_M. The most similar histogram H_M to H_Q would be the histogram minimising $\|H_Q - H_M\|$.

This type of histogram matching is useful for images which contain large proportions of solid colours, or images which are characterised by large areas of certain colours (e.g. sunsets). In Section 3.1.5 we discuss the performance of the Manhatten and Euclidean histogram matching techniques. The results were generally poor for real-world images and we considered fuzzy histogram matching.

3.1.2 Quadratic Histogram Matching

To investigate if this result is a consequence of the L_1- or L_2-distance measures being affected by the equality criterion which quantises the histogram into bins, we can use the quadratic histogram match that will allow for fuzzier colour matching. The way it achieves this is by weighting all the bins against each other dependant on the distance of their representative colours in colour space.

A single bin of a histogram will subtend a volume in colour space. A single colour which represents that bin could be taken arbitrarily in the centre of the volume, or at any corner of the volume. Any pixel which falls into that volume will become equated to the single colour which now represents that bin. The distance between the representative colours, in the chosen colour space, will give the distance between the two clusters of pixels. This distance can affect a weighting on the bin matching process.

A matrix, A, containing the weightings between every pair of bins is generated which will have dimensions $m \times n$, where m and n are the number of bins in the histograms H_Q and H_M, which must be equal. The weightings are given by the Euclidean distance between the points

in the colour space which represent the colour bins, c_i and c_j. In an RGB colour space this would be:

$$d_{c_i c_j} = \sqrt{((r_i - r_j)^2 + (g_i - g_j)^2 + (b_i - b_j)^2)} \tag{3.3}$$

where $c_1 = (r_1, g_1, b_1)$, $c_2 = (r_2, g_2, b_2)$, and so on, are the colours representing the two histogram bins.

The distances for each pair of histograms (H_Q, H_M) are then compiled into a weighting matrix, where $d_{\max}$ is the largest distance between any two bins:

$$a_{ij} = 1 - \frac{d_{ij}}{d_{\max}} \tag{3.4}$$

$$A_{H_Q H_M} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \tag{3.5}$$

The histograms H_Q and H_M are then represented as one dimensional matrices and subtracted to give the difference matrix Z:

$$z_i = H_Q[i] - H_M[i] \tag{3.6}$$

$$Z_{H_Q H_M} = \begin{pmatrix} z_1 & z_2 & \cdots & z_n \end{pmatrix} \tag{3.7}$$

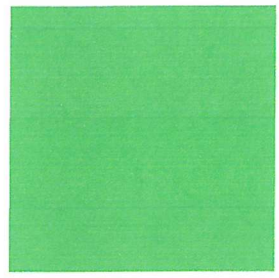
The similarity between the two histograms is then calculated using the quadratic formula:

$$\|Z\| = Z^{\mathrm{T}} A Z \tag{3.8}$$

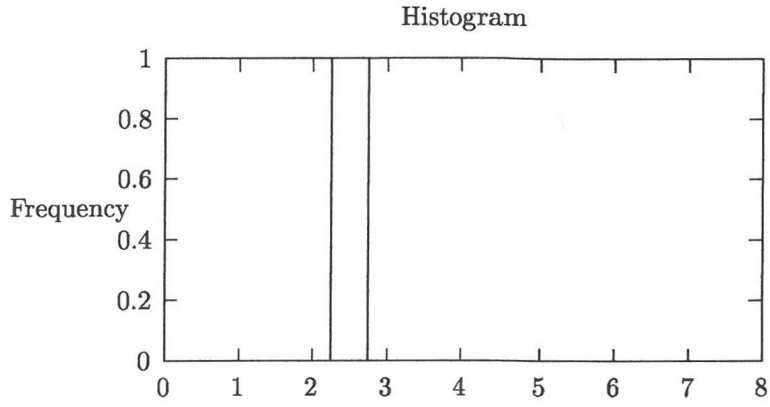
Matching in this way is slower than the L_1- or L_2-distance because for every histogram match there are $n \times n$ calculations, compared to n calculations, that is, the complexity of the operation is $O(n^2)$. In (30) Hafner et al. describe the use of a low resolution filtering mechanism to reduce the number of expensive calculations that need to be done.

A way to decrease the matching time is to effectively pre-calculate the weightings and encode them into the histogram by applying a filter to new entries into a histogram bin. When a new pixel is inserted into a bin the distance between the surrounding bins is weighted by the new pixel and added to the surrounding histogram bins. Performing the intensive calculations during histogram generation then allows simple Manhatten or Euclidian distance to be used during the match. For this test, speed comparisons are not as important as the quality of the matches that are being measured. However, should the histogram technique be required for matching in an application, the response time is important, and this technique would certainly have to be used should the L_2 distance match prove unreliable.

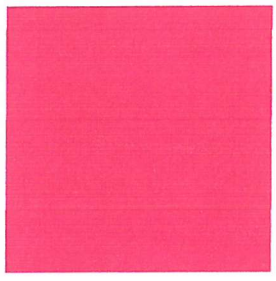
Matching using fuzzy techniques can allow matching of images which should match well but do not due to the equality criterion of the histogram bins. Figure 3.1 shows how the histogram technique places colours which are very similar into different bins of the histogram. The



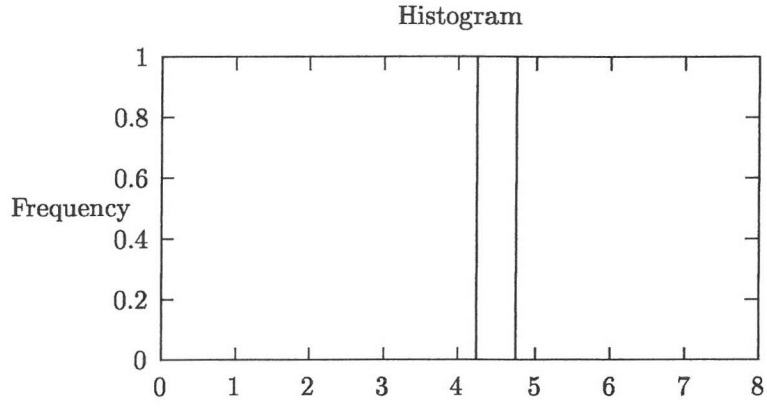
(a) Green Image



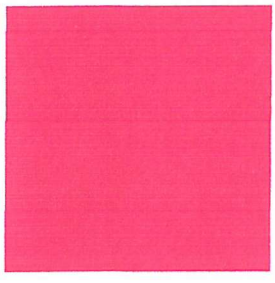
(b) Green Histogram



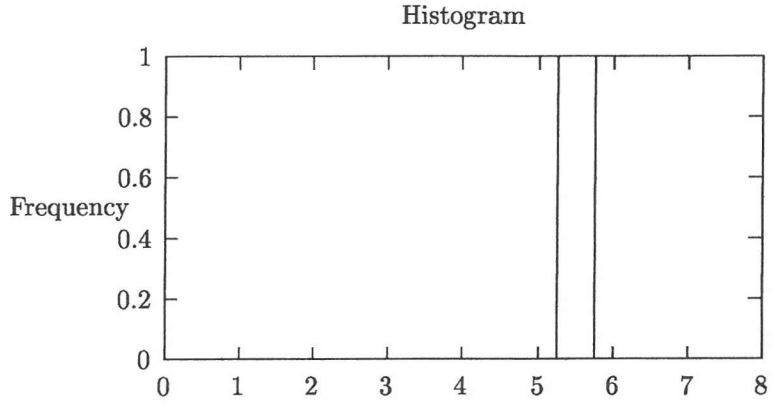
(c) Magenta Image



(d) Magenta Histogram



(e) Magenta Image



(f) Magenta Histogram

Figure 3.1: Single-colour images and their histograms, showing how very similar colours can be placed in adjacent histogram bins due to the equality criterion. This can be partially overcome with the quadratic histogram match.

colour in Figure 3.1b is visually identical to that of 3.1c but has been placed into a different bin of the histogram because of a very small difference in the colour. Matching these images using the L_1-distance, for example, which only matches equivalent bins, the green image in Figure 3.1a matches equally well with the magenta in Figure 3.1b to the Figure in 3.1c with a normalised distance of 2. The quadratic histogram match effectively blurs the boundaries of the histogram bins, allowing the two magenta images to match better (because the bins are near to each other in colour space). With the quadratic histogram match the magenta images have a match score of 1.154 while the magenta and green image still match with a score of 2.

Although you will always get this bin assignment problem due to the simple histogram model, it can be reduced by using this quadratic matching technique.

So despite the quadratic histogram matching being obviously useful for reducing the effects of the arbitrary bin assignment during histogram creation, the recall and precision measures while matching our dataset (see Section 3.1.5) were not much better than using the quicker L_1- or L_2-distance measures. By making the match fuzzier, the range of potential matches increases and therefore the amount of possible false hits also increase.

It seems part of the problem with the histogram matching techniques we used above is that they take a match based on the whole colour content of the image. The images of the cars we have been using have a definite distribution of colour. The background of the images changes little, and generally consists of grass, tarmac and a yellow sign (although obviously specific to this dataset). This implies that the focus of the image, the car, is in the centre region of the image. This is in contrast to the firework images that are randomly distributed, and are retrieved better.

We can impart a restriction on the matching by applying a spatial dimension to the matching process and specifically matching the colour layout of the images, and not just the general distribution. The most pragmatic way to achieve this is with an overlaid grid.

3.1.3 Spatial Colour Matching using a Grid

Whole-image histogram matching might be quick, but the lack of any spatial cues in the matching is a downfall. A naive way to introduce a spatial dimension might be to divide the image into equally sized regions and match those separately. Dividing the image into a regular grid may seem trivial, but it is surprisingly effective at achieving spatial matching (21; 41).

Each grid element consists of a rectangular part of the original image, which itself can be treated as an image, and matched using any of the colour histogram techniques explained above.

For example, dividing an image, I, into an 8×8 grid gives $I_1 \ldots I_{64}$: a number of sub-images which can be considered as single images in their own right. The image which these parts will be matched against, M, is also divided into an 8×8 grid. It is worth mentioning that if M is of a different aspect ratio or of different dimensions, the aspect ratio or dimensions of the

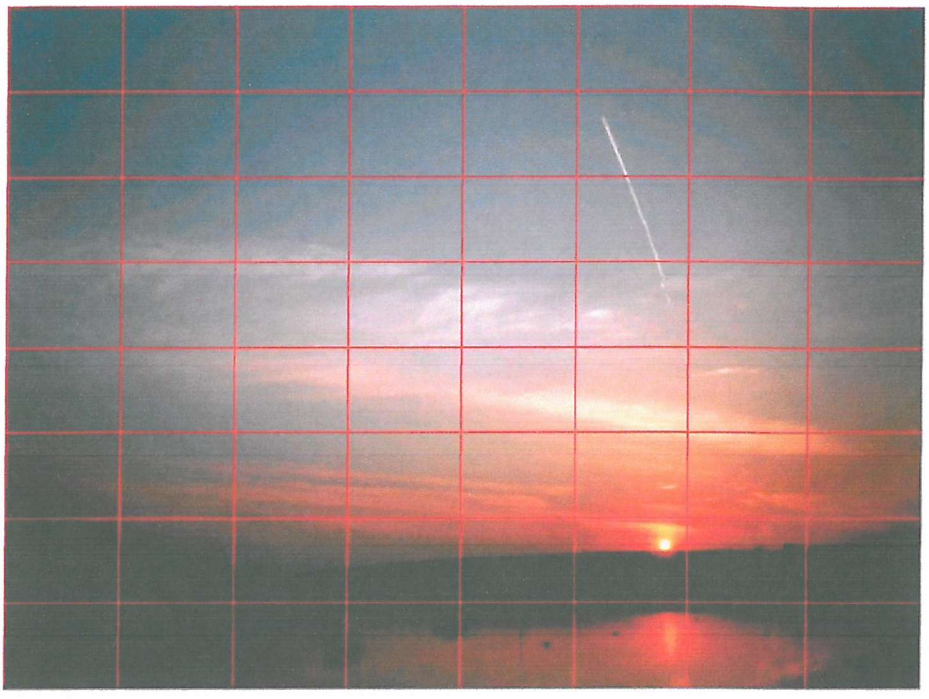


Figure 3.2: An image divided into an 8×8 grid

elements $M_{1...64}$ will not be equal to those of $I_{1...64}$. To search for features which are similarly spaced, dividing the image into an equal number of grid elements is preferable to dividing the image based on equal sized grid elements. Dividing the images based on equal sized elements gives the problem of the scale dependant location feature. Equally divisioned images allow the size and aspect ratio to be irrelevant in the match. This is, generally, the most desirable type of feature.

Figure 3.2 shows an example image segmented into an 8×8 grid. Table 3.1 shows the histogram data for the last four elements in the grid using an HSV histogram of dimensions $3 \times 3 \times 2$. The summation of each histogram (each line in the table) gives the number of pixels in each grid element (17,649 in this example). It can be seen that the data of the third-from-last element's histogram how the saturated red bin contains many pixels, representing the bright reflection of the sunset.

The score, s, of spatial matching a whole image would be achieved by matching each element in I with the equivalent element in M:

$$s = \sum_{i=1}^{n} \|H_{I_i} - H_{M_i}\| \tag{3.9}$$

A database of r pre-segmented images can be compared returning the sorted results, $s_{1...r}$. s_1 gives the most similar image to image I using the spatial match where each grid element is taken into account. It may be desirable to ensure the final distance score is within the range of

Red						Green						Blue					
S_l		S_m		S_h		...											
D	L	D	L	D	L	...											
2311	0	10281	0	2582	0	62	0	24	0	5	0	281	0	1907	0	196	0
216	0	2173	0	5401	7995	11	0	10	0	13	0	50	0	696	0	1084	0
192	0	3967	0	8278	3751	2	0	8	0	0	0	151	0	666	0	634	0
1740	0	6798	0	1002	0	16	0	15	0	6	0	2033	0	5106	0	933	0

Table 3.1: Histogram data for the last 4 elements of the image in Figure 3.2.

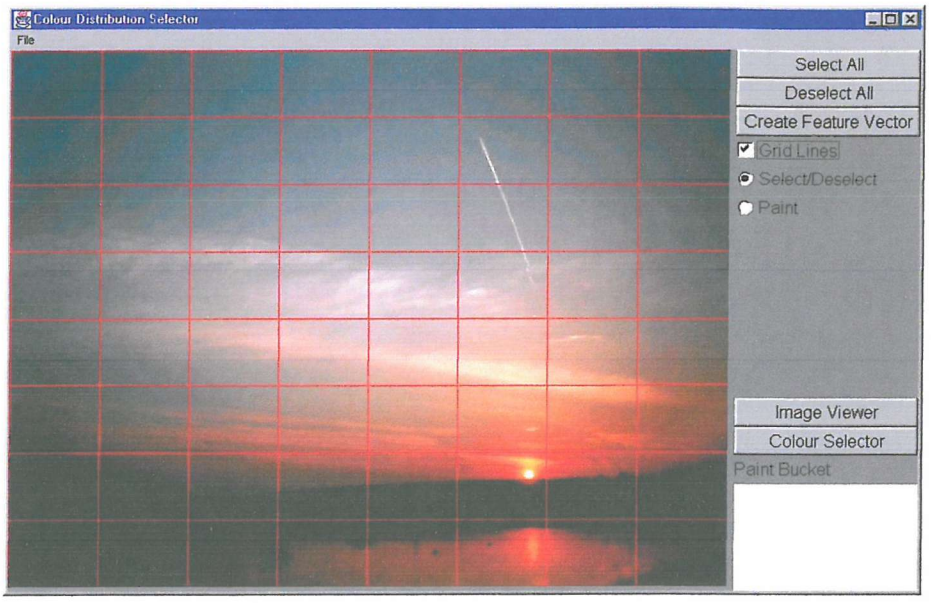


Figure 3.3: The interface allows selection of grid elements or painting into grid elements.

the individual distance scores by averaging the results:

$$s = \frac{1}{n}\sum_{i=1}^{n} \|H_{I_i} - H_{M_i}\| \tag{3.10}$$

A weighting may be applied to each of the match scores for certain situations where the application needs to assume something about the images. For example, while searching for portrait images (images of faces) the assumption might be that the face will be in the centre and the surrounding material is less relavant. A weighting for the grid elements can be applied:

$$s = \frac{1}{k}\sum_{i=1}^{n} \gamma_i \|H_{I_i} - H_{M_i}\| \tag{3.11}$$

where γ_i is the weighting for element i, and k is the sum of all the weightings ($\sum_{i=0}^{n} \gamma_i$).

If binary weightings of only 1 or 0 were considered, the elements could be matched selectively – only match the grid elements with a weighting of 1 and ignore those elements with

a weighting of 0. This will hasten the matching process by decreasing the number of the histograms to be matched against. So, for example, when searching for image portraits (images of faces) we might just ignore the elements around the edge and near the top where the hair might be. Specifying which grid elements should be matched could be fixed by the application (e.g. not edge elements), or could be the decision of the user via a query by example or query by sketch user interface such as that we created for our testing shown in Figure 3.3.

The selective spatial match might be defined as in the equation below.

$$\begin{aligned} \gamma_n &= \begin{cases} 0 & : \text{ if element } n \text{ not selected} \\ 1 & : \text{ if element } n \text{ selected} \end{cases} \\ s &= \frac{1}{k}\sum_{i=1}^{n}\gamma_i\|H_{I_i} - H_{M_i}\| \end{aligned} \tag{3.12}$$

This allows searching through the database for images which have only certain properties (for example, sky at the top of a landscape photograph, or skin in the middle of a portrait photograph) while ignoring all irrelevant spatial locations.

An extra level of functionality afforded by the grid mechanism is to match every selected grid element in I with every element in M to provide basic sub-image matching functionality by finding images which contain similar colours anywhere in the image — but without being affected by inconsequential colours(21).

$$s = \frac{1}{k}\sum_{i=1}^{n}\max_{j=1\ldots n}\left(\gamma_i\|H_{I_i} - H_{M_j}\|\right) \tag{3.13}$$

Without any indexing system, the match times are increased proportionally by the amount of grid-elements, or selected grid-elements, $O(n)$. Pre-filtering the results could be achieved by using low resolution histograms for either every grid element, or for each whole image in turn as described in (29).

Despite the advantages that using the grid method affords, the simplicity of the image segmentation can also cause inaccurate results. If, for example, the part of the query image one wished to search for was too small for a single grid element, then selecting that region would include unwanted pixels in the colour match. Overcoming this requires a higher resolution grid or a different segmentation technique. It's also quite likely that the area of interest is not boundary aligned with the grid borders, also causing background information to invade the match.

Sethi et. al (67) use a grid where all the elements overlap slightly which allows a certain amount of "fuzzy-ness" to be incorporated into the spatial match. They use a nonlinear function based on the average hue and saturation values in each grid elements to perform the colour match.

Leung and Ng (41) use histogram distance estimation techniques over a grid to avoid the matching of background image data, and to improve the calculation time. The histogram

estimation is provided by enlarging or reducing sub-image queries until the histograms are of a comparable size. 'Padding' is increasing the query region size to fit a grid element at a particular resolution with pixels that will minimise the histogram distance. 'Reduction' is reducing the grid element size to the size of the query region by only choosing pixels from the matching block which minimise the distance.

An alternative to overcoming problems with histogram matching on grids, is to create features which overcome the problem, using a different segmentation algorithm. Leung and Ng (41) also considered this and discuss a multi-resolution quad-tree based idea. We also investigated a similar idea, which is discussed below.

3.1.4 Quad-tree Matching

Using an image decomposition method based on the quad-tree (71) (see Section 2.2.6.1) it is possible to achieve more accurate spatial colour matching than the grid based method affords. We experimented with using quad-trees to increase the accuracy of spatial colour matching in our paper (21).

The quad-tree method divides an image based on the content of the regions. A homogeneity criterion is used to divide the image into regions containing a level of homogenous content. This may be based on colour, texture or any region-based feature. An un-even grid of regions is created, again, each of which can be taken as a self-contained image and matched using a colour histogram or feature matching technique. Because the regions are approximately homogeneous, very low resolution histograms can be utilised to increase response times. Indeed, the distance between representative colours for the region can be employed. Often, the use of this algorithm involves merging homogenous regions which are adjacent to one another. However, this loses information, in particular, it loses spatial information, which would defeat the object of segmenting the image. So, for quad-tree matching, the nodes in the tree are left as separate and an example image decomposition is shown in figure 3.4.

Once the image is divided into the quad-tree elements, matching can take place. However, the method for comparing two images requires an extra level of sophistication due to the equivalent elements in the query image, I, and the match image, M, having the possibility of being different sizes.

The matching takes place sequentially over the leaves of the quad tree, using an in-order traversal. The traversal strategy is largely irrelevant so long as every node in the tree is visited. At each leaf node, IL_n, matching takes place with the equivalent leaf node in the match image, ML_n.

However, it is possible that for IL_n in the tree for I, there may be no equivalent leaf node in the tree for M. The nearest leaf node may be at a lower or higher level than IL_n in the tree for I. By converting the nodes of the tree into leaf-codes we can recognise quickly those nodes which are able to be matched as proposed by Sonka et al. in (71). Leaf codes are strings of numbers

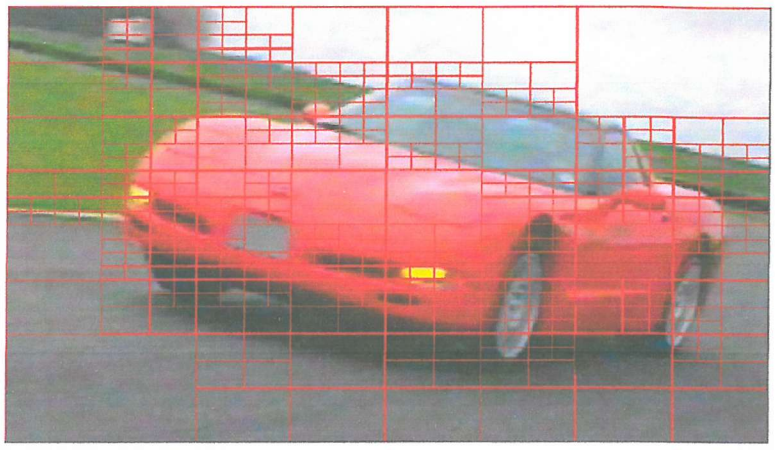


Figure 3.4: An example quad-tree overlaid on a real image using colour as the homogeneity criterion.

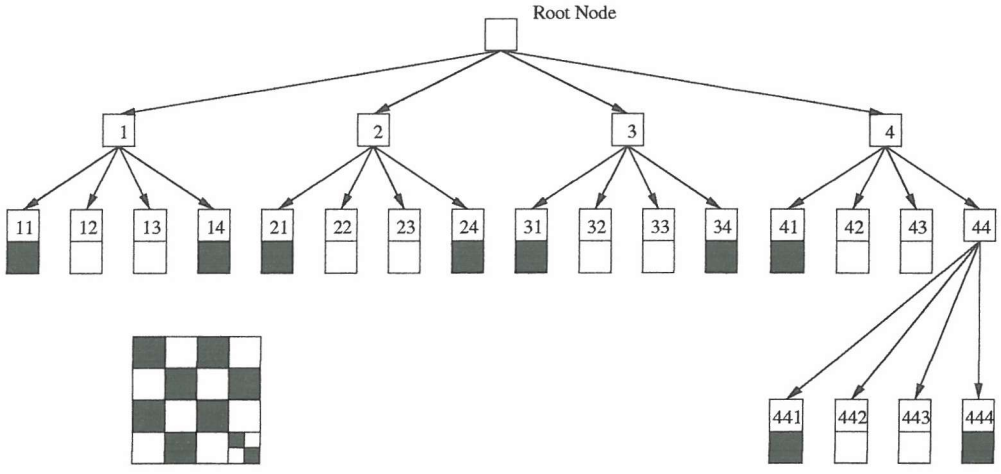


Figure 3.5: An example quad-tree and the leaf code designations.

representing the location of nodes in the tree, where each number represents the branch of the quad-tree to traverse. Quadrants of the image (and branches of the quad-tree) are numbered from 1 to 4 left to right and top to bottom. Each subsequent division is numbered similarly and concatenated to the previous directions, until a leaf node is encountered, as shown in Figure 3.5. This allows every node in the quad-tree to be represented by a unique code, the length of which, $\#LM_n$, is the depth in the tree at which the node resides.

Traversing the quad-tree, we select the leaf nodes, LI_n, and match them with their nearest equivalent by selecting the nearest leaf-code - that is, the lexigraphically nearest. If the nearest leaf-code has the same length (that is the leaves are at the same level), we can immediately match the histograms stored at those leaves.

If the nearest leaf-code in the match image, LM_n, is not a leaf node, then it means the match image is segmented more finely at that point. To perform the match, the histograms

of all the nodes below the node identified in the match image are summed to give a single histogram representing the same relative area as LI_n. The histograms are then matched as usual.

If LM_n is shorter in length, then the query element is smaller than the element in the match image which is closest. Rather than trying to reduce the size of the histogram represented in LM_n, the pragmatic approach suggests that the smaller query area, LI_n, can be directly matched with LM_n from the database. This would seem reasonable because for the match image, M, to be segmented in this way, the features must be close to homogenous, and further division of LM_n would barely effect the normalised histogram data represented there.

Given n leaf nodes, the spatial quad tree matching process can be defined by:

$$\begin{aligned} H_{LM_j} &= \begin{cases} \sum_{k=j}^{i+4} H_{LM_k} & : \quad \#LM_i < \#LI_i \\ H_{LM_i} & : \quad \#LM_i \geq \#LI_i \end{cases} \\ s &= \sum_{i=1}^{n} \gamma_i \| H_{LI_i} - H_{LM_i} \| \end{aligned} \tag{3.14}$$

In a similar method to the non-spatial grid segmentation matching, the quad-tree's leaf nodes can be used to search through the nodes of the match tree to find sub-images which match the selected elements in the query image. So given n leaf nodes in I and m leaf nodes in M, the functionality is defined as:

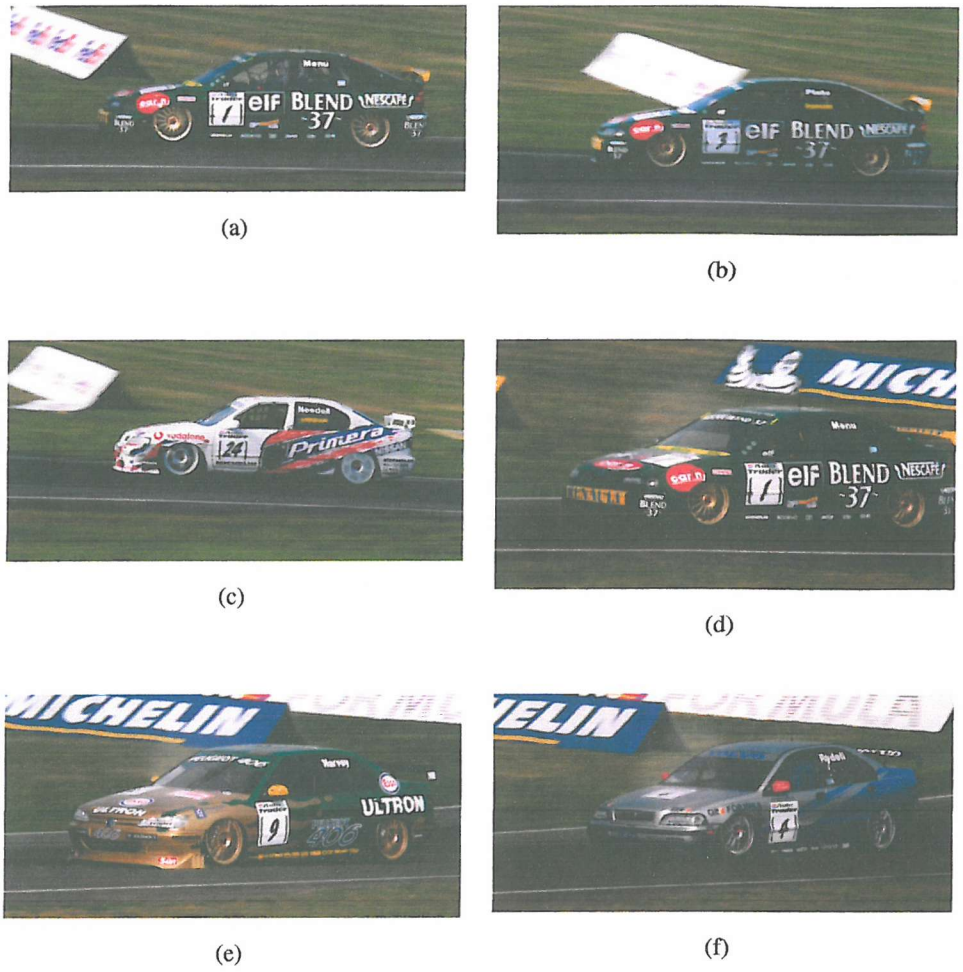
$$s = \frac{1}{k} \sum_{i=1}^{n} \max_{j=1\ldots m} \left(\gamma_i \| H_{LI_i} - H_{LM_j} \| \right) \tag{3.15}$$

3.1.5 Histogram Experimentation

In a simple test, we selected a small set of 76 colour images, taken from a collection of images of touring cars at the Brands Hatch circuit. The images are not artificially generated and have qualities that one would associate with photographic images (cf. graphic images) - that is, they contain noise and colour variances. Most, but not all, of the photos have one focal object (usually a car) which has distinct colours. This means that we can ground-truth this dataset to derive a quantitative accuracy for the algorithms. However, ground-truthing datasets is a difficult process, hence the limited size of the dataset.

The dataset has been subjectively classified into two categories. There are green cars, and there are green Renault Lagunas. The second classification includes all the cars from the same racing team which have the same colours. The first classification includes other green cars from the racing field which are green, but do not have exactly the same colours and decals. Classification 2 is a subset of classification 1. Of the 76 images 17 are of classification 1 (green cars) and 10 are of classification 2.

Our tests of the L_1- and L_2-distance matching on the whole image proved to be fairly poor. Taking an example image from classification 2, we generated an RGB histogram with 6 bins



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

Figure 3.6: The results of histogram retrieval using L_1 distance. The first image is also the query image.

---

in each dimension ($6^3 = 216$ bins in total) and matched it against histograms generated for the rest of the dataset using L_1 and L_2 distance measures. During the tests, the query image was given as the best rank (1), unsurprisingly, with a distance of 0. Both the L_1 distance and the L_2 distance retrieved a similar car in rank 2. Figure 3.6 shows the top 6 images of the match for L_1 histogram matching showing 3 relevant retrievals for classification 2, and 4 relevant retrievals for classification 1.

A well regarded method of quantitatively measuring the effectiveness of a retrieval algorithm is the *recall* and *precision* measures. We can define recall and precision as follows:

$$\text{Precision} = \frac{\text{Number of Relevant Objects Retrieved}}{\text{Number of Returned Objects for a Query}} \quad p = \frac{|A(q) \cap R(q)|}{|A(q)|} \tag{3.16}$$

$$\text{Recall} = \frac{\text{Number of Relevant Objects Retrieved}}{\text{Number of Relevant Objects In Database}} \quad r = \frac{|A(q) \cap R(q)|}{|R(q)|} \tag{3.17}$$

where $A(q)$ is the set of retrieved results, and $R(q)$ the set of relevant images from the dataset.

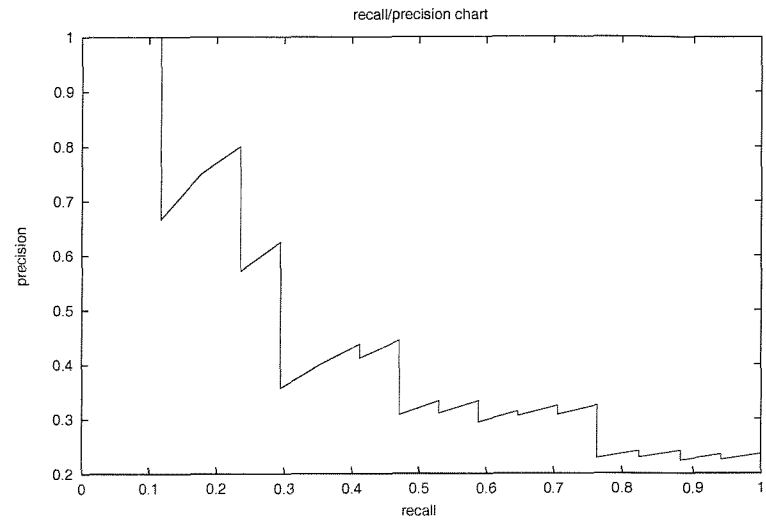
Of course, the precision means nothing when the simple test returns every object in the database because the number of relevant objects retrieved, $|A(q) \cap R(q)|$, will always be all of them! It is artificial to truncate the results, and by doing so we may either never get a recall of 1, or we would end up equating recall and precision (if $|A(q)| = |R(q)|$). However, we are able to plot the precision against the recall for the entire match process. The graph in Figure 3.7 shows a recall vs. precision graph of the L_1 match. It shows that the precision was 1.0 for a recall of upto about 0.2. This means, that the first 20% of the returned results were relevant. As false hits are retrieved the precision falls sharply. At a recall of 1.0, we have retrieved all of the relevant images, and the precision is then the ratio of false hits to relevant images. A perfect precision-recall graph would be straight, horizontal line, where precision is 1.0 at all values of recall.

For classification 1, for both the L_1-distance measure and the L_2-distance measure, the recall returned a poor $r = \frac{4}{17}$. For classification 2 the recall on both measures was $r = \frac{1}{3}$. These results are not exceptional with the false hit rate higher than the retrieval rate. The main reason is that the background data of the images contains so much information, which is also getting matched and skewing the results.

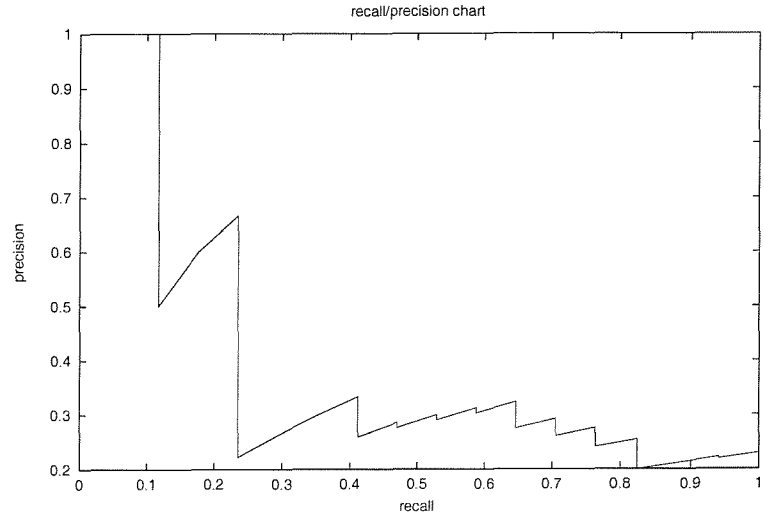
L_1 and L_2 colour histogram matching is useful for situations where the colour defines the scene, for example, a sunset, or fireworks. For example, from a different image dataset containing a mix of graphics, photos, and textures we have 13 images of fireworks from 9908 images. If we run a test across this dataset using an image of a firework as the query (mainly black with about 10%-20% of yellow/red) we find that the recall rate increases dramatically. For the Euclidean distance, the recall becomes $r = \frac{8}{13}$ $\left(\text{notably better than} r = \frac{1}{3}\right)$. However, the results are still not acceptable even when the colour defines the scene. The likelihood is that the quantisation of the histogram is causing errors in the matching. The coarser the quantisation the bigger the errors will be. These type of errors can be minimised by applying a weighting to the bin matching process, which is what the quadratic histogram matching algorithm does.

For the test on the dataset containing images of racing cars, the quadratic match gave a recall measure for classification 1 of $r = \frac{4}{17}$, and for classification 2 $r = \frac{5}{9}$, which is still quite poor.

It seems part of the problem with the histogram matching techniques we used above is that they take a match based on the whole colour content of the image. The images of the cars we have been using have a definite distribution of colour. The background of the images



(a) For L_1



(b) For L_2

Figure 3.7: Recall vs. precision graphs for L_1 and L_2 histogram matching for a particular query.

changes little, and generally consists of grass, tarmac and a yellow sign (although obviously specific to this dataset). This implies that the focus of the image, the car, is in the centre region of the image. This is in contrast to the firework images that are randomly distributed, and are retrieved better.

If we apply this full-image spatial match to our previous test, containing the images of the green cars, we find that the spatial match alone gives little improvement - again, due to the irrelevant colour information stored in the background being matched. Recall that for the L_1, L_2 and quadratic forms of the spatial match the recalls were $\frac{4}{17}$ and $\frac{1}{3}$, for classification 1 and 2, respectively.

If we apply the selective spatial match to our test set of racing cars, we can see the improvement immediately. By matching a selection of 9 sub-images in the centre of the image, where we may expect a car to be, we increase the recall from $\frac{4}{17}$ to $\frac{11}{17}$ for classification 1, and $\frac{1}{3}$ to $\frac{8}{9}$ with the selective spatial L_1 match.

During experimentation we found that the quad-tree algorithm performed no better, but no worse, than the simpler, and faster, grid element decomposition for spatial matching. Indeed, from our dataset of 40 images, the precision and spatial accuracy of the two algorithms were almost identical. The notable difference was that the quad-tree algorithm is more computationally intensive, due to the higher number of histogram matches that need to be calculated merely due to the higher granularity of the segmentation.

The matching of quad-trees without regard to the spatial location of the elements being matched is achieved in the same manner as the grid method - every element is compared against the query element(s). Due to the higher granularity, it is likely there will be more elements to match, meaning the time taken to match the same database could be longer. The quad-tree method could, however, have an advantage over the grid-based method, by using the multi-resolution features implicit in it's structure. By traversing the tree using a pre-order traversal, it is possible to decrease the match time, by discounting a set of histograms from the match. Pre-order traversal visits the nodes of trees first, so, by performing histogram matches at the nodes, it can be decided whether or not it is worth traversing the branches of that tree. This could not only accurately detect sub-images within images, but also decrease the response time of the system, by reducing the number of histogram matches that need to be executed.

3.1.6 Summary

In this section we have discussed some of the work that we originally investigated with regard to content based image matching. Desipte their widespread use, colour histogram techniques alone are not flexible enough to be of use in most situations. Quadratic histogram matching allow fuzzier matching to be applied to colour matching, and as the number of bins in a histogram increase, the more this becomes an issue. We described some of the work we investigated with regard to spatial matching of images based on colour. The grid mechanism affords cheap and

reasonably reliable spatial matching, particularly with regard to selective spatial matching. The extra execution time cost of the quad-tree algorithm limits the use of the quad-tree for selective (one-to-one) and whole-image spatial matching. For sub-image queries, the quad-tree is ideal, because it can avoid lengthly brute force matching of all of the elements, by subjectively dismissing subtrees for matching. However, despite allowing sub-image matching, the quad-tree method is still based wholly on the underlying histogram matching, which is, of course, only based on colour. Should an object in an image appear in various images with differences in the colour the object would not be found - even with the quadratic histogram matching. Something more sophisticated is required, and our work into scale-tree matching is discussed in the next chapter.

3.2 The "Lower Palaeolithic Technology, Raw Materials and Population Ecology" Project

As an example of feature extraction for image matching in a particular application domain, some work on the representation of archaeological artefact images is presented. The Lower Palaeolithic Technology, Raw Materials and Population Ecology project, to which this work contributed, was funded by the Arts and Humanities Research Board (A.H.R.B) between 1999 and 2001. It was directed by professors Clive Gamble of the Department of Archaeology at Southampton, and Derek Roe of the Baden Powell Institute for Palaeolithic Research in Oxford. Employed on the project was Dr.Gilbert Marshall from the Centre for the Archaeology of Human Origins (C.A.H.O) at Southampton.

The initial aim of the project was to generate a large virtual archive of Lower Palaeolithic bifacial artefacts (worked on both faces) from across the Acheulian world. The Lower Palaeolithic or Acheulian refers to a period from approximately 1.5 million to 250 thousand years ago, during which bifacial artefacts were being made from northern Europe to the southern tip of Africa, and across to eastern India. For a comprehensive discussion of the Acheulian, its technology and Hominid evolution see (37), Chapter 5. With the image archive in place and the characteristics of the artefacts measured and described, attributes such as the effect of raw materials, location, and temporality (where between 1.5 million and 250 thousand years) are being assessed. By taking a continental scale view of variability it is hoped that differences can be identified within the Acheulian, a period which until now has been considered as the first convincing evidence for successful global scale proto-human colonisation.

The database is to be hosted by the Arts and Humanities Data Service (A.H.D.S) based at the University of York, from where it is envisaged that it will provide a useful resource for research and teaching. It is also hoped that the archive and ways of using it will evolve as images from new excavations from across the Acheulian world are added. The database current contains no content based matching functionality although the provision is there, and is likely to be

implemented in the future. It is available on the Web at *http://ads.ahds.ac.uk/cfm/bifaces/bf_query.cfm*.

To achieve the functionality required, features are extracted from the digitised images of the flint artefacts and stored in a database along with meta-data about the artefact. Much of the meta-data we extract has been given elsewhere in the literature, however, before now, it has not been automated and incorporated into a system to an extent that will allow access to all the data from one application in a way which archaeological archivists would want to use.

Much of the feature extraction we perform is based on shape measures of the objects, because their colour is largely irrelevant in determining the type of object. Images of the artefacts are taken by a calibrated rostrum camera and digitised directly to computer where they can be processed by the feature extraction software.

3.2.1 Feature Extraction

It is fortunate that the images which are to be used in the system are easily removed from their background due to the fact that the object (the flint axe heads) are laid upon a black backing for photographing. The images are all 960x720 pixels saved as JPEGs. The feature extraction process starts by isolating the boundary of the object using Marr-Hildreth edge detection. The technique involves convolving a Laplacian of Gaussian operator with an image, $\mathcal{P}$, to give the gradient image, $\mathcal{G}$:

$$\mathcal{G} = \nabla^2(g(x,y)) * \mathcal{P} \tag{3.18}$$

The Laplacian of Gaussian operator, with kernel size σ, is given by:

$$\nabla^2 g(x,y) = \frac{1}{\sigma^2}\left(\frac{(x^2+y^2)}{\sigma^2} - 2\right) e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{3.19}$$

The Laplacian of Gaussian operator is derived from the Laplace operator, which approximates the second derivative thereby giving the gradient magnitude. This means edges occur at zero crossings rather than extremum and are easier and more precise to detect. The problem is that the Laplace operator is susceptible to noise. The Gaussian kernel effectively smoothes the image to avoid the response to noise. The Gaussian kernel is used because it optimises the compromise between the smoothing and the accuracy of the edge detection. The Laplacian of Gaussian is the basis of the Marr-Hildreth operator which detects the zero crossings in the convolved image to give closed edge borders, which is very useful when we are going to be tracing the boundary of the object. Other edge detection algorithms, such as Sobel or Canny, do not guarantee to give closed edges. Although the Gaussian operator has potential problems with edge wandering these do not affect the measurements made with the system to a degree which is unacceptable.

Edges occur at large intensity gradients in the image. Because this process relies on the contrast between the background and the foreground there is the potential for errors, for example on objects which are dark, photographed on the black backdrop. To avoid erroneous edge detection the image is thresholded. The thresholding maximises the contrast and is biased

towards black; that is, anything over a certain percentage black is blackened and everything else is whitened. There is the ability to alter the percentage at which the threshold operates for testing purposes. By maximising the contrast, the edges become strong and are detected more accurately.

Before the edge detection takes place, any erroneous noise picked up in the background during the threshold is removed with a Gaussian blur and the image is re-thresholded. A connected set operator, such as the sieve (see section 4.2.1), may provide a better means to removing this noise, as well as extracting the object shape. The edge detection is then performed and a closed boundary of the artefact is available in the edge detection image. These extra blurring and thresholding functions, along with the blurring built into the edge detector, gives the best resilience to noise.

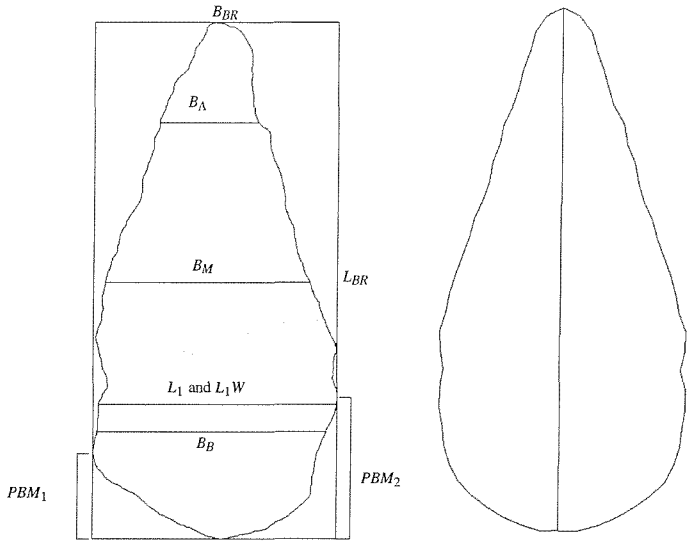
It is still possible that edge detection errors can occur, and these are primarily due to areas within the foreground artefact which are similar to the background colour; a common problem being black lettering, for instance the artefact accession number. This is not a problem when it is within the piece as the edge tracing algorithm can only start from an outer edge. However, when located along the edge it becomes included in the trace. It is very difficult to automatically detect these situations, but they can be discovered by means of the symmetry measure, which we will describe later, and then manually altering the blackness thresholding and running again.

We convert the boundary from pixels in an image to coordinates based on the centre of gravity. The centre of gravity, $(\bar{x},\bar{y})$, is found by the average of all the pixels in the binary image; this technique is called the first order moment as we described in Section 2.2.4.2. The image is scanned for a pixel that is not black. The assumption is that the edge image will contain only edges which belong to the artefact. Once a pixel is found we follow the edge, knowing that the Marr-Hildreth operator gives us closed edges. We can trace the edge to find its length - the perimeter of the object, l. In order to transform the image pixels into a polygon with N points, the edge is traced and every $n = \frac{l}{N}$ pixels along the boundary we save the point as $(\bar{x}-x,\bar{y}-y)$ coordinates which will make up the polygon delimiting the shape of the artefact.

Now there are four domains in which we can perform measurements: the original image, the binary image, the edge detected image, and the polygon points delimiting the shape.

3.2.2 Measurements

The measurements are mainly taken from the polygon points representing the shape, because this requires less processing. Figure 3.8(a) shows these measurements overlaid on an example artefact trace. The bounding rectangle delimited by top-left and bottom-right, $BR = (\min(x),\min(y),\max(x),\max(y))$, is easy to compute and gives us the simplest measurements: length and breadth (L_{BR} and B_{BR}), and consequentially aspect ratio, $R = \frac{L_{BR}}{B_{BR}}$. In the same way as Brande in (12), the width of the artefact at 20%, 50% and 80% of its length from the front of the axe head (B_B, B_M, and B_A respectively) is measured. Measurements of the height from



(a) Measurements taken on the hand axe images

(b) The σ-symmetrical shape for the hand-axe shown in Figure 3.8(a)

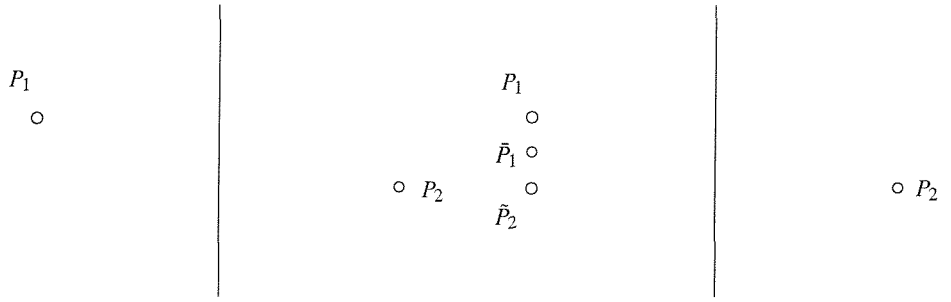
Figure 3.8: The hand-axe feature extraction includes a set of measurements and generation of the nearest symmetrical shape.

---

the front of the axe to where it touches the sides of the enclosing rectangle (PBM_1 and PBM_2), and the height and width of the widest part of the axe (L_1 and L_1W) are taken. We can find the length of the perimeter, c, by adding the length of all the lines adjoining the points which make up the boundary, and the area is provided during the calculation of the first order moment - the number of pixels within the boundary.

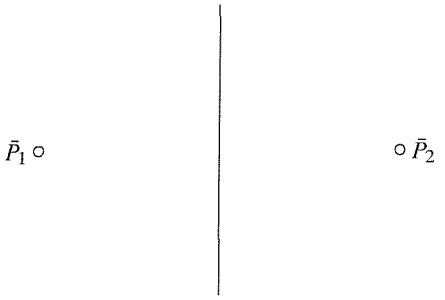
These measurements are all based on the orientation of the tool imposed by the archivist. It is possible to detect the longest axis or find the minimum enclosing rectangle and base the measurements on these computed orientations. The minimum enclosing rectangle can be found by calculating the enclosing rectangle while rotating the points which make up the shape. The rotation need only continue to 90 degrees, because of the symmetry. The angle of the enclosing rectangle with the minimum area gives the minimum enclosing rectangle (and the points of the shape which make it). The longest axis is a useful property extracted using the symmetry measure described below.

A symmetry measure of a shape is not a very useful metric in itself, as it is unlikely that objects can be classified simply by means of their symmetry, unless the application is very specific and the objects being identified have a strong symmetry, for example, finding imperfect plates on a production line. However, it can be used as an extra metric to classify objects more accurately. The technique has been used for classifying ancient flint hand-axes in



(a) Two points (P_1, P_2) about a line of reflection

(b) Point P_2 reflected to point $\tilde{P}_2$ and the average of P_1 and $\tilde{P}_2$ gives $\bar{P}_1$



(c) Point $\bar{P}_1$ reflected back to give point $\bar{P}_2$.

Figure 3.9: The process of reflecting a point about the line of reflection to generate a symmetric pair of points to calculate the σ-symmetric shape for the CSM.

(86; 66) and we also use it for supplementing our measurements taken on flint hand-axes. A symmetry distance measure is often referred to as chirality, which derives from the chemistry term 'chiral', meaning the *lack* of symmetry in an object; a symmetrical object being 'achiral'.

A bilateral symmetry measure such as the Continuous Symmetry Measure (CSM) gives the minimum distance between the actual shape, and an average symmetric shape, called the σ-symmetric(66), and sometimes G-symmetric(86), shape.

A σ-symmetric shape is generated by taking the average of pairs of points around the boundary. The boundary of the shape is split into two, and pairs of points, (P_1, P_2), are taken from relative locations either side of the line of reflection (Figure 3.9(a)). Point P_2 is reflected about the line of reflection to give $\tilde{P}_2$ and the average of P_1 and $\tilde{P}_2$ is taken to give $\bar{P}_1$ (Figure 3.9(b)). $\bar{P}_1$ is then reflected back to give a pair of symmetrical points, $(\bar{P}_1, \bar{P}_2)$, about the line of reflection dividing the two point sets (Figure 3.9(c)). The distance between (P_1, P_2) and $(\bar{P}_1, \bar{P}_2)$ gives the symmetry distance for those points about that particular line of reflection. The sum of all the pairs of points gives the distance measure for that particular symmetrical shape about

that line of reflection. The σ-symmetric shape is the symmetrical shape which minimises the distance between itself and the original shape for all the possible lines of reflection.

An example of a σ-symmetric shape is shown in Figure 3.8(b), which is the σ-symmetric shape of the hand axe shown in the accompanying figure.

The general definition of a CSM, given n boundary vertices of the original configuration in point P_i, and a symmetry group G, is:

$$S'(G) = \frac{1}{n}\sum_{i=1}^{n} \|P_i - \bar{P}_i\|^2 \tag{3.20}$$

where $\bar{P}_i$ are the corresponding points in the nearest σ-symmetric shape. To avoid scale effects, the size of the structure is normalised to the distance between the centre of gravity and the furthest vertex.

The resultant value is a measure of an object's chirality - that is it's distance from being symmetrical.

As a side-effect of calculating the symmetry measure, we also have the best axis for the shape (defined by the best line of bilateral symmetry). The angle and length of this can be calculated.

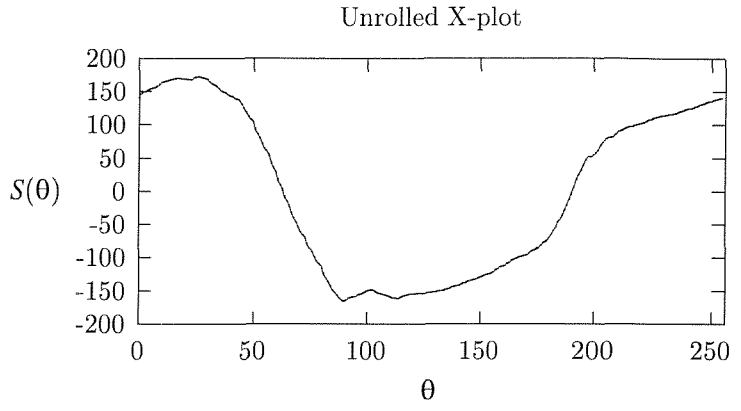
Although all these measures are very useful both for making distinctions between artefacts and for lightening the workload of the archaeological archivist, none of them give a precise (reconstructable) definition of the shape, which could be matched against.

For this we use the Fourier series to decompose a shape into its harmonic constituents. The Fourier descriptor method of artefact similarity measurement has been used before, for example in (27), however it was used to try to provide a classification of artefacts, rather than the ability to find or match similar ones algorithmically.

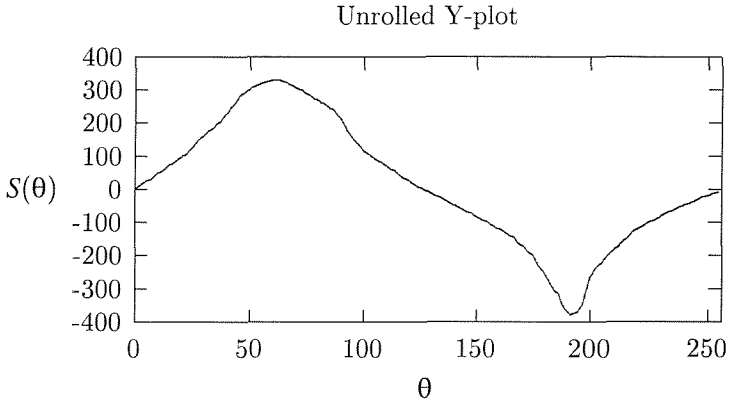
The Fourier transform converts a signal into its harmonic constituents. Once we have a shape signal extracted from the original image we are able to pass it into a Fourier transform. The shape of the artefacts in the system are represented simply as a set of (x,y) pairs. We unroll the shape into two 1-dimensional signals, (x,θ) and (y,θ), as in Figure 3.10, and use the Fourier transform on both to get two sets of Fourier descriptors. From the angle, θ, and the x dimension, it is of course possible to work out the y dimension. However, at certain asymptotic points in the signal y becomes undefined. By using both, we can avoid those situations. To use the radix 2 fast Fourier transform the size of the window has to be a power of 2, so as we unroll the shape we also scale θ into 256 points:

$$\begin{aligned} \theta &= \frac{256}{2\pi} tan^{-1}\left(\frac{y}{x}\right) \\ S(\theta) &= \sqrt{(x^2+y^2)} \end{aligned} \tag{3.21}$$

The signals can be passed through a discrete Fourier transform (DFT) process which returns the amplitudes and phases of each of the sinusoidal signals which combine to create the



(a) Unrolled X-plot



(b) Unrolled Y-plot

Figure 3.10: The unrolled shape signal, as a (θ, s) signal, of the shape in figure 3.8.

original signal. Fourier descriptors are fully invertible and reversing the process through a inverse discrete Fourier transform (IDFT) returns the descriptors to the signal they represent. This means that manipulation can also take place in the descriptor domain.

Knowing that the descriptors represent certain frequencies the values of the descriptors can be altered to alter the original shape. The descriptors represent at what amplitude and at what phase a sine wave of n cycles combine to the signal, for descriptor n. Removing descriptors will simplify the original shape. For example, creating the shape using only the first n descriptors gives the ability to show the shape at different resolutions. Figure 3.11 shows a shape at various resolutions by setting to zero descriptors above the desired scale. The shape at scale 2 is the simplest and is due to only large scale harmonics being combined. It can be seen from Figure 3.12 that descriptor 3 has a relatively large magnitude, and the effect can be

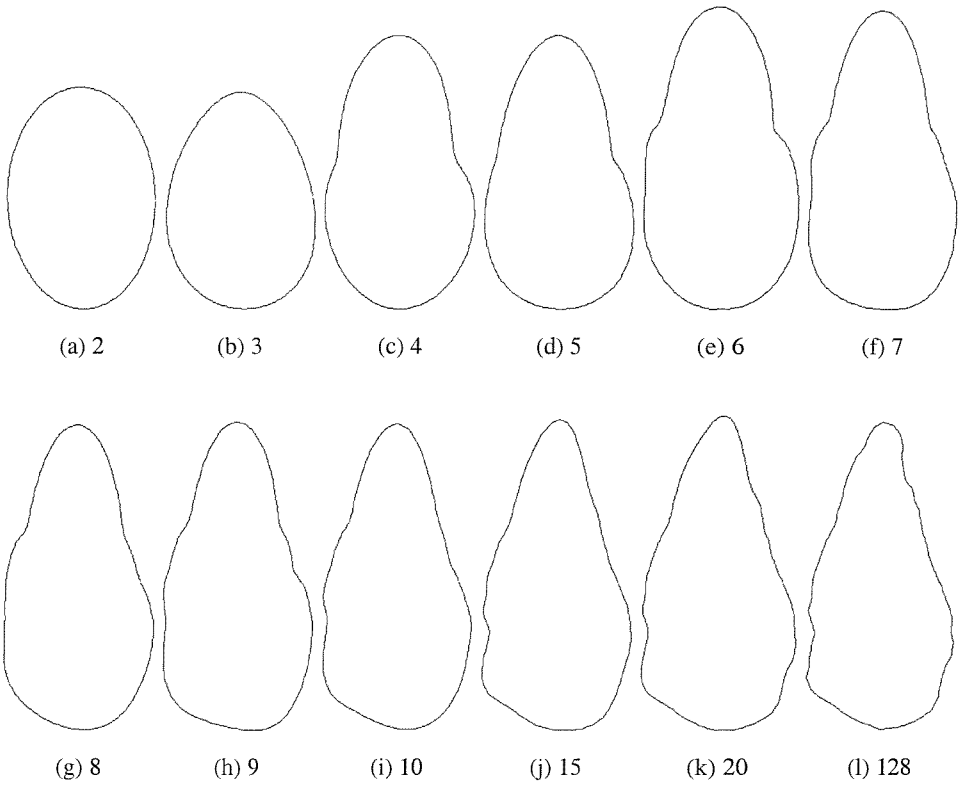


Figure 3.11: Reconstruction of a shape at various scales using Fourier descriptors.

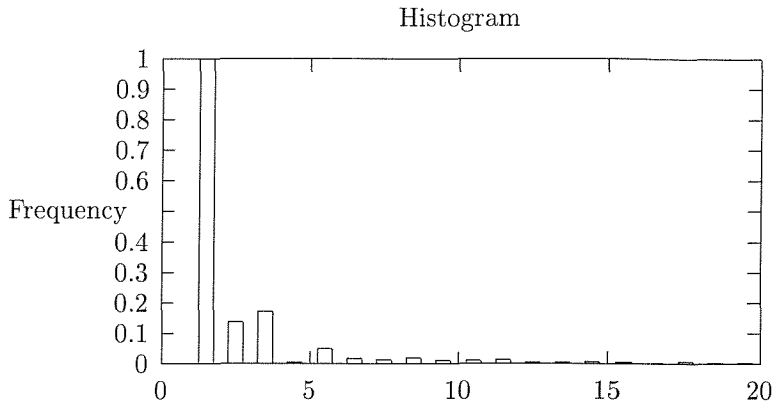
seen between the two reconstructed shapes of Figures 3.11(b) and 3.11(c).

For matching, invariant descriptors are preferred. Fourier descriptors are made invariant through a number of simple processes. For a Fourier descriptor $\mathcal{F}(n) = (\mathcal{P}_n, \mathcal{M}_n)$:

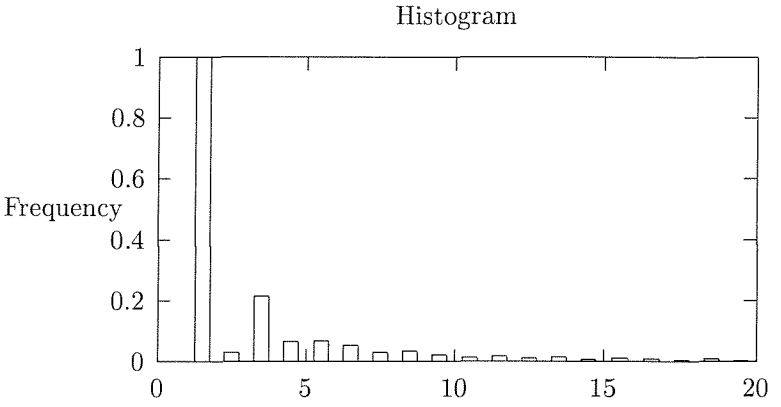
$$\begin{aligned} \mathcal{P}_n &= 1 \\ \mathcal{M}_n &= \frac{\mathcal{M}_n}{\mathcal{M}_1} \\ \mathcal{M}_0 &= 0 \end{aligned} \tag{3.22}$$

The phases are effectively ignored to give rotation independence. This can have consequences on the result, because although the shape becomes rotation independent, every constituent harmonic also becomes independent of every other harmonic. This effectively allows the object to be various shapes, although they will all share characteristics. The magnitudes are all normalised to the size of the second descriptor, $\mathcal{M}_1$, which makes the descriptors scale independent. The first descriptor's magnitude, $\mathcal{M}_0$, is zeroed to make the descriptors translation independent. Figure 3.12 shows the set of invariant magnitude descriptors for the shape in Figures 3.8, 3.10 and 3.11.

The invariant Fourier descriptors are very good for finding similar shaped artefacts. We



(a) X data Fourier descriptors



(b) Y data Fourier descriptors

Figure 3.12: Fourier signatures for the signals given in Figure 3.10.

are able to make the analogy between a set of Fouriers, $\mathcal{F}_Q$ and $\mathcal{F}_M$, (such as those in Figure 3.12) and a colour histogram. Matching of Fourier descriptors can take the form of L_1-distance (Equation 3.23) or L_2-distance (Equation 3.24) measures, where the distance between a pair of normalised magnitudes from equivalent Fourier descriptors, $\mathcal{F}_Q$[j] and $\mathcal{F}_M[j]$, is taken:

$$\|\mathcal{F}_Q - \mathcal{F}_M\| = \sum_{j=1}^{n} \|\mathcal{F}_Q[j] - \mathcal{F}_M[j]\| \tag{3.23}$$

$$\|\mathcal{F}_Q - \mathcal{F}_M\| = \sum_{j=1}^{n} (\mathcal{F}_Q[j] - \mathcal{F}_M[j])^2. \tag{3.24}$$

However, in the same way that the frequencies of pixels in certain bins in the colour histogram can be affected by wayward noise in the capturing process, the magnitude of de-

scriptors may also be affected by noise in the digitisation process of the shape. A slight variation in colour during the creation of the histogram, may cause one pixel to be added to the neighbouring bin. In a similar way, any variation in the boundary of a shape may cause the harmonic representing part of the boundary to be made stronger thus affecting the descriptors. So, in the same way we can reduce the effect of such noise in colour histogram matching using the quadratic histogram match, we can employ the same technique for Fourier descriptor matching. Neighbouring bins in a colour space represent colours which are very similar, and similarly, neighbouring descriptors in Fourier space represent harmonics which are very similar in wavelength. However, where the weighting would normally be based on the distance in colour space between the two colours representing the two colour bins, the distance for Fouriers is undefined. As a simple weighting value for the matrix A, we can use a 2-D Gaussian curve (Equation 3.25). The match characteristics can then be affected by altering the standard deviation of the Gaussian kernel used (i.e. more or less fuzzy).

$$a_{ij} = e^{\frac{-(i^2+j^2)}{2\sigma^2}} \tag{3.25}$$

As for colour histograms, $\mathcal{F}_Q$ and $\mathcal{F}_M$ are then represented as one dimensional matrices and subtracted to give the difference matrix Z:

$$z_i = \mathcal{F}_Q[i] - \mathcal{F}_M[i] \tag{3.26}$$

$$Z_{\mathcal{F}_Q\mathcal{F}_M} = \begin{pmatrix} z_1 & z_2 & \cdots & z_n \end{pmatrix} \tag{3.27}$$

The similarity between the two sets of Fourier descriptors is calculated using the quadratic formula:

$$\|Z\| = Z^{\mathrm{T}}AZ \tag{3.28}$$

Something to be aware of is that, with shapes, the distance score is not as concrete as the distance score for colours. For example, one might consider black to be as far from white as possible, and for shapes, one might consider a square to be as far from a circle as one could get, however, the fourier decompositions both share some similarities. This means that for shape matching with Fourier, the range of the distance measure will be smaller than that of the colour histogram using the same matching algorithm due to the fourier representation of the shapes.

3.3 Summary

In this chapter we have described some of the work we have done in facilitating better content-based retrieval using colour and spatial colour. In particular we discussed the possibility of using a spatial colour segmentation using a grid, and a quad-tree representation and the associated matching techniques for matching sub-images with the foresight to matching objects. However, the techniques, even the quad-tree technique, are too limited to allow matching of

particular objects due, primarily, to their dependence on colour. What is required is a segmentation and matching technique that is both dependent on the content (compared to the grid methods which are imposed on the content) yet independent of the matching techniques used to compare them. We have seen that multi-scale algorithms, such as the quad-tree, are useful in representing the structure of the image, allowing selective matching to be performed on the image. With an algorithm-independent, content-based segmentation, the representation will be matchable by both feature and structure allowing more flexibility and the possibility of object matching and recognition. We describe such an algorithm in the next chapter.

In this chapter, we have also described some on-going work with the Department of Archaeology at Southampton to archive and facilitate matching on ancient flint hand-axes. This work is novel in its field, bringing together many measurement and matching techniques into the digital domain for the annotation and indexing of the artefacts. This project is still underway, and the hope is to provide content based retrieval of archaeological artefacts over the web to other archaeologists. The feature extraction performed in this project can be used for shape extraction and matching in the novel work we are undertaking, which is described in the next chapter.

Chapter 4

Content Based Retrieval with Scale-Trees

4.1 Introduction

The aim of this thesis is to present a new algorithm for achieving content based retrieval with images, that will work towards bridging the semantic gap between low level feature based retrieval and high level semantic based retrieval. The idea revolves around building a topology structure of regions within an image, and using the topology structure along with low level features to match objects. The assumption is that an object has a particular topology of features, and, by using subgraph isomorphism techniques we can search for this topology within a set of graphs and hence find similar objects. It is of course a rather limiting assumption, as many objects do not have particular topologies (e.g. the sky). However, there is the possibility of using this content based retrieval system to find objects which DO have specific topologies, and using those objects to make some postulation about the scene and hence move towards bridging the semantic gap.

The following sections explain how we decompose an image to create a graph, and then how that graph is matched against similar graphs to achieve retrieval.

4.2 Decomposition

As humans, we tend to look at images and, without explicit conscious thought, see what the subject is, and make postulations about the semantics. Currently, in computing, there is no foreseeable way for semantics to be extracted from images without the need to first decompose the image into features which we understand. Indeed, it is thought that the brain actually does decompose the views we see below our higher-level thoughts. The premise is that if we

can decompose the image into regions, from the relationship between features that we can understand, we conjecture about what they represent at a higher level. If such a technique works, then higher level algorithms need only work with semantics much in the way a human's brain does.

The decomposition of images can be achieved in many ways, and we use the sieve idea to decompose an image into an initial scale-space representation from which a topology scale-tree is built. In the following section we describe the sieve and how we use it to build scale trees.

4.2.1 The Sieve

The sieve(9) is a morphological scale space operator developed by the University of East Anglia's School of Information Systems[1]. The sieve has the effect of locating intensity extrema and, using region merging, removes the region at a particular scale to produce a multi-resolution version of the image. The discussion, in section 5.4, explains the use of different intensity scales for extrema detection.

One of the advantages of using the sieve for scale space decomposition, compared to other scale-space techniques, is that the contours of the image are not altered. Also, unlike Gaussian blurring, the sieve preserves scale space causality - the axiom that states that no new extrema are created when moving to a coarser scale. Gaussian scale space decomposition techniques can break the scale space causality axiom by creating multiple extrema from a single extremum in anomalous situations. A well known example of this is that which Lifshitz and Pizer found (62) - the two regions connected by a thin isthmus. Gaussian blurring weakens the isthmus more than the regions which it connects, creating two maxima and a minimum in between - three extrema from one.

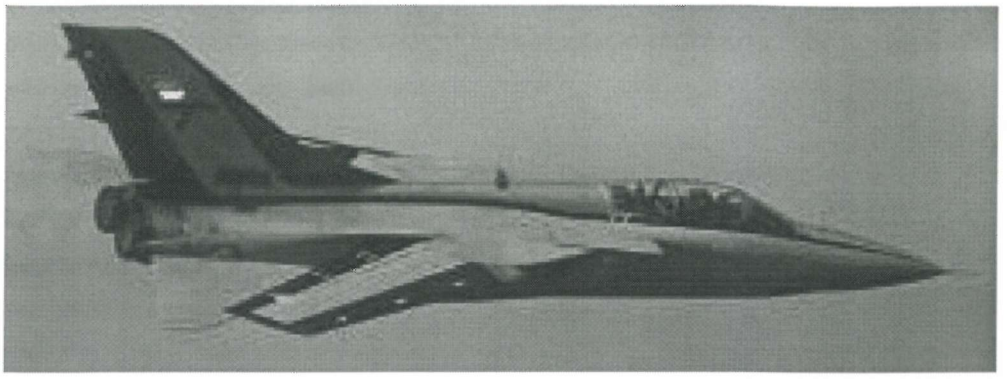
The sieve is also fully invertible, meaning the original image can be re-produced exactly from the scale space decomposition. An example of a scale space decomposition is shown in Figure 4.1. In Figure 4.1(b) the image is sieved to scale 1000 which means any extreme region of area less than 1000 is merged into the image and the sieved image will contain only extreme regions of area (scale) greater than 1000. Many of the small details are removed - although the contours of the remaining shapes in the image are unaltered.

The sieve operates on connected set graphs (33; 9). The array of pixels in the image can be represented as a connected graph (4- or 8-connected for images) $G = (V, E)$, where V is the set of vertices and E is the set of pairs describing the edges. We define the $C_r(G)$ function to give the set of connected subsets of G with r elements containing pixel x:

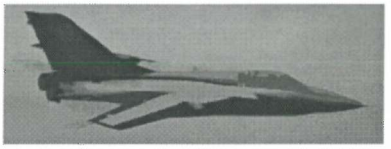
$$C_r(G,x) = \{\xi \in C_r(G) \mid x \in \xi\} \tag{4.1}$$

Morphological operations can be carried out using these connected sets. Morphological opening is achieved by taking the minimum value of every grey-scale value in $C_r(G,x)$ and

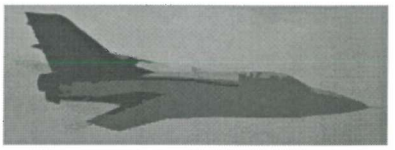
[1] Segmentis Ltd. hold the patent to the sieve (8), *http://www.segmentis.com/*



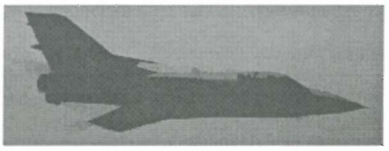
(a) Original Image



(b) 1,000



(c) 3,000



(d) 7,000

(e) 10,000

Figure 4.1: Using the sieve to generate a scale-space.

then taking the maximum of those minima. Closing, conversely, takes the maximum of every grey scale value, and then the minimum of those maxima. The compact definition of opening, ψ_r, and closing, γ_r, is given in equations 4.2 and 4.3.

$$\psi_r f(x) = \max_{\xi \in C_r(G,x)} \min_{u \in \xi} f(u) \tag{4.2}$$

$$\gamma_r f(x) = \min_{\xi \in C_r(G,x)} \max_{u \in \xi} f(u) \tag{4.3}$$

Openings remove maxima in the signal, and closings remove minima. The $\mathcal{M}^r$ and $\mathcal{N}^r$ operators are defined respectively as an opening followed by a closing and, a closing followed by an opening both of which remove all extrema - maxima and minima. The results of the $\mathcal{M}^r$ and $\mathcal{N}^r$ operators are different and to ensure consistency in, for example, an image decomposition, only one of these should ever be used. It is the $\mathcal{M}$ and $\mathcal{N}$ operators that define a sieve.

$$\mathcal{M}^r = \gamma_r \psi_r, \tag{4.4}$$

$$\mathcal{N}^r = \psi_r \gamma_r. \tag{4.5}$$

The regions that are being removed by the sieve at any particular scale are called *granules*. A granule of scale s is removed by a sieve of scale $s+1$ (see Figure 4.3). These granules can be moved into the *granularity domain* - a discrete scale space for the sieve. Addition of all the granules in the granularity domain will return the original image - an example of the invertibility of the sieve.

The output at scale r is defined as:

$$f_1 = Q^1 f, \tag{4.6}$$

$$f_{r+1} = Q^{r+1} f_r \tag{4.7}$$

where Q is one of γ, ψ, $\mathcal{M}$, or $\mathcal{N}$, which means Q^1 is the identity function: $f = Q^1 f$. That is, a sieve of scale r is a sequence of sieves from scale 1 up to scale r. The complete sieve function, given by $CS(f)$, is a function of sieves of all scales that yield granules.

Because only the extreme part of the grey-scale value regions in the image are removed, they are effectively merged into the original image. The effect is to 'slice off' those peaks and troughs (see Figure 4.2 for an example visualisation), always leaving only larger regions in the image - preserving scale space causality. This implies that all granules have 'parent' granules into which they are merged when they are sieved. This is ideal for transferring into a tree structure, such as that in Figure 4.4 where the links between the granules are explicit, facilitating easier manipulation in the scale-space domain.

We use the sieve operator for the image decomposition within our image matching algorithm because of the properties which elevate it above normal scale-space decomposition, which are:

- Contour Preservation

  It is quite important that the contours of an image are preserved when the intention is to use them for matching. The Gaussian scale-spaces round sharp corners, and the edges of objects move around over scales. Anisotropic scale-spaces help to overcome this problem, but the fact remains that still the blurring alters the object boundary - making it somewhat difficult to decide the actual location of it without backtracking. The sieve gives regions with sharp boundaries, and removes the whole region at once, giving well defined regions in the granularity domain, suitable for matching.

- Scale-Space Causality Preservation

  Scale-space causality is an important concept when working in the scale-space domain. The sieve preserves scale-space causality which means that no new extrema are generated as the decomposition moves towards coarser scales. Using this scale-space to build a tree, will ensure that the branches in the tree are consistent and have meaning with regard to the underlying structure. It also allows tracking through the scale-space to find original,

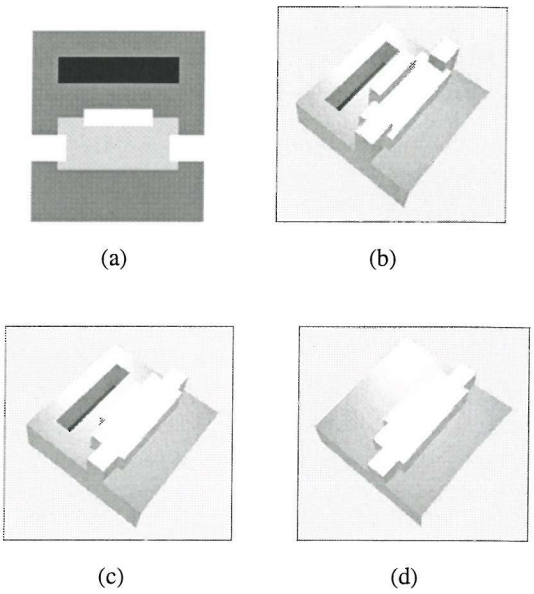


Figure 4.2: The graylevel view of the image in 4.2(a), and its decomposition through scales 4.2(b) - 0; 4.2(c) - 4; 4.2(d) - 16.

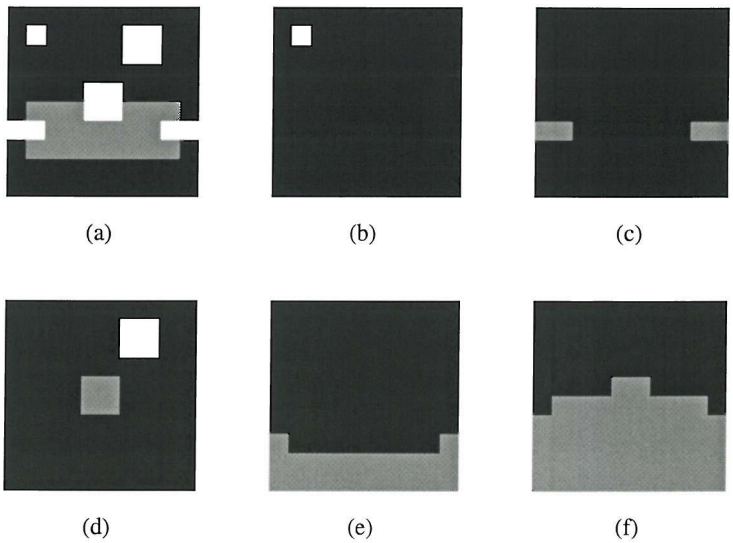


Figure 4.3: The granules removed at scale 4.3(b) - 4; 4.3(c) - 8; 4.3(d) - 16; 4.3(e) - 88; 4.3(f) - 200 from the original image in 4.3(a).

fine detail features, which, had the scale-space not preserved scale-space causality, would not be possible because a final extrema may have been an anomaly. It is important that larger scale regions are actual image regions to make the assumption that a sub-tree may represent an object topology.

- Scale-Tree Construction

  Due to the way in which the sieve operates, construction of a tree from the decomposition becomes relatively simple; the decomposition is a region merging operation yielding children and parents. The tree could be built directly during the scale-space decomposition to avoid the extra overhead of storing a granularity domain.

Our implementation uses an initial segmentation of the image into regions by luminance value. The regions in the image are sorted based on luminance and extremity, and then merging occurs as the morphological sieve would do - merging small extremum to larger regions. At each merge a new node in the scale-tree is created containing the region which is merged. At the end of the algorithm, the scale tree is complete. Although this initially makes our version slower to sieve than the original sieve, it would seem to be more flexible for other types of sieve decomposition. The decomposition need not be based on luminance, but could be based on anything that would generate local extrema (for example, saturation, or a mixture of both saturation and luminance). Aggregations of various types of image decomposition could be combined to generate an initial segmentation from which the scale tree is built.

4.3 Scale Trees

A scale tree is built from the decomposition by using the way that the regions merge into each other to define the relationships between nodes in the tree. The nodes of a scale tree represent the regions in the image at various scales. Regions at the leaves are absolute in the image, and regions at lower levels represent composite regions generated from merging of regions at higher levels. The branches in a scale-tree represent the merging operations. A particular branch will join a node representing a smaller region at a lower scale to a node representing the a larger scale region into which the smaller node was merged. An example scale tree is shown in Figure 4.4.

Building a scale tree is relatively simple, although rather costly. In the following sections we describe how it is possible to build a scale tree from the granularity domain, and how we build one directly during the sieving process.

4.3.1 Scale Tree Creation from the Granularity Domain

From a granularity domain a corresponding scale tree can be constructed. By traversing the granularity domain through the scales from the smallest scale to the largest, nodes are created

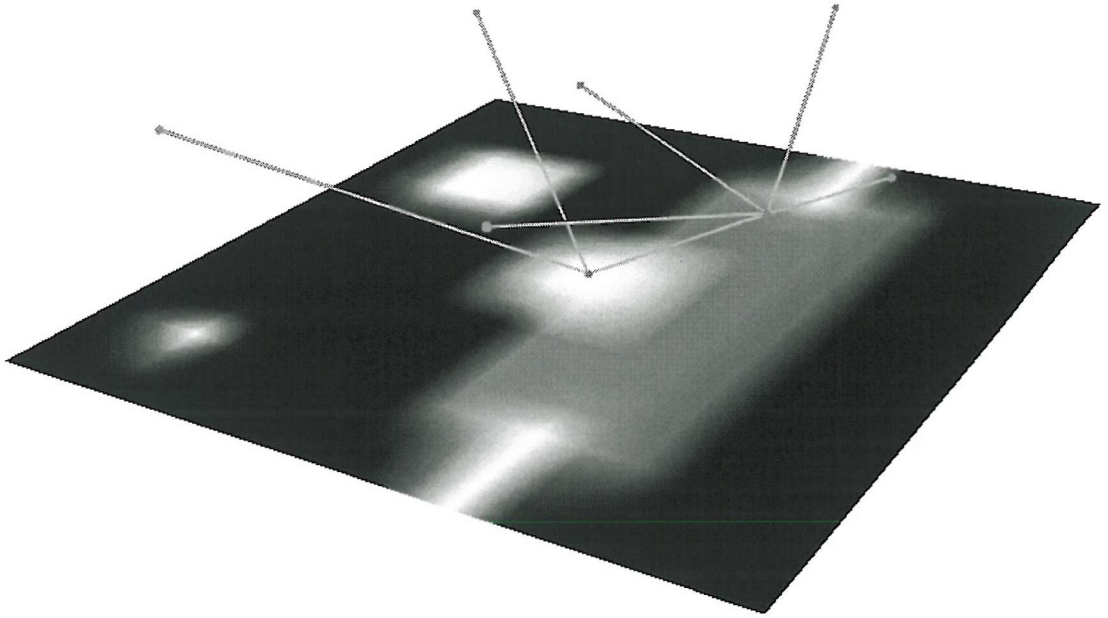


Figure 4.4: A scale tree generated from the sieve decomposition of the image in Figure 4.3(a).

corresponding to granules.

For each granule g_{s_i} at scale s, a node, N_{s_i}, is created. It is temporarily linked to a dummy node which represents all the other points in the image. At the next scale, s', another granule, $g_{s'_j}$, is encountered and the node $N_{s'_j}$ is created to represent that granule.

The father relation, $\mathcal{F}$, is given by Equation 4.8. $[n,m]$ represents an edge, or branch, between nodes n and m, and E represents the set of edges in graph, or scale tree. $\updownarrow(n)$ represents the distance from the root of the tree to the node n - or the level at which the node resides. So, a father relation exists between n and m if, and only if, there is an edge between the nodes in the tree, and they are 1 level apart, with node m closer to the root.

$$\mathcal{F}(n) = m \quad \text{iff} \begin{cases} [n,m] \in E \\ \updownarrow(n) - \updownarrow(m) = 1 \end{cases} \tag{4.8}$$

So, if the the node N_{s_i} is at a smaller scale and is a subset of the node $N_{s'_j}$ in the granularity decomposition, a father relation is created between the nodes N_{s_i} and $N_{s'_j}$.

$$\mathcal{F}(N_{s_i}) = N_{s'_j} \quad \text{iff} \begin{cases} s < s' \\ g_{s_i} \subset g_{s'_j} \end{cases} \tag{4.9}$$

The decomposition continues like this until there are no more granules left in the granularity domain. If there are dummy edges left they are all joined to a single root node containing the plane from which the image is rooted.

4.3.2 Scale Tree Creation During the Sieve Process

The method we use of scale tree building is very similar to the granularity domain decomposition, except that the initial segmentation of the image is generated by region labelling. From the region labelling we perform a sieve-like operation to generate the scale tree, skipping the generation and overhead of the granularity domain stage.

The region labels are generated by numbering each of the connected sets within the image. The sets are stored along with their size and value within the chosen feature space. A sorted list of the regions is generated based on their scale, and their extremity.

The regions are then merged in order of the list until there is only one region left in the list, which will be neither a maximum nor a minimum. Because the merging is explicit and not generated by an overall image filter, it is possible to create two scale tree nodes for each region involved in the merge process, and an edge (father relationship) between them, during the decomposition process. The regions can then be associated with the nodes without the need to have an extra processing step. Due to the sorted list, this, of course, preserves scale-space causality.

4.3.3 Pruning Scale Trees

A typical scale tree will contain thousands of nodes which is very costly to match using any tree or graph matching technique - some of the best algorithms are still quadratic. The solution is to compress the tree into a smaller number of nodes which are more easily used in a matching situation, while still retaining relevant information.

The compression technique compares child and parent, and sibling nodes in the tree. If the distance between the features of any two nodes is small - below some threshold - then the nodes can be merged into one single node representing the larger of the two regions. The larger of the regions becomes the result to ensure that the simplified image contains the same contours.

Generally, noise and blurring in the image cause distortions in the scale tree which are relatively simple to detect.

Noise can occur in two main forms, both of which have distinctive signatures in the scale tree. If there are many child nodes which have small area and have similar colour to their parents, then they are most likely noise on the signal, and can be removed. If a child node is almost the same as it's parent, in size, shape, and colour, then it is most likely a product of blurring within the image, and the child can be merged into the parent. Choosing a threshold for the features is something we discuss in the following sections on matching scale trees.

The most common form of noise is due to interference and inaccuracies in the electronic circuits and storage medium of the capture device. This noise is generally on a very small scale (usually no more than one or two pixels) and occurs as variations in the colour and intensity of the pixel compared to its surroundings. This makes it a very weak extremum. This type of noise causes randomly positioned nodes to appear in the scale tree, although almost always

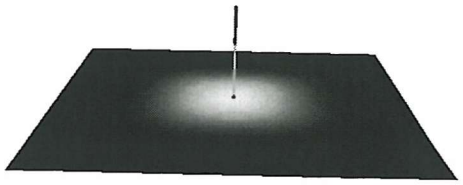


Figure 4.5: Blurring causes long chains of nodes to be generated in the scale tree space.

around the smallest level. They can be detected with a very tightly bound threshold about the parent node. The node n_1 is removed if the feature value of the node, $fv(n_1)$, is within specified limits, x and y, of the feature value of the parent node and the scale of the node (area of region represented at the node), $\uparrow n$, is very small (of less than t). Equation 4.10 shows this formally.

$$merge(n_0,n_1) \quad \text{iff} \begin{cases} fv(n_1) > fv(n_0) - x \\ fv(n_1) < fv(n_0) + y \\ \uparrow (n_1) \leq t \end{cases} \tag{4.10}$$

Aliasing in the image, that is smoothing of edges, is caused by overlapping of the signals from nearby pixels, in a CCD, for example. This blurs edges in the image, which causes chains of connected nodes in the tree. Figure 4.5 shows an exaggerated example of blurring. The figure shows the scale tree of a blurred white box on a black background using grey-level as the basis function of the sieve. The chain of nodes are caused as each successive extrema, at the brightest centre spot, are merged into the next grey-level below.

Chains of nodes can be tested by comparing a node at level o, n_o, with its child node, $n_{(o-1)}$. If they contain similar regions with respect to some feature matching function and their scales are similar, the process is applied again, comparing $n(o-1)$ with its child $n_{(o-2)}$. This process continues until the feature vector of n_i, $fv(n_i)$, is not within the upper and lower similarity threshold limits, x and y, or the scale of the region represented by node n_i, $\uparrow (n_i)$, is not within the scale threshold, t, of $n_{(i-1)}$ or the number of children of the node n_i, $\#children(n_i)$, is greater than one. If n_i has more than one child it can only be at the end of a chain representing a blur. All nodes $n_{(i...j)}$ are merged into n_o. n_o receives the properties (colour or greylevel) and children of the most extreme node that was merged (n_j). Equation 4.11 shows the merging criteria formally.

$$merge(n_o,n_j) \quad \text{iff } \forall o < i \leq j, \begin{cases} \#children(n_i) \leq 1 \\ fv(n_{(i-1)}) > fv(n_i) - x \\ fv(n_{(i-1)}) < fv(n_i) + y \\ \uparrow (n_i) - \uparrow (n_{(i-1)}) \leq t \end{cases} \tag{4.11}$$

Another way to decrease the number of nodes a scale tree has, is to eliminate some of the regions before they are even assigned to scale tree nodes - effectively filtering the image before performing the sieve decomposition, or while building the tree representation. The

sieve is generated from the greyscale representation of the image in some feature space. This greyscale representation can be quantised to a pre-set number of bands. For example, rather than using 256 greyscale values, the image can be quantised to, say, 64 greyscale values. This reduces the number of regions that are initially given to the sieve operator, and therefore less nodes are initially generated, increasing both the speed of the decomposition and the speed of the pruning operations. A quantisation method will not alter the contours of the image, and it therefore ensures accuracy of the decomposition, unlike a blurring operation, for example. Causality is preserved, although it is possible that regions that would otherwise have been in the scale-tree (even after noise and blur tree pruning) will get lost. This compromise may be necessary to allow the graph matching to proceed efficiently.

A well known technique for noise removal on images is that of the median filter. It is a specific instance of the general rank filtering techniques that order a set of pixels in a neighbourhood into a sequence. The result of the pre-processing is some statistic on these values, of which the median is one possibility. The median filter takes the middle value of the neighbourhood set. However, simple median filtering does not preserve edges. Different shaped neighbourhoods can be taken to preserve different features in the image. By taking all possible shaped neighbourhoods, of a certain size, edge contours are preserved.

4.4 Graph Matching and Subgraph Isomorphism Testing

Graphs are used in many applications for representation of objects and concepts, due to their generality and the power which these data structures provide. They have many invariance properties perfect for fuzzy problems. A graph under transforms such as rotation, translation, or mirroring is still the same graph in the mathematical sense, making them well-suited to modelling complex objects or scenes.

When graphs are used in a content based retrieval role, the problem becomes one of graph matching - determining a similarity score between two or more graphs. Typically, vertices in a graph would represent parts of an object, which, in image matching, are likely to be regions of pixels. The edges between vertices would be defined relationships between the pixel regions. Matching the graphs requires matching both the data at the vertices and the relationships between the data.

The main problem with using graphs in content based retrieval situations, is that the finding of graph and subgraph isomorphisms has combinatorial growth - that is, it is NP-complete. In the worst case of a fully connected graph, the algorithm will require at least $O(U^V)$ for graphs with V and U vertices respectively. There are known ways of addressing an NP-complete problem, using stochastic methods like simulated annealing and genetic algorithms, however these methods do no guarantee an optimal solution. Currently any algorithm designed to find an optimal solution will be intractable in the worst case and this applies to the algorithms below, but in the general case, they are faster than a simple graph search.

Of the few subgraph/subtree isomorphism techniques we decided to use the Network Algorithm developed by Messmer and Bunke (see section 4.4.2) as it has the functions we require. Even though it is somewhat redundant having graph matching when we are sure that the structures are trees, it overcomes the need to find an efficient indexing mechanism for the subtree isomorphism algorithms which are designed to only find subtrees between two single trees. This has none of the indexing properties which are necessary in large searchable databases.

Indexing is very important when the technique is designed to search for similarities between a number of models. Without indexing a brute force approach is required - that is, performing a match between the query and every model in the database. This can take considerable time in a database which could contain thousands of representations. Indexing basically reduces the possible number of matches by reducing the dataset during the match based on the query feature. This can be implicit in the data structures used or achieved by literally filtering before hand, as the improved colour histogram quadratic match from QBIC(30).

The Network Algorithm contains both indexing with the binary network, and efficient matching by sharing of common data structures between model graphs. This is due to the design of the special network structure. It is least efficient with highly unbalanced trees, but these are unlikely to occur in scale-trees - only certain blurred subjects causing the feature.

The Inexact Network Algorithm stores edit costs along with instance matches. The edit costs determine how the instances are filtered through the network. STOP lists act as buffers which prevent high cost instances from being forwarded through the network.

The following sections describe in detail the network algorithm which will find a subgraph isomorphisms between one graph and a set of graphs.

4.4.1 Definitions

This section presents some definitions to allow the description of the graph matching algorithms to be more easily understood.

Graphs are the basis of the algorithm, and a description for graphs needs to be introduced. A graph, G, is the union of the set of edges, E, and a set of vertices, V.

$$G = (E, V) \tag{4.12}$$

$$V = v_1, v_2, \ldots, v_n \tag{4.13}$$

$$E = (v_A, v_B), (v_C, v_D), \ldots \tag{4.14}$$

A label, $\mu(v_A)$, represents some value which is stored at the vertex, v_A, in a graph. For vertices which are derived from scale tree nodes these labels will be the attributes of the regions. For example, $\mu(v_1) = RedCircle, 100, 100, 20$, might represent a circle at coordinates $(100, 100)$ at scale 20. It is also possible that the edges will have labels, $\mu((v_A, v_B))$, representing the relationship between the two vertices. For example, an edge $\mu((v_1, v_2)) = Inside$, may represent the relationship that vertex v_1 is contained within v_2. The edges could store redundant

information to increase the speed of the match. For example, pruning speed could be increased by having an edge attribute which was the scale difference between the two nodes.

When comparing two graphs we are effectively looking for a function $f : V \rightarrow V'$ which maps each vertex $v \in V$ onto a vertex $v' \in V'$. For graph monomorphisms (instances within graphs with missing edges), $v = v'$ for all $v \in V$, and $(v_i, v_j) \in E = (v'_i, v'_j) \in E'$. For subgraph isomorphisms (instances within graphs which are the same), the definition is the same as graph monomorphism, with additionally for an edge $(v'_i, v'_j) \in E' = (v_i, v_j) \in E$ - that is, all the edges are the same with no missing and no extra edges. Double subgraph isomorphism testing, finds subgraphs of G_1 in G_2 and subgraphs of G_2 in G_1. It is no more computationally intensive than subgraph isomorphism testing.

4.4.2 Network Algorithm

Messmer and Bunke have developed a graph matching technique called the Network Algorithm which allows matching of an unknown input graph to a number of pre-registered model graphs (48; 51; 13; 52). The graph matching algorithm only works on graphs where the vertex label matching is exact. We have extended the algorithm to allow complex, inexact matching at the vertices.

The network algorithm attempts to speed up the intractable graph matching problem by sharing common subgraphs, thereby cutting down the number of matches that have to be performed. The main requirement in image matching is that one often needs to find sub-images within a set of images. In the graph world, this translated to finding subgraphs within a set of model graphs. It is possible that the set of models will all share some subgraphs, whether they be the ones being queried or whether they are merely a property of the dataset. The subgraphs which are shared between the models are then only represented once in a network structure, thereby requiring only one match to find all the models in which that subgraph exists.

Matching is performed by inputting the query graphs into the network which shares the common subgraphs between the set of model graphs. The queries filter through the network leaving instances of subgraphs at the nodes within the network representing those subgraphs. The graph matching at the nodes is all based on the connectivity of (or properties of the edges between) the graph vertices and the properties (or evaluations, or attributes) of the vertices.

The following section gives a simple description of how the network algorithm works, before section ... which gives more detail on the nodes in the network.

4.4.2.1 The Network Algorithm: An Example

To provide an insight into the idea behind the network algorithm we present here a simple example of the network algorithm in use. In this example we will use images as the origin for the graphs we are matching, for this will show how we are using the network algorithm for content based image retrieval. The basic premise is that we will convert two images into

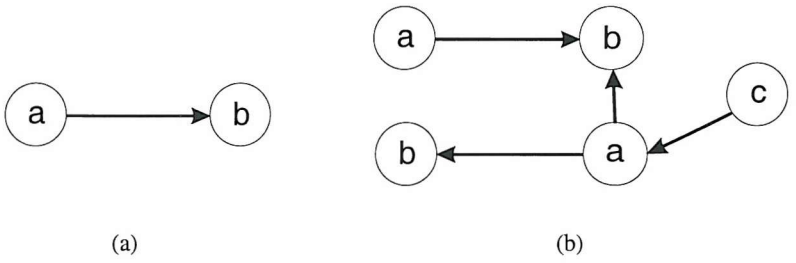


Figure 4.6: 4.6(a) A model graph with two vertices which have different labels, 4.6(b) A model graph with five vertices where some labels are the same.

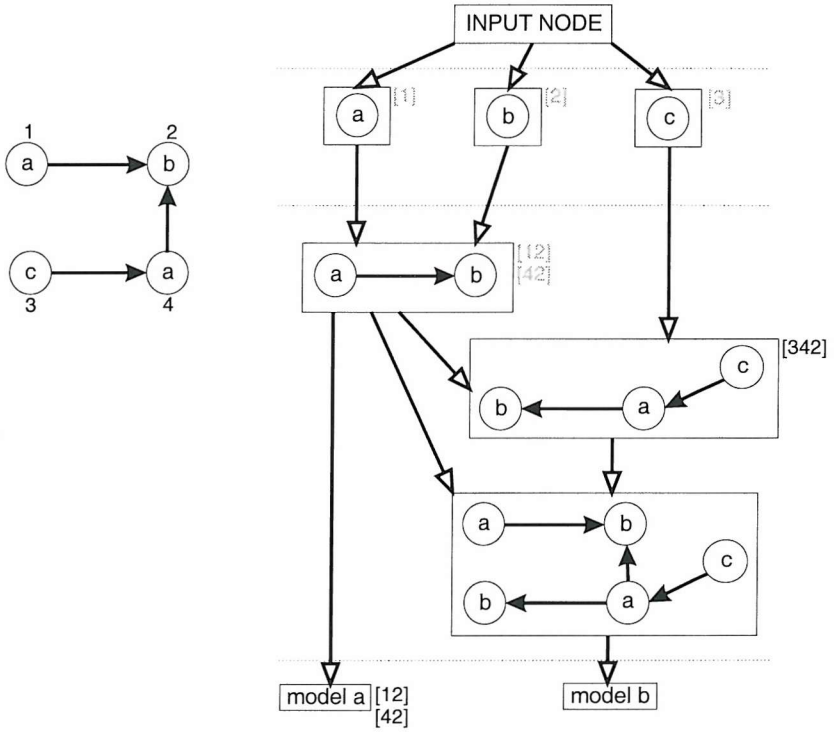
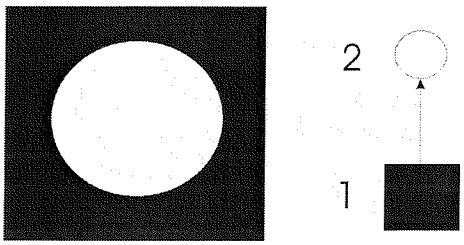


Figure 4.7: A network built with the graphs from Figure 4.6, a) and b). The dotted lines represent the different layers in the network: (from the top), the input node, the l-vertex-checkers, the E-subgraph-checkers, and the m-model-nodes. The graph on the left has been input into the network, and the local memories of the network nodes are shown. The grey memories are ones which are removed as the memories filter through the network, the black memories are those left at completion of the network algorithm. Three subgraph isomorphisms are found between the new input graph, and the two graphs in the database.

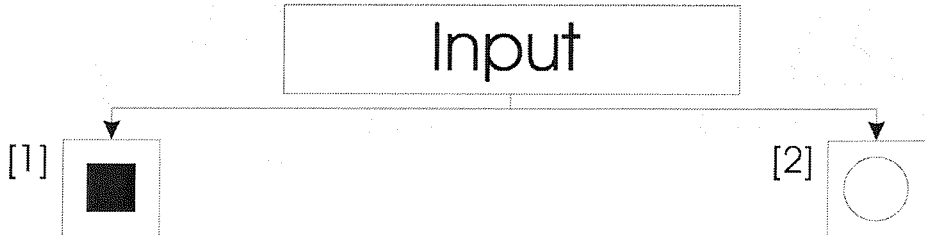
graphs and add the graphs into a network which will share common subgraphs. The network then represents the database, and the graphs in the database represent the image which we will be able to perform matches against.

The following sequence describes the process of adding a graph to a new network, adding a graph to an existing network, and then matching that graph to find isomorphisms taking advantage of the structure of the network to reduce the time for matching.

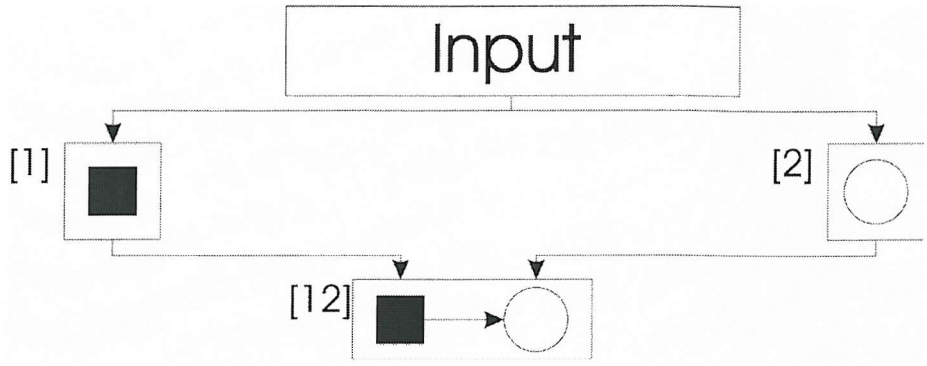


Here we assume that the image shown has been decomposed into the accompanying graph. This is the graph we shall be inserting into the network, and which will become one of the model graphs which we will be able to match against. We first identify the vertices in the graph, by numbering them. This allows us to trace each vertex through the network.

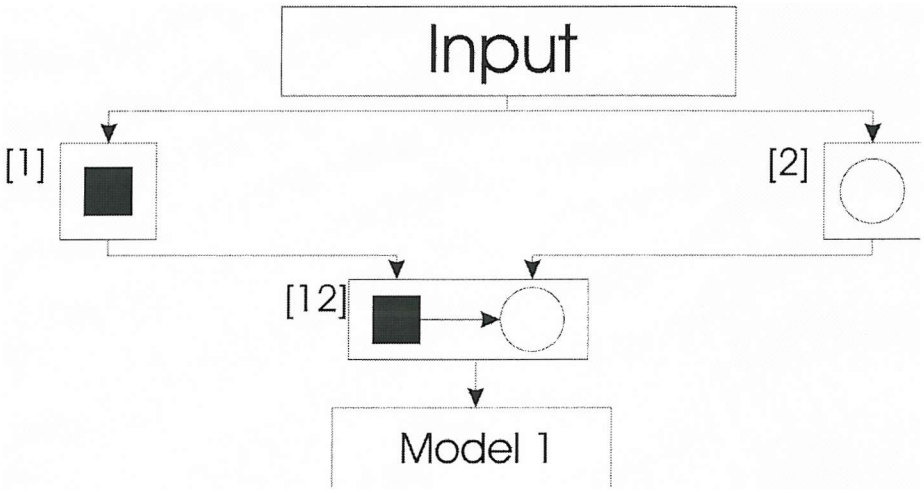
The entrance to a network is always the input node. There is only one of these per network. The network we are building is empty, and so only contains this one node.



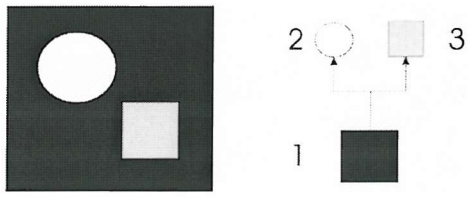
The first layer below the input node consists of an array of nodes, called *IVertexCheckers*, of which there is one of which for every vertex, with different features, in all the models. What constitutes a vertex with different features is based upon the feature vector matchers that are used to match the vertices' features. In this example we are using greylevel to differentiate the vertices, and so we add two IVertexCheckers: one for the black vertex and one for the white vertex. The corresponding vertices from the input graph, which are labelled '1' and '2' are stored at the node to which they match, in the node's *local memory*. At this point there are no more vertices to farm to the input node from the input graph, and a model node has not been created, so it is clear the adding of this input graph has not been completed.



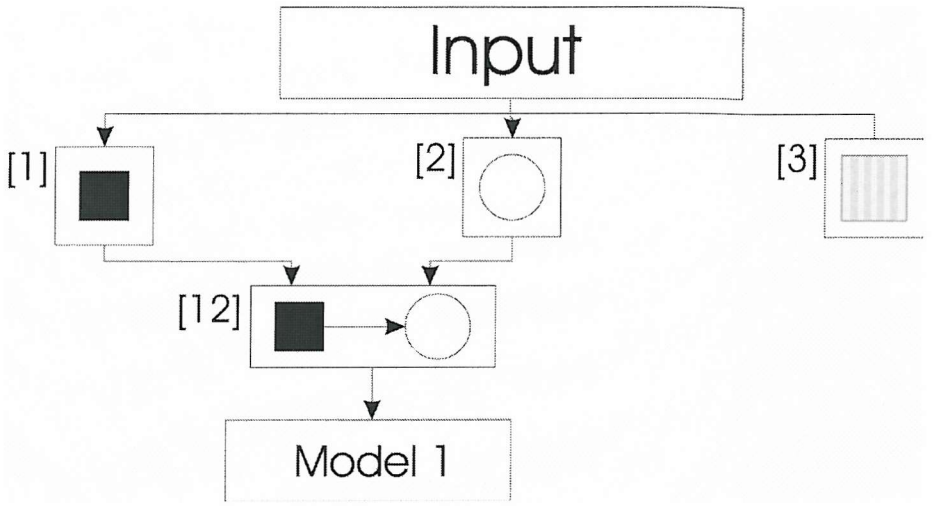
A new network node is created, called an *ESubgraphChecker*, which stores a subgraph of a model. In this case, the instances stored in the lVertexCheckers are used to determine that vertices '1' and '2' are connected in the input graph, therefore this subgraph is stored in this ESubgraphChecker. An ESubgraphChecker only has two inputs, so if there were more instances in lVertexCheckers they would be used in other ESubgraphCheckers.



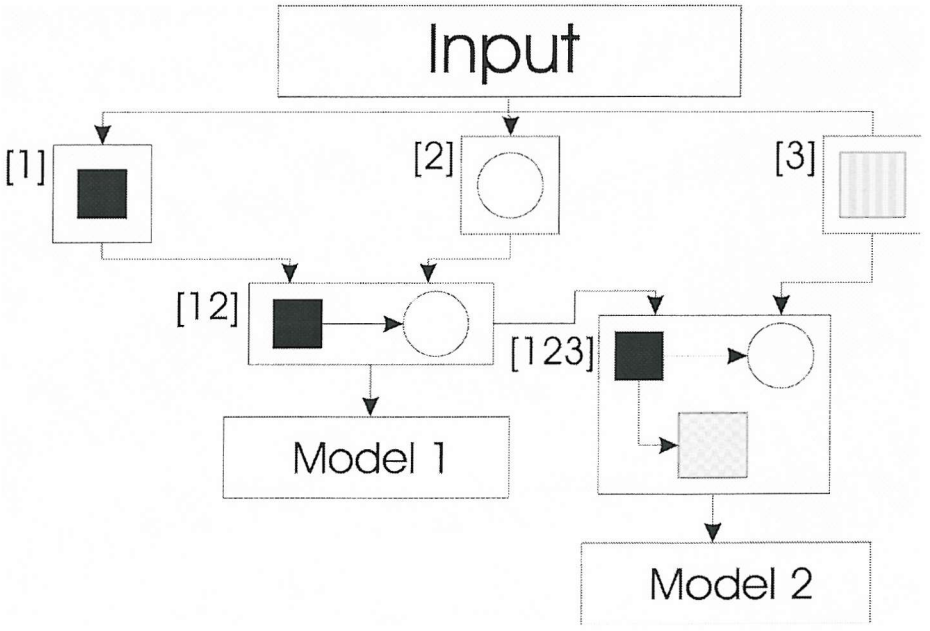
The size of the subgraph instance stored in the ESubgraphChecker's local memory is the same size as the input graph, and it is clear this represents the whole input graph. Therefore we create a model node to represent this input graph. There is only one model node per input graph.



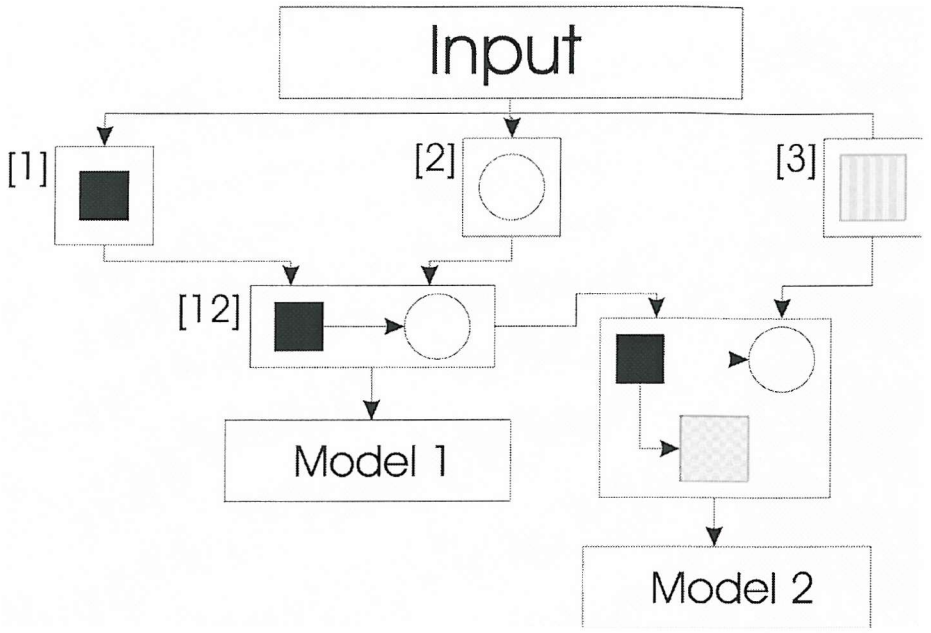
Next, a new image is re-represented as a graph so that we can add this to our existing network. The image clearly has a common subgraph with our previous image (now represented in model node 1), as well as a vertex which is different from those in the network.



Firstly the input graph vertices are farmed out to the existing IVertexCheckers and the appropriate network nodes populate their local memories with the instances of the input graph; this includes a subgraph match at the ESubgraphChecker between vertices '1' and '2'. A new IVertexChecker is generated for the input graph vertex which has no representation in the current network.



Again, there are instances still spread across the network, so it is clear the process is not complete. The two smallest unused instances are taken (the instance in the ESubgraphChecker and the instance in the new IVertexChecker) and checked to see if the instances form a subgraph of the input graph. Because, in this case, the vertices do form a subgraph a new ESubgraphChecker is created to represent this. This new ESubgraphChecker then fully represents the second graph, and is assigned a new model node.



Matching is achieved in a very similar way to the network building process. For example, an attempt to match the first image that was put into the network, with the current network the above situation is reached. Vertices '1' and '2' filter through to the ESubgraphChecker which is linked to the model node 1, as expected, showing we have a graph isomorphism with model 1. However, through the link to the second ESubgraphChecker it is also possible to determine, without any extra calcuation, that there is a subgraph isomorphism in model 2 of the input graph. This is made possible because common subgraphs are shared during the network building stage.

In the next section we explain in greater detail the roles of the various network nodes.

4.4.2.2 The Network Structure

A network is arranged into four layers. Each layer contains a specific type of node with a specific function in the matching process. The top layer consists of a single *Input Node*, the second layer an array of *lVertexCheckers*, the third layer an array of *ESubgraphCheckers* and the final layer consists of a number of *Model Nodes*, one for each of the models in the database. Each of the nodes which represent a subgraph (lVertexCheckers, ESubgraphCheckers, and mModelNodes) contain a memory, called the local memory, which stores the instances of an input graph which match their stored representation.

The following list describes each of the network nodes and their function.

- Input Node

  The input node resides at the top of the network and there is only ever one. It is the entry point into the network, and it receives any inputs which are to be filtered through the network. Its purpose is to farm out instances of individual vertices in the input graph to

the next layer of IVertexCheckers. We required some extra functionality in this node to achieve feature matching, and this is described in section 4.4.3.

- IVertexCheckers

  The second layer of the network consists of an array of IVertexCheckers, which are network nodes representing single vertices. There is an IVertexChecker for every vertex label in all the model graphs. The local memory will contain the vertices from the input graph which match the vertex label stored in this IVertexChecker. Every instance of a vertex from the input graph is sent on to the next layer of ESubgraphCheckers, or possibly mModelNodes (if all models are single vertices).

- ESubgraphCheckers

  The ESubgraphCheckers contain representations of subgraphs which are from at least one model. They only ever have two parents in the network, which are either IVertex-Checkers or other ESubgraphCheckers. During matching they test whether the inputs from the two parents are disjoint, and whether they would join to give the representation which the node contains. If the instances give the same graph as the representation an instance of the represented subgraph exists in the input graph, and it is placed in the local memory. Each composite instance is forwarded to each of the outputs of the network node. The outputs maybe to other ESubgraphCheckers or to the final layer containing mModelNodes.

- mModelNodes

  An mModelNode represents a model which has been placed into the network. It takes its representation and local memory from the single ESubgraphChecker which is its parent. There is one model node for each of the model graphs in the network. If an instance appears at an mModelNode during matching, then a match (graph or subgraph isomorphism) has been found between a database image and the query image.

Building the network is based on the matching algorithm, that is, a match is performed and if no model is found, the network is augmented to include the new model based on the isomorphisms which were found. So, a new model is input into the network, the same way a query graph would be, to find the instances which match subgraphs in the previously entered model nodes (the exact matching process is explained in section 4.4.2.3). At this point new IVertexCheckers may need to be created to take account of new graph vertices which do not already occur in the network. Once the filtering is complete, if the largest instances in the network do not belong to mModelNodes (i.e. the new model did not already exist in the network) new network nodes (ESubgraphCheckers) are created until a new mModelNode is created for the new model. The order in which the new ESubgraphCheckers are created is important to achieving a balanced network, with the most useful setup of distinct subgraphs. The network

algorithm does not generate an ESubgraphChecker for every possible subgraph in a graph, so it is very important to have a balanced network so that as many subgraphs are created as possible.

There are two versions of the matching algorithm. The exact network algorithm finds exact graph monomorphism, and subgraph or graph isomorphisms - where no edits need to be made to the graph to find those matches. The inexact network algorithm finds inexact graph isomorphisms, by applying edits to a distorted input graph, until the input graph matches a model graph with the minimal cost. The following sections describe these two algorithms in more detail.

4.4.2.3 Matching using the Network Algorithm

The exact graph matching algorithm finds graph and subgraph isomorphisms of a query graph within the model graph dataset, where there are no distortions in the graph.

To match a graph with the database, it is used as an input to the network algorithm (NA), and it filters through the network. At the completion of the NA, the network node with the largest instance in its local memory contains the largest subgraph detected in the models. Should a model node be a subgraph isomorphism of the input graph, then the largest subgraph will appear at an mModelNode.

Matching is entirely based on the similarity of the vertex and edge labels. While this may, at first, seem inflexible due to there being no explicit topology matching, it is the most pragmatic way to achieve matching, particularly for applications such as content based image retrieval. It is quite unusual that an application would require the matching of topology only - as opposed to the matching of the topology of certain types of objects. In content based retrieval the properties of the regions stored at the vertices is as important as the topology of the vertices - for example, to match an eye ball, a black region within a white region is correct, whereas a red region in a blue region is not.

The input graph will consist of a number of vertices connected by edges. The vertices will contain information stored in the scale tree nodes. This will include information about the regions - shape, colour, x and y coordinates, and possibly the texture or a histogram.

The graph is initially passed to the processing function of the InputNode, which is the entrance to the network. This node distributes the vertices of the input graph to the IVertex-Checkers based on the labels of the vertices only. This occurs by sending each vertex in turn to the IVertexChecker's processing function.

The IVertexChecker has stored within it a label for the vertex which is represents. It matches this label against the incoming vertex. This is an expensive cost, as it may require a number of feature matches to occur - one for each of the features associated with a vertex label. If the matches are within some thresholds and the vertex is considered to match, it is placed in the local memory of the IVertexChecker. Before processing any more vertices, the IVertexChecker will send the vertex on to the following ESubgraphChecker. If the vertex is

considered to not match the label represented at the IVertexChecker, then it is discarded.

Once all the vertices have been processed the InputNode checks which vertices have been accepted by IVertexCheckers and for those which have not, it creates new IVertexCheckers.

When an ESubgraphChecker receives a vertex from an IVertexChecker it checks in its other parent network node (the node from which the vertex was not received) to see if the vertex passed through will make a union with a vertex in the other parent to create an instance of the subgraph which this ESubgraphChecker represents. More generally, the ESubgraphCheckers check whether an instance graph which is passed through will make a union with an instance graph in the other parent network node to create an instance of the subgraph which it represents. This involves checking the edges between the nodes of each of the instances, and is the most costly part of the network algorithm, other than the feature matching at the nodes. It is here the edge matches take place (feature matching of the edge labels) and also feature matching of the vertices at each end of the edges. To avoid performing excessive feature matching, the vertices can be marked as belonging to a particular IVertexChecker (because an IVertexChecker only represents one label), and the match can then become class instance matching, rather than feature matching.

When an instance is correctly identified in each ESubgraphChecker the instance is passed through to the following node. This may be another ESubgraphChecker which will attempt to match a new graph, or else it may be an mModelNode, in which case the instance is stored as an isomorphism of the model represented by the mModelNode.

In the best case (i.e. the graph or super-graph of a model graph is tested) the instances will flow directly through the graph, ending up at an mModelNode representing the model and input graph. In any other case a model graph will not be found. However, the largest exact subgraph instance will be found and can be detected by checking the network for nodes which contain instances. The largest graph (most number of nodes) represented by a network node which also has at least one instance associated with it will represent the largest exact match.

Inexact matching allows the matching of graphs where the labels do not match (but have the same topologies), where the topology varies, or both. In the next section we explain the matching process using the inexact network algorithm.

4.4.2.4 Matching using the Inexact Network Algorithm

The methods used for creating the input graphs is very likely to be affected by noise in the image. This will cause distortions in the input graph, changing the topology, or the vertex labels of the vertices representing the required object. To overcome this, an inexact network algorithm (INA) has been designed which finds inexact subgraph isomorphisms. The algorithm forces the input graph to be an isomorphism of one of the pre-defined model graphs by label substitution and edge removal, addition or substitution.

An inexact subgraph isomorphism between two graphs G and G' consists of a series of edit

operations Δ applied to G to give a subgraph isomorphism between $\Delta(G)$ and G'. An inexact subgraph isomorphism is optimal if there is no other sequence Δ' such that there is a subgraph isomorphism between $\Delta'(G)$ and G' and the costs of Δ' are lower than those of Δ.

The algorithm begins, as with the NA, by the input node farming the vertices from the input graph to the IVertexCheckers. Every IVertexChecker gets an instance of *every* vertex in the input graph as well as a "dummy" vertex - a vertex which is not in the input graph, and is marked uniquely to identify it as such (usually signified by \$). The labels of the vertices are checked against the label stored at the IVertexChecker and for vertices which do not have the same label, a cost is acquired by that instance of that input vertex - the cost of changing the label. If the input vertex was the dummy vertex, the substitution is equivalent to the deletion of the vertex from the model node and therefore acquires a cost associated with deletion of the vertex.

If all the new instances were passed on, the growth rate would once again become combinatorial, so a new structure is introduced to each network node, called the STOP list. This list hides instances from propagation through the network. We now call the local memory the GO list, which is the list from which instances are allowed to propagate. The decision of which list to place an instance in, is made by the cost of that instance. Costs of more than a particular threshold are put into the STOP list. The first time through the algorithm the threshold is 0 cost. This makes the INA behave in exactly the same way as NA because only instances with zero cost propagate through the network. This stage is then, identical to NA except inexact instances are stored in STOP lists, so we call this algorithm $\tilde{\text{NA}}$.

At each ESubgraphChecker the two incoming instances are checked for disjointedness. If they are disjoint, they are assumed to be an inexact instance of the graph represented by the ESubgraphChecker. However, the cost of the new instance may need to be updated. Firstly, the instance will have at least the cost of the two incoming instances. Secondly, any edit operations that would need to be performed on the edge structure of the new instance have to be included in the cost of the instance. Edges may need to be inserted, deleted, or have their label substituted, in a similar way to the vertices in IVertexCheckers. If the cost of the new instance is above the global network threshold it is added into the STOP list of the ESubgraphChecker, otherwise it is added into the GO list and allowed to propagate through to the next network node.

After $\tilde{\text{NA}}$ has completed, if an instance is available in a model node, then the algorithm stops because an exact instance has been found. However, if no model nodes contain instances, then the algorithm goes back to look at the instances which have acquired a cost through editing. The network node with the lowest cost instance in its STOP list is revisited. To make this step fast the STOP lists with instances are stored in a list called the OPEN list in order of the lowest cost in the STOP lists. The top of the OPEN list is always the network node containing the STOP list with the lowest cost. The new threshold for the network becomes the cost of this instance (currently the lowest cost in the network). The instance is passed on to the next network node in the network in the same manner as with the NA, with that instance being moved

into the GO list. If the network node receiving the instance is unable to find another instance to join with the incoming instance, or the union of the instances gains a cost and is put into the STOP list, then the inexact network algorithm then goes back to the network node containing what is now the lowest cost in the network, sets the threshold, and moves the instance to the GO list to be passed on through the network. This continues until a model node has an instance in it that has a cost lower than all the other instances in the network, and that instance will be the optimal, lowest cost isomorphism of the input graph to the model graphs.

The point at which the edited instance, filtering through the network with INA, has the same number of nodes as the input graph will define the optimal subgraph isomorphism of the input graph within the model graphs which have a path in the network to that point.

4.4.3 Graph Matching for Scale Trees

Conversion of the scale tree representation of an image into a graph representation of an image is a relatively simple task. A tree is simply a specific form of a graph - a directed, non-cyclic graph. Allowing the matching of a scale tree in the network algorithm environment is possible by creating, or trans-coding, a scale-tree representation into the graph representation that the network algorithm uses. Here we describe the extensions we made to the inexact network algorithm to provide scale-tree matching.

However, there are some extra considerations required for the matching of scale-trees with the network algorithm. The most important of these is raised by the label matching that is used to determine vertex equality. This matching is largely ignored in papers on the subject, with vertex labels assumed to be a simple label such as a number or a letter which are easily tested for equality. But, matching scale-trees requires the matching of the nodes within the tree, which become the vertices in a graph, and these contain image features and region descriptions. The matching of region descriptions is carried out by other feature matching modules - such as colour, or shape matching. This implies the network algorithm needs to recognise and allow the ability to match region representations. To avoid the network having to deal with different features and their associated matching technique, feature matchers are programmed as modules conforming to a simple communication interface. A broker is then used to abstract the modules from the network.

The feature broker deals with the brokering of feature matching operations to relevant feature matching modules on behalf of the network algorithm. When the network algorithm requires two vertex labels to be matched they are passed to the feature broker which knows of all the possible feature modules available to perform matches. The labels are distributed from the broker to the active modules which perform a match if they are able to do so. An aggregated score or an aggregated boolean equality result is returned from the feature broker to the network.

Features from the various feature matching methods will return various scores based on

their view of the image feature (the shape, texture, or colour, for example). To get an overall feature score from the modules, the set of returned scores needs to be aggregated. Feature score or equality aggregation can be a very complex subject and is often application dependant, however, the feature broker uses a very simple algorithm to return aggregate results. Feature modules generally would return an unbounded distance measure, which would be incomparable between features, but to allow aggregation in this manner, the feature modules have to return normalised scores (for example, between 0 and 1) otherwise any particular feature module may dominate the results. The broker then aggregates the scores using the mean score of the individual scores returned by each feature module. This bounds the final aggregate score to the same range as any individual normalised feature score. So, for N feature modules, each returning a score s_i from a matching process, an aggregated score, s, is given by:

$$s = \frac{1}{N} \sum_{i=0}^{N} s_i \tag{4.15}$$

where, s_i is the normalised score from feature module i, and N is the total number of feature modules. Normalisation could not be achieved by using any kind of rank-based score normalisation, because each region representation is matched in isolation. Weighted scores could also be used to vary the domination of certain types of feature. Weights could be given to each feature module prior to initiating a query. Each score from a module will be pre-multiplied by the weighting coefficient to give a weighted score:

$$s = \frac{1}{N} \sum_{i=0}^{N} w_i s_i \tag{4.16}$$

where w_i is a weight coefficient for the feature module i.

Equality matches are also aggregated in the most simple way: a match occurs between two vertex labels only if there is a consensus among the feature modules that the labels match. Each feature module is responsible for deciding whether a particular distance score between two vertex labels is a match or not. This thresholding could also be application dependant and altered by the user prior to engaging a query.

Thresholding is necessary in the exact Network Algorithm, where a label has to be classified as the same or completely different to another label to determine whether the instance is propagated. Deciding what value the threshold should take is application dependent, because which of the features is the most important largely depends on the subject matter. Thresholding could take place pre- or post-score aggregation. Aggregating the scores, then thresholding is simple to achieve, but one becomes limited to how the thresholds can be altered for a query. Altering weighting of the score of a particular feature score, then subsequently thresholding is equivalent to altering the thresholds for each feature beforehand, then combining the truth values, but will affect an extra processing cost on the feature modules for score normalisation before aggregation. This latter technique was used successfully in MAVIS-2 (19). Thresholding each of the individual features affords more flexibility, although requires some boolean

logic operations on the resulting truth values. We use this technique because it allows each feature module to define the truth values for a pair of features.

It is worth noting that the more feature modules we use to match the image labels, the more discriminating of labels the network becomes. This means, that the more feature modules that are activated, the more IVertexCheckers will be generated at the first layer of the network, due to the labels being considered less similar because of the better discrimination. This, of course, means that there are less likely to be shared subgraphs! It is therefore important, that the feature modules are not overly discriminating between labels, otherwise there will only ever be a single IVertexChecker for every label ever encountered by the network, and there will never be shared subgraphs. Feature modules should ideally be rotation, scale and translation (RST) invariant, so that a similar region, under these transforms, will be considered the same.

The problem with feature matching is that it is relatively expensive, and there are a number of speed-ups we can make by ensuring that the feature matching is performed as little as possible. During the running of the network algorithm labels are matched regularly to test for label equality, and edge equality and if the labels needed to be matched each time with a feature matching algorithm the matching process would take a very long time. What is required is a global representation of a feature that can be referred to, and matched inexpensively, rather than the actual feature representation.

The network algorithm already supplies such structures in the form of the IVertexCheckers. So, we generate an IVertexChecker-graph representation of the input graph, where each node in the input graph is now represented by the IVertexChecker to which its label matches. Equality of IVertexCheckers is a matter of address equality, and not feature based, and can be done at speed. The extra step to create the IVertexChecker-Graph is performed in the input node of the network, which transforms the input graph into an IVertexChecker-graph before farming the new IVertexChecker-graph vertices to the IVertexCheckers. The main problem with using this technique is that information is lost - the information about the scale and location of the label. For example, an IVertexChecker may represent the set of labels that are black squares. By replacing the actual labels with IVertexChecker references, all black squares, of any scale, and from anywhere within the image, are represented by a single representation. This means that an instance that arrives at the bottom of the network is not accurately reconstructible, unless we store some extra information. At the mModelNodes we store the locations of each of the IVertexGraph-vertices so that sub-tree matches can be back-traced to the mModelNode and highlighted for user feedback.

Another speed-up we can make is to introduce a feature cache which stores which vertex labels have already been matched and what their score was. This avoids the network algorithm having to perform expensive feature matching when the result has already been calculated.

The inexact graph matching allows us to find which models contain the input graph, or which models are contained within the input graph. However, because of the common subgraph sharing, certain attributes of the vertices of the subgraphs are lost (purposely to allow them to

become common); the location of one subgraph is almost certain to be different to that subgraph in a different model and likely to be at a different scale, however, they will be represented by exactly the same subgraph representation. This property makes it very awkward to detect where in the model graphs the matching subgraphs from the input node occur. To allow location retrieval the ESubgraph checkers are augmented with a mapping that maps the original image file, from which the graph was constructed, onto a set of a mappings from subgraph node to a rectangular area (location and size) containing the region the node represents. This allows the application to find where the sub-image represented by any partial subgraph match is located in the model graphs in which that match appears. This is imperative to ensure we are getting correct matches.

4.4.4 Inexact Graph Matching with score filtering

The inexact graph matching, proposed by Messmer and Bunke, allows the matching of input graphs to model graphs when the input graph is a mutation of a model graph. One of the problems with this system is that the costs that are applied to the edits made on the input graph are fixed by the user before the matching takes place. For example, a vertex substitution may incur a cost of 1, and an edge insertion a cost of 2. That is, when vertex label substitution occurs the cost that is incurred is not related to the label that is being substituted or the label it is being substituted with. This means a label being replaced by a very similar label will incur the same cost as a label being replaced by a completely different label, and this means that instances which are very unlikely to produce optimal results will be filtered through the network.

The extension we have implemented is to use the feature distance, produced by the feature matching modules, as the cost associated with label substitution. Substituting a label by a similar label will then incur a cost less than that of substituting a label with a dissimilar label. Both vertex and edge label substitution can incur costs in this way.

Edge addition and deletion can also incur costs based on the similarity of the vertices at the end of the edges, or the similarity between the edge labels, again. This score-based filtering is less important than vertex label substitution score filtering. Edge deletion means there was an edge in the input graph that was not in the model graph and needs to be deleted.

The feature scores are produced once again by an aggregate score-based on the distribution of the match over a number of feature matching modules, as described in the previous section. The mean of the weighted scores gives the final score.

The introduction of score filtering will still generate optimal results, because only the instances with the lowest cost, smallest distance in feature terms, will appear at the model nodes.

Experimentation with inexact graph matching with score filtering for scale-trees is given in chapter 5.

4.5 Summary

In this chapter we have described in detail novel work towards a content-based image retrieval approach that is based on the topology of images with mining provided by graph matching. The hope is that this system moves some way towards providing a semantic based approach to image searching. In the following chapters we describe experiments carried out on a prototype system implementing this retrieval approach, and we described a method for extending the image decomposition and matching algorithm to provide a semantic layer into which the data is linked.

Chapter 5

Evaluation of Matching Algorithms

5.1 Introduction

In the previous chapter we described our work on extending the graph matching algorithm to provide matching of scale-trees generated from a sieve decompositions of images. In this chapter we show experiments we have performed on a prototype content based retrieval system that incorporates the scale-tree decomposition and inexact graph matching, for both the topology of the graph and the vertex label representations.

It is important that, as well as being indexed, to increase the speed of the match the system itself is implemented efficiently. Graph matching, in particular, is known to be NP-complete in the worst case, and with thousands of nodes in a possible scale-tree this has to be considered.

This section explains the way graph matching might be used alongside feature matching to provide content based retrieval based upon the prototype we have implemented. We begin by showing that our implementation of the sieve decomposition gives the correct results, before showing some simple graph matches using exact node matching. In section 5.3.2 we show the pruning of the scale-trees, and in section 5.3.3 the graph matching of scale-trees is shown. Experiments will be presented showing how a graph-based image representation affords new searching possibilities. In section 5.4 a discussion is presented on the experiments, which also details some of the problems with our particular prototype implementation of the system.

5.2 Configuration

Configuring the prototype system is important to the overall success of the experimentation. The amounts of tree pruning, feature matchers, and the feature matcher's attributes all con-

tribute to the match.

The system is set up with five low level features to allow matching of regions. They are all relatively basic features but should be adequate for matching of the simple regions decomposed using the sieve. They are chosen mainly for their speed as well as their relative inability to differentiate very similar regions which aids common subgraph sharing.

The simplest of the metrics measures the monochrome value associated with the region. The regions are generally homogeneous and therefore only have one brightness value. The absolute difference between the greyscale value of region i, v_i, and the greyscale value of region j, v_j, returns the score (normalised to a value between 0 and 1). So, given the maximum greyscale value of N, the score is given by:

$$s = \frac{v_i - v_j}{N} \tag{5.1}$$

The colour difference module returns the colour distance in HSV colour space between the two representative colours of the regions. Each region has associated with it a colour - the region's mean colour. These colours are compared and the distance normalised by the maximum possible distance in the colour space. Given two colour values (H_i, S_i, V_i) and (H_j, S_j, V_j), the distance is given by the Euclidean distance in colour space:

$$s = \sqrt{(S_i cosH_i - S_j cosH_j)^2 + (S_i sinH_i - S_j sinH_j)^2 + (V_i - V_j)^2)} \tag{5.2}$$

This can be extended to allow full histogram matching at each node. Although this may be unnecessary because the regions are generally very homogeneous. This matching technique uses the QBIC quadratic matching technique. The QBIC histogram matching technique is described in 2.2.1.3.

Although Hu moments were also implemented for matching in the tests, a very simple shape matcher was used which provided a very basic method for distinguishing between shapes. It uses the ratio between the area of the enclosing rectangle, $w \times h$, and the actual area of the region, a. The difference is squared to accentuate the range.

$$s = \left(\frac{w_1 h_1}{a_1} - \frac{w_2 h_2}{a_2} \right)^2 \tag{5.3}$$

A rectangular region will fill the enclosing rectangle (and have a ratio of close to 1), whereas a circle would fill it less. The smallest value would be achieved by a diagonal line. This is simple measure is invariant to scale (and unconstrained ratio scaling, i.e. different width and height scaling), quarter rotations (or very small rotations), and translation. However, w and h are dependant on the camera orientation, and this measure is not fully rotation invariant. For the equality match we can statistically generate a threshold value. A circle is to be considered different to a square. If the square has an area w^2, a circle fitting within that square will have area $\pi\left(\frac{1}{2}w\right)^2$. The score for the circle will therefore be $\frac{\pi\left(\frac{1}{2}w\right)^2}{w^2}$. If we take w to be 1, the area

ratio becomes $\frac{1}{4}\pi = 0.79$. This implies the threshold should be no more than 0.21 to ensure circles and squares are differentiated.

The string matcher is used for matching those nodes which are not regions. The edge labels are stored as string labels and require a matching algorithm. The score is 0 or 1 depending on the boolean equality match performed by a string compare.

Tests will be made to show the differences that different algorithms apply to the match, and how the match is altered with multiple matching algorithms activated and to test that the more algorithms that are used the more discriminating the results will become - and the bigger the network should be.

The pruning algorithm is switchable so that the tests can be performed with and without pruning. The pruning variables are alterable before a match takes place, and during the experiments notes will be made of the values of the variables when pruning is activated.

The network is configurable to allow inexact matching to be performed, or not, depending on the application. At first, the network algorithm will be used in its exact mode, and then experiments on the inexact network will take place.

The pre-filtering of images will be disabled for most of these tests, but in the instances where it is activated, a note will be given, stating the value of quantisation used to pre-filter the image. This is only required in the more complex images.

The implementation we give is based on a Java platform. This allows it to be highly portable among hardware platforms and provides a good base for prototyping. Also, the final intention at the beginning of the project was to integrate this with the image search platform called Mavis-2 (see section 6.1.5) which required the implementation to be Java. The downside to using Java as prototyping platform was the issue of speed - Java is interpreted and run within a virtual machine meaning that it runs much slower than natively compiled code. This issue is of less concern here as the implementation is primarily a proof of concept.

5.3 Matching Experiments

To ensure the integrity of the system we need to apply experiments at each level so that we can be sure that the final result is due to all parts working correctly. This section will provide descriptions and examples of how the system provides content based retrieval, and provide some examples of its abilities.

We have a test harness that allows us to examine the results of tests, and visualise the scale-trees. It also allows adjustment of the parameters used within the network algorithm. Figure 5.1 shows the control panel of the harness.

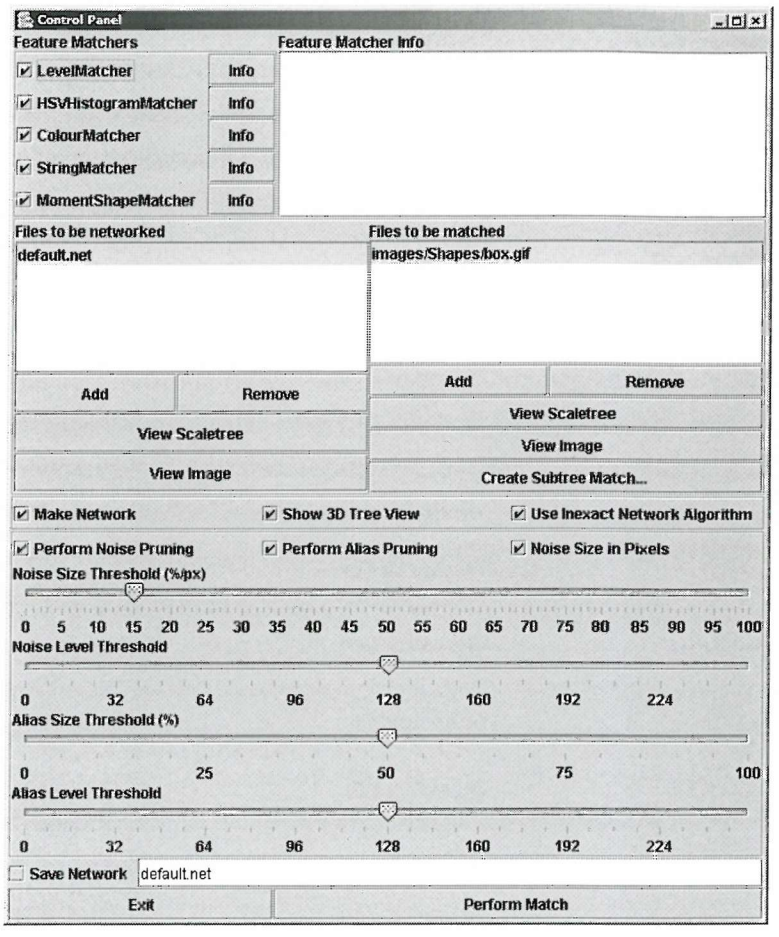


Figure 5.1: The simple test harness.

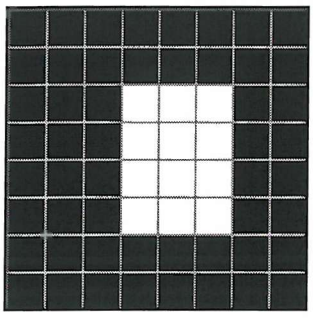
5.3.1 Creating a tree from the granularity domain

The sieve is the basis of the image decomposition algorithm and to test this we use some images to test that our implementation works in the way the theory and the original tests state they should.

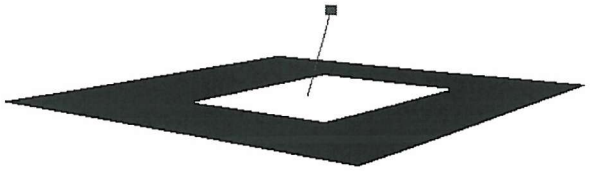
First, the simplest types of images are tested, because incorrect results can easily be detected. Small block images are the correct type of image to test the basic functionality of the sieve, in particular, to test that in this implementation of the sieve the regions are merged correctly - into the correct region, and at the correct scale.

A single block at a specific scale on a black background shows that the region labelling, and scale removal works correctly.

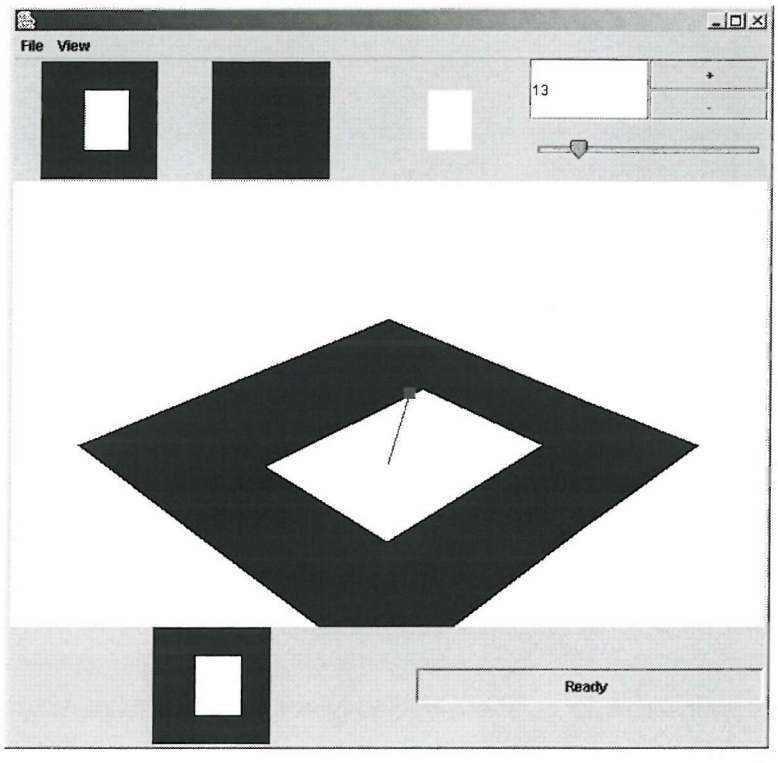
Figure 5.2 shows the first simple test case for the sieve. A white box on a black background is decomposed into a scale-tree with 2 single nodes - one representing the white box, and another representing a black box (the background with the white box merged into it). The white box has an area of scale 12, as shown in figure 5.2(a). The 3D view of the tree is shown



(a)



(b)



(c)

Figure 5.2: Sieve test: Simple image to ensure the region labelling and region merging took place correctly. Figure 5.2(c) shows the scale-tree visualisation harness.

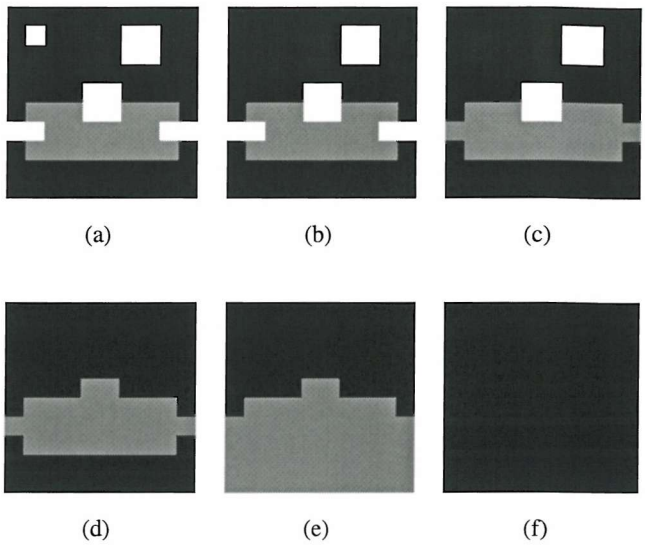


Figure 5.3: The region merging throughout the sieve process on a simple block image (scales 0,4,8,16,88,200).

in figure 5.2(b). As in all the following figures, scale is further away from the base of the 3D graph, with the base representing the size of the image. The node representing the white box is offset due to the white box being offset in the image, and the nodes are placed in 3D space at the centroid of the region they represent. Figure 5.2(c) shows the test harness interface. The three images at the top are the original image, the image rebuilt from the scale tree up to the scale given by the slider, and the third image shows the granules that have been removed (effectively the scale tree down from the leaves to that scale). At scale 13 a new granule appears in that image and disappears from the rebuilt image showing the region of scale 12 is removed at scale 13. This basic test shows that, for the simplest of images, the sieve works.

A more complex image is used to test that the sieve merges regions correctly and that the scale tree is built correctly. It is still a simple block image so that errors can be easily detected.

With a more complex image, we first ensure that the sieve decomposition still works correctly. Figure 5.3 shows the decomposition of this more complex, yet still another simple block, image. The scales shown are scale 0 (the original image), followed by sieves of scale 5, 9, 17, 89, and 201 — the scales at which granules are removed. The granules of size 16, at the left and right of the image, are correctly removed at scale 17 and merged into the region below, which is the large grey area in the centre. It is possible to see that at scale 89, a region that should possibly be considered the background, is merged into the object, as it is a minima of scale 88 in the image. This is expected as, indeed, it is a minima. This begs the question whether the image should always be invisibly padded around the edges with zero values. The problem arises in deciding what should be considered a zero value, so that regions of the image

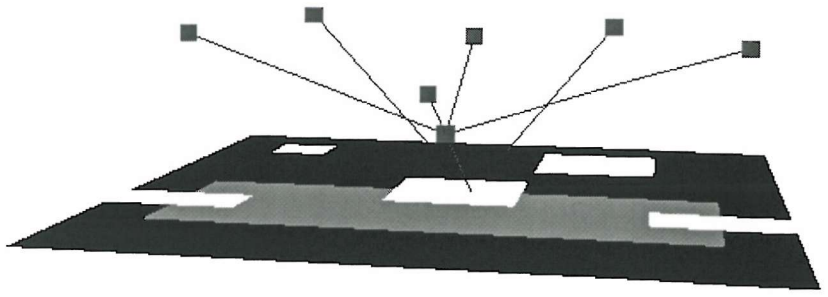


Figure 5.4: A simple block image and its scale-tree

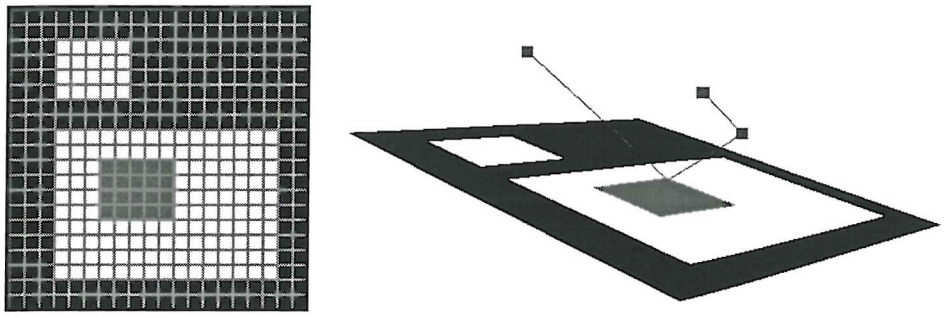
that are relevant are given priority, that is, it is only prudent to pad with zeros if the background of the image is made up of zero values. Images that are photographs could feasibly have no discernible background colour and should probably not be padded at all. However, with this anomaly considered, the sieve merging operations appears to work correctly for this image with no padding. Because the intention is to use this system with a variety of images, we cannot rely on using padding, and therefore the remainder of the experiments run with no padding.

The tree for this image should have 8 nodes. The nodes for the two regions at the left and right should be joined by a branch in the scale tree to the node representing the larger grey area. Also, the anomalous background node will also be joined by a branch to this node. All other nodes merge into the final background region, and they will all be connected by branches to the node representing the background region. Figure 5.4 shows the tree for this image, and the connections described above.

The scale-tree should be built correctly for images with large amounts of pixels as well as small ones. Indeed, the same image scaled to a larger number of pixels should return the same scale-tree (assuming no anti-aliasing was applied during the re-scale). Tests show the trees are identical despite the larger image being of a different aspect ratio and containing nearly 100 times the pixels, which is as expected. The downside is that there is longer execution time for the region labelling stage due to the extra pixels which need to be searched and labelled.

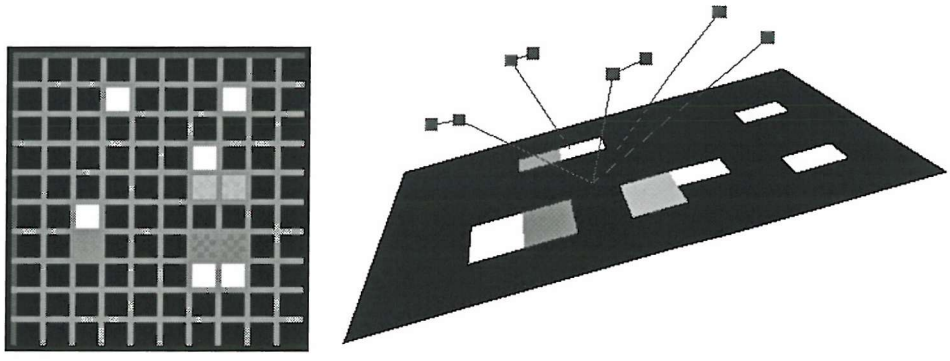
Other simple block images are tested, also, and the correct decompositions and scale-trees are built, giving reasonable evidence that the sieve and scale-tree creation algorithms are working correctly for, at least, simple block images. Figures 5.5 to 5.7 show some examples of other tests designed to test the sieve and scale-tree, where the first image is the original image with an overlaid grid to delineate the pixels, and the second image is the constructed scale-tree angled to show the important features in the tree.

Figure 5.7 tests the merging of surrounding non-extrema blocks. The white block on the right of the image gets merged into the grey block surrounding it. Similarly the three white



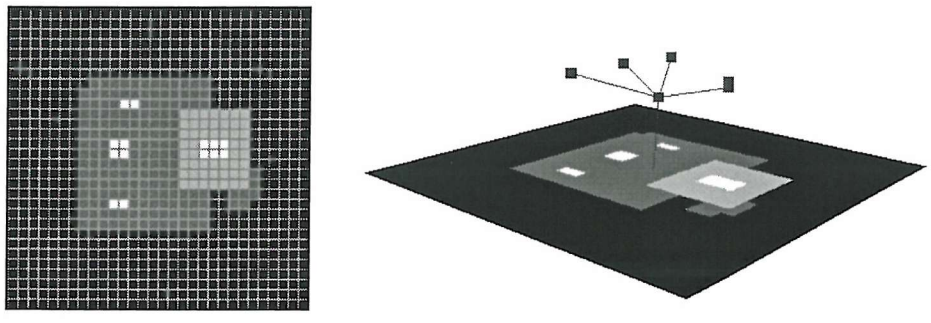
(a) (b)

Figure 5.5: A simple block image and its scale tree.



(a) (b)

Figure 5.6: A simple block image and its scale tree.



(a) (b)

Figure 5.7: A simple block image and its scale tree. This tests the merging of surrounding non-extrema blocks.

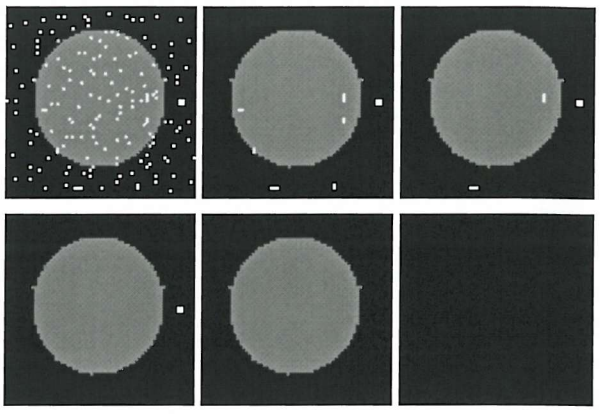


Figure 5.8: The region merging throughout the sieve process on a simple image with 4% salt and pepper noise (scales 0, 1, 2, 3, 4, 1575).

blocks get merged to the darker grey block which surround them. The grey block is eventually merged into the darker grey block. However, a separate darker grey block to the far right will also be merged in at this point. This block will not have a scale-tree node created for it, for it is not a minimum nor a maximum, but merely gets merged into the final grey block. Many similar tests were carried out to ensure other such situations were handled correctly.

5.3.2 Noise and Blurring

Once it is clear the sieve decomposition is correctly removing regions from simple images, and the scale-tree is being correctly generated, we can be fairly confident that for more complex images the decomposition will be carried out correctly. To ensure it is, we conduct some tests on slightly more complex images which contain image features associated with inaccuracies in capture devices. The inaccuracies in the circuitry of capture devices mutate the signal slightly creating noise or blurring in the images. These features will create extra nodes within the scale-tree. These tests ensure the scale-tree is generated consistently.

Figure 5.8 shows an image with a large amount (4%, 169 pixels) of salt and pepper noise (digital noise) on a very simple image. A property of the sieve makes this noise very easy to remove. Because salt and pepper noise is caused by spikes in the signal the regions created by this noise are always extrema - black or white (hence the name salt and pepper). It is possible to see that a sieve of scale 1 removes most of this noise and almost restores the image to its original noise-free state. These granules that are removed by the sieve will appear as leaves in the tree at scale 1. It is therefore a matter of pruning the tree for nodes which are of scale 1, to remove these regions. From the example, it is possible to see that a final noise-free image is generated only at a sieve of scale 4, and the tree would therefore need to be pruned for leaves below scale 4, to return this particular image back to the original. Looking closely at the final noise-free image reveals that the shape of the circle is very slightly altered where the

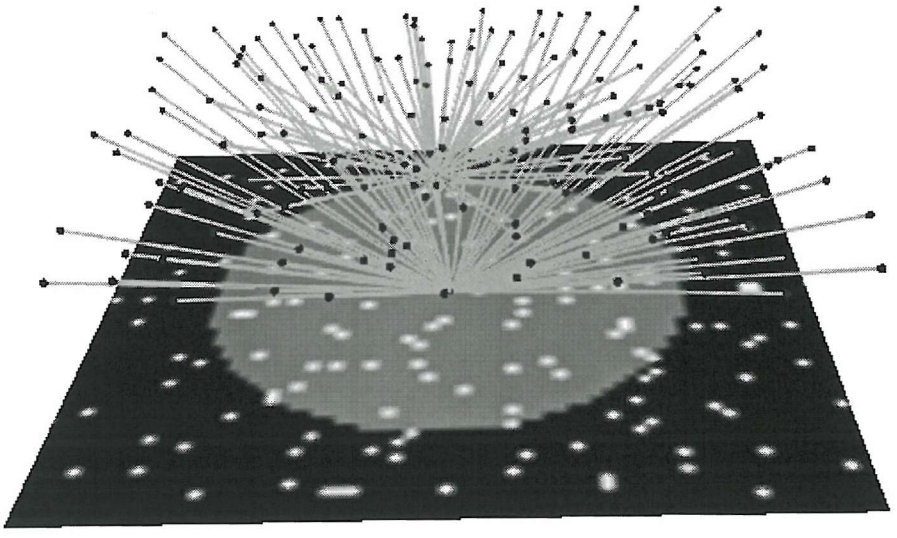


Figure 5.9: Speckled noise is easily removed from images using high-pass sieve - as the noise is nearly always extrema and of high frequency. Here the noise merges to the main elliptical region and to the background region causing two cone shaped structures.

merging criteria of the sieve merged the granules, at, but outside, the edge of the circle, to the next lowest region which was the circle, thereby creating small errors in the contour of the shape. Providing our feature matching modules are robust to small changes in shape, these errors should not adversely affect the matching of trees and even if it does, the nearest shapes will filter through the network first.

Figure 5.10 shows a simple image with some random amplitude noise added in Photoshop, similar to the noise that affects CCD devices generated by small variances in the electric currents flowing through the circuits, and similar to the compression artefacts caused by JPEG compression. This noise is not clearly defined like the spike generated noise, and is more difficult to remove. It affects the whole image, and therefore not able to be fully removed without affecting the image detail level. The noise manifests itself as small variances in the brightness level, and colour, of a pixel in the image creating many small extrema in the image. When viewed from normal viewing distances, the image will look unaffected, but it is possible to see the variances if looking closely. If we sieve to different scales, we see the sieve slowly remove the noise from the image, but it will also begin to remove detail. The problem is that some different variances of the actual region colour will merge into larger regions each of slightly different colours/levels giving a large number of subtree elements. It is only near the removal of the object from the image that the object is cleaned of noise - it is now effectively a silhouette of the object. Again, the noise reduction causes small alterations in the shape of the object when noise falls near the edge of the object. In tree-pruning terms, this type of noise is much more difficult to remove. As seen with the sieve, much of the noise is removed by scale 20,

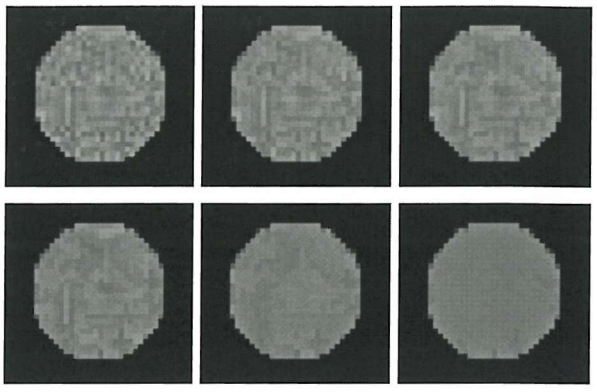


Figure 5.10: The region merging throughout the sieve process on a simple image with random amplitude noise added in Photoshop (scales 0, 2, 3, 5, 20, 500).

and a similar noise pruning effort to the spike generated noise removal can be used - simply removing leaves, or subtrees, from the scale-tree where the root node has a scale of less than 20, for example.

Aliasing is another image feature which is generated by capture devices being unable to accurately reproduce the detail of the real world. Aliasing is when the details in the image are fuzzy due to the resolving power of the capture device being less than that needed to accurately reproduce the detail. It is most noticeable on edges within the image which when viewed closely seem to straddle a number of pixels and the actual location of the edge is difficult to locate. This makes the size of the object difficult to measure. Figure 5.11 shows a closeup of an blurred edge within an image showing blurring on the edge between the object and the background, as well as blurring on details within the object. The blurring causes the particular object or detail to be apparent over a number of scales which is expected with an object whose size is not accurately measurable. Figure 5.12 shows a simple blurred object being sieved at various scales, and how the object is removed over a number of scales. The object appears to change over a number of scales, but which is considered the correct scale is an issue that we discussed in section 4.3.3 - Pruning Scale Trees.

Aliasing causes chains of nodes in an image. This is because each maximum is merged to the next lowest region - which is a very small distance away, and almost of the same size, due to the blurred edges. This causes the scale tree node to appear at almost the same location, and at almost the same scale as the parent region. Figure 5.13 shows an example of a deliberately blurred image to illustrate the effect. The tree contains a long chain of nodes where each node represents a region very similar to the region of its parent and child. Figure 5.14 shows a photographic image and its scale tree. Notice in the scale tree how the two large regions (nearer to the root of the tree) are composed of chains of nodes showing that the object is blurred and is appearing across a number of scales. The sieve of this image can be seen in Figure 4.1 and it can be seen how the object is persistent, even when all details are removed from within it

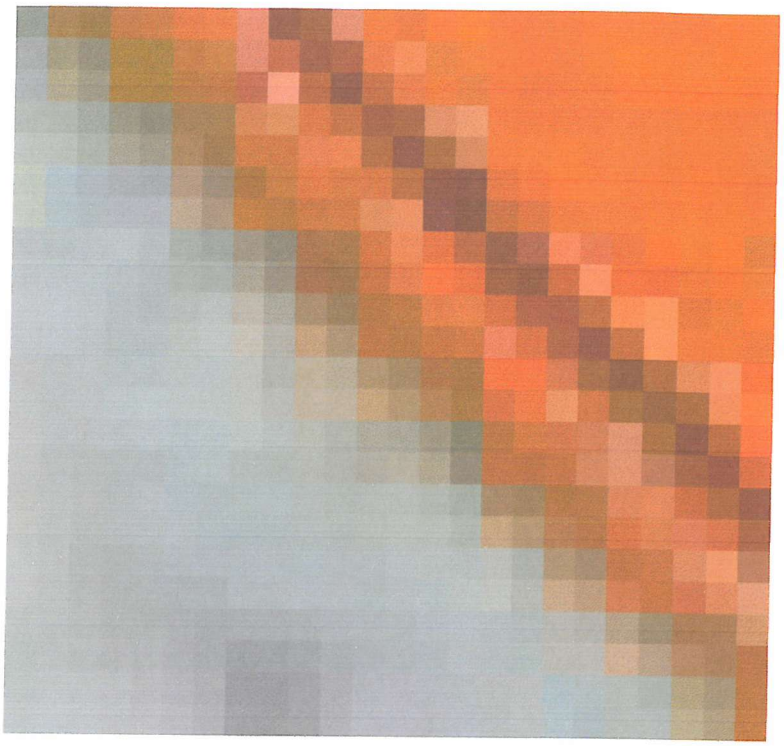


Figure 5.11: Blurred edges within images caused by the resolving power of the capture device

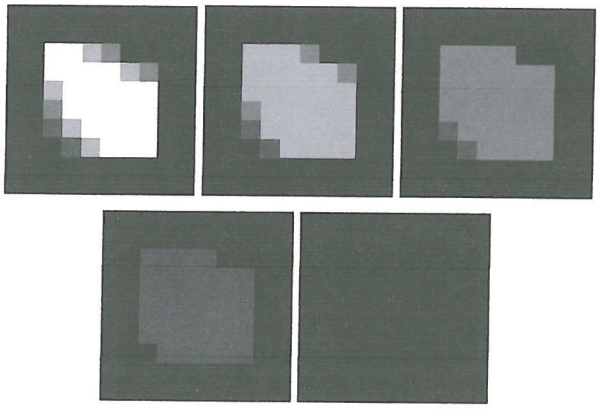


Figure 5.12: The region merging throughout the sieve process on a simple blurred object (scales 0, 24, 28, 32, 34).

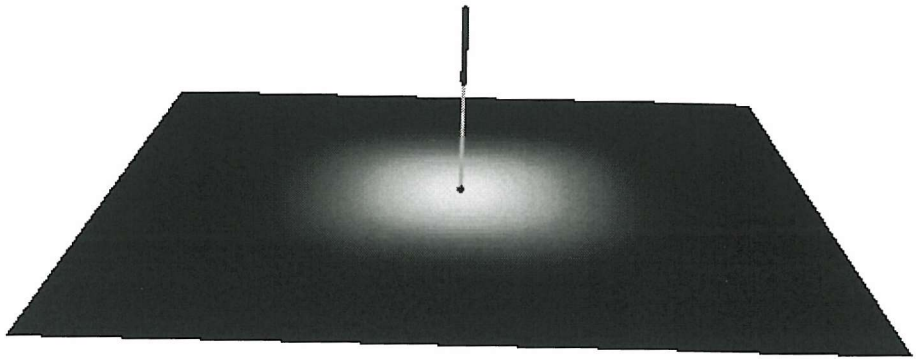
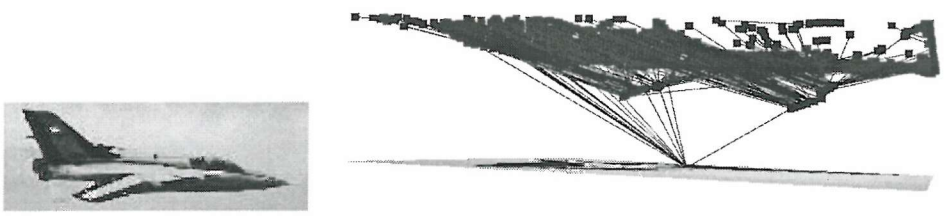


Figure 5.13: Blurred images cause long chains of nodes in the scale tree.



(a) (b)

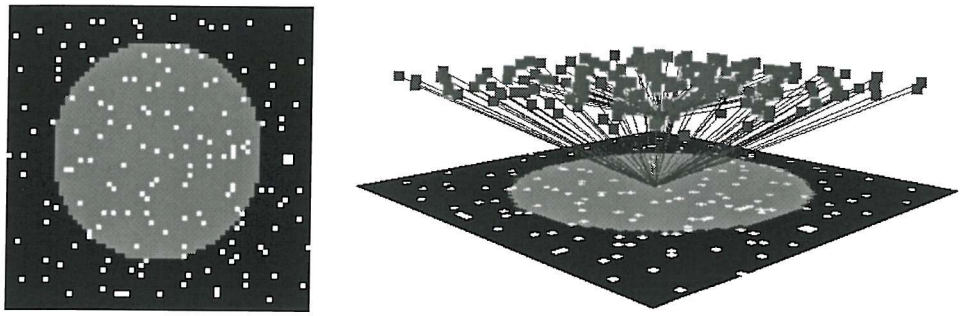
Figure 5.14: An image and its 2300-node scale-tree, showing blurring near the root of the tree.

(between scale 7000 in figure 4.1(d) and scale 10000 in figure 4.1(e)).

The pruning of these trees is important for making the graph matching practicable. Unfortunately, despite advances in graph matching techniques, the speed of the algorithms is still, at best, exponential. Because the blur and noise features are relatively easily detected within the images, we can use the techniques described in section 4.3.3 to remove nodes from the scale-tree which are contributing little to the overall semantics of the image.

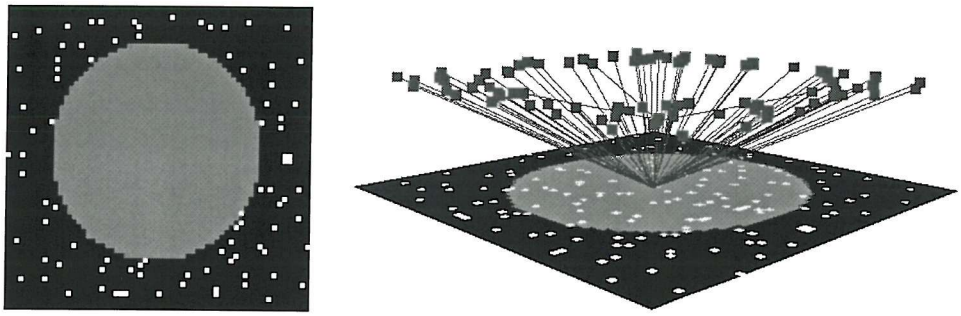
Firstly, we use our simple test-case images shown above to test the tree pruning algorithm.

Figure 5.15 show a simple circle image which has "salt and pepper" noise added to it to simulate signal spikes. Its scale tree has 165 nodes. The noise pruning algorithm is executed over the tree to reduce the number of nodes by attempting to remove the noise from the image. The configuration is set to remove noise regions which are less than or equal to 4 pixels in size (from our original test using the sieve) and to consider them as noise if they are less than or equal to 256 brightness levels (i.e. the whole brightness range) away from the region to which it is connected. This last condition ensures that the salt and pepper noise is considered noise and not details. Figures 5.15(a) and 5.15(b) show the original image and tree. Using the default settings for the noise pruning algorithm, the noise that is greater than 128 levels apart would not be removed (see figures 5.15(c) and 5.15(d)) but using the maximum 256 level threshold



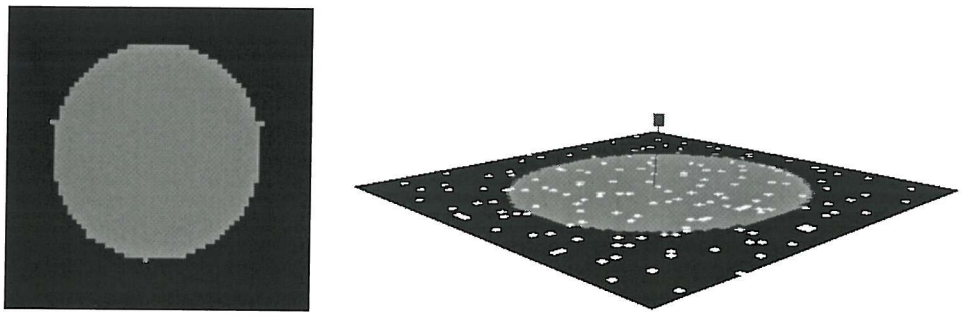
(a) (b)

Original image with its 164 node tree



(c) (d)

Pruned with the default values, the very extreme noise is not removed.



(e) (f)

Using the maximum 256 level threshold, all noise is removed.

Figure 5.15: An image with salt and pepper noise, and its scale tree, and after noise pruning of the tree with 128 and 256 level threshold.

all the noise is removed in figures 5.15(e) and 5.15(f). The final tree has just 7 nodes where there is slight blurring around the circle.

Salt and pepper noise is not very common, and is very unlikely to occur to such an extent as on the previous example. To test a more likely situation we have added some false Gaussian noise to the same image of the circle. This type of noise is much less conspicuous than salt and pepper noise and therefore more difficult to remove. The original scale tree has 2099 nodes (see figure 5.16). The noise pruning algorithm is configured such that the size of region considered noise is less than 15 pixels and, because of the type of noise we could set the brightness range to a small value, in this case it is 128, to avoid removing other details that are not, in fact, noise. Figures 5.16(a) and 5.16(b) show the original image and its tree, and figures 5.16(c) and 5.16(d) show the pruned image along with the pruned tree. In this case the final pruned tree still has 65 nodes. This is due to the merging process. The noise is not merged into a region representing the circle, as it is with the salt and pepper noise, but is merged into a region slightly larger than the noise. This region in turn is then merged into other regions until finally there is a single region representing the circle or the background. However, the regions further along this process will no longer meet the requirements for the region to be noise (e.g. less than 15 pixels) and so will not be pruned from the tree. This effect can be demonstrated by increasing the pixel threshold under which regions are considered to be noise. If the threshold is increased to double its original amount, of 15, to 30, the number of nodes in the final scale tree reduced from 65 to 52. This scale tree and its resulting image is shown in figures 5.16(e) and 5.16(f).

We can test the method on more complex images to consider the result of the noise pruning algorithm on details within images. Figure 5.17(a) shows an image of a sign. The dithering of the graphic has introduced some noise around the edges of the image, which will be removable by the noise pruning algorithm without altering the overall structure of the image. Figure 5.17(c) and 5.17(d) shows the pruned image and its tree. Note how the pruned image is almost indistinguishable from the original, but for the removal of the noise. The details in this image are large enough to be untouched by noise pruning.

Figure 5.18(a) shows another image of a sign. Again it is a graphic and not a photograph and therefore only has dither noise, however, some details in the image (the lettering) are small. Figure 5.18(c) and 5.18(d) shows a final representation of the image and tree after the noise pruning has taken place. Parts of the details of the image (the lettering) have been removed during the noise process. This means the object (the text) has been altered, and may not match well with other similar objects, due to changes in shape and colour. The problem with images with very small details is that the distinction between noise and details becomes blurred. In this case, we can remove the noise, and not the text, by setting the range at which noise can appear, and the size at which noise can appear to be very small. However, this would, most likely, not be the best setting for every image.

The inability for the noise pruner to differentiate between noise and small image details

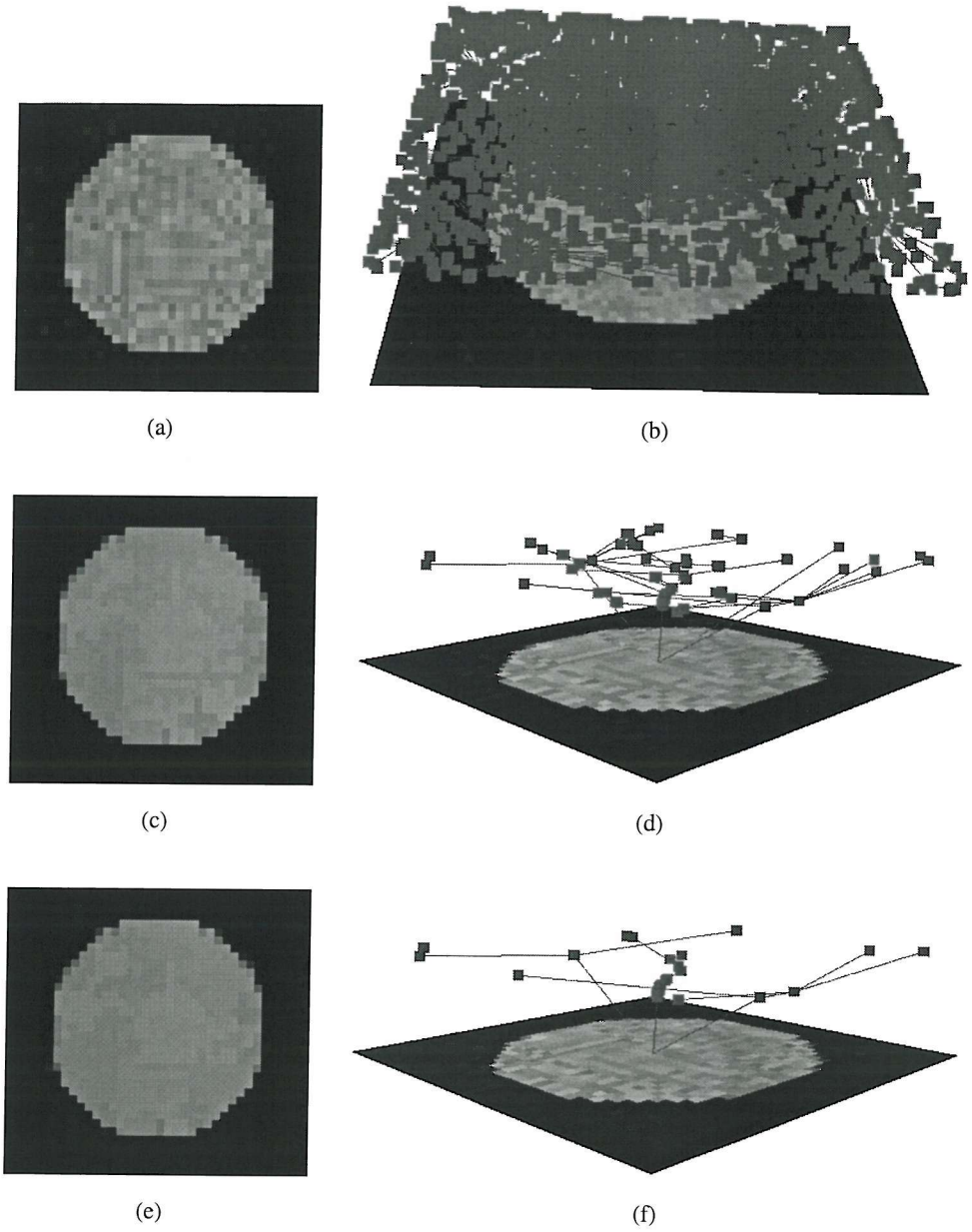
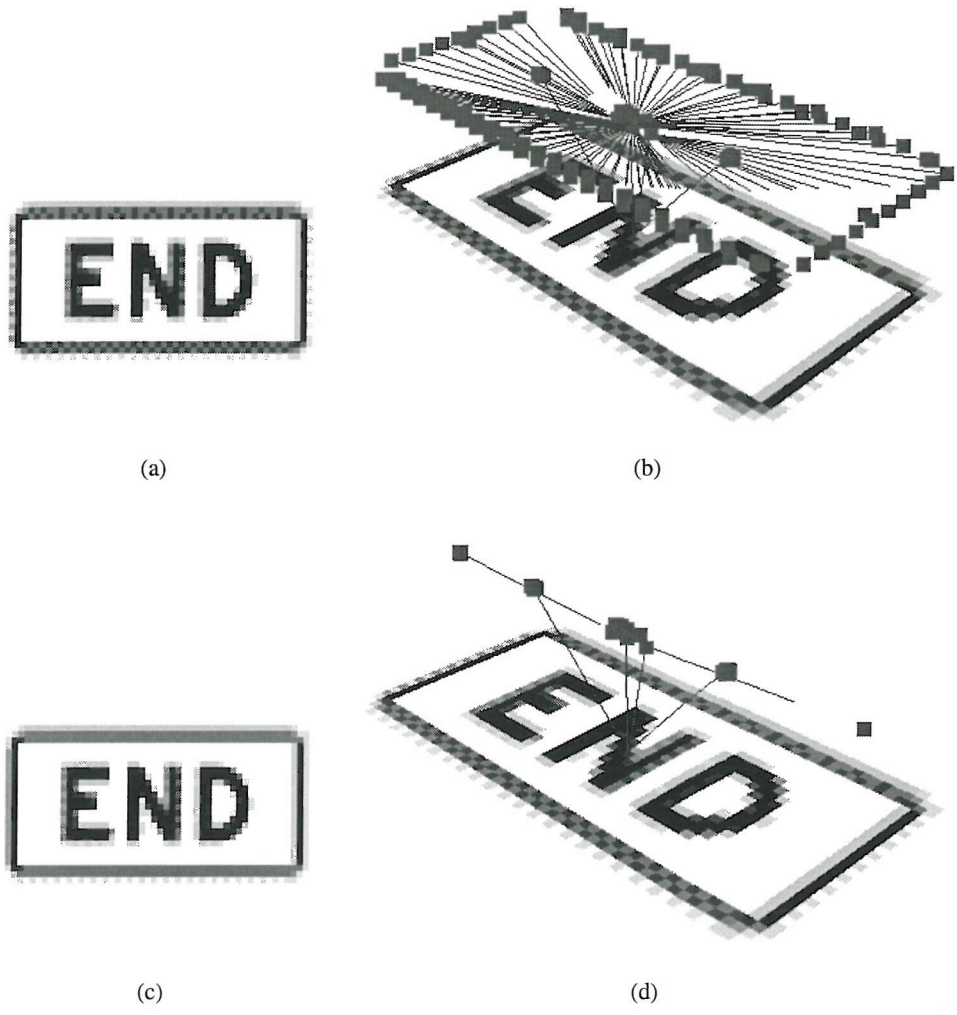


Figure 5.16: An image with Gaussian noise, and its scale tree, and after noise pruning of the tree with a threshold of 15, and 30 pixels.



(a) (b)

(c) (d)

Figure 5.17: This image has noise introduced by dithering. Default value noise pruning removes the dither effectively.

makes it clear that if we are required to prune images, to ensure that the number of vertices in the graph are compatible with the graph matching algorithm, we are going to have to compromise detail finding; that is, small details are going to be removed from images.

We can test the blur pruning similarly, by purposely blurring some well-defined objects and examining the result of the pruned tree. We blur the image using a Gaussian blurring kernel - exactly the sort that would be used for Gaussian scale-space decomposition. The larger the kernel that is used to blur the image, the longer the chain of nodes will be in the scale-tree.

Figure 5.19(a) shows a basic image. The small square in the centre of the image undergoes a Gaussian blurring with a kernel of size 5 to give the image in figure 5.19(b). The tree of the blurred image is shown in figure 5.19(d). Notice the short chain of nodes which occupy the

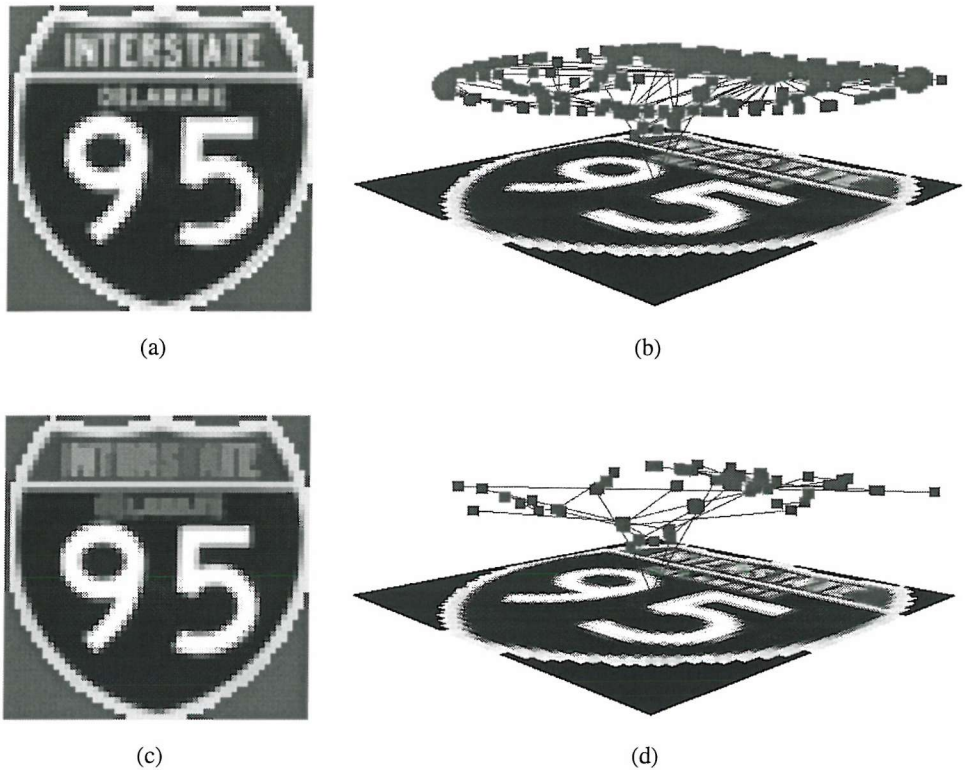


Figure 5.18: This image, which has small detail, loses detail when the noise pruner is set at a threshold to remove enough noise from the large regions so that they become single nodes in the tree.

feature space between scale 5 and scale 1568. These are pruned using the blur pruner. Each region is only a small difference in size and colour from the region that is its parent and so it is marked to be pruned. The tree is traversed until this condition does not hold and the marked nodes are removed. This gives the final tree in figure 5.19(e) and the reconstructed image in figure 5.19(c). The unpruned tree has 158 nodes and the pruned tree has 3 nodes. The size of the region effectively becomes larger (compare figure 5.19(a) to figure 5.19(c)) due to the blurring altering the region's properties to be that of the maximum level. This ensures that scale-space causality is not broken, and that no "gaps" appear in images with multiple objects after pruning. For example, figure 5.20(a) shows an image containing a set of shapes. They are blurred using a Gaussian kernel of size 5. With this blurring the shapes become intertwined. Figure 5.20(b) shows the blurred image, and figure 5.20(d) shows the scale-tree of this image. Using the standard blur pruning parameters, the tree is reduced from 324 nodes to 9 nodes. The tree is shown in figure 5.20(e) along with the reconstructed image in figure 5.20(c). Notice how the objects are now separate, and lack the blurring and are approximately the same as the original shapes. The surrounding region in the pruned reconstruction is the outer-most, almost invisible region caused by the blurring which, again, has been made visible by the alteration of

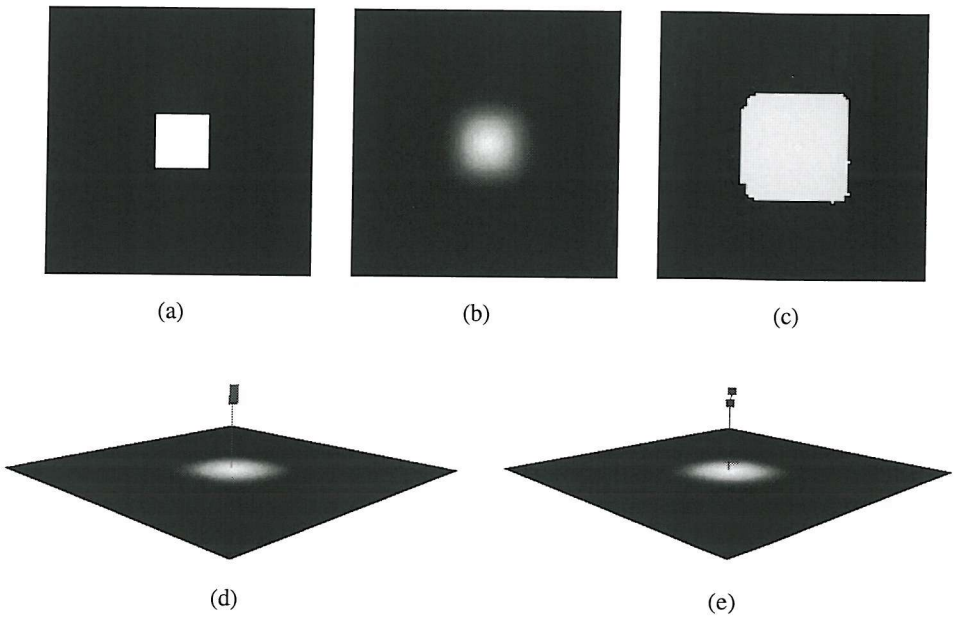


Figure 5.19: This simple image undergoes blurring which is then pruned from the tree

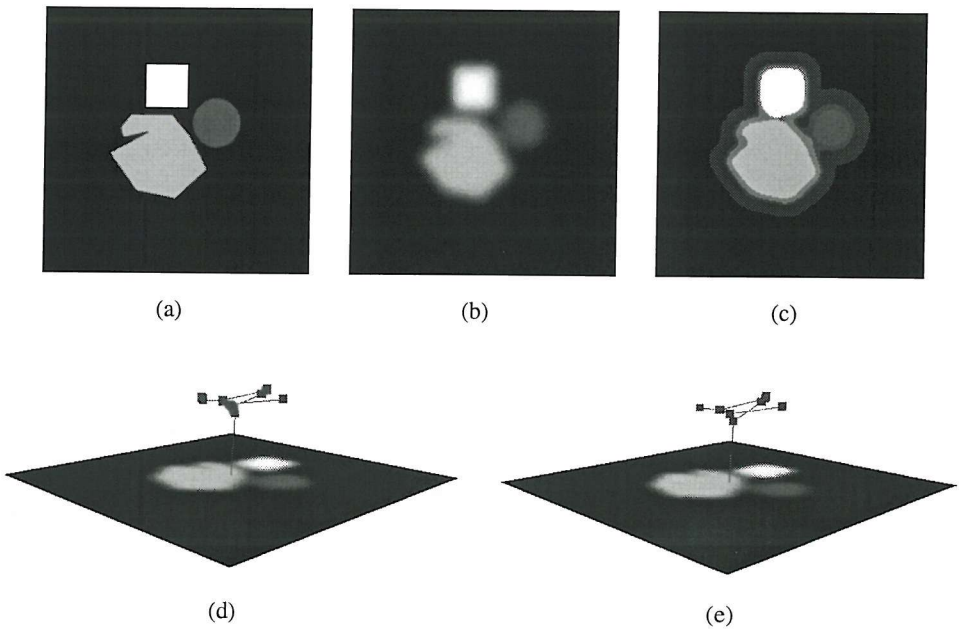


Figure 5.20: Here, the objects merge under blurring, but are clearly separate after blur pruning.

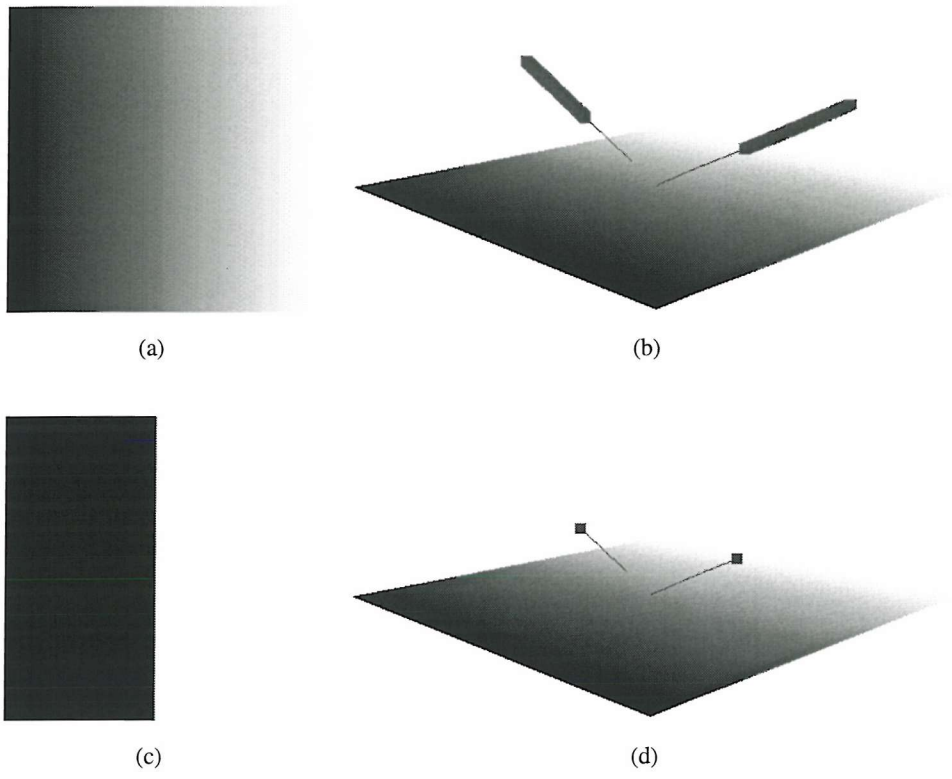
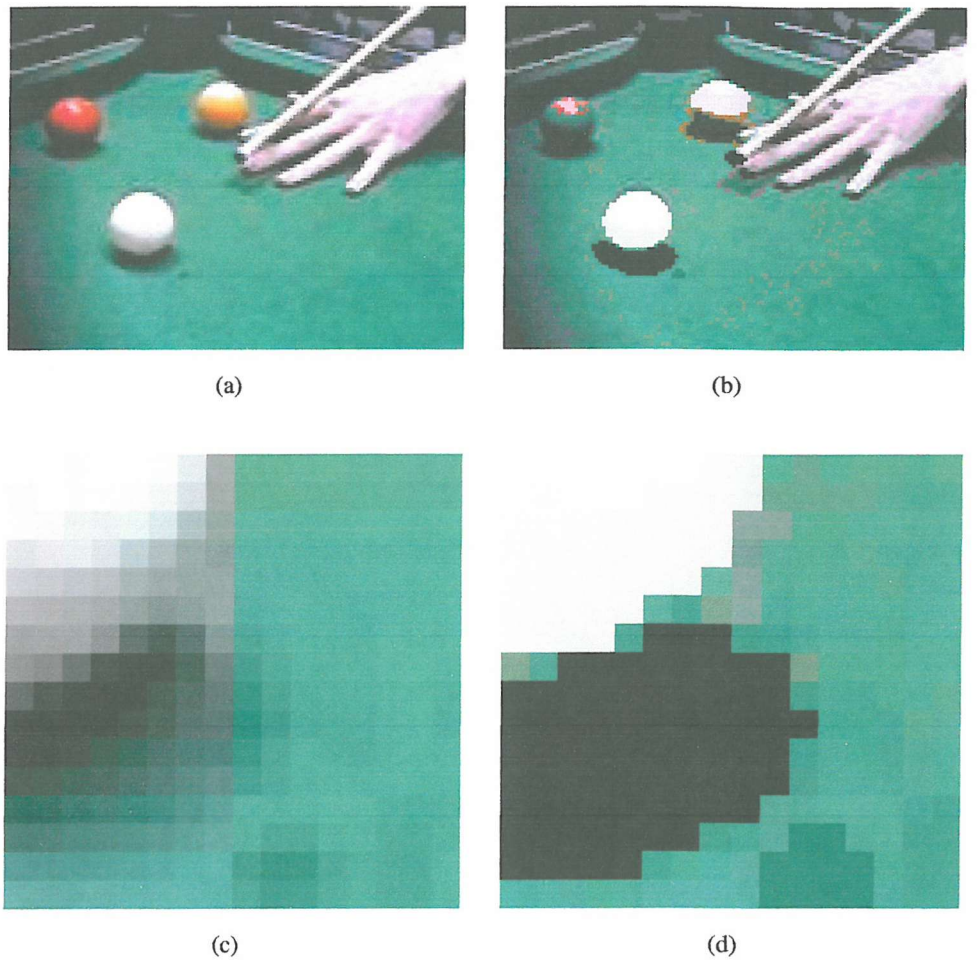


Figure 5.21: A smooth shadow on an object appears as a gradient, which, like blurring, produces long chains of nodes that can be pruned by the blur pruner.

the region properties during the pruning.

Alias pruning is also useful for removing other image-based features. For example, shading over a smooth object will generate chains of nodes in the tree as blurring would, and for the same regions. For example, figure 5.21(a) and figure 5.21(b) shows a gradient pattern and its tree which, run through the blur pruning mechanism, reduces the image to two individual colours and the tree to two node (figures 5.21(c) and 5.21(d)). In the next example we use the blur pruning to remove both shading and blurring.

Figure 5.22(a) shows a photograph of a some snooker balls. This image contains many possibilities for simplification of the tree. As it stands, the image has 4421 nodes. Most of these are noise generated through the capture process, and the compression of the image. Some are image details, but details that are not pertinent to the matching process (for example, the nap of the baize). Ideally, we would want the balls, the hand and the table to each have their own scale-tree node (not necessarily a leaf node), which would aid in our goal to build an object scene-tree. Applying the tree pruner to remove blurring from this image results in the image shown in figure 5.22(b). The most obvious pruning which takes place is due to the shading on the balls. Figure 5.22(c) shows a closeup of the original image, and figure 5.22(d) shows



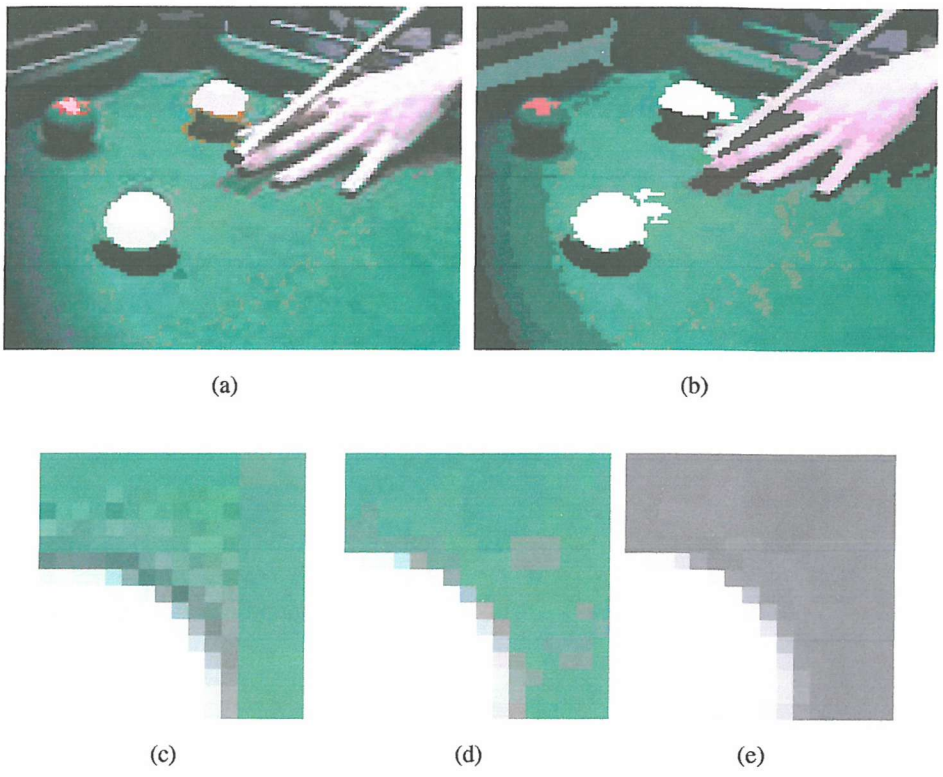
(a) (b)

(c) (d)

Figure 5.22: Alias pruning of a snooker photograph

a closeup of the pruned image, showing that the blurring around the edge of the balls due to digitisation has also been removed. However, the pruned tree still has 2230 nodes, which is far too many to be useful in our graph matching environment. Increasing the fuzziness of the blur thresholding (to remove more nodes) lowers the node count to 2044.

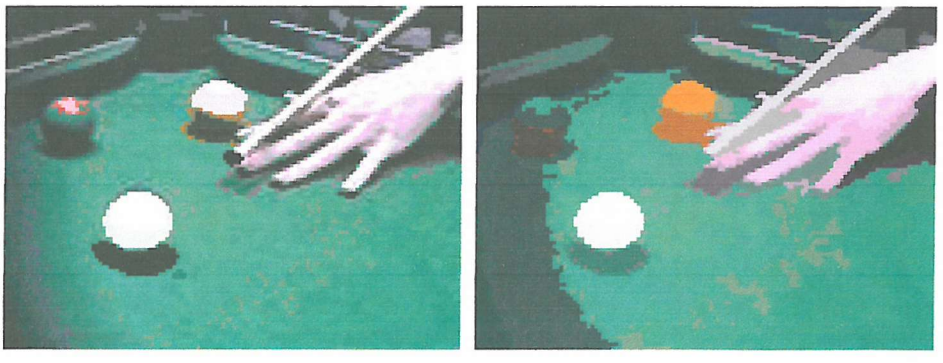
We can use a combination of noise and blur pruning to lower the node count of trees more. For example, if we noise-prune the tree before we blur prune the tree, we are not only removing more nodes, but more likely to get regions which can be merged during blur pruning. For example, using just blur pruning the snooker photograph's tree was reduced to 2230 nodes, and with just noise pruning the number of nodes was reduced to 1394. Using both the methods together our tree was reduced to 142 nodes, however, the consequences to the reconstructed image (which gives an indication of the accuracy of the regions within the scale-tree) are great. In figure 5.23(c) the noise that is effectively providing a barrier between the ball and the cloth



(a) (b)

(c) (d) (e)

Figure 5.23: Noise pruning before blur pruning can cause problems. Figure 5.23(a) shows blur pruning only, at default levels, and figure 5.23(b) shows noise and blur pruning of the image at default levels. Figure 5.23(c) shows a closeup of figure 5.23(a), while figures 5.23(d) and 5.23(e) show a closeup of the image (in colour and greyscale) of the image after noise pruning and before blur pruning.



(a) (b)

Figure 5.24: Setting the noise size threshold high to remove many details, and the blur size threshold low to avoid regions merging that should not be, we can get a better segmentation of the image.

(caused by JPEG artefacting) allows the items to remain separate during blur pruning. However, if we prune this noise before blur pruning, the cloth becomes a constant level colour, at approximately the same level as the slight blurring that is evident around the edge of the ball (see figures 5.23(d) and 5.23(e)). This means that when the tree is pruned for blurring, the region outside of the ball is considered within the size and brightness threshold and is merged to the ball, creating the large extraneous connected region and altering the shape of the ball object (compare figures 5.23(a) and 5.23(b)). Altering the thresholds so that the regions considered to be blurring are only within 2% of the size of the father region allows the pruning algorithm to disregard the cloth as part of the ball object. However, this also increases the node count back to 524, because other parts of the image are no longer pruned. This is, of course, partly due to the way in which the scale-tree is built only upon the greyscale values of the image. The green cloth and the blurring around the edge of the ball all lie very close to each other in the greyscale feature space. With the blurring threshold at 2% the number of nodes increased dramatically, but the number can be cut down to 142 nodes by increasing the size of region considered to be noise (figure 5.24), which will, of course, remove many details from the image, which in this case is required, but would, most likely, not be commutable to other images.

The noise and blur pruning algorithms are very efficient and even on large trees (in excess of 4000 nodes) the pruning time is negligible, and under 1 second even in the prototype.

The region labelling that occurs before the sieve decomposition is an ideal time to cut the amount of calculations that need to be done in the sieve and scale-tree calculation and pruning. There are a few "pre-pruning" methods we can use in this stage. For example, we can remove noise at this position as we know the size of the regions. Pruning of regions of area less that a certain threshold at this point will not be equivalent to pruning the tree for regions of a certain threshold, because the tree contains scale-space (merged) regions, whereas the region labelling structure contains actual image regions. To achieve this, however, the removal of the region will require it to be merged into the appropriate region to avoid leaving a "hole" in the representation of the image. Because this is outside of the scale-tree representation, whether the appropriate region is the next highest level region in the surrounding area, or the next lowest, is a matter of taste. Indeed, the area where this will most affect the scale-tree will probably be at the edges of objects in the blurring. Noise pruning of the tree does not remove these regions because they are not extrema - they are between a background region and an object region. Pruning before this restriction is implied will allow these regions to be removed saving execution time at the costly decomposition and tree building stage, and it is therefore pragmatic to assume that they should be merged into the object which, although not in all cases, is usually a maximum.

The other method for pre-pruning the image is to quantise the image into a set number of bands. This has the effect of grouping together regions which are similar in feature space, and separating those which are more different. This means there will be less regions overall in the image before scale-space decomposition starts, both increasing the speed of the decomposition and the number of final nodes in the scale-tree.

Both the above pre-pruning methods need to be used with care. As with tree-pruning, and maybe even more so, pertinent objects could be removed from the image and because we are removing regions directly from the image, there is the possibility that the properties of regions could be altered in such a way that they are no longer representative of the object.

Despite many attempts to lower the node count of high-node-count images, the results have been mixed. On some images we are able to get a relatively good final segmentation that should aid our graph matching process, but others are problematic. This is partly to do with the sieve mechanism generating scale-space for greyscale images only which causes conflicts between two regions which, although obviously separate in the colour image, become one in greyscale. However, pruning the image and scale-tree to a low number of nodes is very important to achieve image matching using a graph matching paradigm. Graph matching is still an active research topic because it has obvious advantages for structural matching, however, it is still also a highly complex subject and the best algorithms are still slow. The next section describes matching these scale trees.

5.3.3 The Graph Matching

Section 4.4 details the algorithm used to perform graph and subgraph isomorphism testing that we use to prototype a topology based matching algorithm. Here we demonstrate this algorithm and the advantages and disadvantages of using the network based algorithm.

A copy of the graph matching toolkit written originally by Bruno Messmer is available from his homepage (46). The toolkit is written in C and includes the ability to run a number of different graph matching techniques on graphs represented in ASCII, with labels consisting of strings. Unfortunately, this toolkit proved too difficult to adapt to our image matching needs, mainly because it was written in C, which would have been time consuming to integrate with our intended test bed, Mavis-2, because of lack of documentation for the toolkit, and because the feature matching modules we had available were written in Java. It was decided to implement a native Java version of the graph matching toolkit that would allow for vertex labels to be of any type (not just strings) although this raised a few problems that solutions were designed for (for example, the IVertexChecker-Graph described in section 4.4.3). Our implementation was tested for correctness using strings against the GUB toolkit and against examples in their papers, for example (48; 51; 13; 52).

Figure 5.25 gives an illustration of a network built using features, rather than strings. The structure of the network is identical, except that the vertex labels are now representing regions from nodes within the scale tree.

To test the graph matching, as with the sieve process, we first use very simple images, before building up to more complex images. Also we have two querying methods to consider. The only way to query the network is by example, however the example may be a whole image or a part of an image. The first method is where the user finds an image to which they want

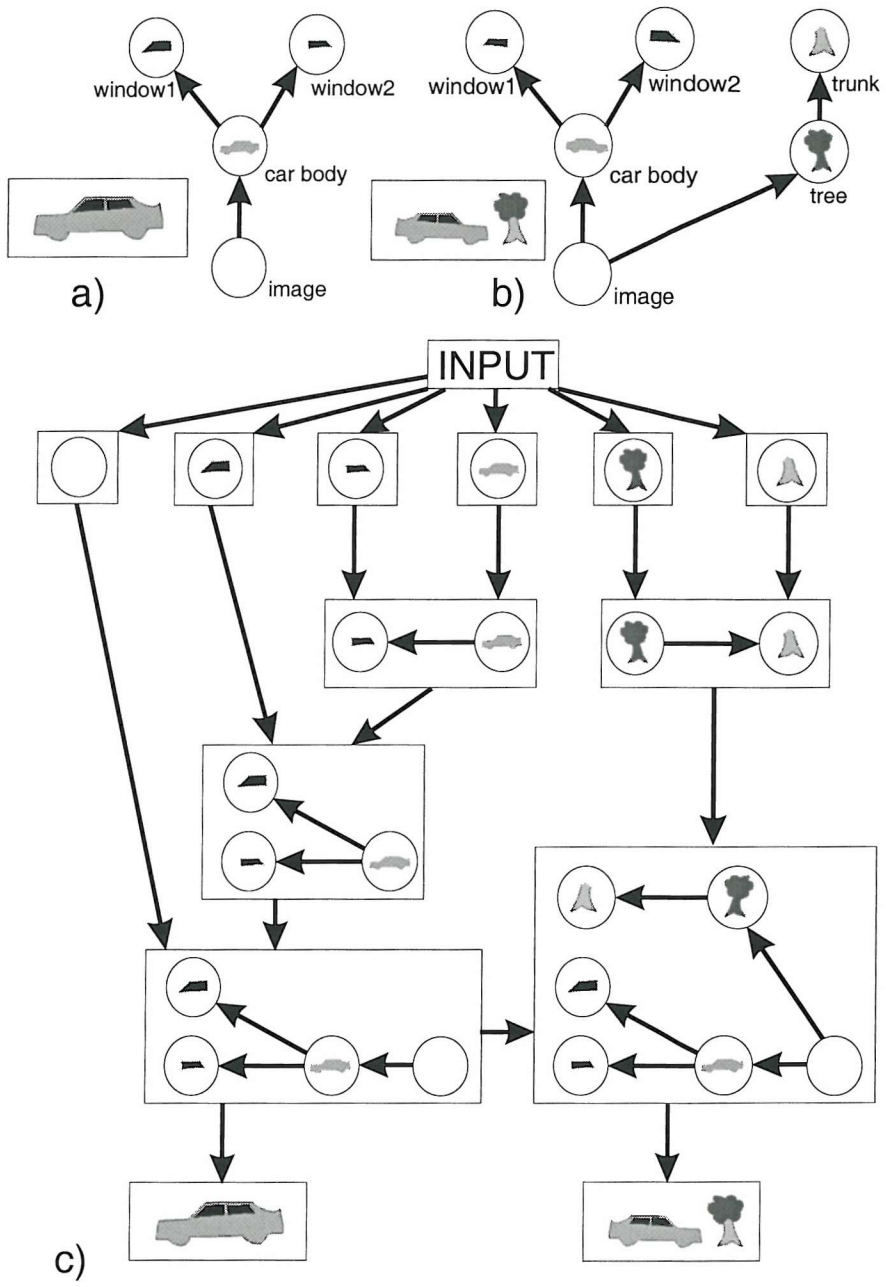


Figure 5.25: a) An illustration of a scale space tree built from the simple image; b) Another illustration of a scale space tree built from an image, where the image has subgraphs in common with the image in a); c) The network when both of these graphs from a) and b) have been inserted. It is possible to see the shared subgraph representing the car.

to find similar images and submit this as a query. Any matching sub-trees in the query image will be found in any of the models in the database, and the model images with the largest matching subgraph may look similar to the query image. The second method of querying is to submit a specific object which one would like to find in the model database. This query could be generated in two ways. An image of an object could be found and submitted as a query image, as in the first method. If the object is well segmented (i.e. on a nondescript, smooth background) there will be a sub-tree of the image representing the object to find. Any matches in the model database will be found. The second way is to generate a subtree by interactive segmentation of an object, using something like GIP (60) for which a segmented object has a tree generated, or by selecting nodes within a scale-tree to select a sub-tree representing an object. This sub-tree representing an object is then used as a query to the network. Both query by whole-image and query by sub-tree methods are tested here.

Before testing matches can take place, the network for a scale-tree has to be correctly generated. Using simple images as inputs into the network algorithm it is possible to display the final network and test the network generation. Only when the networks for trees are being correctly generated can we perform match tests.

It seemed sensible to perform network tests using strings for vertex labels to begin with. This would ensure the network algorithm was working adequately without the extra complication of feature matching introducing fuzzy matches. Because our system is able to use vertex labels of any kind, this is not a problem, and we are able to use the same code to test simple vertex label matching with complex vertex label matching. This allows us to be relatively confident that if the network algorithm works correctly for graphs with strings as vertex labels, it will perform similarly for feature based labels, and we can concentrate on the results.

The following example shows the construction of a network containing a single graph. The graph contains vertices that have labels that are strings. Figure 5.26(a) shows an illustration of the graph and figure 5.26(b) shows an illustration of the network generated from this single graph.

Figure 5.27(a) shows a new graph, also using strings at the vertices. It is plain to see that the graph shares a subgraph with the graph in figure 5.26(a). When we build the network we can expect this subgraph to be shared between the two models in the network. Figure 5.27(b) shows an illustration of the network after the insertion of the graph from figure 5.27(a). This shows how the network algorithm builds networks and shares the common subgraphs between models.

If we insert a new graph that contains a new vertex label (shown in figure 5.28(a)) into this network we see (in figure 5.28(b)) that a new IVertexChecker is created to store that label, and that the model is built correctly, also sharing the common subgraph.

We can perform a simple query by allowing the graph shown in figure 5.26(a) to filter through the network. Obviously, we expect a match at the model node representing that model, which is the result we get. The graph shown in figure 5.29(a) is a mutation of the graph in figure

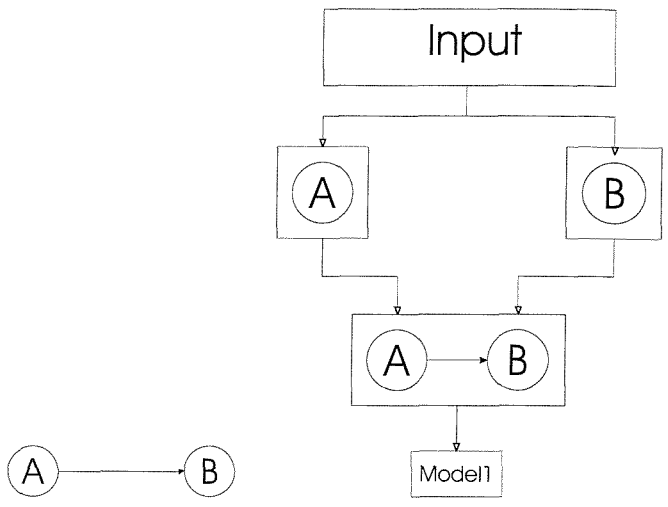


Figure 5.26: Insertion of the simple graph in figure 5.26(a) into an empty network, results in the network in figure 5.26(b).

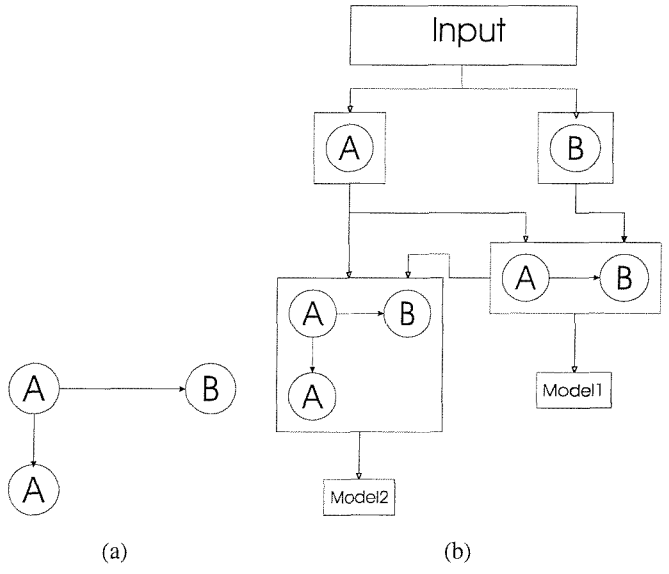


Figure 5.27: Insertion of the graph in figure 5.27(a) into the network from figure 5.26(b), results in the network in figure 5.27(b).

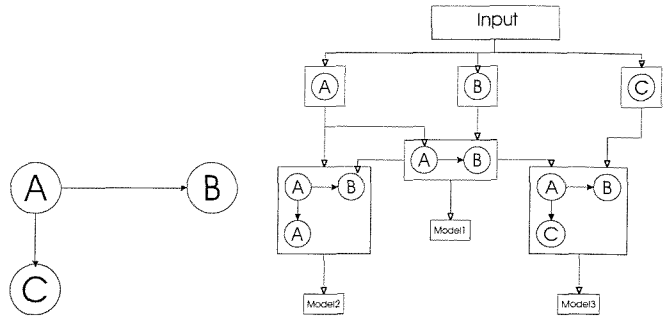


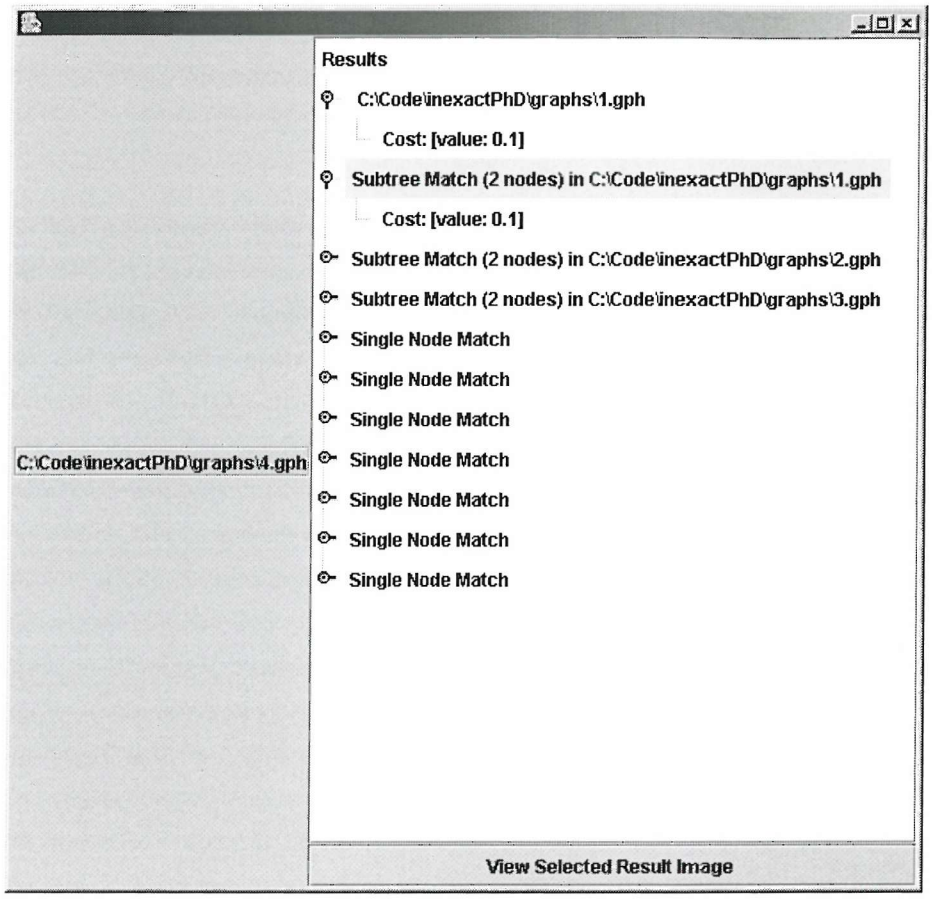
Figure 5.28: Insertion of the graph in figure 5.28(a), containing new, unseen labels, into the network from figure 5.27(b), results in the network in figure 5.28(b).

5.26(a), where one of the nodes has a different label. If a query is performed with this graph we can still expect the best match to be at the model node representing the graph shown in figure 5.26(a). However, there should be a default cost associated with vertex label substitution. The default value for label substitution is set to 0.1 for non-feature-based labels, therefore the cost associated with any matches will be 0.1 - the substitution of label "B" with label "D". Figure 5.29(b) shows the results window from the harness. The three models in figures 5.26(a), 5.27(a) and 5.28(a) are labelled 1.gph, 2.gph, and 3.gph respectively in the example network. The best match appears at model 1.gph, as expected. The results window also shows sub-graph matches in all the models which share subgraphs containing instances. This query finds that, with the same cost of 0.1, there are matches in the other models (2.gph, and 3.gph). This shows the main advantage of the network algorithm approach: matching subgraphs in many models can be found from a single query process, avoiding the need to perform three graph matches.

The above example query demonstrated matching a subgraph within the models. It is also possible for the input graph to be a larger graph in which we can find instances of the models. Figure 5.30(a) shows a larger graph than those that are stored in the network structure as models. There are subgraph isomorphisms of both the graphs shown in figure 5.26(a) and figure 5.27(a). The test harness window, in figure 5.30(b), shows the results of the query using this graph as an input. The isomorphisms of both models are found within the network. This incurs the slowest execution time (without using inexact matching) because the input graph has to filter down to whichever of the matching model nodes contain matches (which may be many). Inexact graph matching using this larger-query-graph paradigm will also take longer than inexact graph matching of sub-tree queries for this reason (without taking into account the time needed for the node label matchers to generate a distance measure for the inexact match).

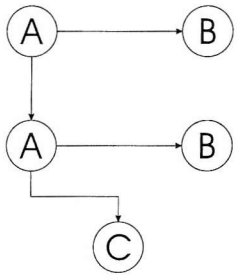
Although this test is not conclusive, other tests can be run to show that our network algorithm implementation correctly generates and matches simple, string-based graphs in the

(a)

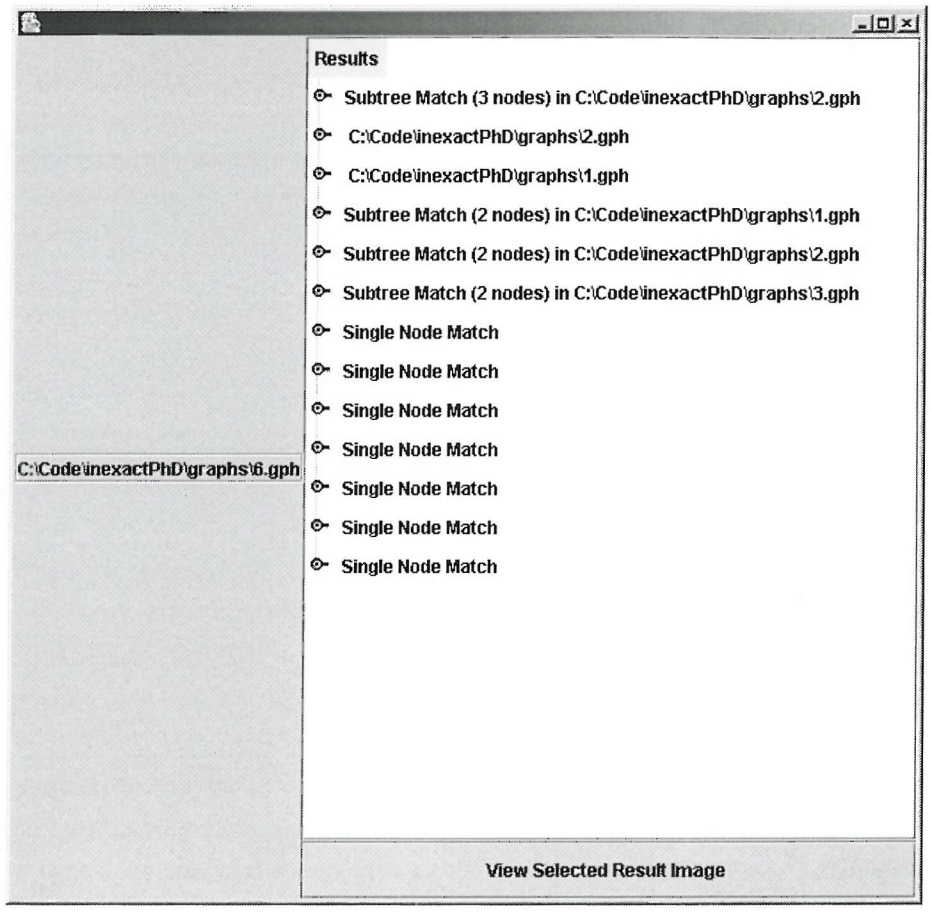


(b)

Figure 5.29: Matching the graph in figure 5.29(a) to the previously generated network (figure 5.28(b), gives the results shown in figure 5.29(b).

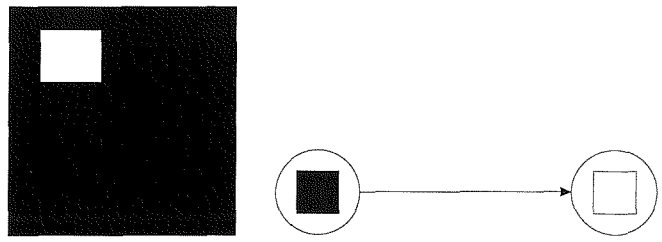


(a)



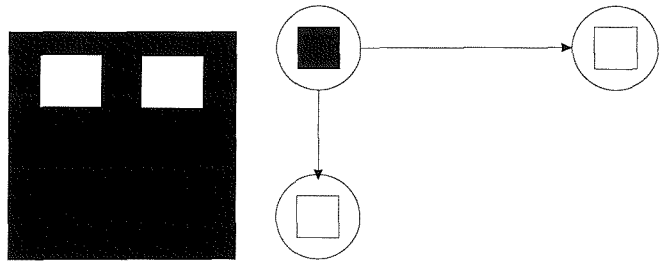
(b)

Figure 5.30: Matching the graph in figure 5.30(a) to the previously generated network (figure 5.28(b), gives the results shown in figure 5.30(b).



(a) (b)

Figure 5.31: Simple image used to test feature-based graph matching and the graph representation of its scale-tree.



(a) (b)

Figure 5.32: Simple image used to test feature-based graph matching and the graph representation of its scale-tree.

expected manner, such as the example above. Note, that the graphs shown above are trees, proving that the network algorithm works with trees.

We can effectively convert the above example graphs into very simple images from which we are able to test the feature-based graph matching. For example, the vertex label "A", can be directly mapped onto a black box, and "B" onto a white box. This means that our mapping of the graph in figure 5.26(a) becomes the image and graph shown in figure 5.31. Because one of the features used to match vertex labels is based on colour, it becomes nonsense to allow a vertex to be connected to a vertex with the same label - the region in question would be invisible (although this is allowable if shape is the only feature used, for example). So, we map the graph in figure 5.27(a) to the image and graph shown in figure 5.32 and build a new network with the two graphs, shown in figure 5.33. As before, a third graph with a new vertex label (figure 5.34(b)) is inserted into the network that contains the scale-tree representation of the image shown in figure 5.34(a). Because, for this experiment, both colour and shape are being used, the new region appears to the network as a completely new vertex label and has a new IVertexChecker generated for it. The figure 5.35 shows the network with the graph for this

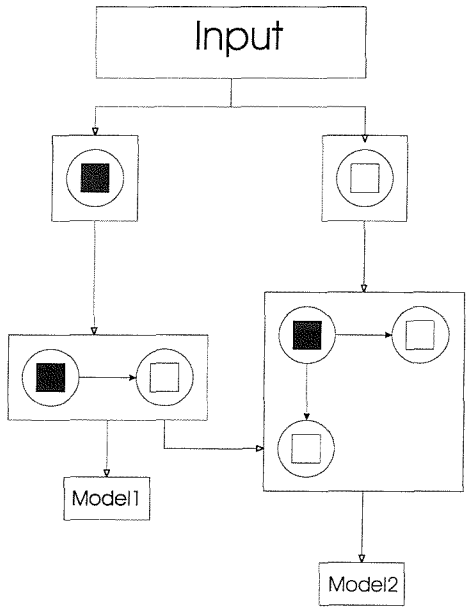
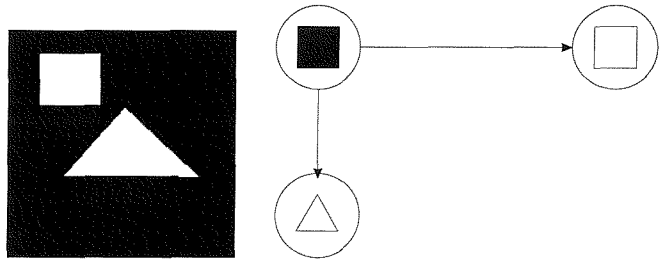


Figure 5.33: Network built from the two graphs in figures 5.31(b) and 5.32(b).



(a) (b)

Figure 5.34: Simple image used to test feature-based graph matching and the graph representation of its scale-tree.

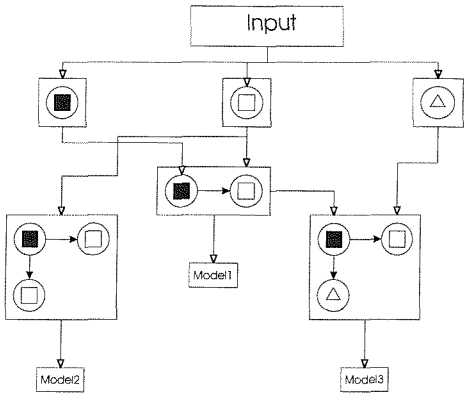


Figure 5.35: Network from figure 5.33 with the graph from figure 5.34(b) inserted.

image (from figure 5.34(b)) inserted. As in the case of the strings, the common subgraph is shared between the models, although feature matching (colour and shape) provided the equality matching functions.

It is worth noting here, that the types of features that are used to match vertex labels will determine the number of IVertexCheckers. It is always assumed in content-based image retrieval that the more features one has representing a region, the better it is for matching because the more discriminating the matching process becomes, and therefore it will find objects nearer to the query object. However, in the case of the network algorithm, having too many features to represent regions can cause a problem. If we extrapolate the idea to the furthest possible extreme, so that every region that enters the network is considered different, we find that every region that ever entered the network has its own IVertexChecker (there will be the same number of IVertexCheckers as there are vertices in every model in the network), and therefore every subgraph that is represented in the network will have its own ESubgraphChecker. This means that no subgraphs will ever get shared. It also means that every incoming query region will need to be compared against every other region in every other database image. Of course, this extreme should never happen, but applying too many discriminating matching algorithms to the network algorithm will actually degrade its performance. Instead, matching algorithms need to be chosen carefully to balance the discriminating power of the region matching against the complexity, and therefore number of shared subgraphs, of the generated network. As an example, compare the final network again, generated for the graphs in figure 5.31(b), figure 5.32(b) and figure 5.34(b). The network in figure 5.36 uses only colour to distinguish between regions, therefore the triangular region from the graph in figure 5.34(b) is considered equal to the square region from the other graphs. The 3-node common subgraph then becomes common between models 2 and 3 (a black region connected to two white regions). It is only if shape is also used that the correct network (shown in figure 5.33) is generated. However, the shape matching algorithm needs to be fuzzy enough to allow any triangular region to match,

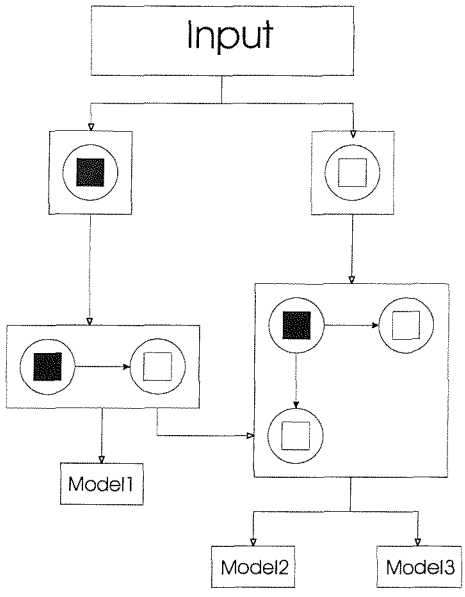


Figure 5.36: Network built from the three above graphs using only colour for the feature matching.

otherwise too many IVertexCheckers will be created (one for every type of triangle). It would be ideal to allow the user to decide the fuzziness of the feature matching functions, however because network generation is a relatively slow process, these decisions would have to be made prior to the matching.

In this section a query was constructed using strings and matched against an example network, a feature-based version that we have generated here. Therefore, analogous to the previous example, we can perform a match on this feature-based network using an image as the query term. For example, using the image in figure 5.31(a) as the query term, we should expect results at the model node representing that image, and sub-graph matches for the other models (which share that common subgraph). Figure 5.37 shows the result window after performing this query. The correct match is found (shown at the top of the list) and sub-matches are shown in decreasing size.

We can mutate one of the graph labels to show how with feature matching the correct models can be found. Figure 5.38(a) shows an image, and the graph of the scale-tree of the image, which contains a single mutated node label. Rather than the white square in model 1, there is a grey circle. The network will edit the input graph, substituting the grey circle for the appropriate region, and adding an appropriate score based on the feature measures. Figure 5.39 shows the results for a query run with this node. Notice how the cost associated with the instance that reached the model node is generated from the feature matching (the average distance between the query and match region for each of the features).

Again, mimicking the small tests performed with the string-based graphs, a larger graph can be input into the network to determine which models fall within the given graph. Figure

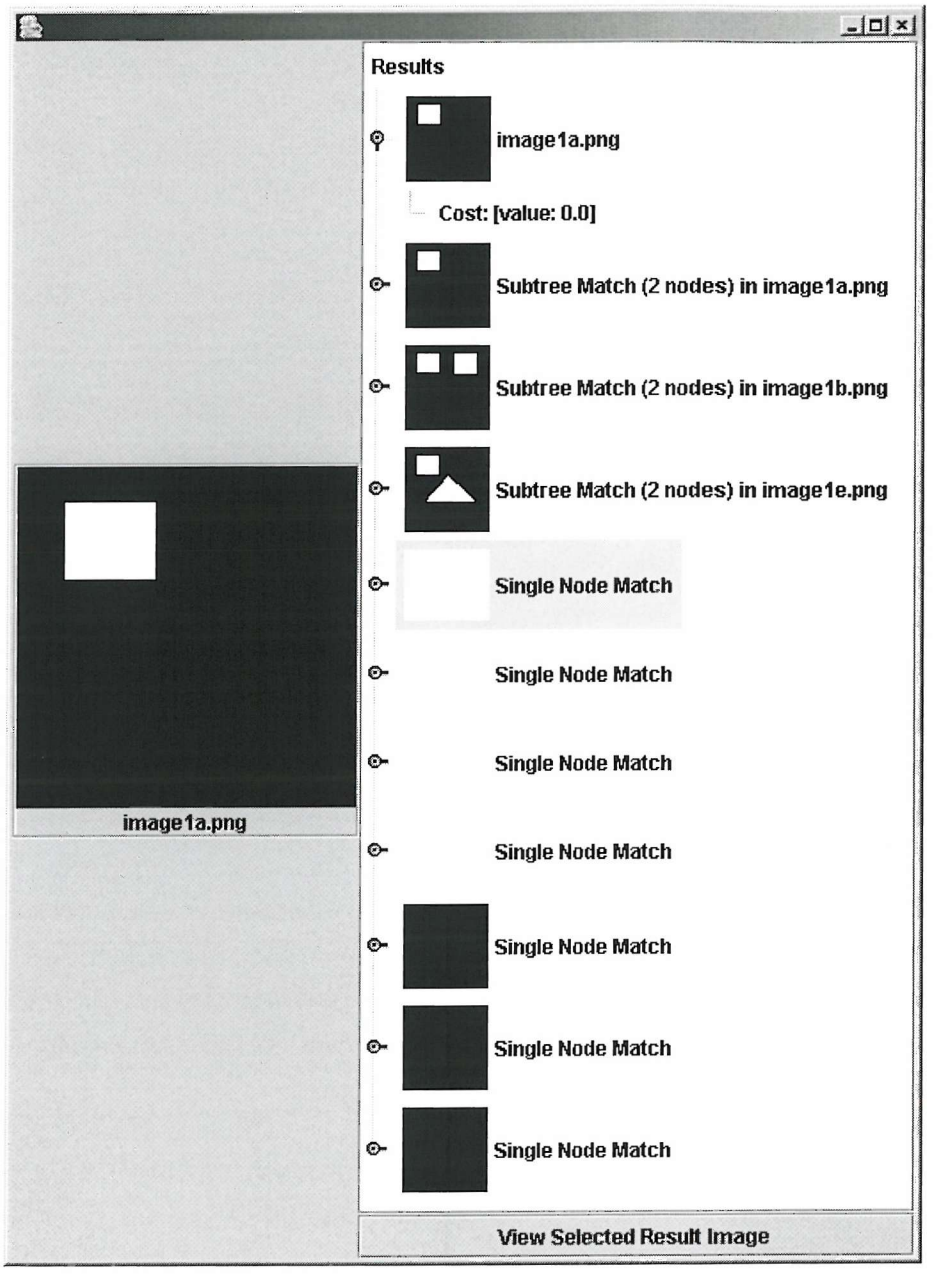


Figure 5.37: Results from querying the network in figure 5.33 with the image from figure 5.31(a).

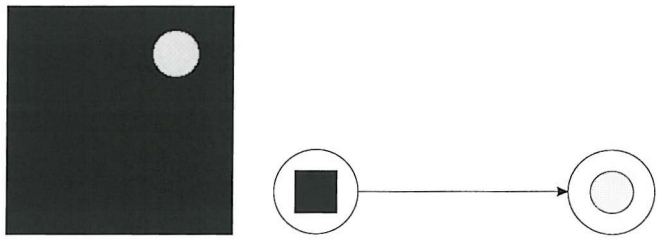


Figure 5.38: Simple image used to test feature-based graph matching and the graph representation of its scale-tree.

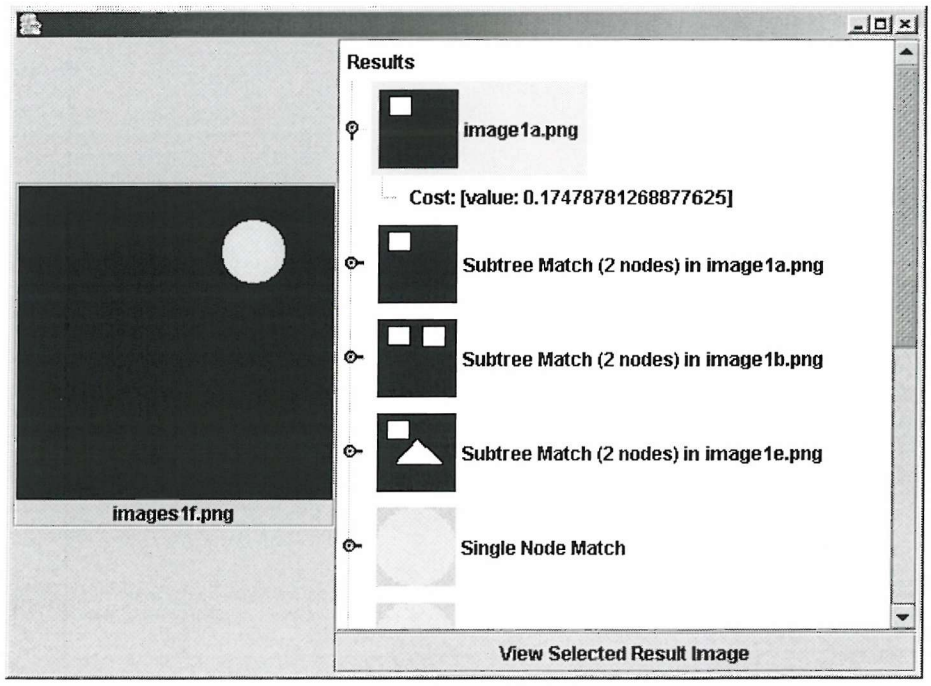
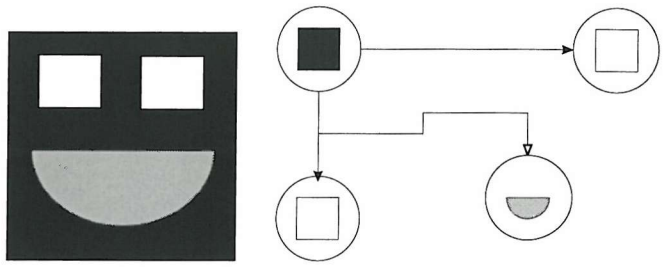


Figure 5.39: Results from querying the network in figure 5.33 with the image from figure 5.38.



(a) (b)

Figure 5.40: Simple image used to test feature-based graph matching and the graph representation of its scale-tree.

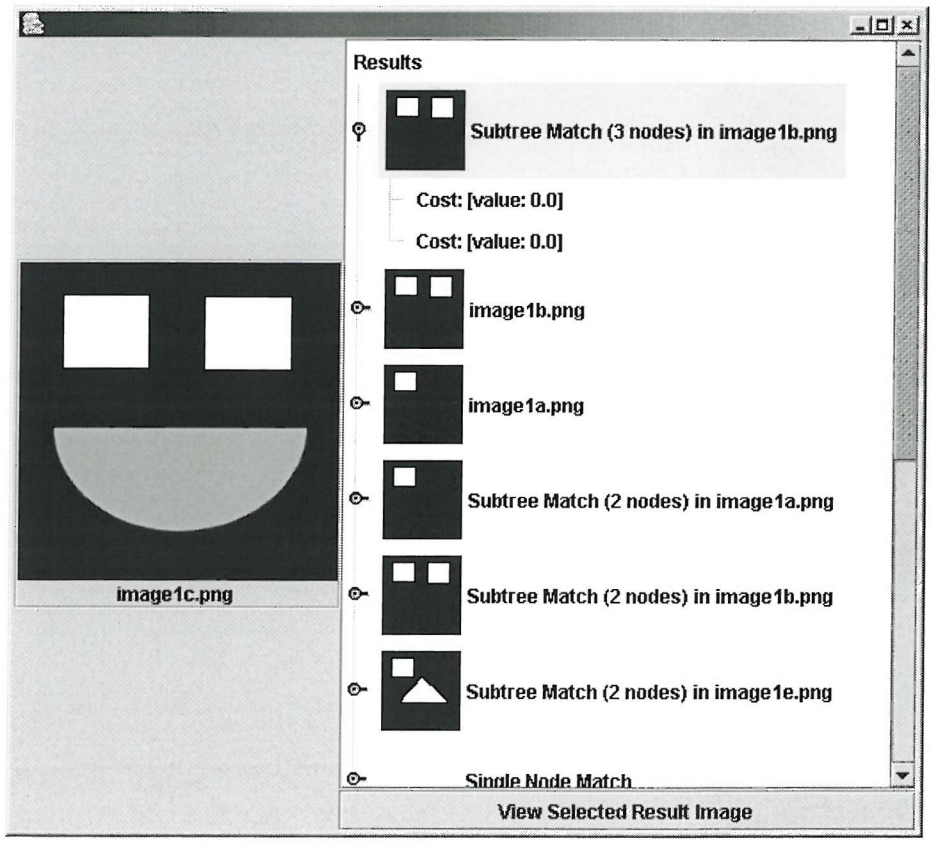
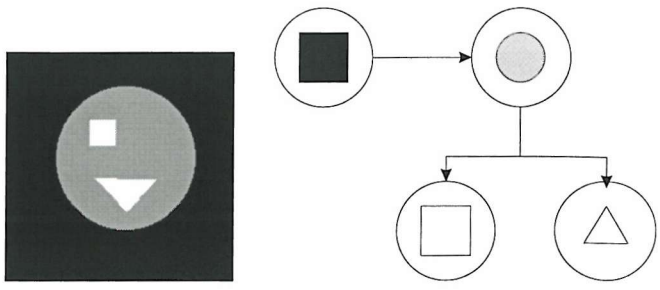


Figure 5.41: Results from querying the network in figure 5.33 with the image from figure 5.40.



(a) (b)

Figure 5.42: Simple image used to test feature-based graph matching and the graph representation of its scale-tree.

5.40 shows an image and the graph of its scale-tree containing four nodes, three of which, as a sub-tree, form model 2 from figure 5.32(b). By using this graph as a query, we should expect to find an exact sub-graph match at the model node representing model 2, and indeed, figure 5.41 shows the results from this query showing this to be the case.

So far we have shown that the network algorithm can be used to find the most similar images to a query image. One of the main novelties of the technique we present here is the ability to match topologies within the scale-tree. In the next example, it is shown how the topologies and sub-topologies of the trees can be matched. For example, figure 5.42 shows an image and the graph of its scale-tree. The image contains an object consisting of more than one node. Again, to demonstrate the idea, it is comprised of simple blocks - a grey block containing a white block and a white triangle. The network generated for this image is shown in figure 5.43. Note how the ESubgraphChecker (marked 'A') represents the object segmented from the background. We can search for this object by supplying a tree, as a query, which represents only this object.

Our test harness allows selection of scale-tree nodes from a 3D representation of the scale-tree, and it is possible to use the selected node as the root to a scale-tree query. Figure 5.44(a) shows the harness and selection of a node. The image in the bottom left highlights the regions in the original image that are represented by the node that has been selected in the tree. Figure 5.44(b) shows the graph for the scale-tree query. Figure 5.44(c) shows the results of performing a query with this tree. Figure 5.44(d) shows the image with the matched region highlighted, showing that the best match was found in the original image in the correct position.

Figure 5.45 shows the test results of searching for the sub-tree in figure 5.44(b) after a transformed image (vertically flipped) has been input into the network as a second model. The network structure shows that the feature matching provided enough accuracy to correctly match the transformed regions, by the fact that the network is identical to that in figure 5.43, but for an extra model at the model node. The results show that the sub-tree was found within both

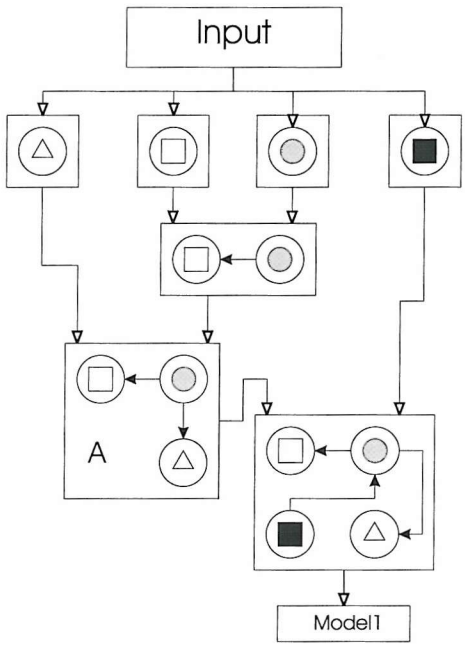


Figure 5.43: A network built containing the image from figure 5.42. The ESubgraphChecker marked 'A' contains the object subtree.

models.

In figure 5.46 scale dependency is tested. Another model is added to the previous network. The model consists of an image with two translated instances of the object, as shown in figure 5.46(a); one is scaled down, and the other scaled down and rotated at 45°. Neither the shape matching, nor the colour matching are scale-dependant and so the query should provide good results. However, the shape matching is susceptible to rotation. For this test, because the rotation and scaling process caused blurring in the images, both the noise and blur filter were used to reduce the number of nodes. The original image, in figure 5.46(a), had 282 nodes, and has been reduced accurately to 7 nodes by the tree pruning for the match. The results for the match with the sub-tree representing the object, from figure 5.44(b), is shown in figure 5.46(b). The previous results are returned along with the new result representing the partial match of the new model. One issue to note is that there is only one instance returned from the query for the new model, when there are plainly two instances of the object in the model. Examining the network constructed for the models reveals that the rotated square, in the lower right instance in the model, is not matched with the squares from the other models, and is considered similar to the triangle by the simple shape matcher, allowing this subgraph to be represented by a separate ESubgraphChecker in the network. The subgraph would have been returned as an instance with a small cost (the cost of replacing this rotated square with the un-rotated square), had no exact instances been found.

The next test shows an example of the feature-based network algorithm providing match-

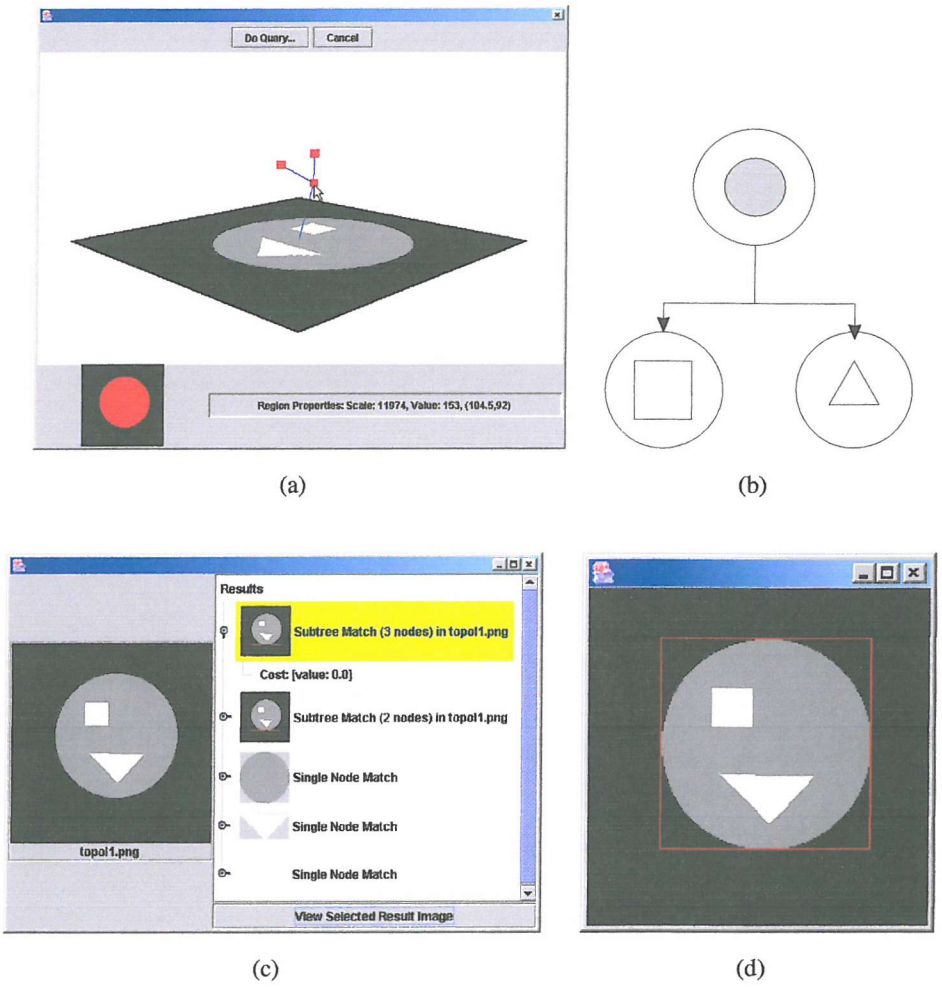


Figure 5.44: Selection of a subtree from an existing image and using that subtree as a query to the network in figure 5.43, the results of which are shown in figure 5.44(c), and the location of the best match in figure 5.44(d).

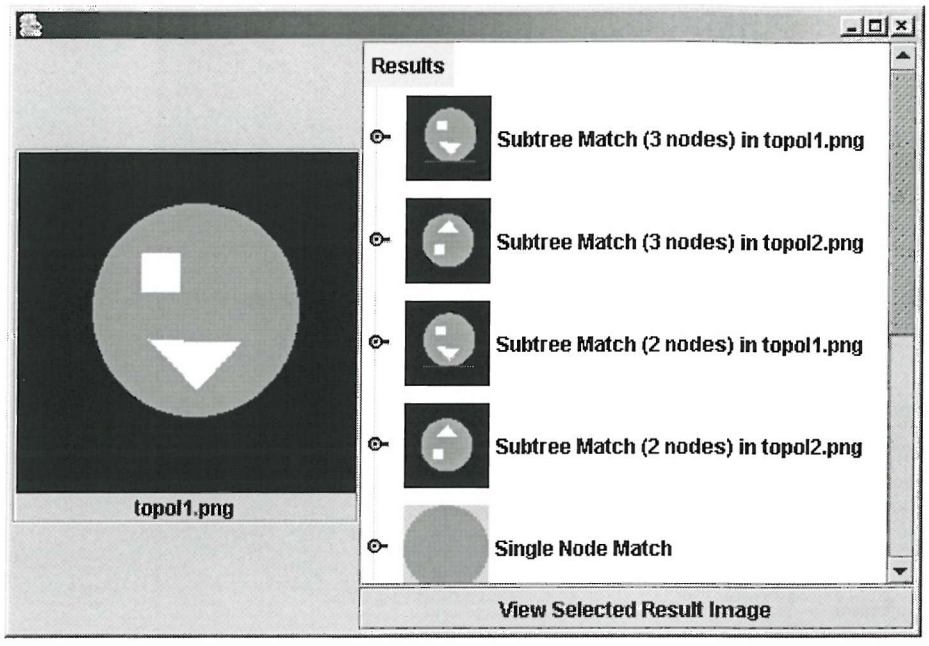
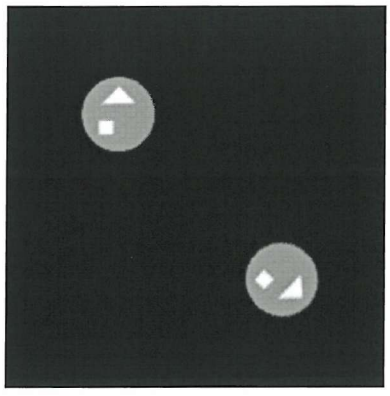


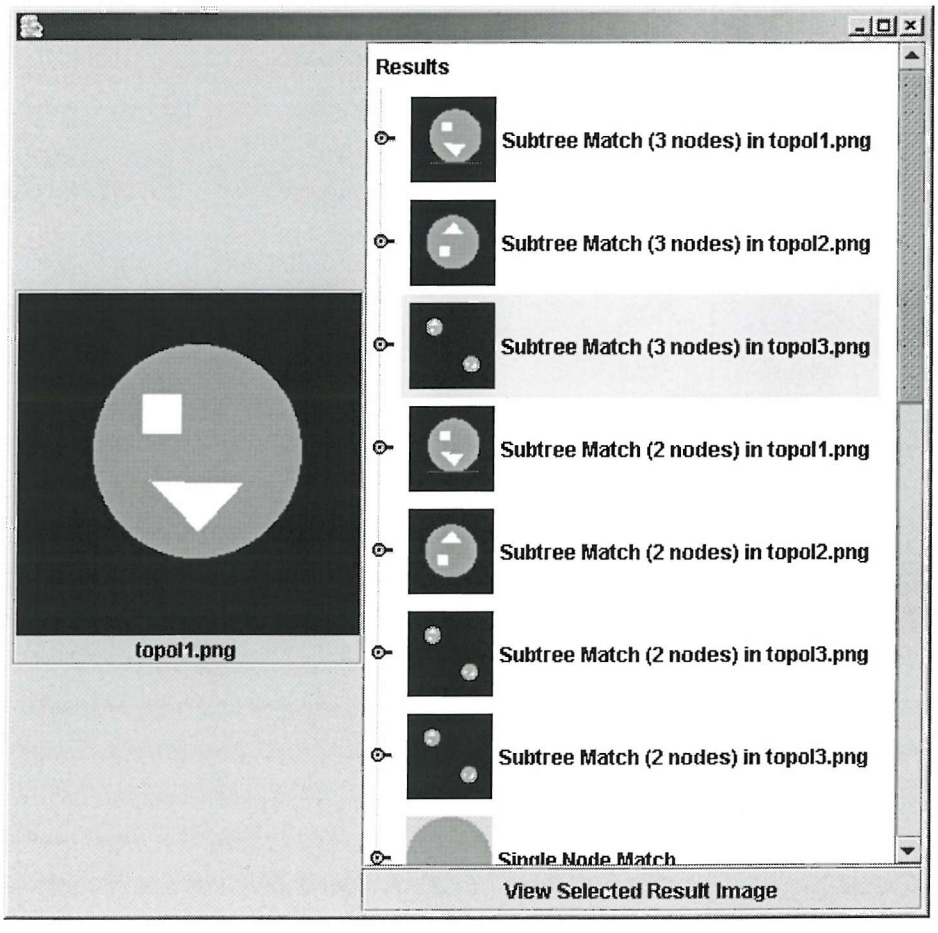
Figure 5.45: Results from querying a network containing the image in figure 5.42(a) and a vertically flipped version of the image, with the sub-tree from figure 5.44(b).

ing of images which are only similar in topology. The models that are used to build the network are shown in figure 5.47(a) and figure 5.47(c). The face image has 8 nodes and the house image also has 8 nodes. These images do not share any subgraphs, and occupy different chains of network nodes in the network. This will ensure that the topology is an important factor in the test. Using the image shown in figure 5.42(a) as the query image, it is possible to see that it looks more similar to the face (figure 5.47(a)) due to the topology of its features. No exact matches are found during the querying process and the inexact graph matching takes over, which causes the match to take considerably longer. Figure 5.48(a) shows the final results for the match. Notice that the query matches only a part of the model (a subtree is returned rather than the model). Although inexact instances are being propagated through the chain of nodes towards the model of the house (figure 5.47(c)) the face model propagates with much less cost, and so is returned as the best match.

Figure 5.49 shows a set of images which are used as models for the database. They're simple road sign images and some have noise and blurring of the pixels. Some of the trees after pruning with the default values had over 170 nodes, so to ensure the network does not grow too large, the noise threshold was raised to the maximum range so that all small regions below scale 50 are removed from the trees. As a comparison figure 5.50 shows the actual pruned images that are used as models. The models are chosen to vary across a range of styles so that the results of the query are clear. Figure 5.51(a) shows the query image that is used - another sign image which does not appear in the database. The final network has 59 network



(a)



(b)

Figure 5.46: Searching in a network containing an images with scaled and rotated instances of the query object.

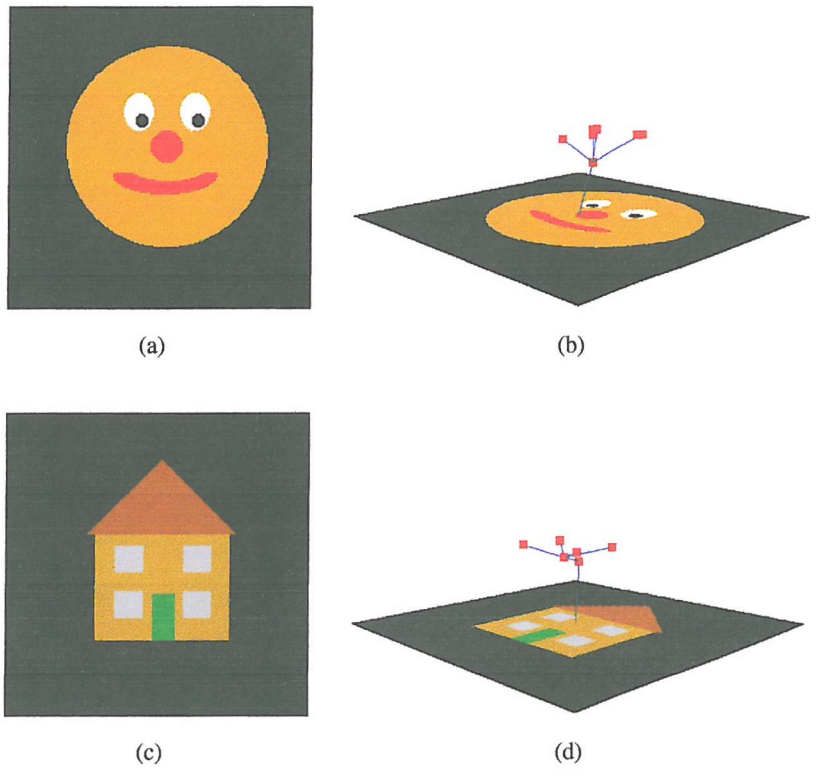
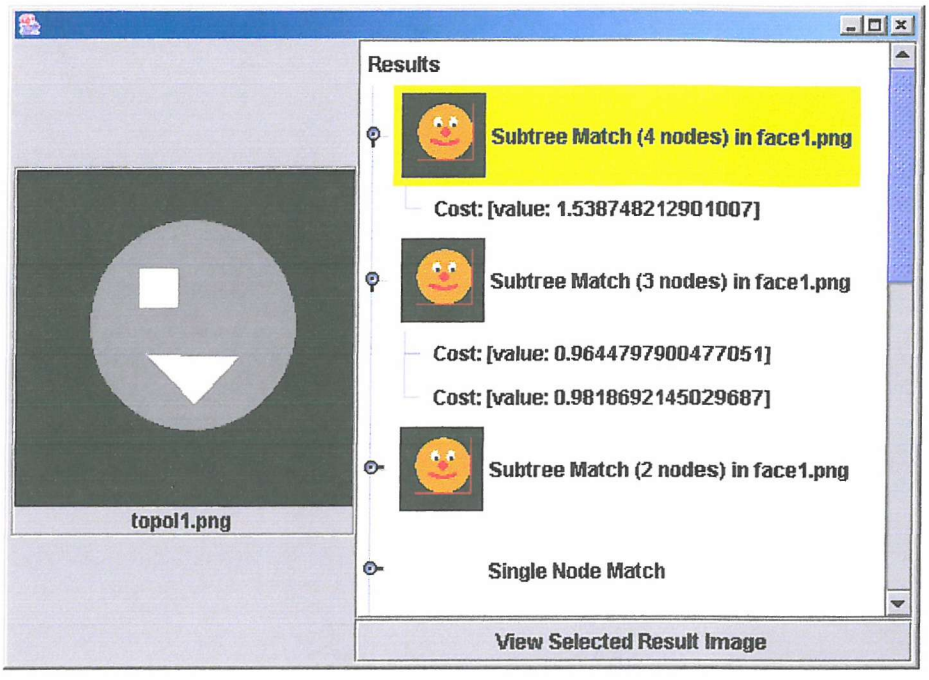


Figure 5.47: Images and their scale-trees used as models for topology matching test.

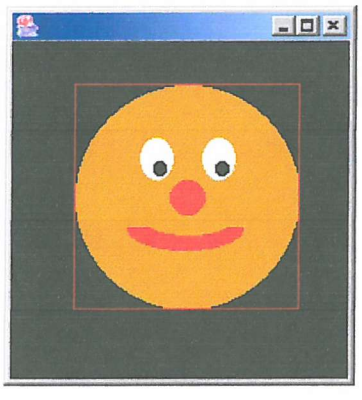
nodes. Figure 5.51(b) shows the results of the query. In the results, it is possible to see that the most similar sign has been found as the best match, while a similar sign (with a similar topology) has been given as another good match. The other signs are not retrieved, as one would expect. Note that the results are exact (there are no inexact results returned if exact matches are found) implying that the shape matcher has considered the red regions to be the same shape.

The following experiment shows the system operating in a more realistic environment - that is, with a photographic image as query. The database contains a set of signs, shown in figure 5.52. Some of the database images are duplicated, as they appear in the original road sign database, and this will not affect the match. The query is shown in figure 5.53(a) and is a photograph of a sign similar to the final one in the database. This experiment utilises luminance and saturation as the greyscale colour space for the image (as explained in the following discussion in section 5.4). The results of the query are shown in figure 5.53(b). The results show that the best match was the expected model, returning the shield shaped road name sign, the cost being the edits to the graph to change the colour of the sign and remove one of the numbers.

These tests have been selected to show how the network algorithm can be used with fea-



(a)



(b)

Figure 5.48: Results of querying a network built with the images from figure 5.47, with the image from figure 5.42(a).

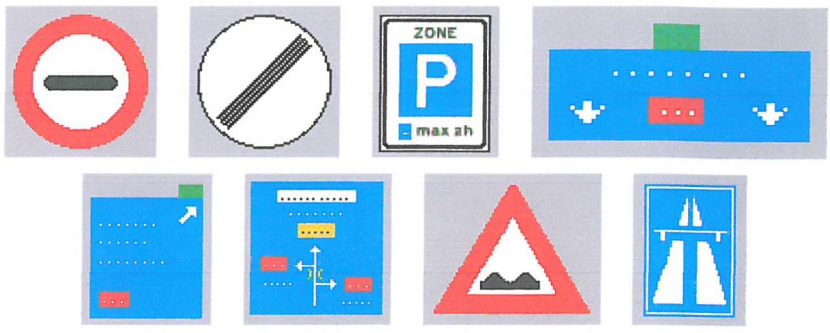
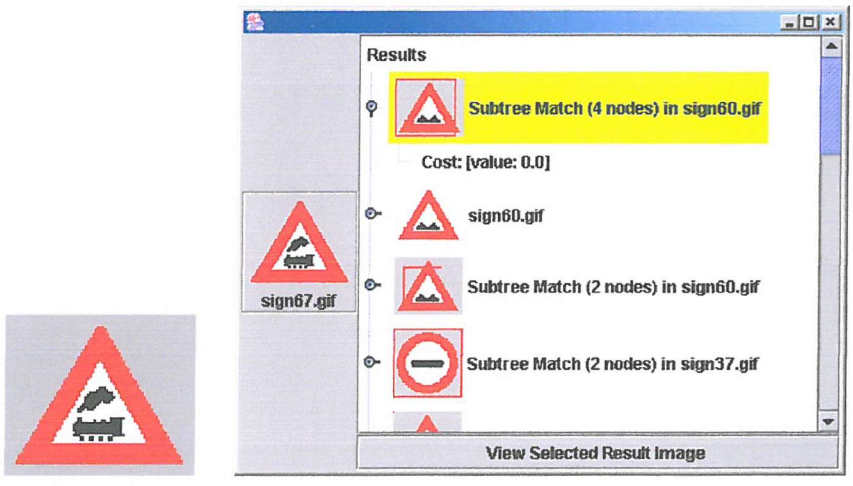


Figure 5.49: Set of model images.

Figure 5.50: Pruned set of model images from figure 5.49.



(a) (b)

Figure 5.51: Results of performing a query with the image in figure 5.51(a) on a network built with the trees of the images from figure 5.50.

Figure 5.52: Set of model images from a database of American road signs.

(a) (b)

Figure 5.53: Results of performing a query with the image in figure 5.53(a) on a network built with the trees of the images from figure 5.52.

ture based matching to provide image and subimage matching. The next section provides some thoughts on the tests, in particular, the problems that were encountered during the testing process, and some of the pitfalls of using the network algorithm, and the sieve for topological image matching.

5.4 Discussion

Although the tests presented above show the idea behind graph matching in order to provide a means for finding objects is sound, there are problems involved with the specific way in which we have chosen to implement this prototype. However, these issues could be overcome with extra development in both topology based segmentation algorithms, and graph matching techniques.

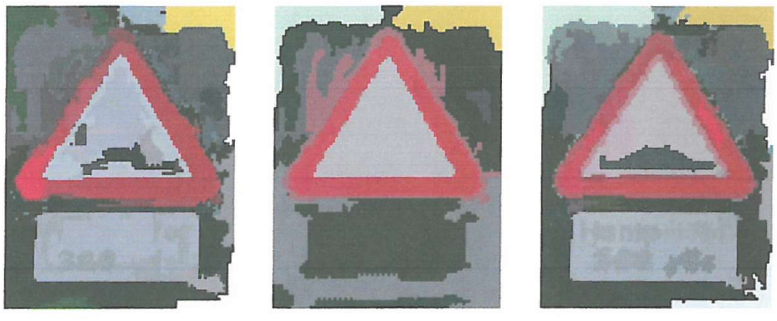
The sieve technique relies on the pertinent regions within the image being extrema within the image domain. Although this is a pragmatic assumption to make, it is often not the case within images. There are few reasons that extreme regions are not necessarily pertinent objects.

Firstly, most pertinent objects consist of many regions, some which will be extrema, others that will not, and, maybe more importantly, background image data will also consist of regions

(a) Original Image

(b) Mean Greyscale

(c) HSB Brightness



(d) HSB Saturation

(e) HSB Saturation × Brightness

(f) HSB Thresholded Saturation × Brightness

Figure 5.54: Reconstructions of decompositions based on different channels of the original image in figure 5.54(a). All decompositions used 32 levels of quantisation, and noise and blur pruning.

of which some will be extrema and others not. The sieve process will then merge regions in a way which is not conducive to an object matching process.

In complex images mutations of the graph are often caused by background image data. Often background image data can have detrimental effect on the pertinent objects in the scene, in particular, when it comes to the merging of regions through the sieve process. Regions of background images can affect the shape, colour and texture of pertinent objects in the scene by being merged into the object simply because of their relative greyscale level (see figure 5.23(b)).

Using a different greyscale colour space could provide a simple solution. Using luminance from the CIELab colour space would provide a perceptually uniform greyscale, providing a more realistic distinction between the different regions. Obviously, this still does not account for the colour channel (i.e. two regions of the same perceptually uniform brightness will still merge in scale-space). Using saturation allows colour variations to provide an input

to the decomposition (combined with luminance gives an input from both). Figure 5.54 shows the reconstructed image from a decomposition of the photographic image in figure 5.54(a) using different methods of greyscale generation. Figure 5.54(b) shows a reconstruction of the decomposition of the model based on the mean greyscale value (calculated by $\frac{R+G+B}{3}$), while figure 5.54(c) shows the luminance from the HSB model. Both techniques suffer from incorrect segmentation, causing the red triangle identifying the sign to be broken (and therefore unmatchable). Figure 5.54(d) shows the reconstruction of the decomposition of the model based on the saturation of the image, while figure 5.54(e) shows the product of luminance and saturation. Saturation allows the red triangle to be segmented better, however, due to saturation being undefined for greyscales (including white and black), the centre of the sign is corrupted. By using a combination of both saturation and luminance a better segmentation is achieved, however, the centre portion of the sign, which differentiates the sign from other warning signs, is obscured. By thresholding the saturation (to avoid the undefined values), a reasonable segmentation is given. Figure 5.54(f) shows a decomposition using saturation thresholding at 10%. However, these still make it possible for certain distinct regions to merge into each other in the new colour space. Mapping the colour to the colour spectrum provides a linear distribution of the colour that can be applied to the image allowing a greyscale representation of the colour to be used, although the idea of extrema then becomes redundant (e.g. why is red less extreme than blue? or more importantly, why is green not considered extreme?). Therefore, spectral mapping of the image colour does not lend a simple solution. However, in any of these cases, it is still possible for incorrect merging to occur because there is a loss of data in the conversion to a greyscale colour space.

There are certain ways to overcome simple instances of incorrect merging, such as using the quantisation technique before sieving. The idea behind the quantisation filter is to provide a flatter image on which to perform the sieve so that it could foreseeably provide a better segmentation, however, this filter is a brute-force approach, and causes artefacts (the banding, causing larger, but not new, extrema where there should not have been). However, using the sieve there is no guaranteed way to segment a pertinent object from a scene without human validation, and if necessary, intervention. This effect is due, on the whole, to the way the sieve process decomposes an image.

Any incorrect merging will cause mutations in the graph which, although overcome to some extent by inexact graph matching, cause other objects which are not similar to the query to be matched, albeit with a high score. Altering the inexact graph matching weightings of costs of graph edits (insertion and removal of nodes) versus feature edits (e.g. vertex label substitution), could possibly go some way to overcoming this. This makes the algorithm dependant on the weightings of these costs. For instance, relatively weighting, say, feature scores more heavily than graph insertions and deletions would cause the new nodes to be inserted and deleted from graphs more readily, allowing unlike objects to be matched more easily. But if we were to weight graph edits highly, features would be the discriminating factor causing the mutated

nodes to match badly allowing unlike objects to match each other and slowing the matching process. A solution is to allow the user to decide at match time which is the correct weighting for the various variables, however this begins to remove the possibility of creating a fully automatic object matching system. Of course, the assumption has been that the user would be wishing to search for pertinent objects, of which there is no guarantee. Although all regions appearing in images that would usually be considered background are encoded by the sieve process into the scale-tree, due to the effect described above, they are not necessarily extrema and represented by their own nodes within the tree, therefore the objects themselves are not searchable, only the agglomeration of a few regions is available to match against. The ideal solution would be to avoid the mutations before tree-creation, i.e. extend the decomposition to provide a better segmentation. In section 6.1.3 we describe a possible method for providing better segmentations based on the sieve.

As well as problems with the sieve method, there are problematic areas with the network algorithm approach. It is, of course, a useful approach for sets of data that share subgraphs. However, the main problem with the network algorithm, is that it does not share all possible subgraphs. Take our earlier examples and the graph in figure 5.27(a). The network, in figure 5.27(b) shares the small two node subgraph from vertex labelled "A" to the vertex labelled "B", however, it fails to contain anywhere within it, the subgraph from vertex labelled "A" to the other vertex labelled "A". This is a serious drawback to the network algorithm as it relies on the network building, and sharing, subgraphs that are to be searched upon, which will not be known in advance. The network algorithm we have implemented uses certain heuristics to attempt to pick those subgraphs that will most likely be searched upon. For example, it will choose nodes representing small regions over nodes representing large regions, and nodes with a high edge degree over nodes with a low edge degree, from which to construct an ESubgraphChecker. However, this in itself is not enough to ensure the choosing of the most likely subgraphs. Indeed, there is no automatic way specific subgraphs could be constructed without prior knowledge of the application domain.

Another problem with the network algorithm is that image data is very diverse and rarely contains sub-topologies (from scale-trees) that are identical. This leads to having to use the inexact network algorithm, which is considerably slower than the exact network algorithm, making matching of large sized trees computationally very expensive. The tree pruning algorithm attempts to overcome this problem by reducing the number of nodes in the tree, while preserving pertinent image data. However, such a small number of nodes is required to allow inexact graph matching, that the tree pruning algorithm tends to damage pertinent image data.

Recently in the literature, the developers of the network algorithm have developed a new matching algorithm that overcomes many of the pitfalls of the network algorithm. This algorithm is faster than the network algorithm (the time complexity is independent of the number of graphs in the database), and it also stores a representation for every subgraph of a model. An algorithm such as this could provide a better basis for graph matching of image topologies

than the network algorithm. However, the technique described in (13) does not have any error correcting mechanism. It is based on decision trees and it is described in the following section 5.4.1.

The final problem revolves about the speed and resource usage of the prototype. Although prototype systems generally are slower and more resource hungry than a final system, the prototype that has been built suffers particularly badly. In fact, it has made it impossible to provide results for all the tests we wanted to run. This is partly to do with the prototype system being a Java application, but also to do with the simple way in which the sieve decomposition has been implemented. The larger the number of regions in the image the slower the decomposition gets. Precision and recall were introduced at the beginning of this chapter as a possible way to provide quantitative measures for the effectiveness of the algorithm, but due to the memory problems no more than about 10 mid-complexity images can be put into the network algorithm at any one time, which would make precision and recall measures less accurate. However, if the system were to be implemented in a native, compiled language, such as C++, and the original patented version of the sieve(8) that uses a recursive-median filter to achieve the same result was used the system would probably produce results in an acceptable time, using less resources. This said, the prototype has proved, with the tests that have been run, that graph matching for content based retrieval is both a worthy and attainable goal.

5.4.1 Decision Trees

Like the network algorithm, a number of model graphs are added into a structure which is used for matching. However, rather than using a network, a compact representation of a decision tree is constructed using the adjacency matrices to represent subgraphs(13). Again the main idea of this decision tree approach is to aid retrieval of similar graphs from a database of graphs, as opposed to finding a similarity score between only two graphs.

The adjacency matrices give the structure of the graph. Two nodes $\mu, \nu \in V$ are said to be *adjacent* (denoted by $\mu \sim \nu = true$) if they are connected by an edge. The adjacency matrix is defined as:

$$M_{ij} = \begin{cases} \mu(v_i) & \text{if } i = j; \\ \mu_i \sim v_j & \text{otherwise} \end{cases} \tag{5.4}$$

The adjacency matrix M is not unique for any graph G, because if M represents G then any permutation of M will also be a valid representation of G.

At compilation time, all possible permutations of the adjacency matrix are computed and transformed into a decision tree. At run time the adjacency matrix of the input graph is used to find those adjacency matrices in the decision tree that are identical to it.

The permutation matrix is a bijective (one-to-one and complete mapping) function that maps the vertices of one graph to another such that, for the adjacency matrices M_1 and M_2 of graphs G_1 and G_2 respectively:

$$M_2 = PM_1P^T \tag{5.5}$$

That is $f(v_j) = v_i, if f m_{ij} = 1$. P and f are both graph isomorphisms between G_1 and G_2. So the graph isomorphism problem becomes a matter of finding a permutation matrix that fits the equation.

$S_{k,m}(M)$ denotes the $k \times m$ matrix given by deleting rows k up to and including n, and deleting columns m up to and including n. Subgraph isomorphism can now be given in terms of the adjacency matrices. Graphs G_1 and G_2 with adjacency matrices M_1 and M_2 of dimensions $m \times m$ and $n \times n$ where $m \leq n$, there is a subgraph isomorphism from G_1 to G_2 if there is a $n \times n$ permutation matrix such that:

$$M_1 = S_{m,m}(PMP^T) \tag{5.6}$$

So, similarly the problem of finding a subgraph isomorphism becomes a matter of finding a matrix P which fits the equation. The set of adjacency matrices can be arranged as a decision tree to help find the matrix P which fits the equation. The decision tree is built according to the row-column elements of each adjacency matrix $M_p \in A(G)$. The root of the tree is a dummy node, but the first level classification is achieved by comparing the first row-column element of the input graph to the first row-column element a_1 of each matrix $M_p \in A(G)$. The element a_1 represents the label of the first vertex in each matrix $A(G)$ so we're simply checking whether the label of the first vertex appears in the input graph. On the second level the second row-column element, a_2, of each matrix is used for the classification. So we're checking if the label and edge occurrences between the first and second vertex of the input graph appear in any model graphs. This continues throughout the decision tree. At each node in the tree, a permutation matrix is stored, and each leaf represents one of $M_p \in A(G)$.

So far, the method only allows matching of a single graph against another single graph, using a decision tree. To overcome the brute force method of just matching every graph against every other graph, the decision trees can be combined into a network, in a similar way to that of the network algorithm. The sets of adjacency matrices for all the model graphs can be treated as if they all belonged to a single graph and hence built into a single decision tree. However, these decision trees may contain some subgraphs which are duplicated - depending on the number of isomorphisms of the graph to itself (auto-morphisms). These can be associated with each other using a redirection pointer creating an interconnected tree - similar to the network in the network algorithm.

5.5 Summary

In this chapter we have shown some experimentation performed on the prototype system built for content based image retrieval using scale-trees. In section 5.3 experimentation is given that shows how the topology of query objects can be found within a set of model graphs generated from model images even if these images are not entirely similar using inexact graph matching. A discussion was presented in section 5.4 about the overall ability of the system, and why it failed at times.

We have shown that graph matching the topology of images is a useful step toward improved image matching and recognition and may help in efforts to bridge the semantic gap.

Chapter 6

Future Work and Summary

This chapter provides details of possible future directions for the work, and summarises the approach for content based retrieval given in chapters 4 and 5.

6.1 Future Work

Currently, the prototype system is able to show some of the advantages of graph based matching, and is implemented using features that provides examples of the concept of feature based graph matching. However, as the implementation stands, it is not practicable to apply the prototype as a content based retrieval system in a real application. To attain the goal where such a system may be available would require extra development, which is described below.

6.1.1 Object Trees and Semantic Networks

Working towards a semantics-based content based image retrieval engine is a very enticing goal. If the semantic gap was bridged effectively enough to allow such a retrieval system, the way we search for images could change dramatically. However, the ability to search by image topology is not enough to provide usable semantics that allow the results of a search to be made semantically relevant.

If we were able to elevate the tree matching to a stage where we could assume that this tree retrieval system was accurately retrieving objects based on their image topology, we could move forward and build possibly the next pier to support our semantic gap bridge: object trees.

We do not see extraction of emotion-based semantics as being attainable. However, the semantics of a scene based on the objects within a scene could be possible, at least semi-

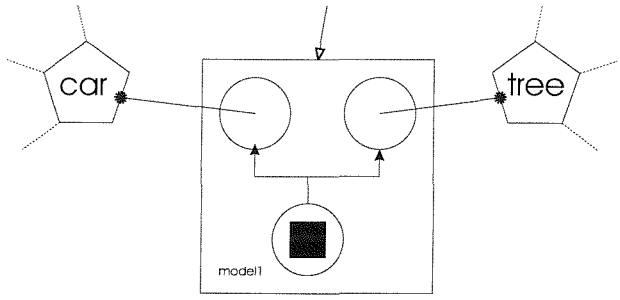


Figure 6.1: Illustration of an object tree, where recognised objects are replaced by associations with concepts in a semantic layer. Concepts will be linked in a network to other concepts (dotted lines).

automatically.

Object trees specify what objects are where within the scene, and the relationships between them. Sub-trees within graphs of images which represent objects could be replaced with concept annotations. The transformation of an image into an object tree would require a predefined knowledge base of model tree representations, for known objects, each linked to one or more semantic labels in a semantic network, perhaps similar to the *Multimedia Thesaurus* idea used in Mavis-2 (74). The illustration in 6.1 shows an annotated graph. Using a multi-layered knowledge base would allow abstract as well as concrete concepts to be directly or indirectly linked to image data.

Querying using these simple object trees could occur with a semantic (concept) based text query in a similar vein to keyword searches, except the annotations were made automatically. Because any retrieved concepts are lead-ins to subgraphs of image graphs, the realisation of that concept, potentially from any view, can be located within images, also.

From the presence of certain objects in the scene and the relationship between them, it should be possible to give some higher level semantics about the scene in general. For example, a street scene of London may contain red buses, black taxis, and people. Various logical operators and semantic network relationships could be used to fully specify this "composite concept", such as "concept contains", or multiple instance relationships. For example, for a scene to match the London composite concept, it must have 2 or more taxis that were black and at least one red bus, which could all be encoded in the relationship information. Figure 6.2 shows an illustration of a semantic layer containing composite concepts.

This networked knowledge base would then allow some relatively high level semantic retrieval. Searching for "London", for example, would find those images containing at least 2 black taxis and a red bus (by image content), which have been matched at previous levels with the tree matching algorithm.

There are a number of ways a semantic layer based retrieval system could be used, which are briefly described below:

TEXT QUERY: The query could be a text based query - such as "find picture of London" -

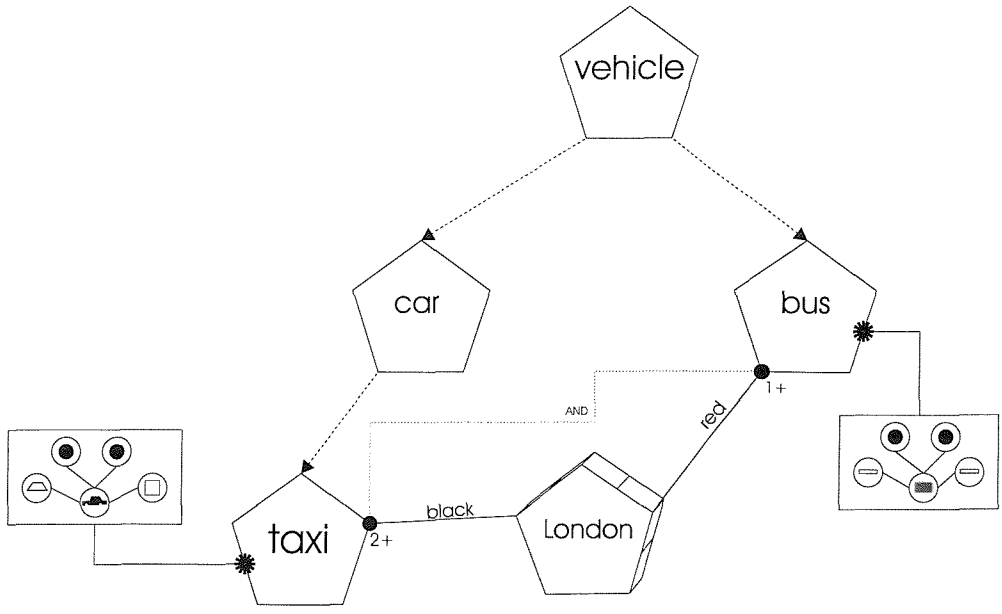


Figure 6.2: Illustration of a composite concept in the semantic layer with attributed multi-instance relationships and a logical relationship between the two relationships.

which would find those images that are associated with the composite concept "London". This concept will have been populated by semi-automatic content-based annotation.

IMAGE QUERY: The query could be an image which contains a London scene for classification. Due to the classification of the objects within the image, any red buses and black taxis will be identified. From the knowledge layer these concepts are both associated with the composite concept "London", which will return all associated images.

RELEVANCE FEEDBACK: Relevance feedback becomes possible by allowing the user to refine a search by using other related trees of the stored images to match a selection from the preliminary results of a query.

So far, the idea of the object tree semantic layer is very appealing, however, it requires the creation of associations between new models and the concepts they represent. The following subsection describes learning and how object trees might help to achieve unsupervised learning.

6.1.2 Learning

The gaining of knowledge is always based on learned material. A computer has no "built in" knowledge, and so everything it learns must be taught to it - usually by the manual annotation of data from a computer operator (the teacher). However, it is possible that such a knowledge base could be built automatically by use of a pre-existing engine that uses meta-data, such as Google image search. Relationships between images and text generated by humans and spidered from the web can be used to guide annotations of the images. Results would need

human validation (the teacher examines the result of the learning process, just like school).

It is also possible to see that the integration with the knowledge layer could go one step further - every new object introduced to the system which has been matched to a concept can be automatically inserted into the knowledge layer as a representation without the necessity for the user to explicitly create the association (unsupervised learning). In this way, the knowledge base gathers more information over time, making the results more accurate (because there are more representations of the same concepts, there is more chance that a query image will match well with one of the representations).

6.1.3 Discussion of Extensions to the Decomposition Process

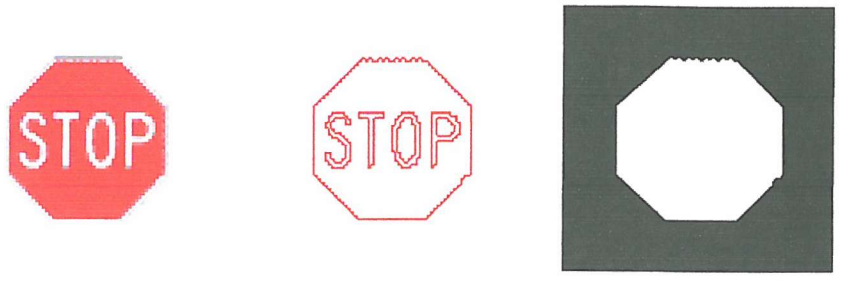
The sieve is a very useful tool for image decomposition, due to its scale-space causality preservation, and its contour preservation. This gives both hard rules for manipulation of structures based on the decomposition and accurate matching of regions. However, with developments, it could potentially be better for decomposing images into objects.

The main problem with the sieve is its inability to distinguish what is a probable object, to background noise. This causes problems with the merging operation which, due to working only in the greyscale image domain, may allow non-object regions to be merged into an object region, and cause some inconsistencies in the object contours and colours. Once the merging operation is performed, it cannot be overcome by tree manipulation. So, the only way to avoid these pitfalls is to alter the sieve process.

We need to be careful when altering the decomposition process that it does not affect the attributes of the sieve decomposition that are useful to the image matching process (scale-space causality and contour preservation). However, the decomposition needs to be generated in a manner which is more conducive to generating trees for objects.

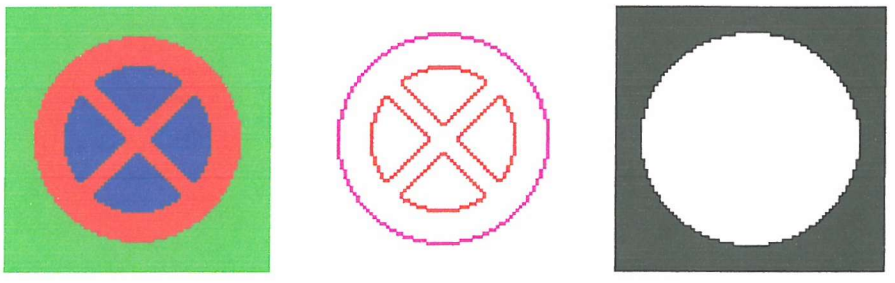
A possible way to achieve this is to limit the decomposition to regions within the image by using an edge detection on each of the channels of the image (RGB or HSV), merging the results, and limit the decompositions to areas bordered by strong edges. This differs from the sieve in that the image is effectively pre-segmented and the sieve only provides topology for the segmented objects rather than the whole image.

For example, figure 6.3(a) shows an image and figure 6.3(b) shows the corresponding edge detected image. By using a technique similar to that described in the preliminary work section on archaeological tool extraction (section 3.2) it is possible to obtain an edge detected image that has continuous edges (and therefore closed borders around regions). A thresholded edge image ensures that only the strong edges are visible within the edge image. The highest level regions (regions which are only within the background region) can be taken as the root of an object tree. The original image can then be masked by each image region in turn and the decomposition process occurs as normal within this image mask area. Figure 6.3(c) shows the highest level region mask in the example image. The scale-tree is built for that region and



(a) (b) (c)

Figure 6.3: Using edges to limit scale-space decomposition to avoid mutations.

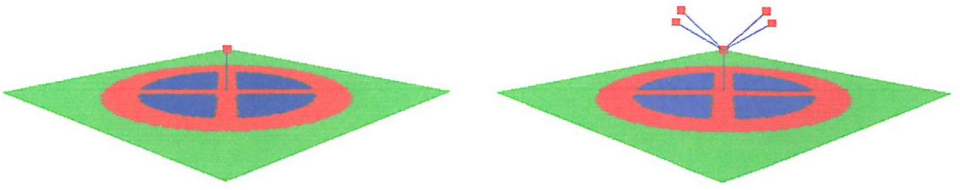


(a) (b) (c)

Figure 6.4: Using edges to limit scale-space decomposition to avoid mutations.

linked to a default background region.

However using only the largest bordered regions would cause problems with certain images such as that shown in figure 6.4(a). The blue segment-shaped regions within the larger red circular region become the same colour under greyscaling. Figure 6.5(a) shows the scale-tree for the image using the normal decomposition process; all the regions are merged during the greyscaling process into one, and therefore only one node appears in the scale-tree. So rather than using just the outer regions, the strong internal regions are also used. Thresholding the edge image will determine which edges are considered strong enough to represent pertinent regions. Starting from the smallest internal region, decomposition takes place within that region, and works outward through larger regions. Each outer region will see only the region represented by the root node of the scale-trees for the internal regions. This allows a topology to be recorded for images which merge in greyscale but actually have objects within real image space. It also avoids the noise around the edges of objects, such as that caused by JPEG compression, from merging into the regions (because the edges for the artefacts will be very weak, suppressed by thresholding, and therefore will not contribute to the mask). However, this makes it possible for a scale-tree node to be connected to a node that represents a region of



(a) (b)

Figure 6.5: The normal decomposition of the tree is shown in figure 6.5(a), and the expected output of the edge-limited decomposition is shown in figure 6.5(b).

the same level as the one being represented at that node. This does not create new extrema in the image, however, and the child node will always be at a smaller scale than that of the father node.

This kind of image decomposition is analogous to anisotropic Gaussian scale-space filtering, where the filtering is confined to avoid mutations to unconnected areas of the image.

6.1.4 Implementation Issues

The current implementation is 100% Java, and as such suffers to some degree. The decision was made originally to implement the prototype in Java so that it could be used with other tools we use in the Southampton Lab, as well as the extra facilities it has built-in for image loading and manipulation, and system prototyping. To use this graph-based matching system in a real-world environment would require some redevelopment of the software, to increase its response time, and make it have a smaller footprint when running. This could be achieved by reimplementation into C++, or a similar natively compiled language.

The graph matching is currently based on the network algorithm developed by Messmer and Bunke. Although this method provided a good basis on which to base a new feature based graph matching algorithm, a different graph matching technique may be more appropriate. In particular, a new algorithm by those authors, described in section 5.4.1, provides total subgraph coverage, while also providing fast graph matching based on shared common subgraphs.

The features used in this project were relatively simple, and although this would seem a disadvantage, it is probably not worth making the features used to match vertex labels more complex. Providing discriminating feature modules would compromise the graph matching speed, while also being slower to calculate features for regions. Keeping the features simple allows larger common subgraphs to be shared, while also ensuring the speed of matching is fast. The best extension to this system would be to include a shape matching module that was discriminating between various shapes, yet fuzzy enough to provide matches between similar shapes. The shape matching integrated into the current system is too basic to accurately distinguish between detailed shapes, whereas the moment based matching only allowed distance

calculations between a query and a known set of shapes.

6.1.5 Application Integration

At the beginning of this project, the idea was to integrate the new content based image retrieval system with Mavis-2. Mavis (Multimedia Architecture for Video Image and Sound) (83; 44) was a research programme undertaken in the University of Southampton to bring generic links into non-text media. Separate modules were responsible for dealing with the processing of a particular media type and as new modules are developed they may be added to the architecture. Mavis-2(19; 17; 43; 74), a follow-up to the Mavis project, was funded by the EPSRC and introduced the "multimedia thesaurus", and a portable Java implementation to the ideas developed in the Mavis project.

Mavis-2 is an ideal test bed for new content based retrieval techniques. Because of the architecture, new algorithms can be added to the system and tested alongside other content based retrieval algorithms, and compared to them. Also, there is the ability to make use of the multimedia thesaurus, which In the long run, the MMT could be used as the knowledge layer to represent scene concepts for object tree recognition, as described in section 6.1.1.

Integration with MAVIS requires the building of a signature module with the correct interface to communicate with MAVIS. It registers itself as a module that is able to match images with the MAVIS broker, which distributes queries and database items to the registered modules. The module is expected to return a list of match distances between the query image and each match image.

The module functionality is not integrated with the other functions of MAVIS, in particular the semantic layer, but affecting other modules from a signature module can be achieved if an appropriate API to other modules is available (such as the semantic layer API).

Although the MAVIS-2 project finished a few years before this thesis, the integration of this system may provide a useful tool for other users. However, it would require that this system still be implemented in Java, which has shown to be too slow for general use.

Other potential applications into which a faster implementation of the system could be integrated include the projects the IAM Group at Southampton are partners in: Artiste (an integrated art analysis and navigation environment) (15), or Sculpteur (Semantic and content-based multimedia exploitation for European benefit) (16). Both projects have a high degree of content-based image analysis functionality. Artiste has goals which allow for sub-image retrieval, something that a graph matching system may make far more intuitive. Sculpteur has goals for semantic retrieval and meta-data augmentation of images and 3D models from a database, which is planned to be achieved with web crawling of trusted web sites. It is possible that using a semantic layer approach with graph based representations of images particular meta-data augmentation could be achieved semi-automatically, while also having the ability to provide sub-image, and object matching. Unfortunately this work culminated too late in the

Artiste project for it to be integrated, while the majority of work in Sculpteur is now moving towards 3D-model based matching.

Of course, a system such as this can also be useful within its own right - providing client based image matching on personal image collections, or on web-based images.

6.2 Summary

During the development of a new technology, the very first method devised for using it often persists until innovative research can provide a more adequate solution; take the motor car as an example. That was originally controlled using a joystick device, before the development of the steering wheel. The sudden growth in image manipulation on computers occurred due to the speed and inexpensive new silicon technologies, however, despite over a decade of affordable computer image manipulation, most of the search algorithms that are available are still based on the idea of query by example using colour and texture feature vectors. Of course, the research field is under-going rapid changes, but until a truly useful, and workable, solution to image searching is developed, the colour and texture features will continue to be the foremost technique.

In this thesis we sought to move from simple feature based retrieval to the use of structured objects in the form of scale-trees. Semantics are very important to users of an image retrieval system. Searching by concept is a far more intuitive way of information mining than query-by-example, particularly if the user does not have an example to hand, or there is no specific example case. Some of the systems already available achieve this with simple textual metadata terms associated with database images, which all have to be inserted manually. Google image search is the only automatic meta-data augmentation system, however, it does not provide content-based image retrieval; it relies on surrounding text in web pages to provide possible meta-data terms which is, of course, no use for simple image databases with no associated textual data.

The idea presented here overcomes problems with simple low level feature matching of images by providing a topology structure representing regions within the image over different scales. It seemed a reasonable assumption that certain objects will always have a similar topology of regions within the image - it is what gives them their appearance - and so by encoding this we could search for the topology within the topology of other images.

In chapter 4 we provide the basis for a novel system, using a filtering mechanism called the sieve to generate a tree structure for an image, and using a graph matching algorithm, called the network algorithm, to provide isomorphism testing between tree structures.

The sieve is a scale-space filter that generates a tree structure based on the scale and inclusion of regions during a defined scale-based merging operation. In the literature the tree is built after the sieve is complete, but we combine the two processes by providing a new way to decompose the image (yet still adhering to the sieve functionality). The advantages of using the

sieve are that the generated structure has definite rules by which it can be manipulated. The tree structure always moves towards smaller regions at the nodes, and allows detection and removal of noise and blurring relatively easily. The enforced structure means that we are able to match trees without any transformations. The trees that are generated are relatively invariant to scale, rotation and translation, and are therefore ideal for object matching where target objects are not guaranteed to be in any particular position. During the testing of the prototype system, we found that the enforced rules of the decomposition can cause problems during the matching process - in particular, when objects get merged into background regions. In section 6.1.3 an overview is given of a possible way to suppress the effect.

The network algorithm is an attempt at providing graph matching without having the very expensive cost of brute force graph matching (which is NP-complete). It is also advantageous by allowing matching against a set of model objects, unlike other graph and tree matching algorithms that only operate on a one-to-one basis. It avoids being NP-complete and provides a speed-up (other than in the worst case) by using a network of common subgraphs of the set of models. This assumes that the models will share some common subgraphs. To allow matching of the topology of regions, the network algorithm incorporates some low level feature matching algorithms. This is a weakness in the algorithm, as if the algorithms are not accurate enough the network can become too large (or too small), and topologies incorrectly matched. However, an inexact network algorithm allows graph edits to be made that are based on the feature scores, allowing the nearest, yet different topology to be matched best. The main weakness with the network algorithm is its lack of coverage of the subgraphs in the models; that is, it does not encode every subgraph in every model. Although this was originally not seen as a major problem because it was foreseen that the inexact network algorithm would allow editing of the graph to allow models to match, it turns out that it can be particularly cumbersome in certain circumstances. A new decision tree approach to graph matching appeared in the literature during the execution of this thesis, and has been described in section 5.4.1.

The experimentation in chapter 5 showed that graph matching for object retrieval is a possibility. We described how the tree structures can be pruned and the effects it has on the images, and also explained how the graph matching can be used to find similar topology images. In section 5.4 we discussed the problems with the method, generally associated with the use of the sieve and the network algorithm.

Matching tree topologies is not enough to bridge the semantic gap, however. Concept retrieval is the ultimate goal, where concepts can be retrieved directly from a semantic network. Of course, such a network would need to be built manually, effectively copying parts of the semantic network that is built into the network author's brain. However, once active and interconnected with a database of subgraphs, it could be used to automatically detect objects from scenes, retrieving meta-data about those objects. If the semantic network is augmented in with composite concepts, whole scenes from images could be automatically annotated. An illustration of this process was explained in this section.

From the success of the preliminary results presented in chapter 5 it is believed that topology matching of objects in images is a worthy goal towards the semantic retrieval of images. Using a semantic layer it is possible that a system based on topology matching could bridge the semantic gap. With continued development in this new field of graph matching of topology trees, techniques for better segmentation and graph matching could provide the object tree ideal presenting an exciting range of new possibilities for image matching.

References

[1] S. Abbasi and F. Mokhtarian. Robustness of shape similairity retrieval under affine transformations. In *Proceedings of the Challenge of Image Retrieval (CIR'99)*, pages 41–50, Newcastle, UK, February 1999.

[2] S. Abbasi, F. Mokhtarian, and J. Kittler. Reliable classification of chrysanthemum leaves through curvature scale space. In *Proceedings of the Scale-Space '97 Conference*, pages 284–295, Utrecht, Germany, 1997.

[3] A.S. Aguado, M.E. Montiel, and M.S. Nixon. Extracting arbitrary geometric primitives represented by Fourier descriptors. In *Proceedings International Conference on Pattern Recognition ICPR '96*, volume B, pages 547–551, Vienna, 1996. IEEE.

[4] A.S. Aguado, M.S. Nixon, and M.E. Montiel. Parameterising arbitrary shapes via Fourier descriptors for evidence-gathering extraction. *CVGIP: Image Understanding*, 69(2):202–221, 1998.

[5] P. Alshuth, Th. Hermes, J. Kreyß, et al. Image-MINER - intelligent retrieval for images and videos. In A.W.M. Smeulders and R. Jain, editors, *Image Databases and Multimedia Search*, pages 241–251, 1997.

[6] Jeffrey R. Bach, Charles Fuller, Amarnath Gupta, Arun Hampapur, Bradley Horowitz, Rich Humphrey, Ramesh Jain, and Chiao-Fe Shu. Virage image search engine: An open framework for image management. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 76–87, 1996.

[7] J. A. Bangham, K. Moravac, R.W. Harvey, and M.H. Fisher. Scale-space trees and applications as filters, for stereo vision and image retrieval. In *10th British Machine Vision Conference (BMVC99)*, Nottingham, UK, 1999.

[8] J.A. Bangham. U.S. patent for Data processing method and apparatus, June 1999. Application Number: 875055.

[9] J.A. Bangham, R. Harvey, P.D. Ling, and R.V. Aldridge. Morphological scale-space preserving transforms in many dimensions. *Journal of Electronic Imaging*, 5(3):283–299, July 1996.

[10] J.A. Bangham, J.R. Hidalgo, and R. Harvey. Robust morphological scale-space trees. In *Proc. Noblesse Workshop on Non-Linear Model-Based Image Analysis (NMBIA'98)*, pages 133–139, Glasgow, July 1998.

[11] J.A. Bangham, J.R. Hidalgo, R. Harvey, and G. Cawley. The segmentation of images via scale-space trees. In *Proc. British Machine Vision Conference 1998 (BMVC'98)*, pages 33–43, Southampton, September 1998.

[12] S. Brande and I Saragusti. A morphometric model and landmark analysis of acheulian handaxes from northern israel. *Advances in Morphometrics*, pages 423–435, 1996.

[13] H. Bunke. Error correcting graph matching: On the influence of the underlying cost function. *IEEE transactions on pattern analysis and machine intelligence*, 21(9):917–922, September 1999.

[14] S. Chan, K. Martinez, P. Lewis, C. Lahanier, and J. Stevenson. Handling sub-image queries in content-based retrieval of high resolution art images. In *Proceedings of ICHIM*, volume 2, pages 157–163, Milan, 2001.

[15] Artiste Consortium. *The Artiste Project: An Integrated Art Analysis and Navigation Environment*. Website: http://www.artisteweb.org/, 1999–2002.

[16] Sculpteur Consortium. *The Sculpteur Project: Semantic and Content-Based Multimedia Exploration for European Benefit*. Website: http://www.sculpteurweb.org/, 2002-2005.

[17] M Dobie, R. Tansley, D. Joyce, M. Weal, P. Lewis, and W. Hall. A flexible architecture for content and concept based multimedia information exploration. In *Proceedings of the Challenge of Image Retrieval (CIR'99)*, pages 99–110, Newcastle, UK, February 1999. Also available from http://www.iam.ecs.soton.ac.uk/publications.

[18] Mark R. Dobie, Robert H. Tansley, Dan W. Joyce, Mark J. Weal, Paul H. Lewis, and Wendy Hall. A flexible architecture for content and concept based multimedia information exploration. In David J. Harper and John P. Eakins, editors, *The Challenge of Image Retrieval, Newcastle, 1999*, pages 1–12. University of Newcastle, February 1999.

[19] Mark R. Dobie, Robert H. Tansley, Dan W. Joyce, Mark J. Weal, Paul H. Lewis, and Wendy Hall. Mavis 2: A new approach to content and concept based navigation. In *Proceedings of the IEE Colloquim on Multimedia Databases and MPEG-7*, volume 99, pages 9/1–9/5, London, UK, January 1999.

[20] N. Duffy and G. Lacey. Colour profiling using multiple colour spaces. In *Proceedings of the British Machine Vision Conference*, volume 1, pages 245–255, Southampton, UK, 1998.

[21] D. Dupplaw, P. Lewis, and M. Dobie. Spatial colour matching for content based retrieval and navigation. In *Proceedings of the Challenge of Image Retrieval (CIR'99)*, pages 129–131, Newcastle, UK, February 1999. Also available from http://www.iam.ecs.soton.ac.uk/publications.

[22] J. Eakins and M. Graham. Content-based image retrieval: A report to the JISC technology applications programme. Technical Report JTAP-039, Institute for Image Data Research, University of Northumbria at Newcastle, January 1999.

[23] eVision Global. A new vision for internet search - a technical white paper from eVision. Technical report, Available from http://www.evisionglobal.com/, July 2001.

[24] G. Finlayson and Gui Yun Tian. Colour indexing across illumination. In *Proceedings of the Challenge of Image Retrieval (CIR'99)*, pages 127–128, Newcastle, UK, February 1999.

[25] M. Flickner, H. Sawhney, W. Niblack, et al. Query by image and video content: The QBIC system. *Computer*, 28(9):23–32, September 1995.

[26] D. Forsyth, J. Malik, and R. Wilensky. Searching for digital pictures. *Scientific American*, pages 72–77, July 1997.

[27] J. Gero and J. Mazzullo. Analysis of artifact shape using Fourier series in closed form. *Journal of Field Archeology*, 11:315–322, 1984.

[28] T. Gevers and A.W.M. Smeulders. PicToSeek: A color invariant retrieval system. In A.W.M. Smeulders and R. Jain, editors, *Image Databases and Multimedia Search*, pages 25–37, 1997.

[29] J. Hafner, H.S. Sawhney, W. Equitz, M. Flickner, and W. Niblack. Efficient color histogram indexing for quadratic form distance functions. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 17, pages 729–736, July 1997.

[30] James Hafner, Harpreet Sawhney, Will Equitz, Myron Flickner, and Wayne Niblack. Efficient color histogram indexing for quadratic form distance functions. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 17, pages 729–736, July 1995.

[31] A. Hampapur, A. Gupta, B. Horowitz, et al. Virage video engine. In I.K. Sethi and R.C. Jain, editors, *Storage and Retrieval for Image and Video Databases V*, pages 188–197, 1997.

[32] R. Harvey, J.A. Bangham, and A. Bosson. Some morphological scale-space filters and their properties. In ..., November 1996.

[33] Javier Ruiz Hidalgo. The representation of images using scale trees, 1998. Master Thesis.

[34] M.K Hu. Visual pattern recognition by moment invariants. In *IRA Transactions on Information Theory*, volume 17-8, pages 179–187, Feburary 1962.

[35] J. Huang, S.R. Kumar, M. Mitra, W.J. Zhu, and R. Zabih. Image indexing using color correlograms. In *Computer Vision and Pattern Recognition 1997 (CVPR97)*, pages 762–768, 1997.

[36] T. Kato. Database architecture for content-based image retrieval. In *SPIE*, volume 1662, pages 112–123, 1992.

[37] R.G. Klein. *The Human career: Human Biological and cultural origins, second edition.* University of Chicago press, Chicago, 1999.

[38] Michael Kliot and Ehud Rivlin. Invariant-based shape retrieval in pictorial databases. *Computer Vision and Image Understanding: CVIU*, 71(2):182–197, 1998.

[39] J.J. Koenderink. The structure of images. *Biol. Cyb.*, 50:363–370.

[40] A. Lenaghan, R. Malyan, and G.A. Jones. Matching structural descriptions of handwritten characters using heuristic graph search. In *Proceedings of IEE Third European Conference on Handwriting Analysis and Recognition*, pages 10/1–10/4, Brussels, July 1998.

[41] K-S. Leung and R. Ng. Multiresolution subimage similarity matching for large image databases. In *Proceedings of SPIE Storage and Retrieval for Image and Video Databases VI*, volume 3312, pages 259–270, San Jose, California, US, January 1998.

[42] Man-Wai Leung and Kwok-Leung Chan. Object-based image retrieval using heirarchical shape descriptor. In *Proceedings of CIVR 2002*, pages 165–174, London, July 2002. Springer.

[43] P.H. Lewis, H.C. Davis, M.R. Dobie, and W Hall. Using image content for retrieval and navigation. In *Journal of History and Computing*, volume 10, pages 112–119, 1999.

[44] PH Lewis, HC Davis, SR Griffiths, W Hall, and RJ Wilkins. Content based retrieval and navigation with images in the microcosm model. In *International Conference on Multimedia Communications*, pages 86–90, 1995.

[45] J. Matas, R. Marik, and J. Kittler. Illumination invariant colour recognition. In *Proceedings of the British Machine Vision Conference*, Birmingham, September 1995.

[46] B. Messmer and H. Bunke. *GUB: The Graph Matching Toolkit.* Available at http://www.iam.unibe.ch/ fki/projects/GraphMatch.html, 1996.

[47] Bruno T. Messmer. *Efficient Graph Matching Algorithms*. PhD thesis, University of Bern, Switzerland, 1995.

[48] Bruno T. Messmer and H. Bunke. A network based approach to exact and inexact graph matching. Tech. Rep. IAM 93-021, Institut für Informatik, Universität Bern, Switzerland, September 1993.

[49] B.T. Messmer and H. Bunke. Subgraph isomorphism in polynomial time. Technischer Bericht IAM 95-003, Institut für Informatik, Universität Bern, Schweiz, 1995.

[50] B.T. Messmer and H. Bunke. Fast error-correcting graph isomorphism based on model precompilation. Technischer Bericht IAM-96-012, Institut für Informatik, Universität Bern, Schweiz, 1996. Filed: no, PSFile: no.

[51] B.T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–503, May 1998.

[52] B.T. Messmer and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognition*, 32:1979–1998, 1999.

[53] F. Mokhtarian and J. Abbasi S., Kittler. Robust and efficient shape indexing through curvature scale space. In *Proceedings of the sixth British Machine Vision Conference on Multimedia Computing and Systems*, pages 53–62, Edinburgh, UK, 1996.

[54] W. Niblack, R. Barber, W. Equitz, et al. The QBIC project: Querying images by content using color, texture, and shape. *Computer Science*, RJ9203, February 1993.

[55] W. Niblack, Z. Xiaoming, J.L. Hafner, et al. Updates to the QBIC system. In I.K. Sethi and R.C. Jain, editors, *Storage and Retrieval for Image and Video Databases V*, pages 150–161, 1997.

[56] Nils Nilson. *Principles of artificial intelligence*. Tioga Publishing Co., Palo Alto CA, 1980.

[57] I. Park, I. Yun, and S. Lee. Color image retrieval using hybrid graph representation. In *IVC*, volume 17, pages 465–474, 1999.

[58] G. Pass, R. Zabih, and J. Miller. Comparing images using color coherence vectors. In *Proceedings of the Fourth ACM International Conference on Multimedia '96*, pages 65–73, Boston, MA, US, November 1996.

[59] P. Perona and J. Malik. Scale space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 12(1):629–639, 1990.

[60] S. Perry and P. Lewis. Novel image viewer providing fast object delineation for content based retrieval and navigation. In *SPIE Conference on Storage and Retrieval for Image and Video Databases VI.*, volume 3312, pages 436–445. SPIE, January 1998.

[61] E. Persoon and K. Fu. Shape discrimination using Fourier descriptors. In *IEEE Transactions on Systems, Man, and Cybernetics*, volume 7, pages 170–179, 1977.

[62] S. Pizer and L. Lifshitz. A multiresolution, hierarchical approach to image segmentation based on intensity extrema. *IEEE transaction on Pattern Analysis and Machine Intelligence*, 12(9):529–540, 1990.

[63] Kriengkrai Porkaew, Michael Ortega, and Sharad Mehrotra. Query reformulation for content based multimedia retrieval in MARS. In *ICMCS, Vol. 2*, pages 747–751, 1999.

[64] Yong Rui and Thomas S. Huang. Image retrieval: Current techniques, promising directions, and open issues. *Journal of Visual Communications and Image Representation*, 10:39–62, 1999.

[65] P. Salembier and L. Garrido. Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. In *IEEE Transactions on Image Processing*, volume 9, pages 561–576, 2000.

[66] I. Saragusti and I. Sharon. Quantitative analysis of the symmetry of artefacts: Lower paleothic handaxes. *Journal of Archeological Science*, 25:817–825, 1998.

[67] I.K. Sethi, I. Coman, B. Day, et al. Color-WISE: A system for image similarity retrieval using color. In I.K. Sethi and R.C. Jain, editors, *Storage and Retrieval for Image and Video Databases V*, pages 140–149, 1997.

[68] K. Siddiqi, A. Shokoufandeh, S.J. Dickinson, and W. Zucker. Shock graphs and shape matching. In *Proceedings of International Conference on Computer Vision*, pages 222–229, Bombay, India, 1998.

[69] A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 22, pages 1349–1360, December 2000.

[70] J. Smith and S. Chang. VisualSEEk: A fully automated content-based image query system. *ACM Multimedia 1996*, pages 87–98, November 1996.

[71] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Thompson Computer Press, 1996.

[72] G. Sprigg (ed.). Fourier transform and waste recycling. *Image Processing Europe*, pages 30–32, March/April 1999.

[73] Michael Swain and Dana Ballard. Color indexing. *International Journal of Computer Vision*, 7(11):11–32, 1991.

[74] Robert Tansley. *The Multimedia Thesaurus: Adding A Semantic Layer to Multimedia Information*. PhD thesis, University of Southampton, August 2000.

[75] Robert Tansley, Colin Bird, Wendy Hall, Paul Lewis, and Mark Weal. Automating the linking of content and concept. In *Proceedings ACM Multimedia 2000*, pages 445–448, New York, 2000. ACM Press.

[76] G. Toussaint. *Introduction to Pattern Recognition*. Available on the web from http://cgm.cs.mcgill.ca/ godfried/teaching/pr-notes/, 1997.

[77] T.Pavlidis. A review of algorithms for shape analysis. In *Proceedings of Computer Graphics and Image Processing*, volume 7, pages 243–258, 1978.

[78] C. Venters and M. Cooper. A review of content-based image retrieval systems. Technical Report JTAP-054, JISC Technology Application Program, 2000.

[79] P. Walcott and T. Ellis. A colour object search algorithm. In *Proceedings of the British Machine Vision Conference*, volume 1, pages 296–305, Southampton, UK, 1998.

[80] Wei Wang, Yuqing Song, and Aidong Zhang. Semantics-based image retrieval by region saliency. In *Proceedings of CIVR 2002*, pages 29–37, London, July 2002. Springer.

[81] J. Wei, M. Drew, and Z. Li. Illuminant invariant video segmentation by heirarchical robust thresholding. In *Proceedings of SPIE: Storage and Retrieval for Image and Video Databases VI*, volume 3312, pages 188–201, San Jose, California, US, 1998.

[82] J. Weickert and B. Benhamouda. Why the Perona-Malik filter works. Technical Report DIKU-TR-97/22, Department of Computer Science, University of Copenhagen, 1997.

[83] RJ Wilkins, S Griffiths, PH Lewis, W Hall, and HC Davis. The mavis microcosm extensions for content based navigation and retrieval. In *Proceedings of the LIRM International Workshop on Hypermedia Design*, pages 77–80, Montpelier, France, June 1995.

[84] Andrew P. Witkin. Scale-space filtering. In *Proceedings of 8th International Conference on Artificial Intelligence*, pages 1019–1022, Germany, August 1983.

[85] Andrew P. Witkin. Scale-space filtering. *From Pixels to Predicates*, pages 5–19, 1986.

[86] H. Zabrodsky and D. Avnir. Continuous symmetry measures: Chirality. *Journal American Chemistry Society*, 117:462–473, 1995.