University of Edinburgh
Division of Informatics

Fast Colour Based Hierarchical Image Region Segmentation

4th Year Project Report
Artificial Intelligence and Mathematics

Jonathan Betts

May 26, 2004

Abstract: This paper presents a fast colour based image segmentation technique based on recursive application of a modified version of the K-Means clustering approach. The algorithm works on multiple scales to produce a hierarchical representation and reduce computational demands. Several adaptations to the traditional K-Means algorithm are presented to extend it to the segmentation of multichannel images and to enhance performance at the cost of accuracy. The effects of these extensions and overall performance are considered in terms of speed, subjective performance and correlation to hand segmented images.

Acknowledgements

Bob Fisher for is guidance and patience.

Penny, Sally and Dave for their perseverance.

And as always Tor.

Contents

1 Introduction 1
- 1.1 Dissertation Outline . . . 1
- 1.2 Applications . . . 1
- 1.3 The Segmentation Problem . . . 2
  - 1.3.1 Complications . . . 3

2 Current Techniques 5
- 2.1 Flat Methods . . . 5
  - 2.1.1 Edge Detection . . . 5
  - 2.1.2 Region Growing . . . 7
  - 2.1.3 Clustering . . . 9
- 2.2 Hierarchical Methods . . . 10
  - 2.2.1 Split and Merge . . . 11
  - 2.2.2 Graph Based . . . 12
- 2.3 Pre/Post processing . . . 13
  - 2.3.1 Common Filters . . . 13
  - 2.3.2 Convolution . . . 15
  - 2.3.3 Edge Preserving Blurs . . . 15
  - 2.3.4 Mathematical Morphologies . . . 17

3 Flat Methods 19
- 3.1 Region Growing . . . 19
  - 3.1.1 Random / Modulo Arithmetic Seeding . . . 21
  - 3.1.2 The Homogeneity Criterion and Distance Measures . . . 22
  - 3.1.3 Symmetric Nearest Neighbour (SNN) . . . 23
- 3.2 K-Means . . . 26
  - 3.2.1 Distance Measures . . . 30
  - 3.2.2 Random / Spaced / Region Growing Seeding . . . 31
  - 3.2.3 The Effect of Iteration . . . 33
  - 3.2.4 ID Filtering . . . 35
  - 3.2.5 spatial / Non spatial Colour Merging . . . 37
- 3.3 Evaluation . . . 41
  - 3.3.1 Speed . . . 41
  - 3.3.2 Comparison with Hand Segmentations . . . 47
  - 3.3.3 Subjective Comparison . . . 52

4 Hierarchical Methods 59
4.1 Graph Based Hierarchical Segmenter 59
4.2 Recursive Application Two Phase K-Means Hierarchical Segmenter 62
4.3 Evaluation 64
4.3.1 Speed 64
4.3.2 Subjective Comparison 65

5 Conclusion 71

A General Appendix 73
A.1 Proof of Modulo Arithmetic Traversal 73
A.2 Results of the Flat Methods Speed Tests 74
A.3 Results of Speed Test of Varying k 75
A.4 5 Hand Segmentations 75
A.5 Results of the WSCM Tests 78
A.6 Results of the Hierarchical Speed Tests 80

B Code Appendix 81
B.1 KMeans 81
B.2 TwoPhaseKMeans 87
B.3 HierarchicalTwoPhaseKMeans 88
B.4 RegionGrower 91
B.5 AveragingRegionGrower 93

Bibliography 95

1. Introduction

1.1 Dissertation Outline

Two flat segmentation techniques have been developed in this project: The Two Phase K-Means segmenter (**Section 3.2**) based upon the K-Means procedure and the Region Grower (**Section 3.1**). Subjective results of the processes can be observed in **Section 3.3.3** which illustrate the impressive object centric segmentations that can be produced with suitable settings.

Limiting factors of the K-Means approach such as the output of a fixed k segments has been addressed though the joint application of Spatial and Non-Spatial Merging to provide more connected and representative regions (see **Section 3.2.5**).

Additionally filtering methods appropriate to each methods are presented to help each cope with noise. Symmetric Nearest Neighbour (**Section 3.1.3**) for Region Growing and ID Filtering (**Section 3.2.4**) for K-Means.

Both techniques require seeds to initialise parts of the algorithm. In both cases seeding schemes have been developed to increase the quality of the segmentations produced. In the Region Growing technique an image traversal based on modulo arithmetic is presented to avoid generate and test solutions and reduce runtime(**Section 3.1.1**). In the case of K-Means a technique was developed to improve the spacing of the seeds produced and so improve the quality of the segmentation (**Section 3.2.2**).

Both of these techniques were then extended to deal with the hierarchical segmentation problem. The Hierarchical Two Phase K-Means (**Section 4.2**) was developed to tackle the problem of generating hierarchies based on features at different scales and the Graph Based segmenter (**Section 4.1**) was developed to generate hierarchies based on the containment of regions by others.

Evaluations of the speed and quality of all of these methods can be observed in the relevant sections.

1.2 Applications

Region segmentation as a representation of an image is of great interest in a number of fields due to the object centric nature and compactness of the representation. Recently hierarchical segmentation methods have been applied to video compression techniques [6], often based on arranging information about

the image at different levels of abstraction in a tree structure [4], with higher levels of abstraction close to the root. These methods are primarily concerned with achieving maximum fidelity at the lowest possible bit rates, harnessing the structural information produced by segmentation.

Segmentation techniques, the most basic being thresholding, are frequently used in all manner of visual systems and often play the role of low to mid level processing. Segmentation of an object from background, non-object areas of an image, is used extensively in automatic video surveillance and object tracking systems, with application in both robotics and security. Mobile robotic systems in particular are often concerned with object identification, with a view to navigation and avoidance [5]. In this context it is imperative that the object extraction is performed at a rate that is comparable to the speed of the world to facilitate sensible reaction times.

Visual attention models attempt to explain the shifting of attention of the relatively small but sensitive fovea area in the eye to extract important information from a larger image. Many models have been proposed that are concerned with movement between feature points in the image, often corners or other single points that meet some criterion. More recently it has been proposed that visual attention is more preoccupied with areas, or segments of interest, rather than points; in particular moving from areas to sub areas at multiple scales. Hierarchical segmentation, particularly at high speed, is therefore a necessary prerequisite to a complete vision system.

Since the advent of satellite imagery intense interest has arisen in the segmentation procedure as the first stage in automatic object detection. Many scientific, military and political applications exist in satellite image recognition from identifying military vehicles and troops to analysing river swell and crop sizes.

Another area of enormous interest is that of automatic processing of medical imaging data to produce an interactive and intuitive representation, to facilitate the physicians understanding of a possibly large data set. A crucial first step to this process is the classification of regions of interest. Other medical applications include cell counting on slides, atlas construction and disease diagnosis.

1.3 The Segmentation Problem

The classic segmentation problem is, given a source image I and homogeneity criterion H where;

$$H(R) = \begin{cases} true, & \text{if } R \text{ is Homogenenous} \\ false, & \text{otherwise} \end{cases}$$

Find a segmentation S of the image into connected regions $R_0 ... R_n$ such that:

1. $H(R_i), \forall i$
2. $R_i \cap R_j = \oslash, \forall i \neq j$
3. $H(R_i \cup R_j) \wedge adj(R_i, R_j) \longrightarrow i = j$
4. $\bigcup_{i=0}^{n}(R_i) = I$

Stating that each region must be homogeneous(1), all regions are distinct(2), no two adjacent regions should form one larger homogeneous region(3) and that the segmentation should cover the entire image(4).

Another possible formulation of the segmentation problem is to state that a segmentation should divide the image into regions of interest. Working from a human centric view, as is inherent in the formulation of the colour spaces and image file formats, we would wish our segmentation to approach the areas that a human might highlight as important or distinct.

1.3.1 Complications

There are a great deal of complications associated with this simple outline of the problem which contribute to the difficulty of the segmentation problem.

- **Semantic**

  What is a homogeneous region? Is it specified by local image level data such as pixel intensity, or is it defined by higher level statistical data such as texture? Often one would wish the segmentation to be comprised of regions which include objects, whereby the homogeneity criteria is defined as some measure of an areas correlation to a particular object; requiring knowledge outside that contained simply within the image.

- **Human Centric**

  Humans recognise regions which are not necessarily connected; a dense pattern of dots for example. So should these be considered a single region or should the classic description be adhered to?

- **Uniqueness**

  This formulation of the problem makes no claims about the uniqueness of a particular solution. So are all segmentations according to the same homogeneity criterion equally valid? For that matter how can the quality of a particular segmentation be quantitatively assessed at all?

2. Current Techniques

2.1 Flat Methods

There are a wide variety of algorithms concerned with the flat segmentation problem. In this context flat implies that while the implementation of an algorithm may rely on recursion or tree based methods its output is a single flat segmentation which uniquely attributes each pixel to a segment without any concept of hierarchy or connectedness (although many attempt to respect this).

Broadly all segmentation techniques fall into several categories [3]:

- Edge Detection
- Region Growing
- Clustering
- Split and Merge

Note: Split and Merge is covered in **2.2 Hierarchical Methods**.

2.1.1 Edge Detection

Edges in an image are areas of high discontinuity in the frequency domain. The interest in finding these edges for the application of segmentation is that they often lie along the boundaries of regions. The basic approach of an edge detecting segmenter is to identify the edges using a suitable preprocessing step and then to link these edges so as to form region boundaries.

An example of the edge detection process can be seen in **Figure 2.1** where an example image edges are detected and thresholded in an attempt to isolate

Figure 2.1: An image of a helicopter (left) with edge detection applied (centre) and thresholded (right).

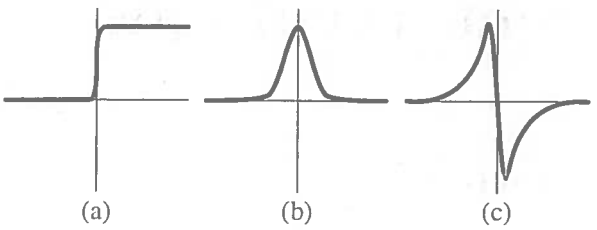


Figure 2.2: Zero crossings, here we see an edge in intensity values of the image (a), the gradient of these values (b) and the second derivative (c) showing a zero at the edge location.

sufficiently strong edges.

Zero Crossings

Many edge detectors work on the basis of finding zero crossings in the second derivative of the intensity values of the image. **Figure 2.2** (a) shows a large change in the intensity values in the source image, which the process is attempting to classify as an edge. The one dimensional derivative of the intensity values can be seen in (b) where the peak indicates the location of the edge. In the second derivative (c) this is indicated by the zero crossing point.

The Laplacian and Laplacian of Gaussians

Theoretically this can be implemented as the Laplacian of the image. The Laplacian $L(x, y)$ of an image of intensity $I(x, y)$ is given by:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

The Laplacian

As the Laplacian is often strongly effected by noise, it is common to apply a Gaussian filter first. As these effects can both be approximated using convolutions, which are additive, it is also common to compose a kernel which performs both Gaussian filtering and the Laplacian operator in one pass. This approach is known as the Laplacian of Gaussians. Such kernels are an approximation to the equation:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right] e^{\frac{x^2+y^2}{2\sigma^2}}$$

The Laplacian of Gaussians with a standard deviation of σ.

Whilst very accurate, the Laplacian of Gaussians approach only yields an approximation to the second derivative which must be further processed to give the location of the zero crossings as opposed to zero regions where no edges occur.

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

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

Figure 2.3: The Sobel G_x (left) and G_y (right) edge detecting kernels [7].

Convolution based approaches

In practice edge detection is often implemented as two linear convolution filters which approximate the spatial gradient in orthogonal directions. If these convolutions are G_x and G_y then the magnitude of the gradient at a point can be calculated as $|G| = \sqrt{G_x^2 + G_y^2}$. Which is commonly approximated to $|G| = |G_x| + |G_y|$ to increase speed and thresholded to remove weak edges. The common Sobel edge detection kernels for G_x and G_y can be seen in **Figure 2.3**.

Edge based approaches problems

Unfortunately edges frequently occur in the interior of regions due to texture and noise and additional processing is required to attempt to reconstruct the regions from the edges without including these false edges. One approach to this is hysteresis tracking [1], whereby weak edges are only permitted to contribute to real edges if they are adjacent to high magnitude edges.

Due to the prolific nature of edges in images, and the addition processing requirements to take an edge detected image to regions, pure edge detection methods have not been applied in this project.

2.1.2 Region Growing

Region growing is a local approach to segmentation whereby regions are initialised from seed pixels and connected pixels are considered for inclusion in to each region. If the region that would be formed by adding the new pixels passes the homogeneity criterion then it is added to the region and the region spreads from that pixel to all surrounding pixels recursively. Although region spreading can be implemented in a recursive manner it is still a flat segmentation technique as its output is a single assignment for each pixel.

The typical random seed segmenter will choose an arbitrary non assigned pixel and spread from it until no further spreading occurs, at which point the segment is finished and the process continues until all pixels are covered. **Figure 2.4** (left) shows a seed pixel located within a region spreading to its connected neighbours

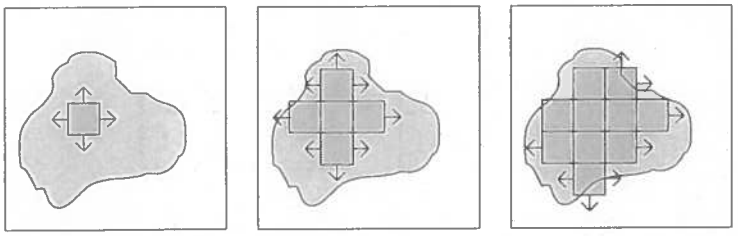


Figure 2.4: A growing region from a seed pixel (left) spreading (centre) and ceasing to grow at discontinuities (right).

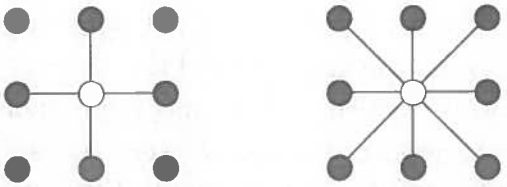


Figure 2.5: A centre pixel with 4 connectedness to the surrounding pixels (left) and 8 connectedness (right).

(centre) and finally some of the second generation pixels failing to spread any further (right).

Although it is possible to formulate other topologies, most spreaders work on either 4 or 8 connectedness (see **Figure 2.5**) to the surrounding pixels. The main effect of different levels of connectedness is whether a chequerboard pattern is considered connected and a twofold increase in run time (as twice as many pixel are checked at each step). In region growing a number of redundant checks are made when pixels already attributed to a region are marked for spreading by other neighbouring pixels from different directions. For this reason 4 connectedness is often chosen as a balance between speed and accuracy as less redundant checks are performed.

Benefits of Region Growing

As region growing is a local approach, fast implementations can be realised and the process is well suited to parallel computation. Its simplistic nature makes region growing procedures easily adaptable to new approaches, such as re-segmenting arbitrarily shaped regions, unlike the Quadtree structure which will discussed later. Region growing facilitates intuitive human computer interactive segmentation, where the human selects the centre of a region, essentially the seed pixel, and the computer performs the segmentation.

Limitations of Region Growing

Region growing is sensitive to the choice of seed pixel; an initial choice of pixel which lies on an edge, for example, would lead to a false classification of a thin region between genuine areas of the image. If a seed pixel is unduly affected by noise then it may not spread at all causing a small false region. This can be counteracted by a post processing step of assigning regions under a certain size to their most similar neighbouring region or by an adaption of the close morphological operator.

Pixels are biased toward regions created first over those which follow, as they become irreversibly claimed. For example if two neighbouring regions have similar colours, yet their union still fails the homogeneity criterion, then there may be boundary pixels between the two which are sufficiently similar to both regions to be included in either. In the case of a serial spreading algorithm, whichever region is generated first will claim the similar pixel region, rather than a more intelligent approach which might assign the pixels to the regions they most closely resemble. One way to combat this effect is to grow all regions simultaneously; this can be achieved with a fixed number of seed pixels which spread and merge as they meet other suitably homogeneous regions. Assuming the starting number of seed pixels is suitably large, the image will not be under segmented.

2.1.3 Clustering

Clustering is a general method for grouping arbitrary data in some space, into like regions based on a distance measure. This has been applied to the segmentation problem of grouping pixels in colour space, Euclidean space, or a combination of the two.

K-Means Procedure

The K-Means procedure assumes a predefined k regions in which the data is to be grouped. Each of these regions are seeded with initial values for the means $m_0 ... m_k$. Each of the data points are then attributed to the region with the mean that it most closely resembles. The means are then updated to reflect the change in the contents of the regions. The procedure is repeated until the means either do not change, change is below some predefined threshold or some predefined termination condition is reached.

Although other measures may be used, the most common measure of distance between two points $x, y \in \mathbb{R}^n$, is the Euclidean distance:

$$d_E(x, y) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}$$

The Euclidean distance between two points in n dimensional space.

The effect of this procedure is to create approximations to the centres of k regions, which it is hoped correspond to k underlying regions in the data. Over time these approximations move to better approximate the true centres.

Applying K-Means to Colour Segmentation

K-Means can be used to segment the image purely in colour space, whereby all red pixels might compose one region regardless of connectedness. Another possibility is to cluster the points in a higher dimensional space composed of the two spatial dimensions of the image along with a number of colour dimensions. For this application is it necessary to scale each dimension so as to give proportional representation. For example if the x component ranges from 0 to 1024 and each colour component between 0.0 and 1.0, the colour of pixels will be all but ignored. One further possibility is to segment the image twice, once in colour space and again in Euclidean space.

Limitations of the K-Means Procedure

The initial choice of means is crucial, as some may swallow a disproportionate percentage of the space by fortuitous placement, while others may attract no points whatsoever. Optimality of the eventual partitioning of the space is therefore not guaranteed. A common approach is to randomly allocate the means.

Often one would wish to discover the number of regions in an image rather than specify them, as a guess to the number of regions in a completely unseen data set is unlikely to be accurate. To combat this we can ask for a small number of regions and regard these as the major regions and then re-segment them until some criteria is satisfied.

2.2 Hierarchical Methods

Hierarchical methods in this context are those which provide a multi layered or structured output whereby each pixel may belong to one or more regions at different levels of abstraction. These processes provide a mechanism for analysis of the source data at either different resolutions or conceptual levels of abstraction.

For some techniques (Quadtree) the main thrust behind the technique is to gain a speed advantage by processing larger portions of the image in one go at higher levels. Others (Binary Space Partition Trees) seek to gain a compact representation of the source image more suited to compression.

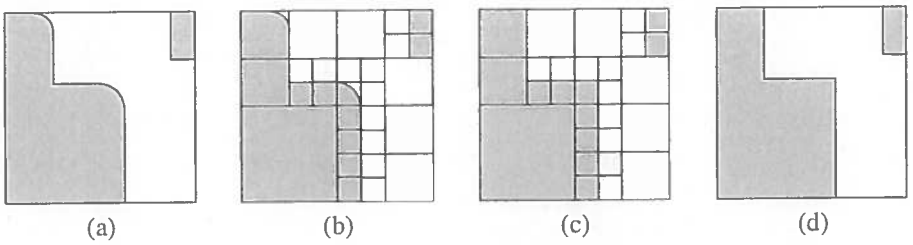


Figure 2.6: The quadtree process upon an original image (a), the divisions after the splitting phase (b) with the split approximation (c) and the final merged approximation of the original (d).

2.2.1 Split and Merge

Split and merge approaches work by recursively diving an image until homogeneous regions are achieved and then subsequently merging similar regions. The first splitting phase begins by testing a portion of the image (often the whole image) for homogeneity: if this region is suitably homogeneous then the process stops, otherwise the region is divided into some regular arrangement and the process continues recursively. A second merging phase then merges adjacent regions if their union passes the homogeneity criterion until no further merges are possible. The splitting phase is often quick and computationally elegant, whereas the merging phase naively requires each region to be compared with every other, leading to exponential runtime.

Quadtree Split and Merge

Quadtree split and merge is a common technique based on the quadtree data structure in which each node represents a square portion of the image with four children corresponding to the four quadrants of the portion. This technique is most elegantly applied to square images with width 2^n. The splitting phase can be efficiently implemented due to the quadtree structure, however the merging phase is complicated by the need to merge regions at different depths in the structure; indeed the problem of deciding when all possible merges have taken place is non trivial in itself.

The quadtree process and many other splitting methods based on regular patterns introduce artefacts based on those patterns. Quadtree segmentations have a much stronger response for horizontal and vertical edges than those at 45^o. So for images which exhibit strong non axial features the segmentation may include a large number or regions. In the pathological case of a small scale chequerboard at 45^o, the quadtree approach may give very little improvement on merging the original pixels to form regions.

If only the leaves of the quadtree are inspected then the process as presented is

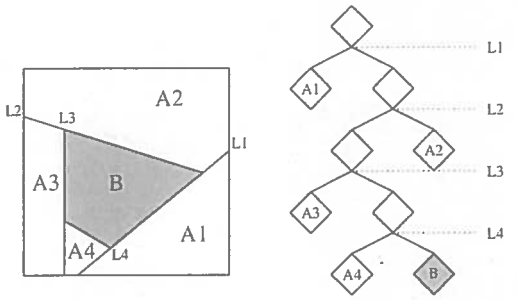


Figure 2.7: An image containing two regions A and B divided by lines L1 to L4 (left) and the corresponding BSP tree produced with the line cuts marked (right)

another example of an algorithm that produces a flat output despite a hierarchical process. It is when the tree is viewed at each depth instead as a representation of the source image that the the quadtree process is truly hierarchical.

Binary Space Partition Trees (BSP) [4]

BSP trees are a technique for classifying any space into sectors by recursively dividing the space using a cutting plane. In the case of image segmentation the image is repeatedly divided by a cutting line fitted to the data (see **Figure 2.7**) using some other image recognition technique (possibly the Hough Transform or RANSAC algorithm). This process continues until the homogeneity criterion is met for each of the sectors.

The process produces a binary tree which can be used to efficiently code the image for video compression. Similar to the Quadtree process this technique introduces artefacts as it has a stronger response for polygonal structures than smooth curves or gradients.

2.2.2 Graph Based

Graph based methods are those that view the image as a set of nodes connected by arcs. A flat segmenter can be constructed by searching all arcs in the graph, merging any node whose union passes the homogeneity criterion.

These nodes may represent segments from a previous segmentation procedure with the express aim of providing a large number of small segments for the graph method to group appropriately. Another approach is to initially view each pixel as a node, in this case the region grower could be thought of as an implicit implementation of a graph based approach. Explicit graph based methods can be used to monitor and control the workings of the merge phases of split and merge algorithms, assuming some sensible conversion from the native representation of the method to a graph based representation can be formed.

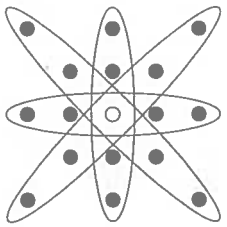


Figure 2.8: Elliptical regions used in one multiple median filter technique.

The real power of graph based approaches is to facilitate abstracted analysis of the connectedness of the regions. A great deal of mathematical graph theory exists and can be directly applied. In particular analysis to identify groups of regions that surround others can be employed to convert the graph into a tree hierarchy of objects which contain others. However many of these graphing problems are in them selves mathematically hard problems. A graph based approach for this is presented and dealt with in more detail in **4 Hierarchical Methods**.

2.3 Pre/Post processing

Images collected from almost any type of sensor are likely to be corrupted by noise. The purpose of all of the preprocessing filters is to reduce the effect of noise and enhance the segmentation process. The question of the quality of a particular filter therefore becomes the balance between the effectiveness of the filter in facilitating good segmentation and its speed of application.

2.3.1 Common Filters

Median Filter

The median filter is one of the simplest filters, attempting to reduce the noise at a particular pixel by replacing its value with the median of the surrounding pixels in some neighbourhood.

This can be achieved in a simple $n \times n$ square region or in elliptical regions around the centre pixel. In this elliptical method the median of the five areas are taken independently and the centre pixel is replaced with the median of these values (**Figure 2.8**). There is no specific ordering of colours in $\mathbb{R}^3$, and so the median is undefined for colour images. One way to overcome is this to apply the filter in each of the colour channels independently, or to order by lightness alone.

The effect of the median filter is good suppression of salt and pepper noise, with

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \quad G(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Figure 2.9: The Gaussian Distribution for one (left) and two dimensions (right) with a standard deviation of σ

a reduction in Gaussian noise. Edges are adversely effected by the median filter; a characteristic blocking effect results.

Mean Blur

The mean filter again is one of the most intuitive filters where the centre pixel, in a typically square neighbourhood, is replaced with the mean of that neighbourhood. The mean filter can be applied using a normalised convolution kernel of all ones. As the mean filter is symmetrical and linear it can also be decomposed into two linear kernels, with attendant benefits to processing time. These kernels are themselves linear mean filters in the horizontal and vertical directions (kernels of all ones).

The mean filter is highly destructive to edges and, when applied in a square kernel, introduces square artefacts around edges due to the high effect of extreme values upon the mean. The mean is more effective in removing Gaussian noise than salt and pepper due to the effect of extreme values.

Gaussian Blur

The Gaussian filter is one which follows the Gaussian distribution give in **Figure 2.9** to decide the effect of surrounding pixels on the final pixel value. The effect is one of a smooth blur and is hence very edge destructive.

The Gaussian filter is a mathematical ideal, as such all actual implementations are approximations; in particular as the kernel is infinitely large all pixels have an effect on every other pixel, which would make its application computationally intensive. In practice the simplifying assumption is made that after a set number of standard deviations the effect is negligible.

The Gaussian filter is often applied using an integer approximation to the true values in a normalised convolution kernel. Like the mean filter, the Gaussian can be decomposed into two subsequent applications of the same linear convolution kernel in the horizontal and vertical direction. A good approximation to the values of this linear matrix is obtained from the coefficients of the binomial series (the values of Pascals triangle). Unlike the mean filter subsequent applications of the Gaussian filter do equate to one application of a larger kernel.

2.3.2 Convolution

Convolution is a process which can be used to achieve a wide variety of filtering operations by the application of a variable kernel at every pixel value in an image. This kernel is centred on a pixel and specifies a weighted sum for all of the surrounding pixels. It is common to ensure that these weights sum to one to ensure that the output of the convolution is in the same range as the input; such a kernel is said to be normalised.

The convolution process can be used to implement the mean filter easily using a square kernel of all ones. As with any symmetric kernel, it is possible to decompose the two dimensional kernel into successive applications of linear kernels. This has a direct implication to runtime as the application of linear kernels is an $O(n)$ process with respect to the number of pixels, whereas two dimensional kernels require $O(n^2)$.

The Gaussian blur is commonly approximated as a combination of linear kernels as it is symmetric. Furthermore, repeated applications of a smaller Gaussian kernel produces the same effect as the single application of one larger kernel. Therefore a square Gaussian kernel approximation of any size can be achieved with judicious application of a single 3×1 kernel.

2.3.3 Edge Preserving Blurs

The aim of edge preserving filters is to achieve the reduction of noise within homoegeneous regions without destroying the information at the edges of regions.

Symmetric Nearest Neighbour Filter (SNN)

The symmetric nearest neighbour filter (or SNN) is an example of such a filter which attempts to suppress noise whilst preserving edges.

One method of applying the SNN filter is to test each member of the 8 connected neighbourhood with its symmetric opposite beyond the centre pixel (see **2.10**). The mean of the four most similar pixels, to the centre pixel, then generates the output value for centre pixel. Greater effects can be achieved by repeated applications of the filter.

Another method is to apply the same principal in some neighbourhood about the centre pixel. For example in a square neighbourhood each pixel at (i, j) relative to the centre pixel, would be compared to $(-i, -j)$. In this case greater effects can be achieved using a larger neighbourhood.

The effect of the SNN filter is to reduce noise but to sharpen the edges; a characteristic crystalline structure of similarly coloured patches appears in the resultant

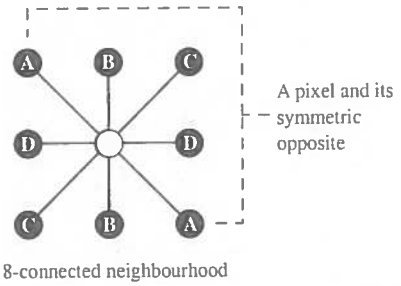


Figure 2.10: A centre pixel with symmetric pixels labeled

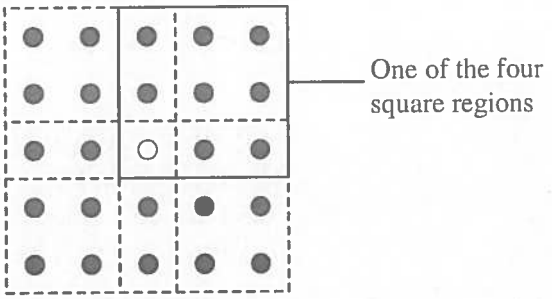


Figure 2.11: The centre pixel showing the surrounding four regions in which the variance and mean are calculated.

image. The filter achieves its edge preserving qualities by making the assumption that at a local level edges are straight lines. This line ideally divides the pixels either side into pixels which are like the centre pixel and those which are not. The centre pixel is therefore replaced with the mean of only those pixels which are on the same side of the edge.

Kuwahara Filter [2]

The Kuwahara filter attempts to preserve edge sharpness and position (like SNN) by considering four square regions about the centre pixel; one such region is highlighted in **Figure 2.11**. For each of these regions the mean and variance are calculated. The centre pixel is replaced by the mean of the region with the lowest variance.

The effect of the Kuwahara filter is similar to the SNN, but slower to calculate and with a lower capacity to remove salt and pepper noise.

2.3.4 Mathematical Morphologies

A mathematical morphology is an operator based upon the relative positions of pixels rather than their value[1]. Commonly they are only applied to binary images although there are adaptions to allow their use in grey scale and colour images.

Erode and Dilate

The most basic morphological operators are the erode and dilate operators. The dilate operator returns the maximum value in some neighbourhood B about a pixel (acting upon an image X this is denoted at $X \oplus B$). In binary images this translates to returning 1 if there is at least one positive pixel. The effect of the dilate operator is to grow white regions. The erode operator (denoted $X \ominus B$) is the complement of dilate, returning 0 if there is at least one zero pixel in the neighbourhood about the centre pixel.

Open and Close

$$\text{Open: } (X \oplus^n B) \ominus^n B$$

$$\text{Close: } (X \ominus^n B) \oplus^n B$$

The open and close operators are comprised of repeated applications of the erode and dilate operators. The effect of the open operator is to remove speckles and smooth boundaries. The close operator has the effect of healing small holes of negative pixels in the midst of positive ones. Again the two operators are the complement of each other. These operators are often used to clean the results of a binary segmentation.

3. Flat Methods

3.1 Region Growing

It is assumed here that the outline of the general region growing process has been read in the current techniques review **Section 2.1.2**. This section relates to the specifics of the implementation of the region growing implementation developed in this project. The JAVA code for this can be seen in **Appendix B: Code**. An example output of the process described here can be seen in **Figure 3.1**.

Overview

In the broadest overview the process of region growing presented here can be described as the application of several processes, each of which is discussed in detail in later sections.

1. Filter input image.
2. Choose a seed pixel.
3. Grow a Region.
4. If all pixels have been included in regions end, otherwise goto 2.

The Basic Structure

The most basic implementation of the region grower attempted was instantiated as a recursive call to a seed pixel location. This call initiates a region including that pixel and forms a recursive call to the pixels in either a 4 or 8 connected neighbourhood about that pixel. Each of these pixels are compared to the region,

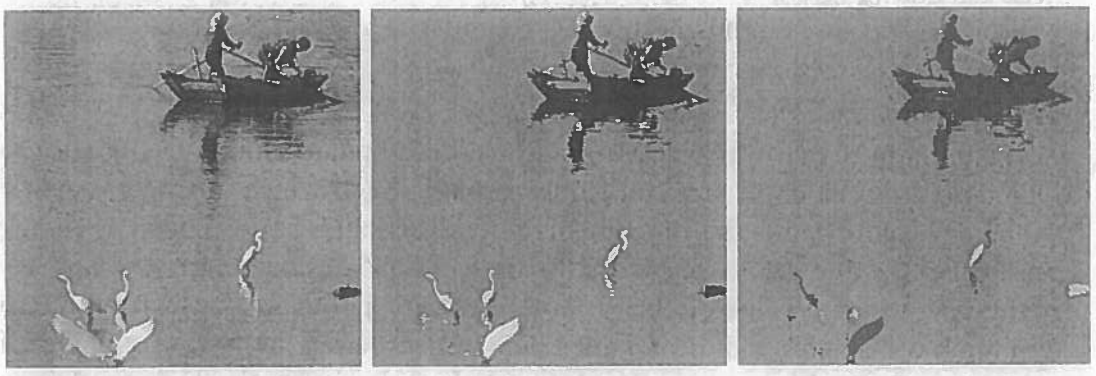


Figure 3.1: An example image (left) and the result of the SNN filter and region growing process in mean region colour (center) and false colour (right)

```
1   segment[] RegionGrow(image I)
2      I = SNN(I);
3      (width, height) = I.getDimensions();
4      int ID[width][height];
5      int count = 0;
6      int seg = 0;
7      segment s[];
8      while(count != (height * width - 1))
9         (x,y) = pickSegmentSeed(ID);
10        s[seg] = doGrowing(x, y, seg, I, ID);
11        seg = seg + 1;
12     endwhile
13  return s;
```

Figure 3.2: The Pseudocode for the Region Growing control loop

if they are suitably similar then the pixels are added and a call is made to pixels in the neighbourhood about the new pixel. If the pixel is already within the region or too dissimilar then the recursion ceases at that branch.

It was found that this recursive method placed far too much information on the stack and the process rapidly ran out of memory. To resolve this the process was serialised and implemented as an agenda of pixels (see **Figure 3.3**) to be considered for addition to the region. The process was then initialised by adding the seed pixel to the agenda (**Figure 3.3 line 3**). The agenda is processed one pixel at a time in a similar fashion to the recursive method, however recursive calls are emulated by placing the new pixels that are spread to, onto the end of the agenda (**Figure 3.3 lines 11-14**). If the agenda is empty then the region has ceased to spread and the process terminates.

To eliminate search when determining whether or not a pixel has already been allocated to region, an array of values is maintained which mirrors the pixels in the image and records their membership to a particular region coded as an integer (**Figure 3.2 line 4** and **Figure 3.3 line 8**). This array is referred to here as the ID array, where a value of -1 indicates that the pixel has not yet been allocated and any other integer indicates membership to the region by that number. This technique increases memory requirements but decreases the time to determine membership from $O(m)$ where m is the number of pixels allocated to the region in question to constant time $O(1)$.

```
1   segment doGrowing(int x, int y, int segID, image I, int ID[][])
2      stack agenda;
3      (x, y) → agenda;
4      segment s;
5      while( agenda.size() != 0)
6         (x,y) ← agenda;
7         if( getDistance(s,(x,y)) < distanceThreshold )
8            if( ID(x,y) = -1 )
9               (x, y) → s;
10              ID(x, y) = segID;
11              (x+1, y) → agenda;
12              (x-1, y) → agenda;
13              (x, y+1) → agenda;
14              (x, y-1) → agenda;
15           endif
16        endif
17     endwhile
18  return s;
```

Figure 3.3: The Pseudocode for single Region Growing

3.1.1 Random / Modulo Arithmetic Seeding

The region growing process is very sensitive to the placement of the seed pixels that initalise the regions. In the case of a serial region grower the choice of seed pixels is one that must be made, as the program runs, each time a new region is spawned. The following methods implement the function `pickSegmentSeed(int ID[][])` in **Figure 3.3 line 9**. **Image Order Traversal**

The most naïve method of seed choice is to pick each pixel in turn, left to right, top to bottom; if the chosen pixel is already in a region then move to the next. The disadvantage of this procedure is that it introduces a large bias toward the pixels in the upper left hand corner. The first region is always seeded at $[0, 0]$, which will at some point reach an edge and cease to spread. The next seed pixel is therefore guaranteed to be on this boundary as it is the next scanline pixel after the first segment. This introduces a large number of false boundary regions. In its favour only one position has to be recalled (the last seed pixel tested), and the total number of checks that will be made is exactly the number of pixels.

Random Generate and Test Traversal

One alternative method is to randomly generate locations over the whole image and test for a pixel which has not been allocated yet, each time a seed pixel is required. While this works very well for the initial segments, the time taken to

$$(d_E(x,y))^2 = \sum_{i=0}^{n} (x_i - y_i)^2 \quad d_M(x,y) = \sum_{i=0}^{n} |x_i - y_i|$$

Figure 3.4: The Sum of Squares distance (left) and Manhattan Distance (right) of two vectors $x, y \in \mathbf{R}^n$

randomly generate a pixel that has not already been allocated increases dramatically as the percentage of unallocated pixels decreases.

Modulo Arithmetic Traversal

To combat these problems a method based on modulo arithmetic is presented which combines the benefits of pseudo random traversal of the image with the benefit of only having to store one position value. If the images are constrained to dimensions of 2^n, with $p \in [0, 2^{2n})$ as the position though the image and $k, n \in \mathbb{N} \cup \{0\}$ then the traversal function is as follows:

$$f(p) = 3^k p \bmod 2^{2n}$$

The claim is that $f : [0, 2^{2n}) \to [0, 2^{2n})$ is a bijection. That is to say that each input value of p determines a unique position within the image. The implication of this in the case of our application is that if p is incremented from 0 to 2^{2n} then each of the pixels in the image will be covered once and only once, and if k is suitably large, in a pseudo random fashion. A proof of this fact can be read in the appendix.

3.1.2 The Homogeneity Criterion and Distance Measures

The homogeneity criterion links a candidate pixel with a region and determines if the region would still retain homogeneity after inclusion of the pixel. For the purpose of this implementation the homogeneity criterion has been implemented as a comparison between some distance measure and a threshold. The distance measure used is the Euclidean distance squared, or sum of squares between the candidate pixel and the segments average colour (**Figure 3.4** left). The sum of squares gives identical performance to the traditional Euclidean distance save for the fact that the threshold must be squared to remain at the same level. This has the benefit of avoiding a square root calculation which can greatly speed execution.

Distance Approximations

The Manhattan distance or Taxicab metric (**Figure 3.4** right) is an approximation to the common Euclidean distance measure used to save on computation

$$m_{t+1} = \frac{(n-1)m_t + v}{n}$$

Figure 3.5: The equation for incremental update of a mean over time where v is the new value

(as the modulus avoids calculations of squares and roots). This approximation was used but was found to be only negligibly faster at the cost of segmentation performance. The reason for this degredation can be seen when the isosurfaces of the function are considered. In 3 colour space the isosurfaces of the Euclidean distance form spheres about a centre point, whereas the Manhattan distance form concentric cubes. This causes colours toward the corners of the cube to be rated as unreasonably close to centre point compared with the centre points of the sides.

Averaging Colour Segments

The segments used in this project are plain coloured, where the plain colour represents the average colour of the pixels within the region. It is this value to which all candidate values are compared. To avoid full recomputation of the mean value after each pixel is added, the mean can be altered incrementally using the equation in **Figure 3.5**.

It is possible to use only the seed pixel colour value for comparisons to avoid the computation of the mean of the region. It was found that this caused reduced segmentation performance due to increased sensitivity to the seed pixel and any noise which might be present at that point. As the time saved was only marginal over the incremental mean method this approach was not used.

Another possibility considered was a leaky accumulator approach whereby the old mean is reduced by some fixed factor α, and summed with the new colour multiplied by $(1-\alpha)$. The effect is to allow each new pixel to have some fixed effect on the accumulated mean value. This process increases chaining errors where a long string of pixels which are suitably similar connect two dissimilar regions which violate the homogeneity criterion. As the leaky accumulator is forgetful of early values, it is possible that such errors could occur; indeed region drift was observed over gradients. This drifting effect and the minimal computational saving caused this method to also be abandoned.

3.1.3 Symmetric Nearest Neighbour (SNN)

The Need for Filtering As region growing is a purely local operation it is

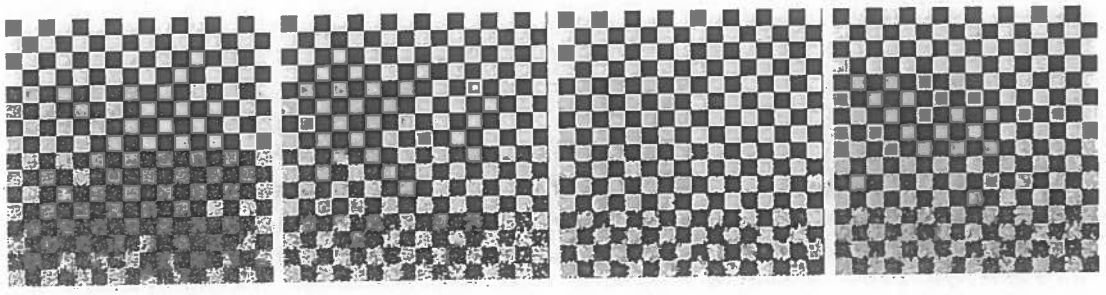


Figure 3.6: A test image with varying levels of Gaussian and salt and pepper noise(left), the effect of one SNN pass (centre left), the long term effect after 50 passes(centre right) and a comparison with Gaussian Blur with a 1 pixel radius

sensitive to local noise effects. In particular salt and pepper noise will cause over segmentation as single pixels will fail to spread and form their own regions. While these can be dealt with via a process of morphological type operators, it is desirable that they do not arise in the first place. As one of features we are interested in is the boundaries between regions, it is desirable that the filter we use does not disturb these boundaries. For this reason, along with the high speed, the SNN filter was chosen as the preprocessing step to facilitate region growing.

The Implementation The implementation used realised the SNN filter with a 3×3 neighbourhood. An array of four colour values was stored for each of the the opposite pairs (N-S, NE-SW, E-W,NW-SE), which retained the value out of those two pixels which most closely resembled the centre pixel using the sum of squares.

The sum of squares distance measure was used as it omits the need to calculate a square root. As the calculation is performed many times this can add up to a good saving in time without any loss of accuracy. There is no loss of accuracy as the sum of squares orders pixels in exactly the same fashion as the Euclidean distance measure, only the exact lengths are different.

After these four most similar points are acquired, they are averaged to provide the eventual output for the pixel.

The code for this method can be seen in **Appendix B: Code**

The Effect of SNN Filtering upon Noise

On the left hand side of Figure 3.6 a chequerboard patter can be seen that has been corrupted with increasing levels of Gaussian noise (on the left side) and salt and pepper noise (on the right) to illustrate the ability of SNN to remove said noise. The centre left image shows the effect after one application of the filter, with the centre right image giving the long term effect of multiple applications of the filter. In this case 50 passes were applied, but the result is much the same

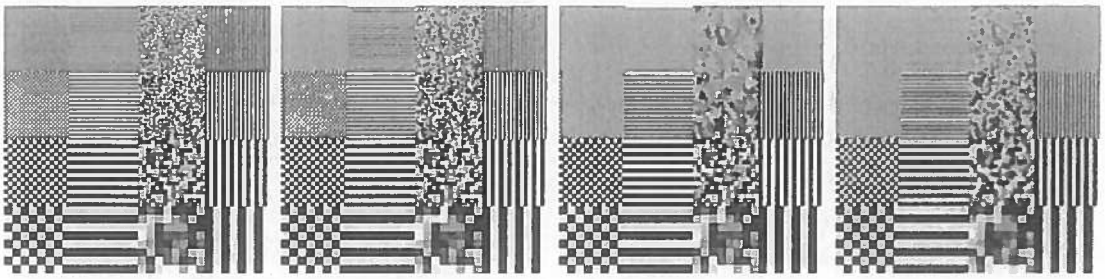


Figure 3.7: A test image with varying levels of detail(left), the effect of one SNN pass (centre left), the long term effect after 50 passes(centre right) and a comparison with Gaussian Blur with a 1 pixel radius

as after 6.

The effect upon the salt and pepper noise is particularly marked, with the underlying pattern being almost completely reclaimed in the lower levels of noise after only one pass. The effect upon the Gaussian noise is not as pronounced but some reclamation of the pattern is achieved.

The edge preserving qualities of the SNN filter can be observed when compared to the one pixel Gaussian blur (far right), where the edges of the chequerboard have been noticeably disturbed.

The Effect of SNN Filtering upon Structure

In Figure 3.7, a test image is given with various binary pixel patterns at different resolutions. Left to right they are the chequerboard, horizontal lines, random noise and vertical lines. From top to bottom the patterns are scaled at 1,2,4 and 16 pixel sizes. The aim is to illustrate the effect of the SNN filter upon deliberate structure.

The SNN filter works in a 3×3 neighbourhood, which roughly corresponds to a one pixel radius from the centre pixel. Unsurprisingly the structures at the level of one pixel are significantly altered with the chequerboard removed in the first pass and the lines removed in the long term. Interestingly in the long term the 2 pixel chequerboard is also destroyed, due to converging grey values generated in the first pass. The noise is first regarded as structure at the point of 4×4 pixel blocks (before random regions are generated).

As a guide it can be concluded that any deliberate structure above the size of 4×4 should remain largely unaffected, and most features at the scale of 2 pixels should also remain unaffected provided they are part of a larger structure (a $2 \times x$ line for example). By comparison the Gaussian blur can be seen to adversely affect the structures at all scales after just one pass.

The Speed of SNN Filtering

Image Size	Colour?	Mean Time (ms)	Pixels/ms
64 × 64	Yes	16.38	250.06
64 × 64	No	16.31	251.13
128 × 128	Yes	52.68	311.01
128 × 128	No	53.46	306.47
256 × 256	Yes	219.41	298.69
256 × 256	No	215.03	304.78
512 × 512	Yes	827.82	316.67
512 × 512	No	825.09	317.72

Figure 3.8: Runtimes for the SNN filter applied to various images.

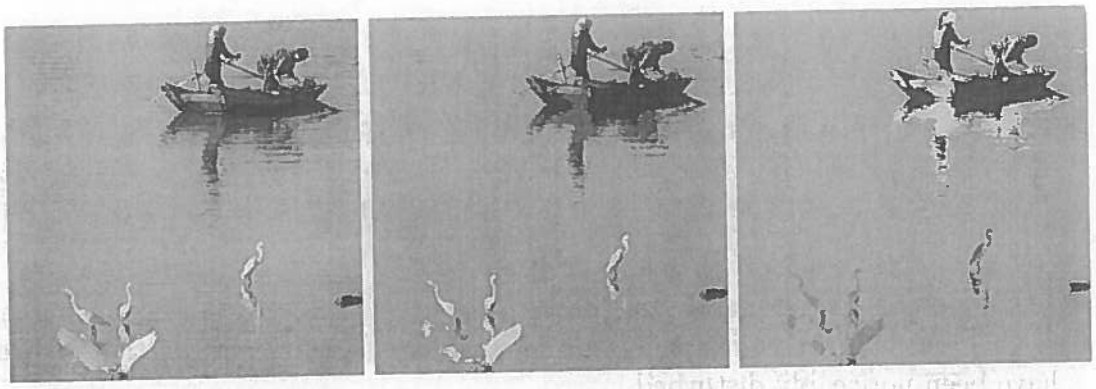


Figure 3.9: An example image (left) and the result of the Two Phase K-Means process in mean region colour (center) and false colour (right)

To evaluate the speed of SNN, the filter was run upon a test set of images acquired from the Hypermedia Image Processing Reference (HIPR2)[7] image library. The library contains 189 greyscale and 59 colour images of a wide variety; ranging from assembly parts and medical images to normal photographs. All images were scaled to square images of width 64, 128, 256 or 512 for compatability with the program. The SNN filter was applied to each photograph 10 times and the resulting mean times can be observed in Figure 3.8.

The process is swift and never exceeds one second even for 512 × 512 images. This implementation is in JAVA, no doubt a swifter implementation could be formulated in C.

3.2 K-Means

It is assumed here that the outline of the general K-Means process has been read in the current techniques review **Section 2.1.3**. This section relates to the specifics of the implementation of the various K-Means implementations devel-

oped in this project. The JAVA code for this can be seen in **Appendix B: Code**. An example output of the process described here can be seen in **Figure 3.9**.

Overview

K-Means in not a sole purpose segmentation technique. It can natively be made to segment an image into k disjunct regions which have the same colour by simply applying the basic process to the set of image points considering only their colour value. However any segmenter should also produce regions which are spatially adjunct.

Unlike the region growing implementation, several variants of K-Means were produced which build upon each other. The vanilla K-Means implementation segments an image either based on a spatial distance measure, a colour distance measure or a combination measure. The most successful process named Two Phase K-Means, consists of first segmenting in the colour space as above, then subsequently re-segmenting each of these regions under a spatial measure. A further variant of K-Means was created to allow successive iterations to run at different scales to improve performance but due to the fact that large numbers of iterations do little to improve the quality of the segmentation produced, this approach was abandoned.

The most basic overview of the plain K-Means implementation can be viewed as the following:

1. Setup k means
2. Allocate all pixels to nearest mean using distance measure
3. Update all means to reflect changes in pixels allocations
4. If iteration limit is not reached GOTO 2
5. Filter the IDs produced.

The Basic Structure

To begin with, the means are seeded with (x, y) locations (**Figure 3.10 line 7**) and the colour value from the image at that point is read into the mean colour for that region. Each mean also has an attendant value which records how many pixels have been attributed to that mean, which is initialised to 0 (**Figure 3.10 line 5**).

One iteration of the procedure consists of moving though each pixel in the image and comparing it with each of the k means (see **Figure 3.11**) under the current measure. Which ever mean it is closest to, by that distance measure, is updated to reflect the inclusion of the pixel and the size for that mean is also incremented. In a similar way to the region growing procedure an ID array is use to store the membership of each pixel. This allows a fast check to see what mean the pixel

```
1  segment[] KMeans(image I, int k, int iterations)
2     (width, height) = I.getDimensions();
3     int ID[width][height];
4     color means[k];
5     int size[k];
6     segment s[];
7     (means, size) = seedMeans(means);
8     for( i=1 ...  iterations)
9        doOneIteration(I,means,size);
10    endfor;
11    s = readIDsToSegments(ID);
12 return s;
```

Figure 3.10: The Pseudocode for the K-Means control loop

```
1  color[] int[] doOneIteration(image I, color[] means, int[] size)
2     int old, new;
3     for( x=1 ...  width )
4        for( y=1 ...  height)
5           old = ID(x,y);
6           new = getNearest(x,y,means);
7           if( old != -1 )
8              reduceMean(means, old);
9              size[old] = size[old] - 1;
10          endif;
11          increaseMean(means, new);
12          size[new] = size[new] + 1;
13       endfor;
14    endfor;
15 return means;
```

Figure 3.11: The Pseudocode for one K-Means iteration

was attributed with previously. If the pixel was not attributed to any mean (a value of -1) then no action is taken, however if the pixel did belong to another mean, that means size is decremented and the mean value is adjusted to reflect the removal of that pixel (**Figure 3.11 lines 7-10**).

This iteration procedure is repeated up to a predefined iteration limit (**Figure 3.10 lines 7-10**). Initially a method was attempted that detected the change in the means between iterations and terminated when said change dropped below a certain threshold. It was found that the total number of iterations require for the means to stabilise is rarely more than 4 making such a procedure was unnecessary.

Two Phase K-Means

Many of the processes discussed in this chapter are designed as solutions to the limitations of the K-Means approach as a segmentation technique. The most fundamental of which is using it to cluster the data in the image in a way that ensures that pixels in the same region are both close in colour space and Euclidean space. As a unified distance measure which achieved both was not found (see **Distance Measures** in this section) it was thought that a far more native approach for K-Means would be to segment the data twice. Once in colour space, which K-Means achieves very competently and then again for each of the segments produced in Euclidean space. The rationale being that if the image is separated into regions of dissimilar colour then each of the regions formed will represent the union of a number of spatially disjunct regions; as any regions which are connected at this point are already of suitably similar colour and therefore form one region by definition.

The new problems created by this approach are that the segments produced in the initial colour phase may make it impossible to create any sensible connected regions in the second. To combat this shortcoming a technique of region allocation filtering (see **ID Filtering**) was developed to allow the removal of very small clusters of pixels.

A further problem of the K-Means approach when used with the common Euclidean distance measure for spatial segmentation is that while this may guarantee good spatial clustering it makes no concessions for connectedness. To combat this, a technique of merging based on spatial information was developed to conservatively ensure that no connected regions were segmented into different regions in the output (see **Spatial Colour Merging: Section 3.2.5**).

As this process works by merging regions which are connected by any pixels it is preferable to increase k in the second phase so that in essence the process becomes an abstracted region growing method. In this sense the nature of the two phase mechanism works in our favour as if the image is segmented into k colour regions each of which is then re-segmented into k spatial segments. This

$$(d_E(x,y))^2 = \sum_{i=0}^{n}(x_i - y_i)^2$$

Figure 3.12: The sum of squares function for distance between elements $x, y \in (R)^n$ space.

leads to the situation where the effective k at the colour level is in fact k^2. A rare example of combinatorial explosion working in the favour of an algorithm.

Abstractly the flow of Two Phase K-Means is as follows:

- **Phase One:** K-Means in Colour Space
- ID Filtering
- Non-spatial Color Merging
- **Phase Two:** K-Means in (x, y) space (on all color segments)
- spatial Color Merging

3.2.1 Distance Measures

The notion of how close a pixel is to the mean of a region is completely dependant on the distance measure used and in which space one is interested in defining that measure. In both cases the sum of squares measure (see **Figure 3.12**) is used in either $\mathbb{R}^3$ in the case of colour or $\mathbb{R}^2$ in the case of (x, y) space. The full Euclidean distance is not used as it includes the calculation of an unnecessary square root, which can slow calculation significantly. The sum of squares is sufficient since only the order of the distances is important (which is preserved in the positive reals by the squaring function).

Higher Dimensional Measures

In the case of this application the ideal measure would separate the pixels in both colour and Euclidean space such that a small distance would relate to the pixel being simultaneously similar in colour and close to the region. This would allow full segmentation of the image in one fell swoop. One such function could be the higer dimensional sum of squares of the vector composed of both the colour and spatial coordinates.

One issue with this method is that in order for colour and spatial distance to have an equal effect they must both be normalised. Irrespective of the particular scaling used however it is always the case that at a particular distance the colour

value will either dominate, or be completely ignored. This leads to regions being indiscriminate about the colours near their centres and impossibly strict about colour at the extremities. In practice it was found that the higher dimensional approach is a poor compromise compared with segmentation in colour followed by segmentation in the spatial dimensions and so was not used.

Nearest Neighbour Distance

For spatial distance measuring, simple distance measures from the mean point of a region lead to undesirable allocation of pixels when the underlying regions are not circular. For example consider a large round region horizontally next to an extremely tall and thin region, with a pixel equidistant from their centre points. The pixel should obviously be attributed to the large round region as its distance to the edge of that region will necessarily be less. However the naïve method is equally likely to attribute the pixel to either region. Other pathological situations include large 'C' shaped regions where none of the points in the true region are close to their mean; using the conventional method such features are unlikely to be detected at all.

To combat this issue a method was developed that compared a pixel with a local neighbourhood and returned the distance corresponding to the nearest pixel allocated to a region. If no pixels in the region were allocated or if more than one pixel from different means were equal, then the conventional distance measure would be used. It was hoped that this would encourage the formation of connected regions that were not necessarily round as a pixel could be close to a region by being close any other member of that region.

In actuality it was found that whilst the process did indeed encourage connectedness within the scope of the neighbourhood size, even very small regions increased the computational demands vastly. Consider a small square neighbourhood of distance 5 pixel in each direction from the centre pixel, the total number of pixels to be considered is $(2\times5+1)^2 1 = 120$. This is already a large number of comparisons at each pixel and a neighbourhood of this size only encourages connectedness on the scale of 11 pixels squared (certainly not enough for global clumping). For this reason the method was not used.

3.2.2 Random / Spaced / Region Growing Seeding

Although the K-Means procedure is sensitive to the placement of the seed pixels it is not critical that they be placed in specific locations as it is accepted that over time the mean points will drift toward the centres of significant regions. For this to be achieved the means must be distributed representatively across the space in which the image is to be segmented. For example if the image is to be segmented using a spatial distance measure it is enough to ensure that the pixels

are distributed in (x, y) space across the image. However if one wishes to segment in the colour space then the positions of the means must correspond pixels which have dissimilar colours, regardless of physical location. **Random Seeding**

The simplest method of seeding is again to randomly specify (x, y) positions in the image. In the spatial domain this is an effective method of guaranteeing good distribution. In the colour domain colours have a likelihood of representation in direct proportion to their density of distribution in that colour space. That is to say if 50% of the pixels in the image are red, one would expect on average for 50% of the means to sample red pixels. In general this is not a desirable property; consider the situation where a number of different coloured object lie against a constant colour background. If the background fills a significant enough proportion of the image then the likelihood of any particular mean sampling a pixel in the object of different colour decreases. Furthermore as the K-Means procedure continues there is little chance of the different coloured regions being able to influence the means as a large number of background pixels will be attracted to each one.

Separated Seeding

One possible method to attain good separation in the spatial dimension would be to simply randomize the colour values of the means. Unlike the spatial domain however the colour space is not necessarily uniformly distributed over the image. A random sampling might therefore produce a large number of means that do not correspond at all to any particular feature in the image.

To attempt to overcome this problem without adding undue processing before the K-Means has even begun, a simple method for increasing the spread of the means has been developed. The idea is to create a simple generate and test method to find a solution to the problem of finding means which are all mutually spaced by at least the specified figure in the colour domain and are present in the image.

The process works by taking a user specified spacing (in colour space) which all means should hopefully attain from each other. Initially all of the means are randomly seeded from the image, at which point each pairing of means is tested to see if the distance between them is greater than the specified spacing. If this test fails then first of the pairs (x, y) position is randomized and the colour value from the image is read in as the new mean. If this occurs for any pairing then the process is flagged as incomplete and the checking-randomizing loop begins again, up to a specified number of trials.

Clearly if all of the means are at the specified distance from each other then the process will return a solution. This method is certainly not guaranteed to find a solution at all but it does in general produce means which are separated in colour space but also represented in the image.

Region Growing Seeding

Whilst the above process is swift and gives good separation, no information about the regions in the image is utilised. One large draw back to the K-Means procedure is the necessity to specify a fixed k, which will cause the process to generate k region irrespective of the contents of the image.

To combat both of these drawbacks a method was developed which used a region growing technique on a small scale representation of the image to quickly generate a rough segmentation. The means of segments above a specified size were used as the seeds for the procedure. The hope was to simultaneously automate the process of choosing a k for the process and generate means that correspond not only to strong features in the colour domain but in the spatial domain also.

In practice it was found that in order to capture important but small features in the image, the size threshold for inclusion to the process had to be set to a level which generated a prohibitively large number of seeding regions. Furthermore, many of these segments tended to correspond to the same feature, where some small interuption had caused the growth of two regions rather than one. In addition, with the extra time required to not only perform the pre-segmentation but to rescale the image to a suitably small size, it was decided that the process was not as suitable as either random or separated seeding.

3.2.3 The Effect of Iteration

K-Means is an iterative process that attempts to obtain more accurate representations of underlying groups in the data by refining the means over each successive iteration. As the number of iterations increase, so will the accuracy of the result, at the cost of an increase in the runtime of the process. But at which point does the cost in increased time outweigh the benefit of the increased accuracy?

To answer this question a small batch of tests were performed to ascertain the change in the centre points and colour values of the means between iterations. This distance was measured as the average Euclidean distance in $\mathbb{R}^5$ composed of the three colour channels and (x, y) values. The result of a batch of ten runs with ten iterations and $k = 10$ can be seen in **Figure 3.13**. For this test all additional processes such as filtering were turned off.

It is clear that almost all of the movement of the means is conducted in the very first step. Additionally the change in iterations 2 and 3 is roughly equal to the total change in the following 7 iterations. As the effect of more iterations is, predictably, a linear increase in runtime the default value for iterations is recommended as 3, with 2 for speed.

Iterations	Change	Time (ms)	ms/Iteration
1	234.7	319.5	319.5
2	4.8	632.9	316.5
3	4.0	933.4	311.1
4	2.6	1380.9	345.2
5	1.7	1548.3	309.7
6	1.4	1857.6	309.6
7	1.4	2161.2	308.7
8	1.1	2470.5	308.8
9	0.8	2776.0	308.4
10	0.7	3194.6	319.5
		Mean:	315.7

Figure 3.13: A table of the change measured as the average distance between the mean positions before and after an iteration measure in combined colour and Euclidean space and runtime over 10 runs.

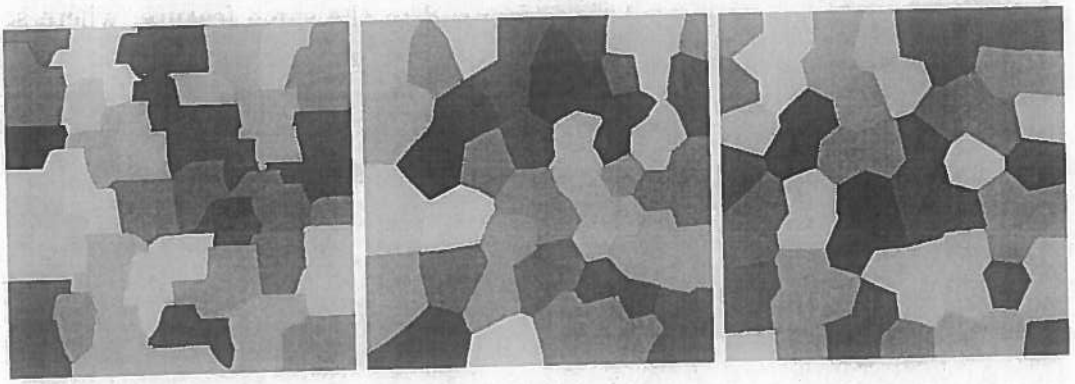


Figure 3.14: Three separate segmentations in Euclidean Space only after 1 iteration (left) 2 iterations (centre) and 3 iterations (right) shown in false colour.

It is worth noting that although it may be tempting to only perform the first step looking purely at the movement of the means, significant refinement of the pixel allocations is achieved in the second iteration. Theoretically under the sum of squares distance the boundaries between regions in (x, y) space should be polygons formed from the lines of equidistance between the centre points (The same is true of colour space although the meaning is harder to visualise). Therefore once the regions have settled to these polygons of equidistance the process should not show further improvement.

This allows an intuitive visual appraisal of the progress of the iteration procedure as can be observed in **Figure 3.14** where the far left image shows regions with significantly curved boundaries due to the mean shifting over the process of the iteration; the centre image exhibiting some curvature and the third iteration

showing almost completely polygonal boundaries. In this context the recommendation for 3 iterations becomes more clear, as at this point the boundaries are almost completely straight and therefore will exhibit little further movement.

3.2.4 ID Filtering

An initial segmentation in purely colour space provides no guarantees about the distribution of the resultant pixels. Often the pixels will be spatially adjacent as the regions they emerge from are often similar in colour but often when an area of an image is close to equidistance from two means in colour space, the resulting segmentation contains a large cloud of single pixels or small islands. Ideally it is hoped that the overall process will produce connected regions; in order to facilitate a spatial segmentation to achieve this it is necessary to remove these small islands and single pixels. To achieve this a process known as mathematical morphologies is often employed (see **Section 2.3.4 Mathematical Morphologies**). These morphologies as presented work with binary images, in this instance a variant will be used that will operate on the ID array produced during segmentation.

ID Array Filter

The ID filter counts the number of pixels allocated to each mean in a square neighbourhood about a centre pixel. The allocation of the centre pixel is replaced to correspond to the mean with the maximum number of occurrences. A second formulation was made that only changed the centre pixel if the mean selected accounted for a certain percentage of the neighbourhood. This was done to reduce the destructiveness of the filter and the parameter is refered to as the filter threshold.

The Effect of ID Filtering

The effect of the ID filtering is to remove small areas at the scale of the neighbourhood and to merge those regions into larger coherent regions. This comes at the cost of losing small details below the filter aperture size. To combat the loss of these details the filter threshold can be increased. This prevents areas where the pixels are evenly distributed to all of the means from arbitrarily assigning new values where one mean may happen to just hit the maximum. On the whole higher apertures should be accompanied by larger threshold values to combat the increasingly destructive effect.

The effect of ID filtering is best observed on an example image, as its effect upon the resolution test image is negligible at any settings due to the nature of its effects upon regular patterns. The results of various levels of filtering can be seen in **Figure 3.15**.

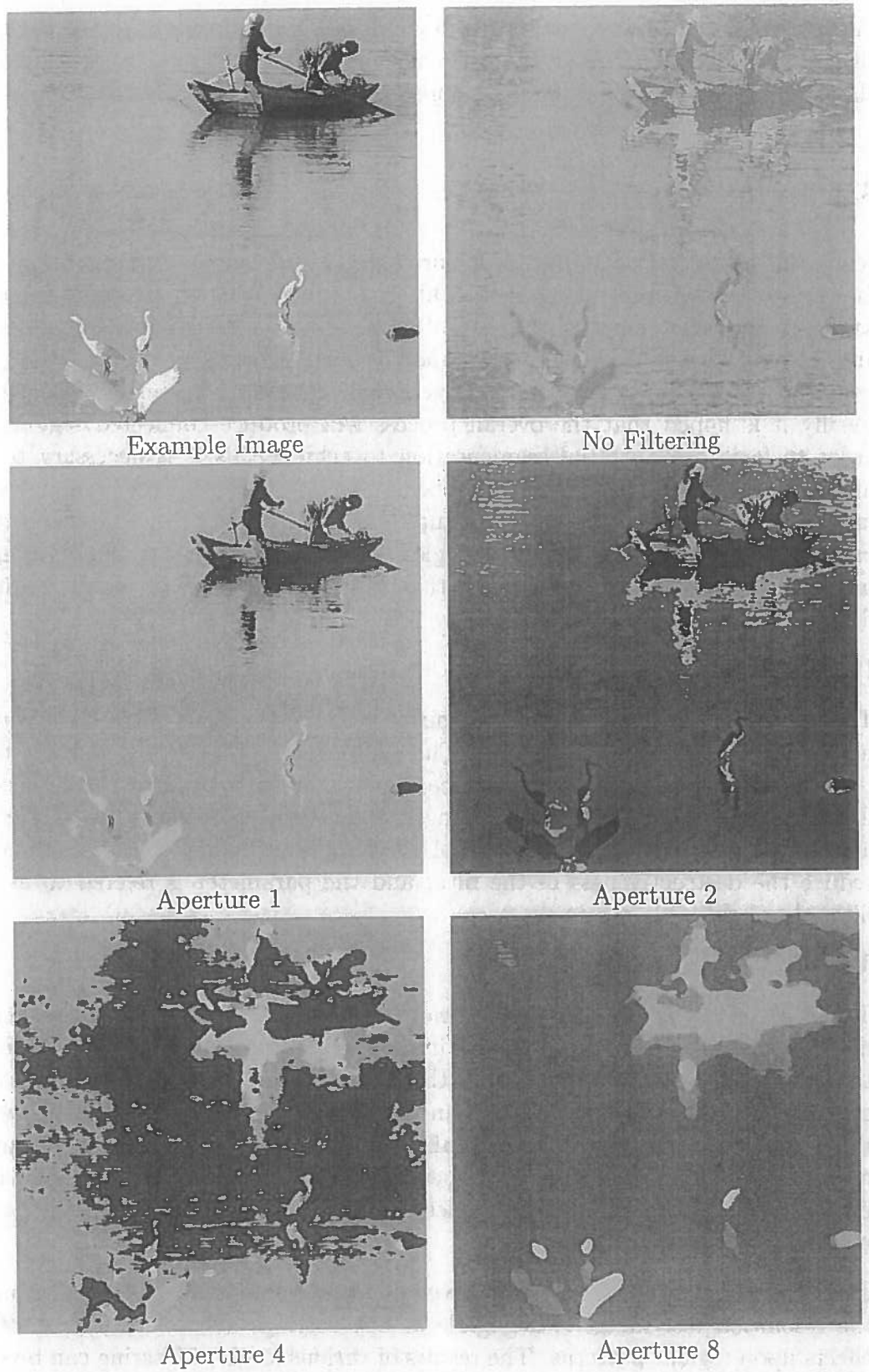


Figure 3.15: An example image and 5 false colour segmentations at various levels of ID filtering with a filter threshold of 0 % .

It can be seen that very small apertures provide good reduction of small regions without destroying overall structure. Better quality can be obtained at larger aperture sizes by increasing the filter threshold level however this is largely counter productive as these larger sizes also require more time to compute. It is worthy of note that any value for the filter threshold below $\frac{1}{k}$ is essentially the same as 0, as no region can have a majority with less than this value.

The Speed of ID Filtering

To test the speed of the ID filtering process a small batch of 10 runs at different aperture sizes was conducted. All non essential processes were disabled and only one iteration of K-Means was conducted. In order to isolate the time spent on filtering, the process was run 10 times without filtering to get a base line value ($1050.2ms$), which was subtracted from the measured runtimes to produce the values in **Figure 3.16**. The process was run with all other processing steps removed from the K-Means process using one iteration and a fixed filter threshold of 0 % .

The filter completes extremely quickly and is capable of processing an aperture of 5 (corresponding to a square region of 121 pixels) faster than the SNN filter processes a neighbourhood of 9 pixels. ID filtering is able to achieve these speeds as the process is only concerned with a single integer as opposed to a triple of integers to denote colour. This vastly reduces the time required to perform operations over the same neighbourhood.

The time per pixel relates to the time per pixel in the neighbourhood. It can be seen that the time is roughly linear with the number of pixels in the neighbourhood and thus in proportion with the square of the aperture size. Due to the extremely fast performance at low aperture setting, along with the ability to preserve small details apertures of 1 or 2 are recommended.

3.2.5 spatial / Non spatial Colour Merging

The K-Means process is limited by its strict adherence to producing k regions irrespective of their suitability. In general under-segmentation can be remedied by increasing k, over segmentation (the likely consequence of increasing k) is harder to deal with. To remedy over segmentation two merging techniques were developed which merge regions with similar colours, one with consideration to spatial placement, one without.

Non-spatial Colour Merging

Pure colour space segmentation of an image can result in separation into more different colours than are broadly speaking present in the image. Of course in any real coloured image there will be thousands of distinct colours by value, in

NHood Radius	Pixels in NHood	Time (ms)	(ms)/pixel
1	9	83.5	9.28
2	25	160.5	6.42
3	49	331.8	6.77
4	81	495.0	6.11
5	121	732.4	6.05
6	169	1041.8	6.16
7	225	1612.6	7.17
8	289	1846.0	6.39
9	361	2312.6	6.41
10	441	2860.4	6.49

Figure 3.16: The timings of segmentations with different ID filter apertures with filter threshold of 0% .

this context it is the extraction of possible underlying structure that is of interest. That is to say if there is a largely red object in the image then the potentially large number of distinct reds within that region are not of interest, instead it is the representative red that classifies the region.

Under this definition an over segmented image will exhibit regions which are very similar in colour value. To counter this all regions that fall below a certain colour distance threshold can be merged. The benefit to this procedure is that a high value of k can be used in conjunction with a colour merge threshold and the process will return *at most* k regions with at least the specified distance in colour space.

spatial Colour Merging In spatial K-Means segmentation of an image, under the sum of squares distance measure, connectedness is not guaranteed. This creates the dual problem of single regions which are not connected and multiple regions which are connected being separated. Single disjunct regions can be remedied to some extent by increasing k leading to smaller more connected regions at the cost of more over segmented connected regions.

Such multiple regions which are not connected but should be can be remedied by a process of comparing each spatially adjacent pixels attributed to different regions and merging the parent regions if the distance is below a specified threshold.

At the extreme, where k is the number of pixels, this process is exactly region growing, however in general when k is much less than the number of pixels the process is more efficient than region growing. The process presented assumes that an ID array has been produced by the earlier segmentation process which consists of an integer representing membership to a similarly numbered region at each (x, y) position.

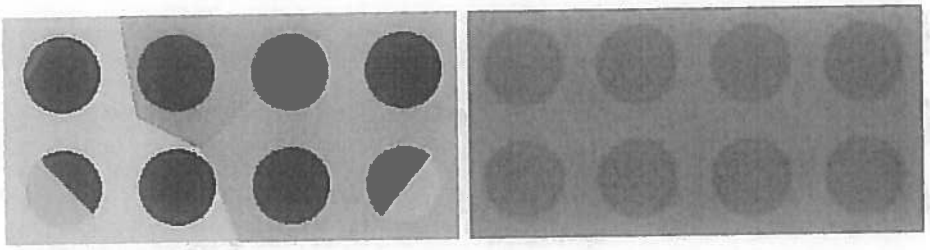


Figure 3.17: An example of over segmentation (left) and the results of spatial colour merging (right)

At each position in the array the value need only be compared with the pixels directly to the right and directly below independently to guarantee that all connections between pixels are explored. If the values are identical then no further action need be taken, otherwise the means of the two regions specified by the ID values should be compared as in non-spatial colour merging and merged as appropriately. The benefit over classical region growing is that most pixels will be in the same region as those adjacent to them whereby the computation required is only two integer comparisons rather than a sum of squares value. Additionally no redundant checks are made as the process only needs to be able to check all connections rather than follow them (possibly in 4 directions).

In practice the process completes extremely quickly (in the 100s of ms) and produces the effect that no connected regions fall in two segments. Due to the fact that the process merges *all* pixels in two regions that share at least one connection, the regions produced are often the union of disjunct regions. In the example of **Figure 3.17 (left)** an image can be seen in which spatially adjunct areas of similar colour are divided into separate regions (the dots and background). After the colour merging process (**Figure 3.17 (right)**) the connected areas of similar colour have been merged, resulting in the background being correctly classified as one region and the debatable effect of classifying all of the circular regions as one. This is not an entirely undesirable quality as a human observer may view a dense area of dots as one region rather than as many separate and unconnected regions.

To decide at which point discontinuous regions become merged consider the whole image to contain n pixels and let the overall density of 'in' pixels that are being considered for the spatial colour merging procedure to be d_0 then the total number of 'in' pixels is $d_0 n$. Let a region R be circular with radius r, then the area of R is clearly πr^2. If a proportion d of R are 'in' pixels then the total number of 'in' pixels in R is $d\pi r^2$. Assuming the k means are distributed randomly though out the 'in' pixels we would expect the number of means in R to match the percentage

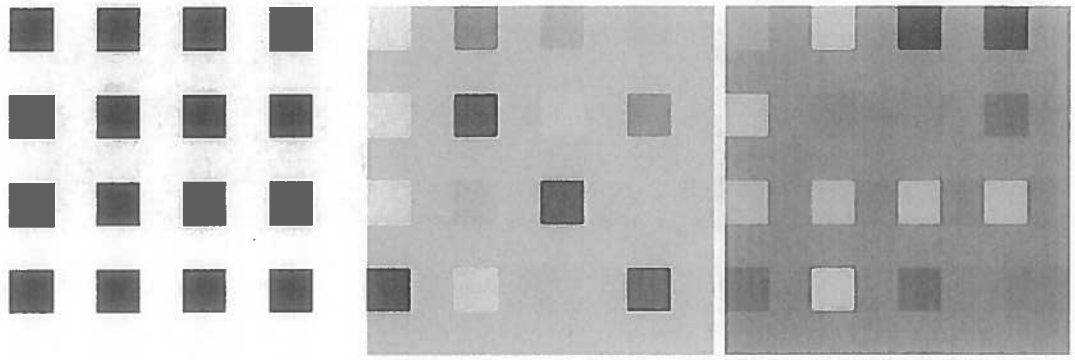


Figure 3.18: A regularly spaced pattern of squares (left) with the segmentation produced in false colour at $k = 21$ (centre) and $k = 81$ (right). Note the centre image exhibits 3 linked regions from the top left down.

of 'in' pixels in R, thus the mean number of means (K) centred within R is:

$$K = \frac{d\pi r^2}{d_0 n} k$$

Now let $K = 1$ then the following relations hold:

$$r = \sqrt{\frac{d_0 n}{d\pi k}} \qquad k = \frac{d_0 n}{d\pi r^2}$$

That is to say in a region of local density d one would expect a region of radius r to contain one mean centre point on average. If we consider the region to be centred on the mean then R gives us some idea of the average spacing at a given density. The second equation gives us an idea of what k to specify to maintain a certain radius. Note that the d in this case is only the local density at the centre of that particular region, for an image of varying local density a more sophisticated application of this formula would be necessary.

As a simple validation of this result consider a 512×512 image split into 16 square regions each of which has the upper left quadrant coloured black so as to form a regular spacing of 64×64 black squares at 64 pixels apart. The overall density d_0 is clearly $\frac{1}{4}$ and as a simplifying assumption for any suitably large neighbourhood the local density can also be viewed as a constant $\frac{1}{4}$.

As the gap between the squares is 64 pixels wide then this is the minimum radius at which we would the regions could be considered separate as over this value a seed in one square may claim pixels in another. Thus using a radius of 64 pixels the result is:

$$k = \frac{d_0 n}{d\pi r^2} = \frac{0.25 \times 512 \times 512}{0.25 \times \pi \times 64^2} \simeq 20.37$$

Reassuringly this value is above the bare minimum of 16 regions needed to account for the 16 squares. Note that this value only guarantees that the *average* spacing will be 64 pixels apart, so unless all of the means are spaced exactly 64 pixels apart then some regions certainly will be merged at this value. Instead this value should be considered the absolute minimum at which separation is possible. It is worth noting that as the values in the spatial segmentation are randomly attributed there is no value at which separation is guaranteed.

As a rule of thumb however using a radius of roughly half of the minimum required separation yields good results. Accordingly halving the radius will quadruple the number of regions needed (in this case to about 81). The results of this can be seen in **Figure 3.18**, where as predicted linked regions can be observed at $k = 21$ where the top three squares are merged in the far left row of the centre image.

3.3 Evaluation

3.3.1 Speed

The speed of the processes presented is of critical importance to the environments that they are employed in and is of central concern to this project. A thorough investigation of the effects of the various parameters upon the speed of the process was therefore conducted.

To evaluate the speed of Region Growing and Two Phase K-Means each was run upon a test set of images acquired from the Hypermedia Image Processing Reference (HIPR2)[7] image library. The library contains 189 greyscale and 59 colour images of a wide variety; ranging from assembly parts and medical images to normal photographs. All images were scaled to square images of width 64, 128, 256 or 512 for compatability with the program.

Region Growing

The Region Grower was tested with colour merging thresholds of 50, 100 and 200 (with the Symmetric Nearest Neighbour Filter (SNN) on and off for each) upon the colour and black and white images at the four resolutions. No repetitions were performed due to the large time taken to run the tests. Full results can be seen in **Appendix A.2**.

Superior Greyscale Performance

It is immediately clear that the process appears to work a good deal faster on the greyscale images than the colour images. While this may indeed be the case it is worth noting that this could instead be due to the fact that *these* greyscale images are quicker to segment than the coloured ones, rather than all greyscale

images. There is no reason for greyscale images to run any faster merely by virtue of their lack of colour as no special optimisations are used upon them as they are represented in the same manner (three colour channels). Therefore any speed improvements will be due to the distribution of the greys in the colour space and the effect his has upon segmentation.

For example the process will be slowed by the generation of large numbers of segments as there is a fixed overhead in initialising the data structure regardless of how many pixels will be attributed to it. Another effect upon the runtime is the number of colour comparisons made. These comparisons can only be avoided by a pixel already being claimed by another region, therefore the longer pixels remain unclaimed the more colour comparisons may be made against them. For this reason large regions which claim many pixels are preferable as they will minimise future comparisons.

Many of the greyscale images are, barring noise and resizing effects, monochrome. This would lead naturally to larger regions and so possible higher speed.

Disproportionally Good Performance on Small Images

The performance of the region growing process is obviously far greater upon small images, therefore to compare performance more directly between the images sizes, the timings were scaled to a value showing the milliseconds taken to process 1000 pixels. The graph can be seen in **Figure 3.19**.

As one would expect there is a slight decline in performance for the very smallest images, as certain constant time procedures common to all image sizes may exhibit a greater relative performance hit. But surprisingly the performance for the largest size (512×512) is significantly worse per pixel the previous sizes.

One possible explanation for this is the nature of the image test set. Many of the images scaled to the 512×512 resolution were of similar size, therefore noise in these images would be largely preserved. In addition many of the colour images from the HIPR2 library are 8-bit indexed colour images; that is to say that they only possess 256 unique colours. This is often accompanied with dithering to fool the eye into believing there are a greater number of colours. The effect of dithering is to create a perceived gradient between two distinct colours by scattering the pixels of each colour with a varying percentage of colour belonging to each. For example an apparent purple colour can be obtained by a scattering of red and blue pixels. The relevance of this to the region growing procedure is that the ends of boundaries will not be described by the smooth change from one colour to another but rather a ragged scattering of colours. To the region growing process this effect appears as noise, increasing the number of regions created and so increasing the time taken.

The effect of resampling the images is such that the smaller images do in fact

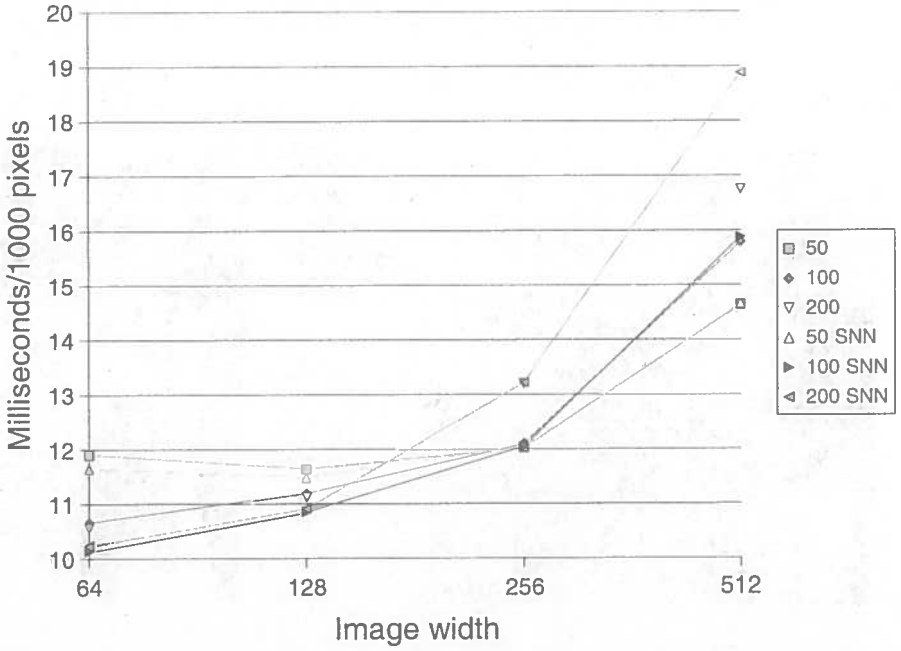


Figure 3.19: A graph of the relative performance of the Region Growing method with various parameters. The timings shown are in $ms/1000$ pixels.

contain smooth gradients as each pixel is the average of many. Therefore it is reasonable to expect a greater proportion of noise and dithering effects to survive resampling to similar or larger sizes. This effect can be seen in **Figure 3.20** whereby a 128×128 section of a 512×512 image has been selected to be in pixel to pixel corresponce to a smaller 128×128 image. Far more dithering effects and noise are visible on the larger image.

To test this hypothesis the region grower was run upon two test images, one of uniform neutral grey and one with random coloured noise. The noise was generated at each different resolution to avoid the softening effect introduced by resampling. If the swift performance of the method is due to the reduced noise in smaller images then the resu already described with threshold 200 and no SNN filtering (to preserve the noise). The results can be seen in **Figure 3.21**.

In addition to the time and per pixel performance another figure is given showing the time of the process at the current resolution divided by the time for the previous resolution. As the process is theoretically linear with the number of pixels it is expected that this figure should be 4 for all processes (as the image quadruples in area each time).

It is clear from the results in **Figure 3.21** that the process did not obey a linear

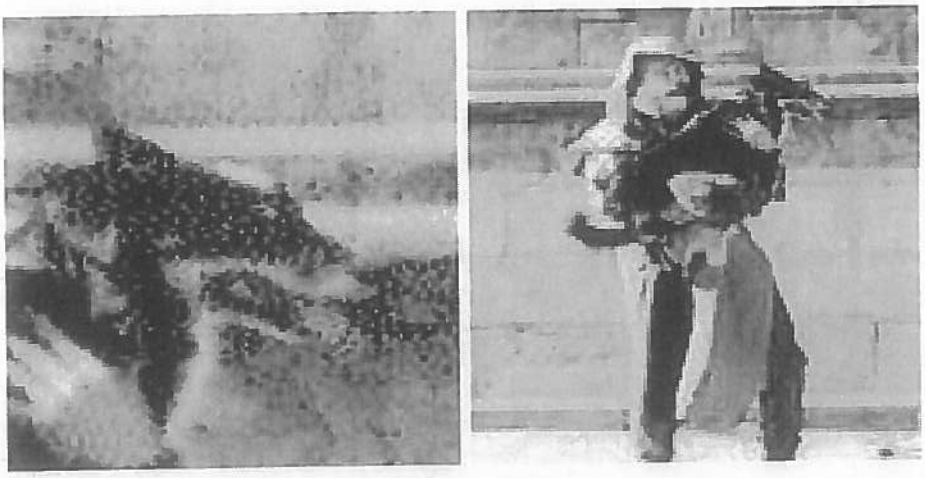


Figure 3.20: A magnified section fo a 512 × 512 image (left) exhibiting more dither produced noise than the resampled 128 × 128 image (right)

Image	Measure	64	128	256	512
Neutral Grey	Time (ms)	39.40	177.90	887.93	5149.37
	ms/1000 pixels	9.62	10.86	13.55	19.64
	% of prev run	n/a	4.52	4.99	5.80
Random Color	Time (ms)	120.83	460.00	1989.57	Mem Error
	ms/1000 pixels	29.50	28.08	30.36	n/a
	% of prev run	n/a	3.81	4.33	n/a

Figure 3.21: A table of various results of 30 repetitions of the region growing process running upon a neutral grey image and a random color image.

Threshold	64	128	256	512
50	-2.06%	-1.35%	0.24%	0.12%
100	-4.90%	-3.19%	-0.45%	0.59%
200	-3.10%	-1.95%	0.12%	12.71%

Figure 3.22: The percentage increase of runtime upon enabling SNN filtering for various thresholds in the Region Merging procedure.

time for the neutral grey image, but instead followed a similar pattern to the previous results; increasing time at higher resolutions. This is clearly visible from the values of the multiples between image sizes which are all over 4. It is clear that the non linear effect is not due to differing levels on noise in the image and that some other unpredicted effect is at work. One possibility is the increased activity of the Java garbage collector when the memory demands are higher.

It is worth noting that as predicted the random image provided significantly poorer performance than the neutral grey. Furthermore the time for this process was consistently around 30 $ms/1000$ pixels. As this value is higher than any of the other results is is possible that the process is bounded above by this linear time, and therefore is linear itself. On the 512×512 image the Java virtual machine experienced an out of memory error. This will be due to the large number of regions which would be generated by such an image as each region will require a fixed minimum amount of memory.

SNN Filtering Reducing Runtime

The results of enabling and disabling the SNN filter have been summarised in **Figure 3.22** as the percentage extra time required dueing the segmentation phase when filtering is enabled. Therefore negative values represent a time saving and the filtering can be observed to save time in most cases except for the 512×512 images. However apart from the threshold 200 run the added time is negligible. In the worst case enabling filtering here increases the time required by 12.71%, which translates to an increase of 558 ms. Note that these values do not include the extra time take to actually perform the filtering; in depth results for the speed of SNN can be gained in **Section 3.1.3**. Furthermore the quality of segmentation is increased when the SNN filter is enabled.

Two Phase K-Means

The Two Phase K-Means process was run upon the HIPR2 image library test set. The process was constrained to $k = 6$, separated seeding and one iteration though the set due to time constraints. Various features of the K-Means procedure were enabled and disabled to form a better idea of their impact upon the runtime of the whole process. The full results of the test can be seen in **Appendix A.2**.

Superior Greyscale Performance

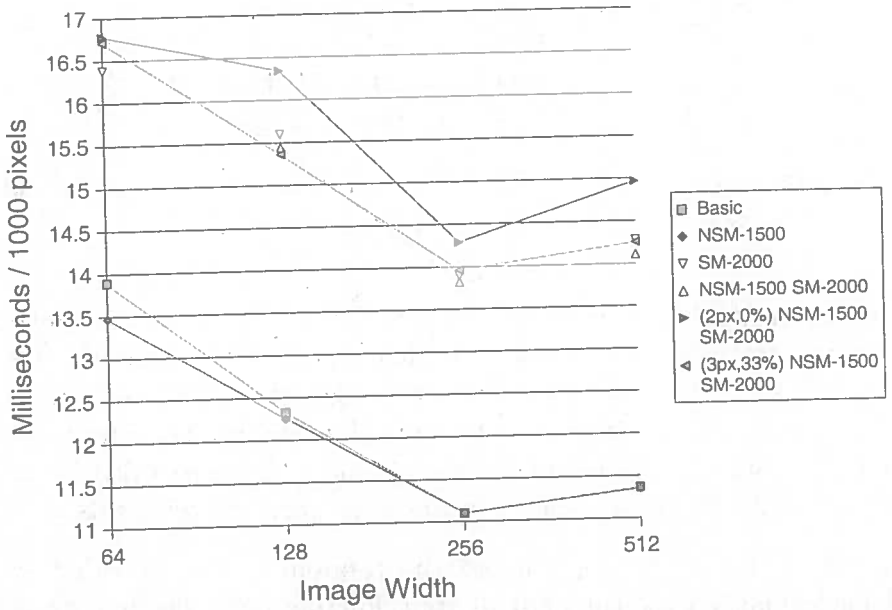


Figure 3.23: A graph of the relative performance of the K-Means method with various parameters. The timings shown are in $ms/1000$ pixels.

Again K-Means presents superior performance upon the greyscale images than the colour images. However in this instance the trend is less marked the larger the source image. The K-Means approach will increase in runtime when the pixels allocated in one iteration shift positions in the next. This is due to the additional processing that has to be effected upon the means of the region the pixel leaves and joins to reflect the change. Otherwise the procedure should have nearly identical performance regardless of the contents of an image.

Constant Time Behaviour Over Varying Image Size

Unlike the Region Growing process, K-Means is very well behaved when it comes to varying the scale of the image. The results of scaling to $ms/$ 1000 pixels can be seen in **Figure 3.23** and clearly show the process to be bounded above. In the figure SM and NSM relate to the presence of spatial and non spatial colour merging thresholds whereas the tuples (2px,0%) and (3px, 33%) represent the presence of ID filtering at 2 and 3 pixel radii and 0

Decrease in Performance due to spatial Colour Merging

The speed of the process under different settings can be broadly separated into those that employ spatial colour merging (denoted SM in the graph) and those that do not. Other than this distinction the runtimes are so similar as to be virtually indistinguishable.

While this might indicate a reason to disable spatial colour merging it should be remembered that without this step the second spatial phase is prone to generating very over segmented results. spatial colour merging should therefore be regarded as an essential part of the process. It is the authors belief that with suitable time a solution could be found to greatly reduce the time spent on spatial colour merging as most of the computation of the process is actually spent converting between the contents of the ID array and segments rather than merging regions. For this reason if an implementation that omitted these conversion steps were introduced, the processes speed would benefit greatly.

Increase in Performance due to Non-spatial Colour Merging

As the non-spatial colour merging step works directly with the values in the ID array, and serves to reduce the number of regions to undergo segmentation in the second phase, a small but uniform increase in performance can be observed when it is enabled. The effect can be observed in **Figure 3.23** where in both bands of performance the fastest procedures were those with non spatial merging enabled. As the process also increases the quality of the segmentation by eliminating extremely similarly coloured regions it should be enabled by default.

Linear Increase in Runtime with k

A set of 20 runs though the colour and greyscale image was performed with the Two Phase K-Means with k ranging from 1 to 20. The raw results can be seen in **Appendix A.3**. **Figure 3.24** summarises the results of the tests and clearly shows the procedure to be linear with respect to varying k.

3.3.2 Comparison with Hand Segmentations

To attempt to quantitatively assess the performance of the quality of the segmentations produced by the procedures it was necessary to produce a measure that compared the segmentations to some ground truth. This could easily be achieved by checking to see if the segments produced exactly matched those in a hand segmentation, but several problems with this approach exist.

- Non-Unique Solutions

  Two different segmentations may be of equal quality. If one single ground truth is used to evaluate them how can it possibly differentiate?

- What is a Good Segmentation?

  The items that might be of interest in one context may be completely useless in another. Where the boundaries lie depends on what objects you are trying to find.

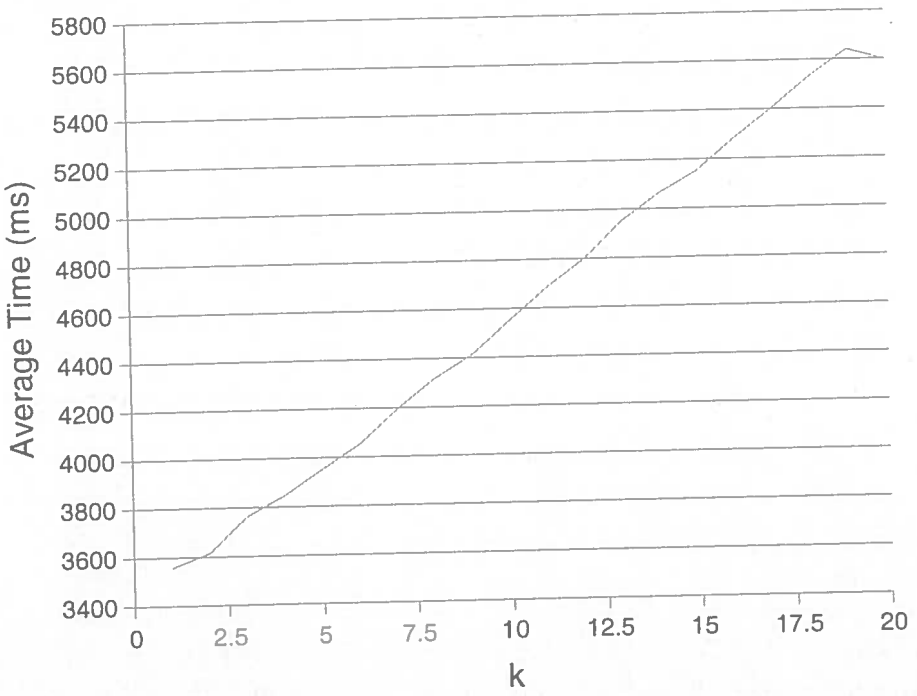


Figure 3.24: A graph of the performance of the K-Means method, vaying k from 1 to 20. The timings shown are in ms

To answer both of these questions it was decided that the hand segmentations produced should follow the boundaries of objects that any object centric segmentation should pick out. This means that the regions defined in the segmentation follow the boundaries of the most significant objects in the scene rather than the most prominent boundaries of colour. It also means that in the instance where a human observer would find it hard or ambiguous to place an exact boundary none was placed. This situation arises, for example, when tracing the outline of a tree; although the basic shape can be seen when the edge is inspected closely it becomes increasingly difficult to decide what is leaf and sticks and what is background.

To answer the question of non unique solutions, we have already stated that the objects we have highlighted in the hand segmentation are so obvious that any segmentation should pick them out. This means to say that although there may be many segmentations all should contain at least these objects.

The Weighted Segment Coverage Measure (WSCM)

The WSCM was developed to allow the comparison of two segmentations that may differ but follow the same boundaries of the hand example and conclude that they represent the same quality. Briefly the WSCM is a measure that will

return 100% for any segmentation that has regions that never cross the boundaries defined in a specified hand segmentation and 0% for a completely random allocation of pixels.

For clarity the predefined regions will be refered to as objects and the regions being considered for evalutation will be refered to as segments. If $S_{1...n}$ are the segments to be compared with the defined objects $O_{1...m}$ then the WSCM is as follows:

$$W_{SCM}(S_{1...n}, O_{1...m}) = \left(\left(\sum_{i=1}^{n} \sum_{j=1}^{m} \frac{|S_i \in O_j|^2}{|S_i||O_j|} \right) - 1 \right) \left(\frac{1}{m-1} \right)$$

The inner summation of the equation is a weighted sum over object O_j of the percentage of segment S_i in O_j weighted by the percentage of O_j that it covers. The percentage of S_i in O_j is given by $|S_i \in O_j|/S_i$ and the percentage coverage of O_j is given by $|S_i \in O_j|/O_j$. The sum of all the fractions of O_j will be 1 and therefore the inner summation will be at a maximum when the percentage of S_i in O_j for every i is also 1, that is to say that each S_i in O_j is wholly in O_j.

As this sum is performed for all m objects the central section is divided by m to produce a number in $[0, 1]$. This would be the case if the minimum of the summation over m were indeed 0, but it can be shown for a purely random assignment of segments the minimum is in fact $\frac{1}{m}$. For this reason the further scaling evident in the equation is performed to return the measure to the full range of $[0, 1]$.

Properties of the WSCM

- A completely random allocations of pixels will have a WSCM of 0% for any predefined segmentation
- Any segmentation with regions that do not cross any boundaries in the defined segmentation will have a WSCM of 100%.
- Therefore a segmentation consisting of every pixel in its own region will have a WSCM of 100%.
- The larger a number of segments a segmentation contains the closer to this situation it will be and so the higher WSCM it will have.
- Therefore values of WSCM are only comparable between segmentations with equal numbers of segments upon the same predefined segmentation.

Testing Segmenters: Crystal, Grid and Random

The fact that WSCM values are only directly comparable between segmentations with equal number of segments (as it is easy to get a high WSCM with a high

number of segment) is a severe draw back, as this will undoubtedly be the case for most examples. To combat this several testing segmenters were developed to randomly segment the image in a specified way into a specified number of segments. As these testing segmenters completely ignore the contents of the image they can be deemed to represent the value that is returned by utterly uniformed segmentation. Therefore the performance of any segmentatation can be evalutated in terms of how significantly it beats these random segmenters. The different segmenters are as follows:

- **Crystal Segmenter**

  This segmenter is in fact the K-Means segmenter set to perform spatial segmentation over the whole image without any filtering of merging. As there is no colour information to work with this will produce polygonal segments independent of the contents of the image. The segmenter is called the Crystal segmenter due to the crystal like output. This segmenter is highly useful as it can produce a random clumped segmentation with any number of segments by specifying k.

- **Grid Segmenter**

  The Grid segmenter chops the image into a regular grid using the square number which most closely approximates the required number of regions. As the Grid segmenter cannot exactly match the number of regions required its WSCM value will always be given as a tuple including the actual number of segments produced.

- **Random Segmenter**

  The Random segmenter allocates the pixels in an image to a specified number of regions completely randomly. The WSCM value of this segmenter should always be very close to 0%.

The WSCM Results

A series of tests were conducted to obtain the WSCM measures for various settings of the Region Grower and Two Phase K-Means processes. Twenty repetitions of the algorithms were performed upon a test set of 5 images with hand segmentations. The Region Grower was run at thresholds 50, 100 and 200 with SNN on and off. K-Means was run with k =6,12 and 18, spatial merging at 2000, non spatial merging at 1000 and ID filtering on a 2 pixel radius with 0% threshold. For full results and details of the WSCM results collected see **Appendix A.5**.

To allow the WSCM values to be compared between the various techniques it is necessary to account for the fact that high numbers of segments are likely to increase the WSCM value. This was achieved by dividing the WSCM value

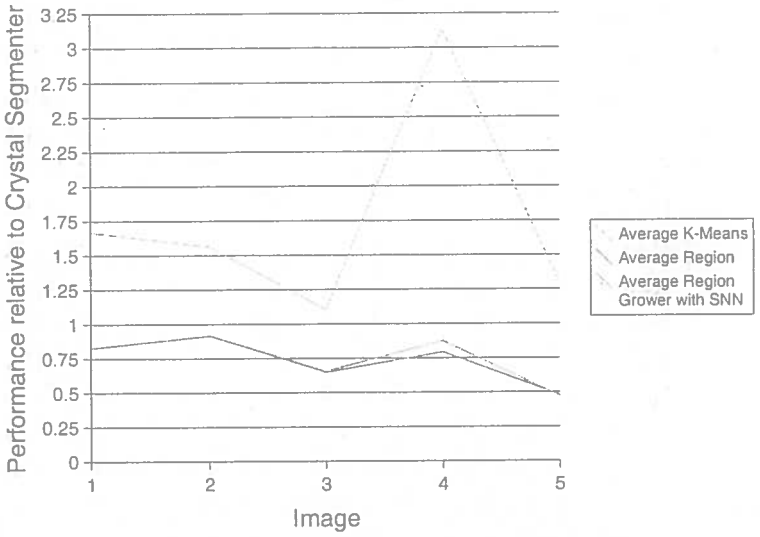


Figure 3.25: A graph of the performance of the adjusted WSCM values for the average of the Two Phase K-Means and Region Grower runs.

obtained from a run of the segmenter in question by the WSCM value obtained from the Crystal Segmenter set to produce the same number of segments that were produced in that run. This gives an estimator for the performance relative to an uninformed segmentation technique and is referred to as the adjusted WSCM.

Figure 3.25 shows the average adjusted WSCM performance of the Two Phase K-Means, Region Growing and Region Growing with SNN. It can be seen that the K-Means approach achieves a significant improvement over the Region Grower due to the vastly smaller number of regions produced. **Figure 3.26** shows this more clearly with the average number of regions produced and the ranking of the algorithms based on the adjusted WSCMs for each image.

The K-Means based approaches all rank above the Region Growers and all runs of the Region Grower with SNN enabled rank above the same threshold run without. While the increase in significance may only be small for enabling SNN, the number of regions is always significantly reduced (usually around the order of one half). In this ranking the low threshold Region Growers perform well but the number of segments produces is very high and so the significance of the segmentation is reduced. Other than this the subjective comparisons which follow confirm that this ranking accurately represents the significance of the segmentations produced; particularly as the Two-Phase K-Means based approaches achieve a very high WSCM measure with extremely low segment counts compared to the Region Grower.

Process	Mean Rank	Mean Segments
k=6 (2,0%)	1.2	9.1
k=12 (2,0%)	2.0	16.2
k=18 (2,0%)	2.8	21.0
Reg 50 SNN	5.6	2161.6
Reg 100 SNN	5.8	501.0
Reg 50	6.2	3943.0
Reg 100	6.4	864.0
Reg 200 SNN	7.0	175.6
Reg 200	8.0	287.0

Figure 3.26: A table of the relative rankings of the procedures based on the adjusted WSCM and the average segments produced during segmentation.

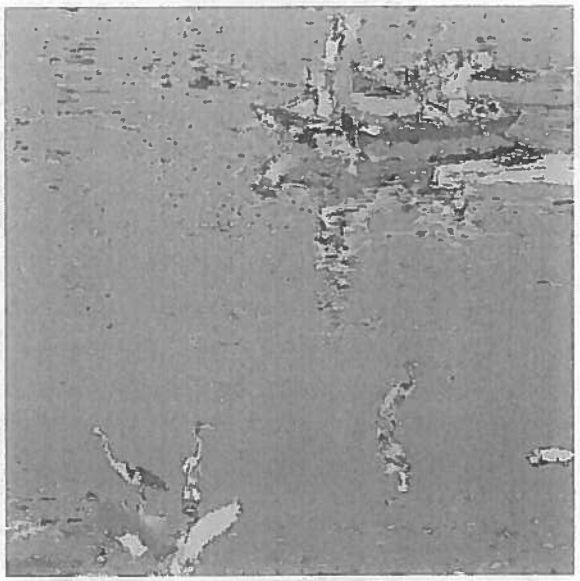
3.3.3 Subjective Comparison

Subjective Performance A difficult but important topic to discuss is the quality of segmentation as it appears to a human observer. Necessarily such assessments of quality will be subjective but serve to highlight important features of the performance of the various procedures that other measures do not. Note that all images given in this section are presented in false colour to highlight the different regions.

Region Grower

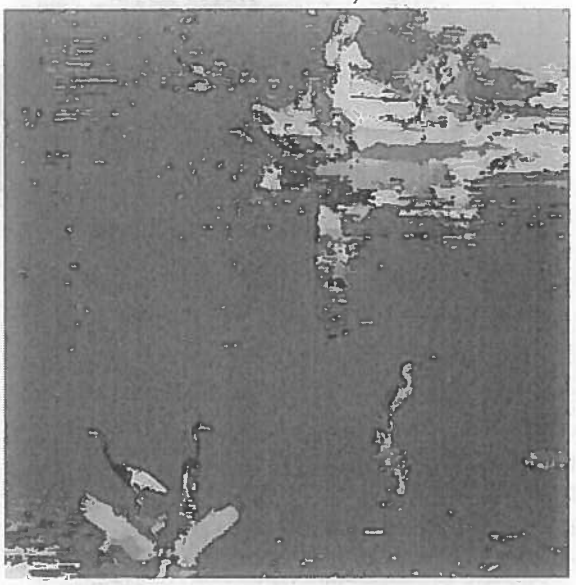
To produce example images for the region grower the process was run at various colour thresholds with and without SNN filtering to illustrate the effect. WSCM values are also included. As this mesure is only directly comparable with values produced from segmentations resulting in an equal number of segments values are also given for the Crystal Testing segmenter and Grid segmenter. As the grid segmenter can only segment to the nearest square number, values are given as a tuple of the actual number of segments produced and the WSCM.

Threshold=25



Segments = 9227 WCSM = 91%
Crystal = n/a Grid = (9216, 62%)

Threshold=25, SNN



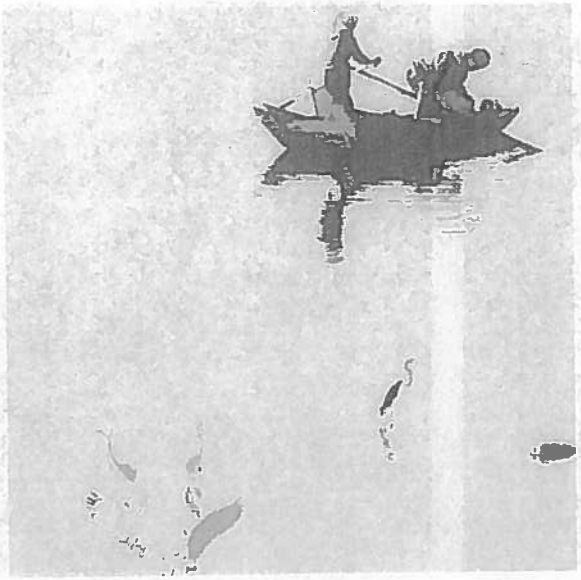
Segments = 4987 WCSM = 89%
Crystal = 84% Grid = (4900, 69%)

Obviously such a small threshold has caused the image to be highly oversegmented in both cases (at nearly 10000 segments). Although the WSCM is high, with this many segments almost any segmentation would achieve a good measure. It can be seen that the segmentation is not scoring significantly above the Grid segmenter. The Testing Crystal segmenter did not complete after 20 minutes and so the attempt was abandoned. This is not surprising as it is essentially identical to running K-Means with $k = 9227$.

The application of the SNN filter nearly halves the number of regions produced while preserving almost the same WSCM. Many small noise segments are visible in the water section of the image without filtering which are subsequently removed by the filtering process. Although some structure is visible a less strict threshold is obviously required to differentiate between noise and structure. It is difficult even for the human observer to pick out the structure of the boat and its occupants with this many subdivisions.

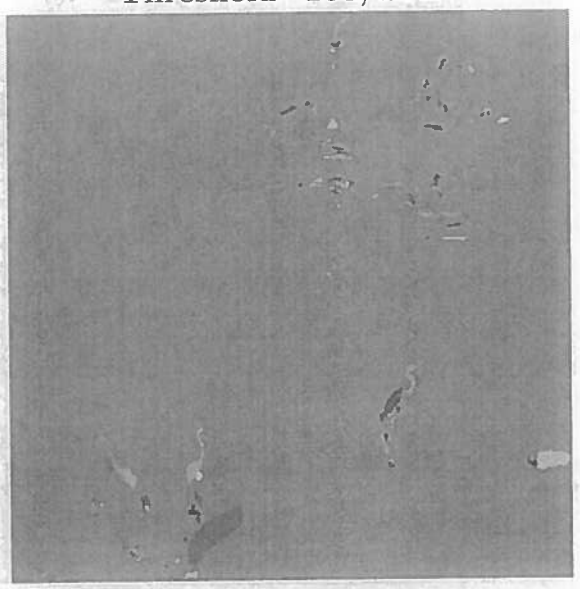
The nature of the region spreader is to segment, very precisely, each patch of colour under the specified threshold. In the case that two very similar regions are separated by even one pixel there will be no merging of the regions. This can be seen in the water where the ripples each form their own individual regions despite strong similarities and spatial proximity.

Threshold=100



Segments = 353 WCSM = 55%
Crystal = 56% Grid = (324, 55%)

Threshold=100, SNN



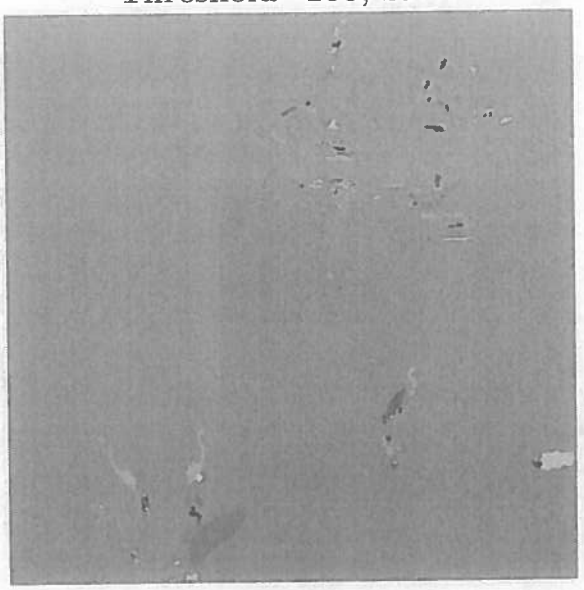
Segments =273 WCSM =48%
Crystal = 52% Grid = (256, 48%)

It is immediately clear that the segmentation at threshold 100 is significantly better with the large scale features such as the boat and birds visible. This improvement is not apparent in the WSCM where in both cases the WSCM performance has been equivalent to the random segmenters. In this instance the measure is unrepresentative as many of the larger segments do indeed correspond well to features in the image.

In tandem with the clarity of structure in this segmentation the number of segments produced has been dramatically reduced. Again enabling the SNN filtering reduces the segment count, however in this case the WSCM has been more significantly effected.

Event though many of the large scale features are well represented the segment count is still very high compared to the number of significant segments observable in the image. The temptation would be to increase the threshold in order to try and eliminate these smaller regions. However it can be observed in both images the the birds wing in the lower left hand corner has already been absorbed into the water segment and any further increase in threshold will on exacerbate the problem.

Threshold=100, SNN



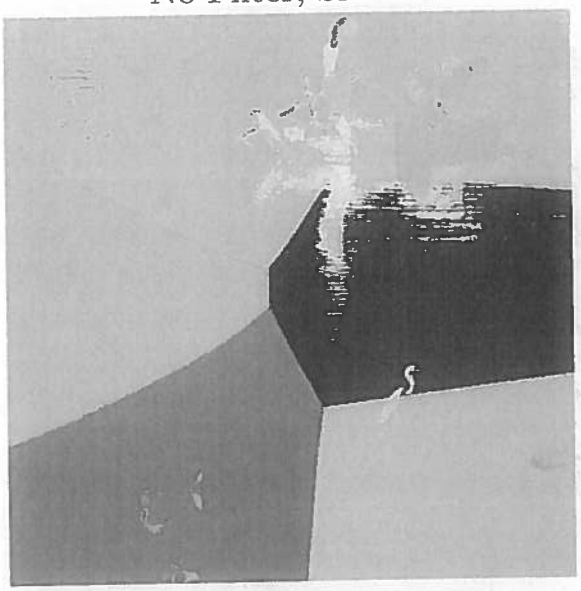
Segments = 118
WCSM = 18%
Crystal = 40%
Grid = (100, 33%)

When the threshold is reduced again many of the smaller features are obliterated as predicted. The WSCM suffers accordingly and the process significantly under performs the random segmenters. Although this is the case for this image it is possible that an image with a larger dynamic range could benefit from such a high threshold.

Two Phase K-Means

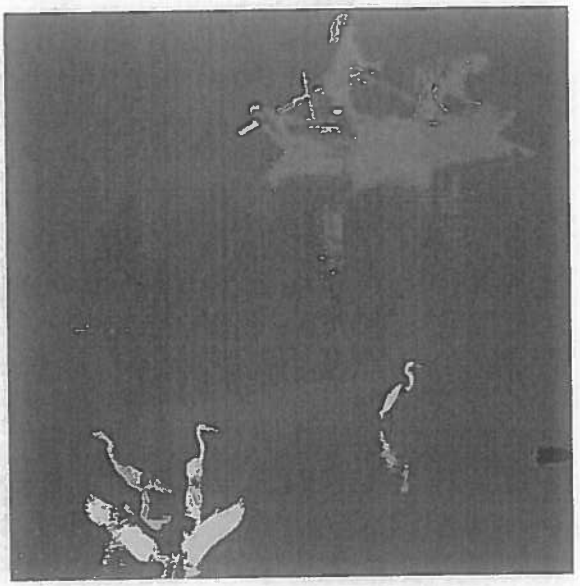
For the Two Phase K-Means example images settings of $k = 6$, Non-spatial Colour Merging (NSM) threshold of 1000 and a spatial Colour Merging (SM) threshold of 2000 were used. A setting of 3 iterations was used and filtering was set to 2px radius and a threshold of 0%.

No Filter, SM off



Segments = 22 WCSM = 61%
Crystal = 19% Grid = (16, 15%)

No Filter



Segments = 14 WCSM = 62%
Crystal = 14% Grid = (9, 11%)

This example was created to illustrate the effect of disabling spatial merging. As is clear from the left image there are large artefacts in the form of false boundaries in continuous regions; most notable in the background and boat. For this reason the use of spatial merging is strongly recommended. In the right image the results can be seen on a different run where the similar regions in the water particularly, have been merged into one. Notice how the segmentations are different due to the randomised seeding points.

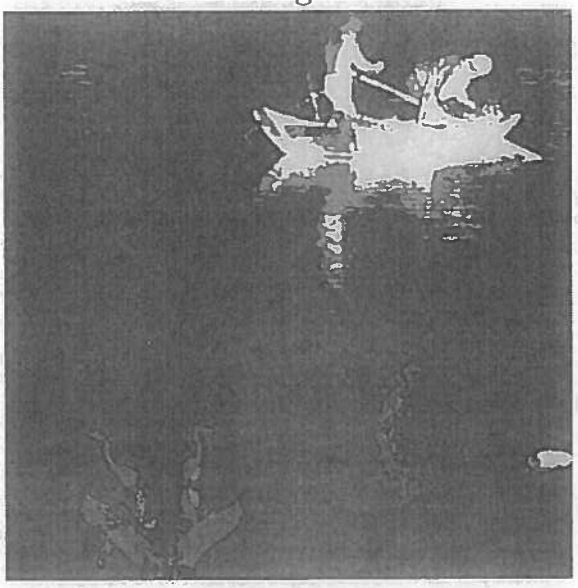
The number of segments produced is far lower than any of the region growing and is capped by k^2 which in this case is 36. Despite this limit on numerous segment the procedure manages a WSCM far in excess of the testing segmenters. In addition the spatial merging reduces the segment count again without effecting the WSCM. This implies that each of the segments in the merged region are of higher significance. Indeed most of the regions in the merged image can be seen to strongly correspond to a feature in the image.

Note that unlike the Region Grower the Two-Phase K-Means approach classes all of the bodies and wings of the birds in the bottom left hand area as one region in the non merged image. In this case this is an acceptable interpretation as a likely human response to the question "What are the important features of this image?" may well be "The birds there". In other cases this linking of regions is not a desirable property and can be counteracted by increasing k.

There are detectable single pixel and small island parts of regions visible particularly in the bottom left hand corner of the merged image. These are the results of the colour phase producing highly non-connected regions. This is exactly the

noise that ID filtering is designed to prevent.

ID Filtering Enabled



Segments = 8
WCSM = 65%
Crystal = 11%
Grid = (9, 11%)

Once the filtering has been enabled the resultant segmentation exhibits far fewer small island regions. In addition the individual segments are more contiguous. In this example run a very respectable WSCM of 65% was achieved. In comparison with the region grower the 8 segments produced here correspond more strongly to the image than the more than 300 produced in the threshold 100 runs.

The effect of undesirable linking however can be observed where the chest and feet of the lone bird are classed as the same as the lighter region of the rock.

Summary of Subjective Results

The segmentation performance of the K-Means based approach can be seen to far exceed that of the region grower. However Two Phased K-Means is a variable process and is not guaranteed to always give such predictable results as the region grower. In some of the runs above areas of the birds wings were omitted from the segments produced by the Two Phase K-Means segmenters, but on occasions the birds themselves are omitted altogether.

Furthermore the image used plays to the strengths of the Two Phase K-Means approach as it contains a limited number distinct objects separated by generous gaps. As can be seen from the results of the spatial Colour Merging section when the proximity of distinct objects reduces, k must be raised correspondingly to compensate. Thus on very densely populated images where under segmentation

is not acceptable region growing may be preferable.

4. Hierarchical Methods

In this section two techniques are presented that tackle different versions of the problem of extracting hierarchical information from an image. The Graph Based Segmenter generates a hierarchy based on connected component analysis of a previous segmentation; whereby the hierarchy corresponds to unions of regions which contain the unions of other regions grounded from the edge of the image. The Two Phase K-Means approach however produces hierarchical results based on the scale of details in regions of an image.

While both may be examples of hierarchical segmentation techniques their results as quite different as can be observed in the subjective comparison section of this chapter.

4.1 Graph Based Hierarchical Segmenter

The graph based hierarchical segmenter is in fact the flat region growing method with a post processing step applied to detect the connectedness between pixels and construct a Regional Adjacency Graph (RAG). From each region this RAG catalogues the regions which it is connected to. A graph processing step is then taken that converts the RAG to a Regional Hierarchy Graph (RHG) which attempts to capture the notion of the union of regions surrounding other regions.

RAG Representation

The RAG in this case is stored as a set of nodes of the format (Node# ,{nodes connected to}). The zero node is taken to represent the outside region.

For example in **Figure 4.1** the segmented image (left) with labelled regions

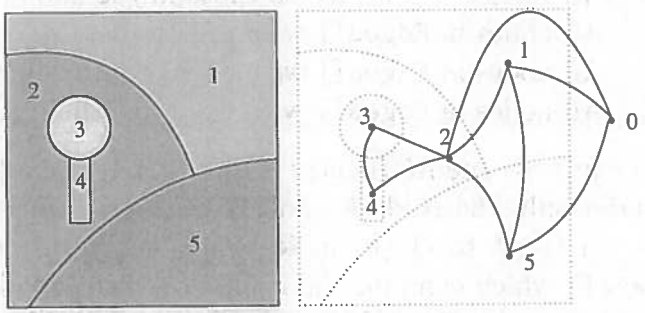


Figure 4.1: An example image and 5 false colour segmentations at various levels of ID filtering with a filter threshold of 0 % .

and the corresponding Regional Adjacency Graph (right) the RAG would be as follows:

0: {1,2,5}
1: {0,5,2}
2: {0,1,3,4,5}
3: {2,4}
4: {2,3}
5: {0,1,2}

Considering how to construct a hierarchy, it is not immediately clear what the parent of regions 3 and 4 should be as neither is completely surrounded by 2. It is clear that in some sense 2 surrounds both regions and so should be above both 3 and 4 in the hierarchy: in this method we will say that 2 surrounds the new region composed of 3 and 4. It is important to state that this is not the same as merging regions 3 and 4; we are merely stating that they have similarities in the hierarchical placement. We can also clearly say that region 0 is the root of the hierarchy from which all other regions follow. Given that 1,2 and 5 do not surround each other in any way, they must be at the same depth in this sense. Therefore the hierarchy we can construct is as follows:

{0} parent of {1,2,5} parent of {3,4}

This should be interpreted as region 0 surrounding the multiple region group {1,2,5} which in turn surrounds the multiple region group {3,4}. Note that in this representation the fact that 2 is the sole parent of {3,4} is not preserved.

RAG to RHG Algorithm

To retain all of the information of this nature and facilitate processing consider a new data type, called an RHG entry (Regional Hierarchy Graph entry), which contains :

`Nodes[]`	A list of nodes in this region group (replacing the Node#)
`Depth`	An integer depth from the 0 group.
`Edges[]`	All vertices connected to those in the node list.
`Above[]`	All nodes in `Edges[]` with greater depth
`Same[]`	All nodes in `Edges[]` with equal depth
`Below[]`	All nodes in `Edges[]` with lesser depth

To initialise the process one RGH entry is made for each entry in the RAG where `Nodes[]` includes only the Node# , and `Edges[]` contains the edges as in the RAG. The depth is set to -1 (to indicate not set yet) and all other sets are empty, bar `Same[]` which contains the Node#. A list `Todo[]` is composed of all the entries. An empty list `Done[]` to store the results and another intermediate empty list `Processing[]` are also initialised. For this process it is assumed that all nodes are connected to themselves.

The RAG to RHG algorithm is then:

```
Begin with only the entry with node list \{0\} in Processing[].

WHILE Processing is not empty:
    Remove the first entry e in Processing[]
    FOR every node n in e.Edges[]:
        IF the entry e2 for n is in Todo[]
    set e2.depth = e.depth
            remove e2 from Todo[]
            add e2 to the end of Processing[]
        ELSE e2 = the entry containing n in its node list
                    (either in Processing[] or Done[])
            IF e2.depth = e.depth
                merge all sets of e and e2 into e
                remove e2 from Processing[]
            IF e2.depth >= e.depth
                add e2 to Above[]
    END FOR
    IF e.nodes[] = e.same[]
        add e to Done[]
    ELSE
        add e to the head of Processing[]
END WHILE
```

The effect of this process is to generate a depth based on the distance from the edge of the image. Any segment connected to the edge of the image will have depth 1 and any regions which are enclosed by a union of these regions will have a depth of 2 and so on. The depth that a region will be placed at can be calculated by observing the minimum number of regions that you have to cross though to get to the edge.

For example, the finished RHG entries for the example above looks as follows:

Nodes[]	Depth	Edges[]	Above[]	Same[]	Below[]
{0}	0	{0,1,2,5}	{}	{0}	{1,2,5}
{1,2,5}	1	{0,1,2,3,4,5}	{0}	{1,2,5}	{3,4}
{3,4}	2	{2,3,4}	{2}	{3,4}	{}

Note that the {3,4} group retains the information about its 2 being its single parent in the Above[] list. It can be observer that this representation contains all of the information of the previous formulation of an RHG along with the specific information about the parents of a region.

4.2 Recursive Application Two Phase K-Means Hierarchical Segmenter

The Hierarchical Two Phase K-Means is a recursive application technique based upon the Two Phase K-Means process. The process operates on a multi resolution representation of an image in order to simultaneously reduce computational demands and extract information about the image at different levels of abstraction. Unlike the Graph based approach this method is both hierarchical in is procedure as well as output. **Multi Scale Image Representation**

The process works upon an image pyramid where the lowest level is in 1-1 correspondence with the source image and each subsequent layer on top is exactly half the height and width. The Pyramid is obtained by averaging each four pixels in a square regions to generate the pixel value for the layer above. The process runs up to a specified depth, which determines the maximum number of recursions as well as the hight of the pyramid and so the resolution of the smallest image at the top of the pyramid.

Each layer in the pyramid has an attendant Two Phase K-Means segmenter initialised on the image at that depth, this is to prevent the additional costs of adjusting one segmenter to work at the different depths as the procedure continues.

The Flow of Segmentation

The process is begins by initialising a root node for a tree based representation which includes then whole image at the smallest level, represented as one region. This region is the segmented using a regular Two Phase K-Means segmentation upon the region and the image with matching scale. Each of the segments produced in this step are considered as children of the parent region and compared against various recursion termination conditions. This is to prune recursion in unsuitable regions which can allow large time savings as each pixel processed at level $n+2$ represents 4 pixels in level $n+1$ and 16 in level n and so forth.

Children are pruned if they fall below a certain size as it is deemed that all the important information has been gained at this level. Heuristically this level was set to $\frac{5}{k^2}$ of pixels in the image at the current resolution to allow the value to alter with scale and k used. If only one child was produced then this was also pruned as no further information has been gained. Lastly if the maximum recursion depth has been reached recursion is also terminated.

For segmentation to take place at a more detailed resolution it is necessary to rescale the parent segment prior to re-segmentation in to the the new larger resolution. This is achieved by writing the value of each pixel in the parent region into the corresponding four pixels below. This results in a blocky approximation

to the true lower region; the higher in the hierarchy a region is produced the more blocking effect produced.

Output produced

The process produces a tree of regions where each region represents a segment of the image at the scale of the image at the specified depth. The hierarchy can be viewed as a series of slices whereby each depth in the tree is painted to a separate image. In this visualisation approach holes can be seen in layers where recursion has prevented any data from being produced at that level. Another method to visualise the output is to paint all of the leaves of the tree to one image (scaling as appropriate). These leaves should form one non overlapping segmentation for the image.

Complexity

The runtime of the Two Phase K-Means process has been empirically shown to be linear as one would expect. As there is a finite and fixed maximum number of calls to the Two Phase K-Means possible for a set k and maximum depth d_{max}, the hierarchical version will also have linear runtime. While this is comforting, it is possible that this linear runtime will also be extremely long. To show that this is not the case consider the number of pixels processed at a specific depth in the tree. For convenience let the depth d be labelled so that the original image is at depth 0 and all layers above are numbered accordingly. Then processing will begin at $d = d_{max}$ and descend at most until $d = 0$. At the level d the fraction of pixels compared to the image at depth 0 is $\left(\frac{1}{4}\right)^d$ as each image is one quarter the size of the image below. One Two Phase K-Means call will be made at this level and process $\frac{1}{4}^d$ of the base number of pixels. This will generate at most k^2 regions which will be re-segmented on an image of size $\left(\frac{1}{4}\right)^{d-1}$, but none of these regions share any pixels so each will process on average $\frac{1}{k^2}$ of these pixels. Therefore the total number of pixels processed at each depth will simply be those in the image at that resolution. This argument can be continued recursively to show that every pixel in each of the image is processed at most once.

If relative to the base image each image is $\frac{1}{4}$ of the size, then the upper limit on the number of pixels it is possible to process can be calculated. Let n be the number of pixels in the base image and let S be the limit of the number of pixels processed as $d_{max} \rightarrow \infty$.

$$S = n + \frac{n}{4} + \frac{n}{4^2} + \frac{n}{4^3} + ...$$

$$\frac{S}{4} = \frac{n}{4} + \frac{n}{4^2} + \frac{n}{4^3} + \frac{n}{4^4} ...$$

$$S - \frac{S}{4} = n$$

$$4S - S = 3S = 4n$$
$$S = \frac{4}{3}n$$

Therefore the number of pixels processed will never exceed $\frac{4}{3}$ of those in the base image. While the Two Phase K-Means process is linear with the number of pixels there is a cost associated with initialising the process to begin segmentation upon a region. In general for a maximum depth of d the total number of calls to the segmenter that can be made is k^{2d}. This is obviously highly exponential and for suitably large d this would produce a very slow algorithm. In general there is little requirement to set the maximum depth to more than 3 or 4. In this case for a suitably small k the process will run in a timely fashion. This is also the worst case; for many segmentations the recursion termination conditions prune a great enough proportion of the process to make the hierarchical formulation of K-Means faster than the flat method.

4.3 Evaluation

4.3.1 Speed

As the runtimes of the two undelying processes are well understood a smaller range of tests were performed to show the new behaviour of the hierarchical algorithms. To compare the speeds the two processes were run upon the HIPR2[7] image set. Only one repetition was performed due to the lengthy nature of the runs. The Graph Based segmenter was run with SNN enabled at thresholds of 100 and 200. The Hierarchical Two Phase K-Means was run with spatial merging at 2000, non spatial merging at 1000, ID filtering on at radius 2 pixels with 0% threshold at $k = 6, 12$ at a maximum depth of 3. The full results can be seen in **Appendix A.6**.

As before the values for the runtimes have been scaled to millisconds / 1000 pixels the results of which can be seen in **Figure 4.2**

It can be seen that the K-Means based approach appears once again to be linear, coincidentally running under the 30 ms /1000 pixel boundary for all runs apart from $k = 12$ on image width 64. Although a slight increase in runtime appears for the 512 wide images. The graph based approach however appears to exhibit strongly non-linear runtime with the per pixel runtime rapidly increasing with the number of pixels. This is a highly undesirable property and is most likely due in part to the complexity of the RHG algorithm but also due to the non linear behaviour exhibited by the underlying region grower in previous test. From experince the authour has indeed noted a strong increase in time based upon

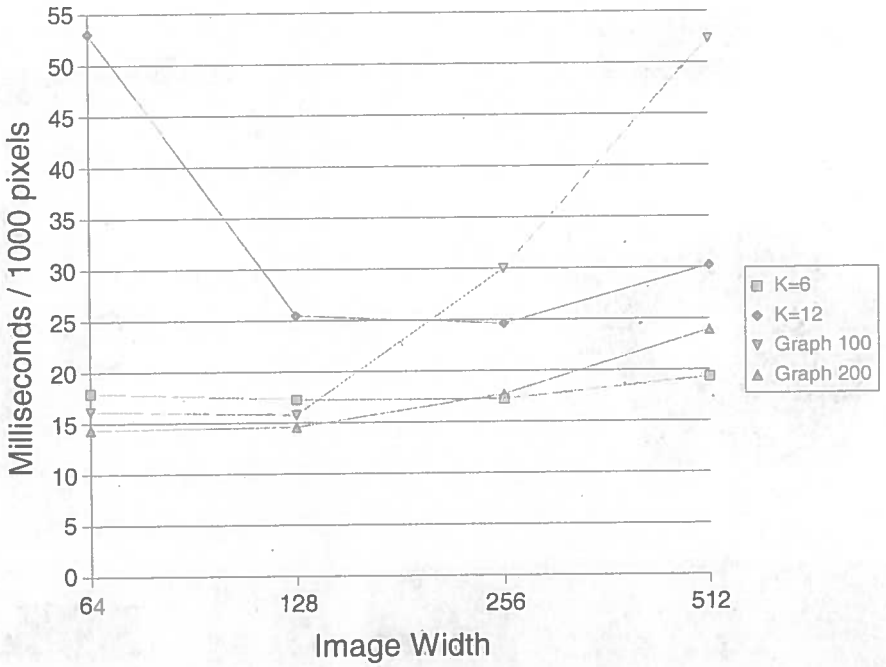


Figure 4.2: A graph showing the relative per pixel performance of the Hierarchical methods with various parameters.

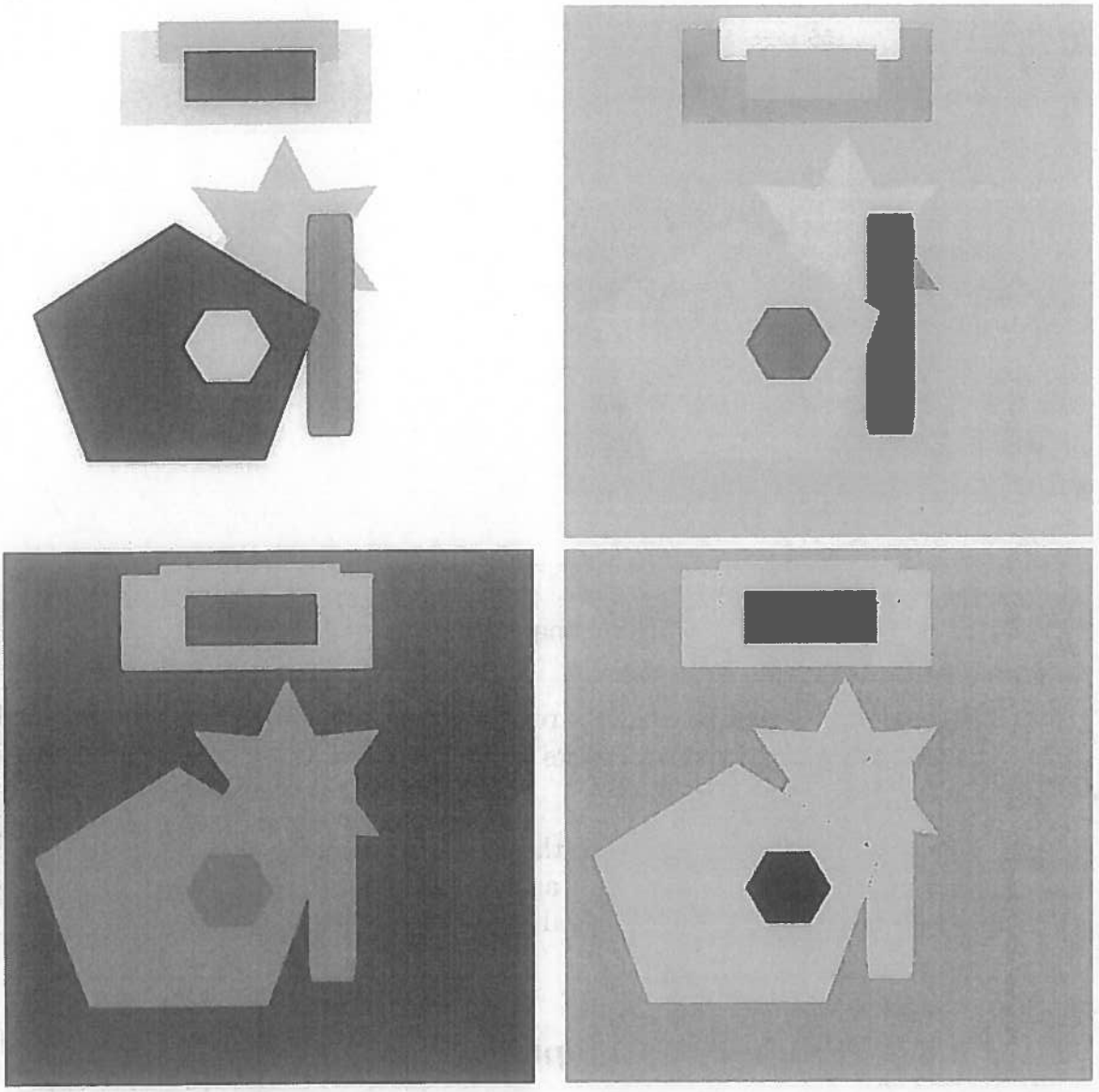
the number of nodes passed to the RHG algorithm, although this has not been empirically shown here. Once again the Region Grower based method is out performed by the K-Means based approach for larger images.

4.3.2 Subjective Comparison

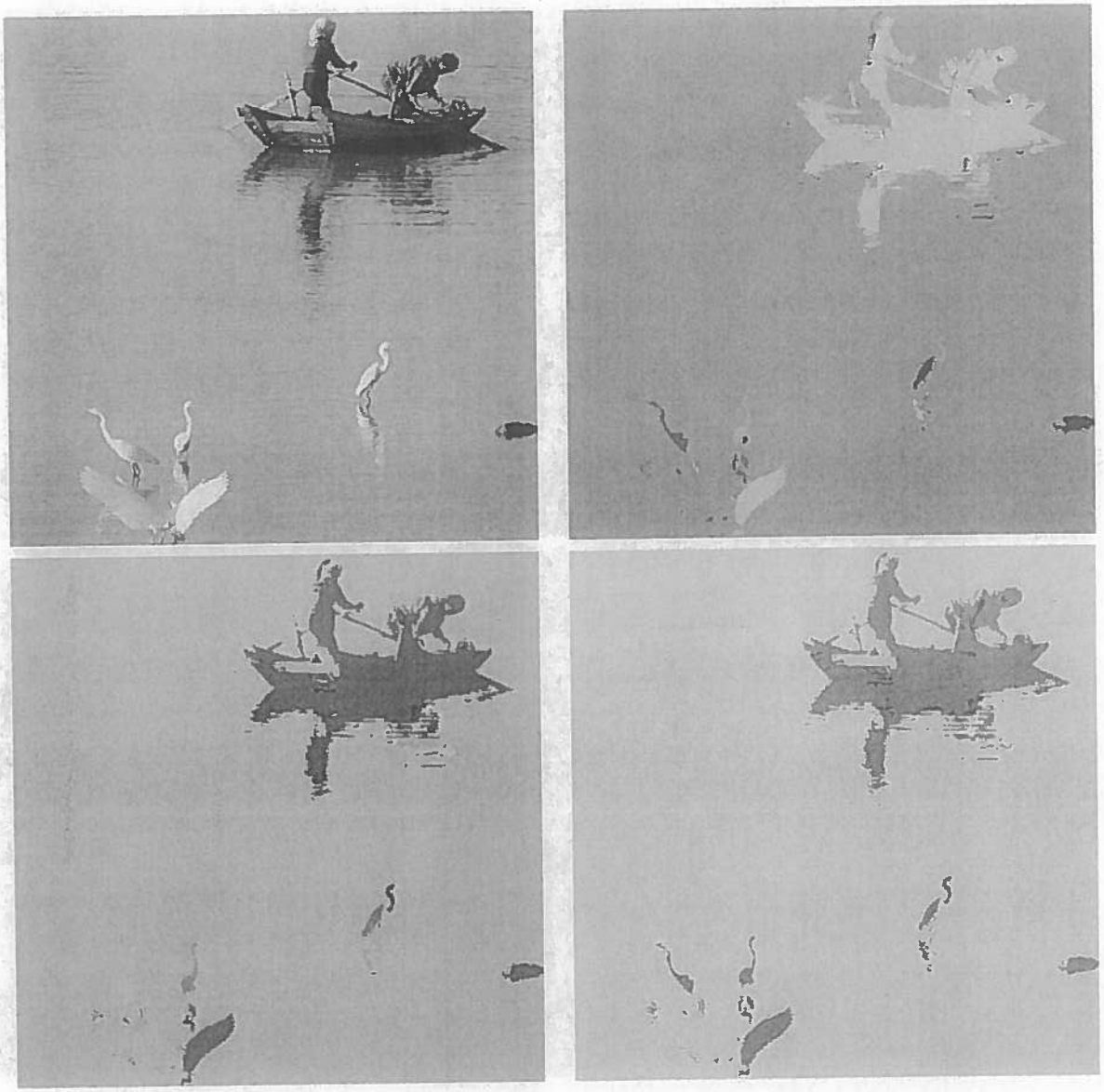
To compare the subjective performance of the Graph based and the K-Means based hierarchical methods, the two have been used to segment an artificial image in which the hierarchy is obvious and a real image where is is not so. In order to visualise the hierarchy of the graph based approach two new shading methods are used. One which highlights in false colour all segments which form region sets in the RHG and one that goes further by highlighting those at equal depths also.

To visualise the output of the K-Means based approach the sliced images at each resolution are given in tandem with the composite image obtained from painting all of the leaf segments to one image.

Graph Based Region Growing Method

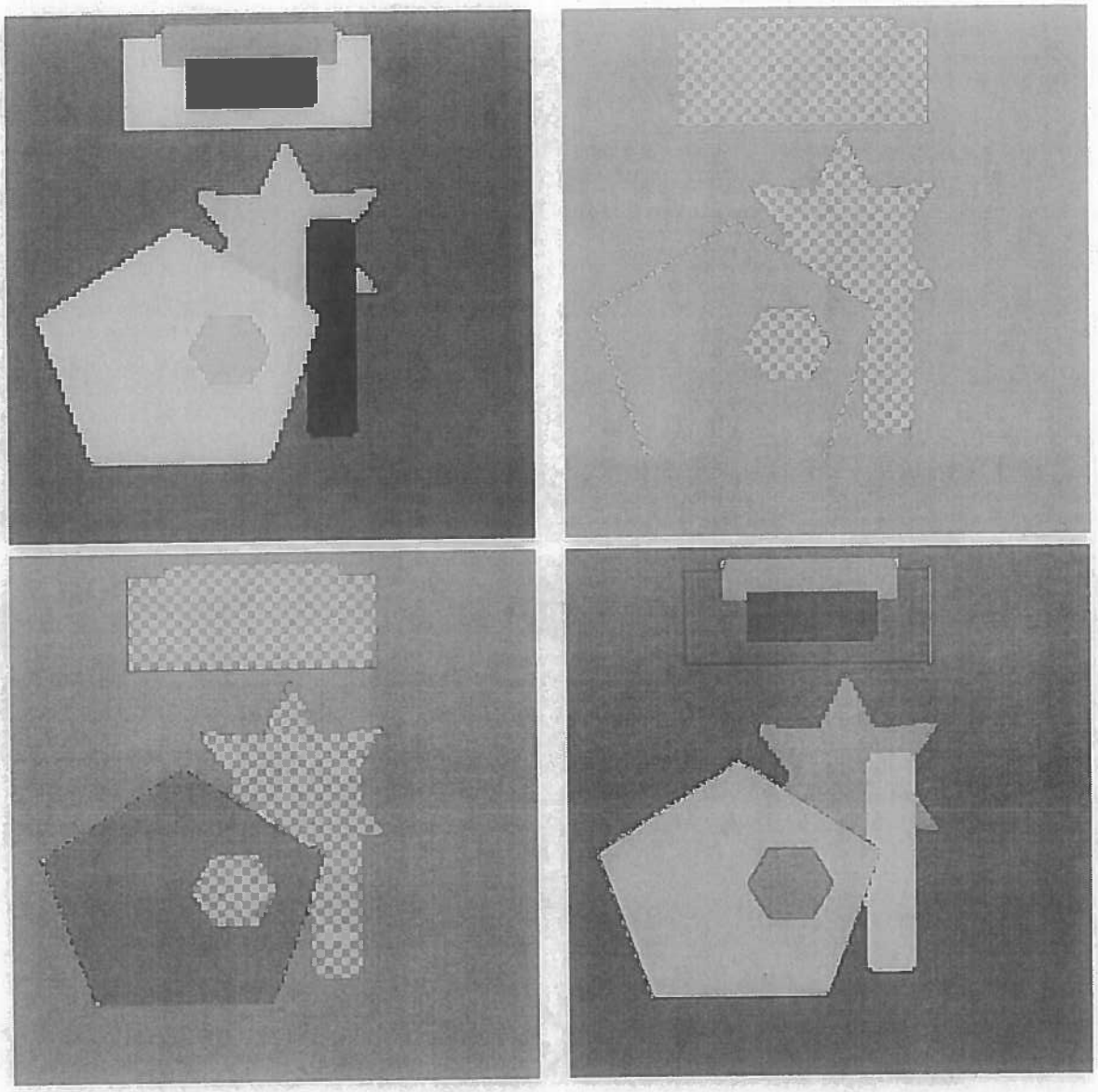


A source image is show in the top left with the false colour segmentation result of the region growing process on the top right. The bottom left image shows the regions which have been grouped together by the RHG algorithm into region groups and the bottom right colours these groups by depth. It can be seen that the method correctly groups the objects which contain the smaller shapes. This is most noticeable in the section of overlapping rectangles where the central rectangle is not solely contained by either of the surrounding regions. The depth hierarchy shows each of the regions to be grouped as one would expect, with all of the object within the background at the same depth and the children of those regions at the same depth.



The four images provided here show the same information as before. Example top left, segmentation result top right, region groups bottom left and depth graph bottom right. Here we can see that the process has had a far harder job obtaining any meaning full hierarchical information from the image. The boat and elements of the birds are correctly children of the background lake area, however, the only children of these sections are tiny details. In this a human response to assigning a hierarchy would likely involve the concept of objects within the scene which are not available to the program. For example a human observer may indicate that the head is part of the person which is in the boat. If the human observer is constrained not to use any such knowledge it is debatable whether a more concrete hierarchy could be obtained.

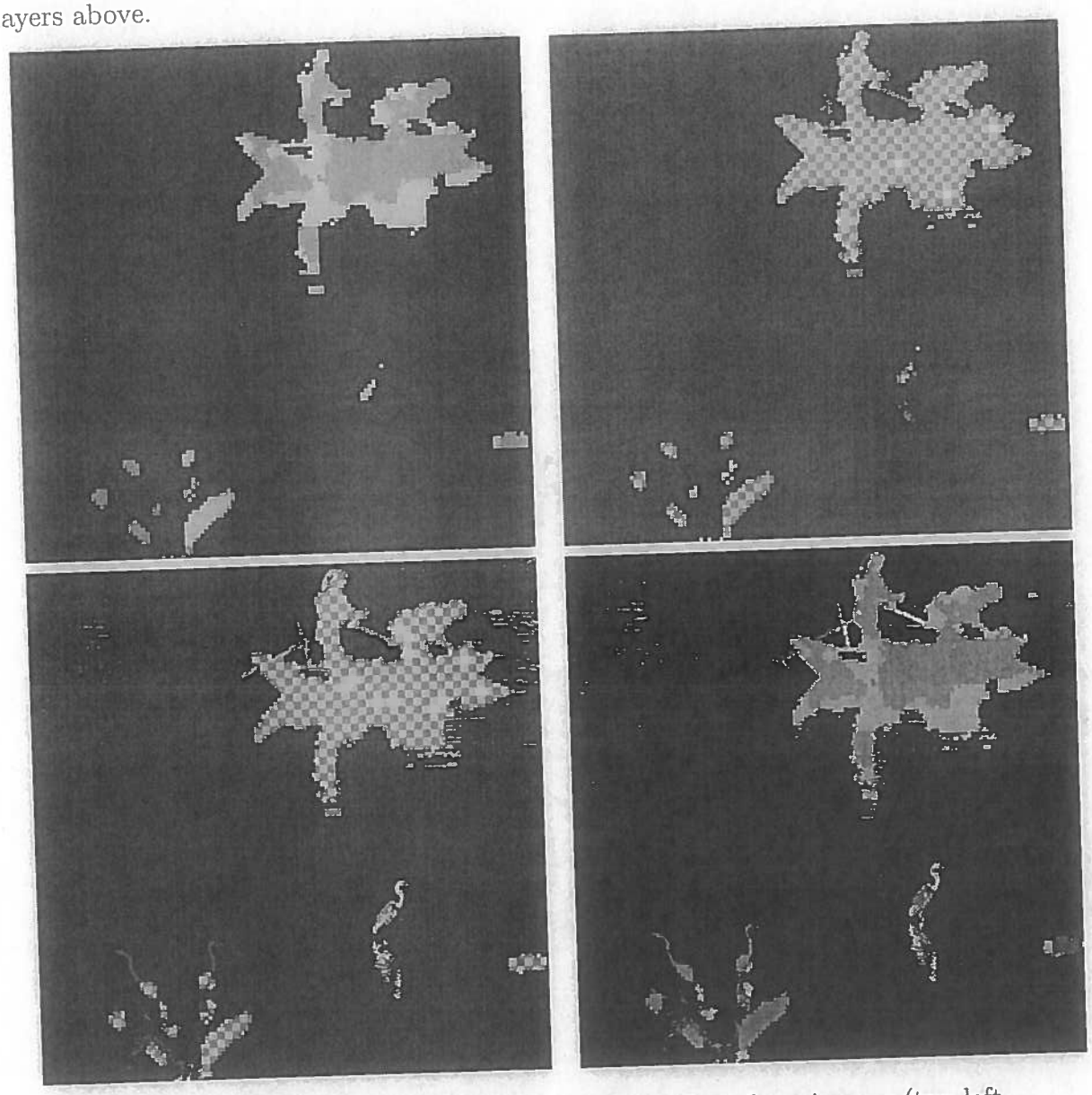
Hierarchical Two Phase K-Means



The images shown here are for the same example image as for the graph based method. The top left shows the results of the first level of segmentation upon the smallest scale image. All the images here have been scaled to the same size for clarity but the top left images source size is only 128×128. The top left image shows the second layer of the output at 256×256 where areas which contain no information have been filled with a grey chequerboard pattern to differentiate them from grey regions. These areas without information correspond to areas where recursion upon the region at the level above has been terminated. The bottom left shows the lowest layer of the image at the full 512×512. Finally the composite of the leaves of the tree structure are shown in the bottom right.

It is immediately clear that this process is not extracting the same level of information as the graph based segmenter. In fact all of the useful information from

this hierarchy is gained at the highest level of abstraction. All further levels only generate small regions to compensate for the blocky nature of the segments from layers above.



Again the layout here is the same as above with the first three images (top left, top right and bottom left) showing the process at the scales 128, 256×256 and 512×512 respectively and the composite of the leaves in the bottom right. Here the type of information that this process is able to extract can be seen. At the top levels the regions correspond to large scale features. As the resolution of the scale increase the features picked out at each level increases in detail. This distinction highlights about the fundamental difference in the operation of the graph based technique and the K-Means based technique. The graph based technique provides a hierarchy based on surrounding of regions based from the edge of the image,

whereas the K-Means based approach provides a hierarchy based on the scale of detail for different regions of the image.

5. Conclusion

The Two Phase K-Means Segmenter produced in the course of this project proved to be a highly effective flat segmentation technique, highly configurable with good runtime, producing a small number of significant regions. A number of quality improving measures were introduced which successfully increase the significance and connectedness of the output produced by the procedure. Using suitable k and a suitable subset of these quality improving measures the runtime of the algorithm can be altered to fit a range of specified time budgets. The Two Phase segmenter as it stands is not without its problems. For example the current implementation still includes inefficient code converting internally from ID arrays to vectors of segments. Further more in the nature is the Two Phase K-Means is a lack of predictable results and the production of non connected regions.

The Region Growing algorithm by contrast is deterministic in its output and generates connected regions by definition. Though the use of the SNN filter the procedure can be made more robust to noise and a significant reduction in the number of regions can be achieved. The region grower is still prone to generating large numbers of segments for any given image unless the threshold is set to a point where details are lost.

Two hierarchical extensions of these flat techniques have been developed to tackle the different issues of generating hierarchies that represent a notion of containment and hierarchies that capture different scales of detail in an image.

The Graph based approach has been demonstrated to produce reasonable hierarchies representing containment based upon the notion of a Regional Hierarchy Graph. An algorithm for converting from a standard Regional Adjacency Graphs (RAGs) to these RHGs was presented although the speed of the algorithm is poor for large numbers of nodes and remains to be proven.

The Hierarchical Two Phase K-Means process can capture scale in an image at different levels of abstraction although the hierarchy strictly represents different scales of detail these details are often associated with regions that contain then rather than the objects in the scene that they are associated with. Despite the complications of recursively applying the Two Phase Process to an iamge at many different scales, the runtime of the algorithm is comparable with the flat Two Phase K-Means due to the application of recursion termination conditions.

All of the work performed in this project was worked upon the native RGB environment for compatibility with computing and imaging devices. This does not necessarily make it the best colour space to work in for object identification. Much good work has been performed on distance measures based on the HSV or

HSI spaces. An none of the techniques here rely on a specific measure this work could be applied with relatively little effort to the techniques provided here.

Although the adjusted WSCM goes some way toward an independent comparison of the quality of two segmentations a more concrete statistical method would add rigour to notions of one segmenter out performing another.

Further optimizations can always be applied to improve the performance a procedure and there is still plenty of room for improvement in the JAVA code produced for this project. A C coding of the methods provided here might also provide a better basis for comparison with previous work.

Appendix A. General Appendix

A.1 Proof of Modulo Arithmetic Traversal

$$f(p) = 3^k p \bmod 2^{2n}$$

The moduluo arithemtic traversal function for input position p

The claim is that each p maps uniquely onto one output value for $f(p)$. If $f(p) = r$ then the relationship can between p and r can be expressed as:

$$3^k p = 2^{2n} q + r$$

For some $k, n, q \in \mathbf{N} \cup \{0\}$ with $p, r \in [0, 2^{2n})$.

Proof:

Assume that $\exists p, p' \in [0, 2^{2n})$, $p \neq p'$ st.

$$3^k p = 2^{2n} q + r$$

$$3^k p' = 2^{2n} q' + r$$

Therfore:

$$3^k p - 2^{2n} q = 3^k p' - 2^{2n} q'$$

$$3^k (p - p') = 2^{2n} (q - q')$$

Let $M = 3^k(p-p') = 2^{2n}(q-q')$ then as $(q-q'), (p-p') \in \mathbf{Z}$, M is clearly divisible by both 3^k and 2^{2n}. As 2 and 3 are both prime, then by the uniqueness of prime factorisations $M = N3^k 2^{2n}$ for some $N \in \mathbf{N} \cup \{0\}$. Therefore:

$$M = N3^k 2^{2n} = 3^k (p - p')$$

$$N2^{2n} = (p - p')$$

$$p = N2^{2n} + p'$$

As $p < 2^{2n}$ and $p' \geq 0$ then $N = 0$ which implies $p = p'$.

Contradiction: However in our assumptions it is stated that $p \neq p'$. Therefore we must conclude that there exist no such p and p' and that each p uniquely specifies a particular r.

A.2 Results of the Flat Methods Speed Tests

The full results of the Region Growing and flat Two Phase K-Means speed tests. For more deatials about the setup used in the test please refer to **Section 3.3.1**.

Threshold	SNN	Test Set	64 × 64	128 × 128	256 × 256	512 × 512
50	Off	BW	44.57	183.01	769.57	3820.30
		Col	61.95	215.75	847.81	3900.98
50	On	BW	44.25	179.58	772.59	3825.51
		Col	58.76	215.95	845.98	3904.07
100	Off	BW	40.70	180.80	794.56	4150.02
		Col	53.12	192.46	788.34	4101.36
100	On	BW	38.10	174.63	785.10	4178.53
		Col	52.46	187.61	803.68	4112.64
200	Off	BW	38.47	173.31	846.55	4150.02
		Col	58.75	212.02	924.69	5161.32
200	On	BW	36.92	169.24	844.84	4923.71
		Col	58.07	210.10	934.64	5028.14

The timings of the Averaging Region Grower run upon two sets of images with various parameters. All timing given are in *ms*.

ID Filter	NS Merge	S Merge	Test Set	64	128	256	512
Off	Off	Off	BW	51.61	194.92	713.01	2975.21
			Col	73.85	224.93	773.80	2989.03
Off	1500	Off	BW	51.61	194.10	714.92	2968.38
			Col	66.58	222.32	767.32	2994.27
Off	Off	2000	BW	61.04	245.84	891.43	3739.42
			Col	86.54	286.68	972.80	3753.51
Off	1500	2000	BW	61.10	245.69	892.49	3683.08
			Col	93.37	276.32	948.07	3742.68
2px, 0%	1500	2000	BW	61.52	255.42	913.94	3918.47
			Col	91.83	307.22	1008.00	3937.58
3px, 33%	1500	2000	BW	61.35	245.69	902.88	3730.71
			Col	91.14	270.86	953.71	3752.46

The timings of the Two Phase K-Means segmenter run upon two sets of images with $k = 6$, separated seeding and various other parameters. All timings given are in *ms*.

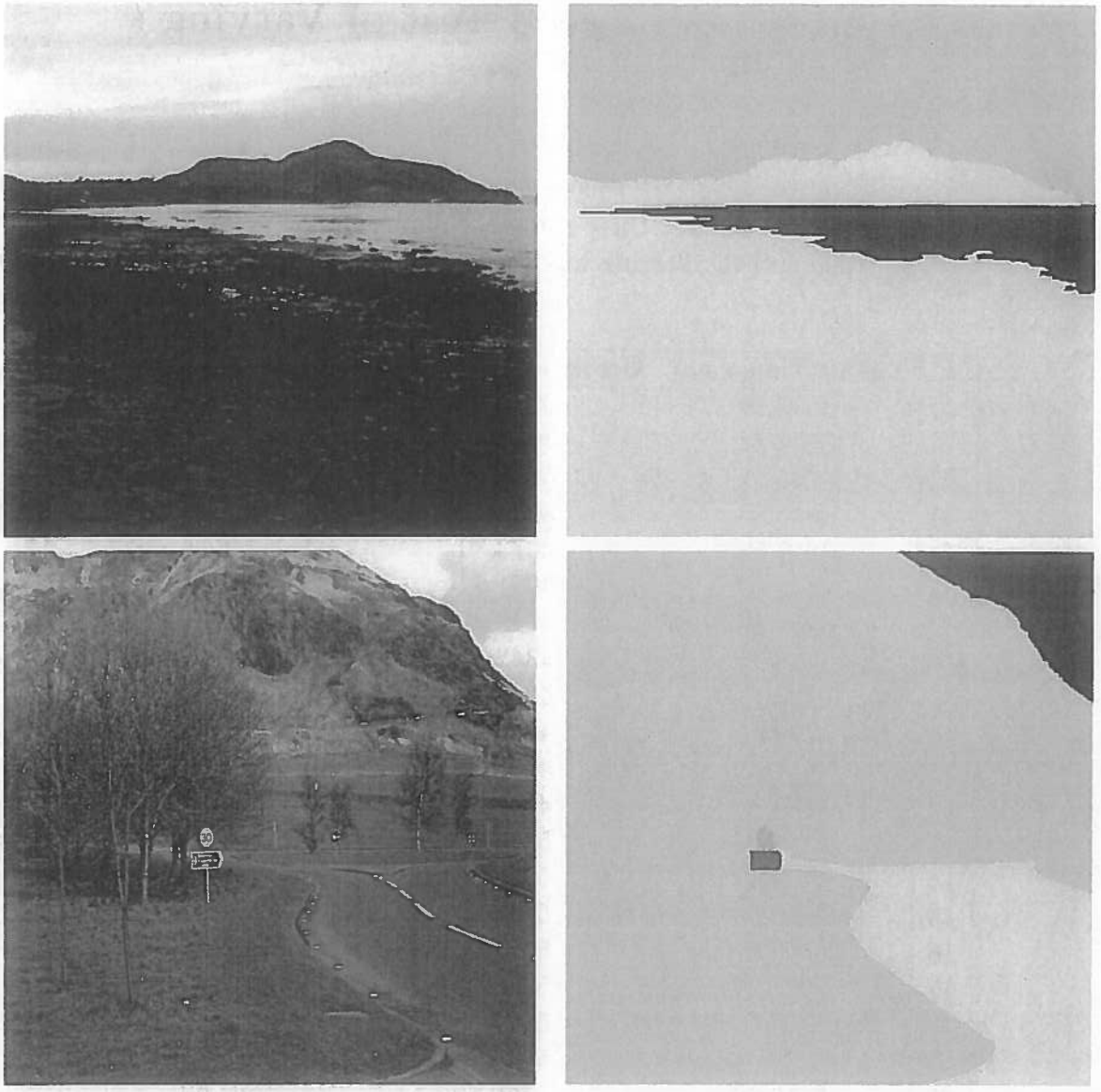
A.3 Results of Speed Test of Varying k

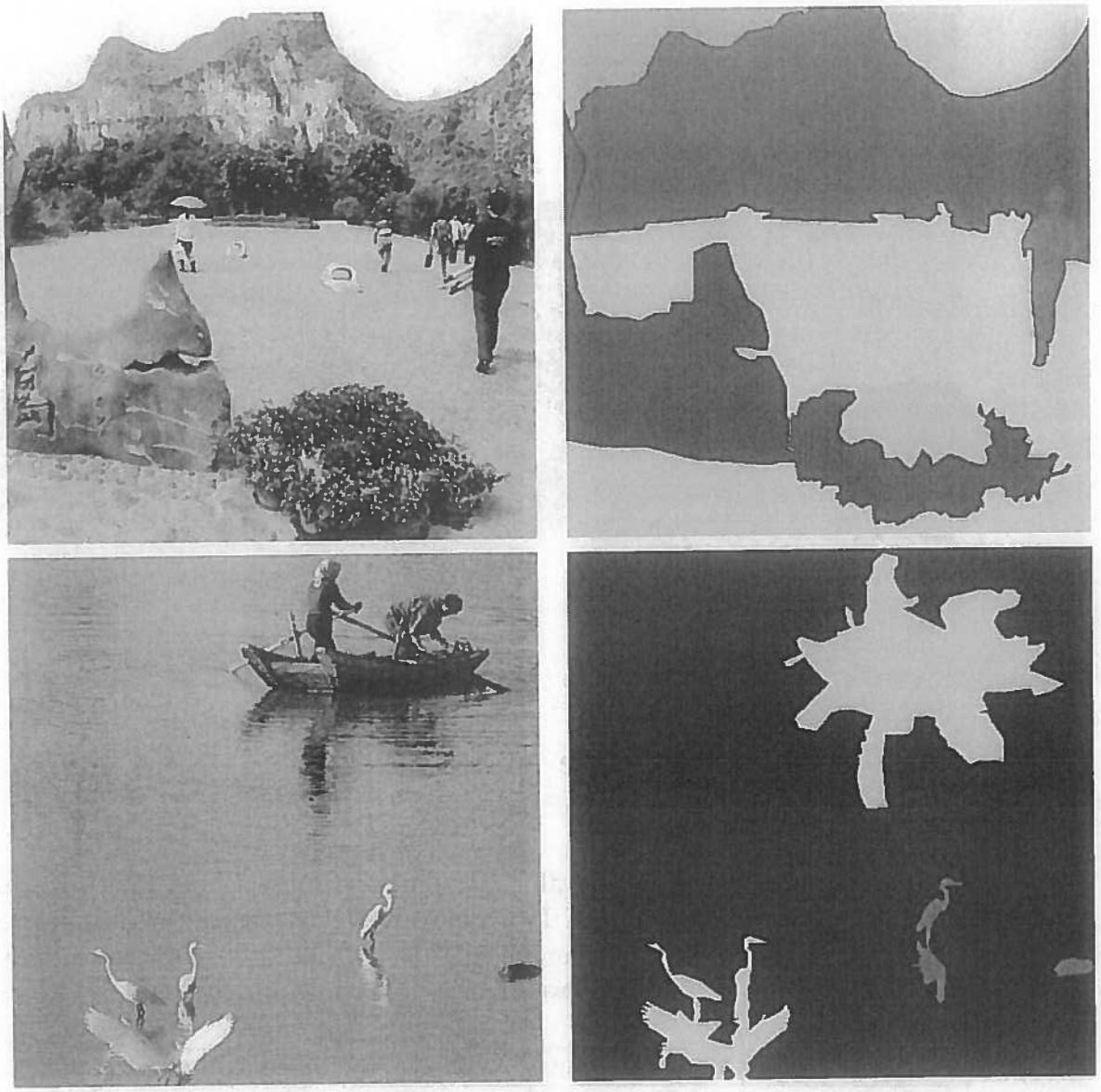
The results of running Two Phase K-Means upon the HIPR image library with k ranging from 1 to 20. For these runs spatial merging was set to 2000, non spatial merging 1000 and ID filtering was set to a radius of 2 pixels and 0% threshold.

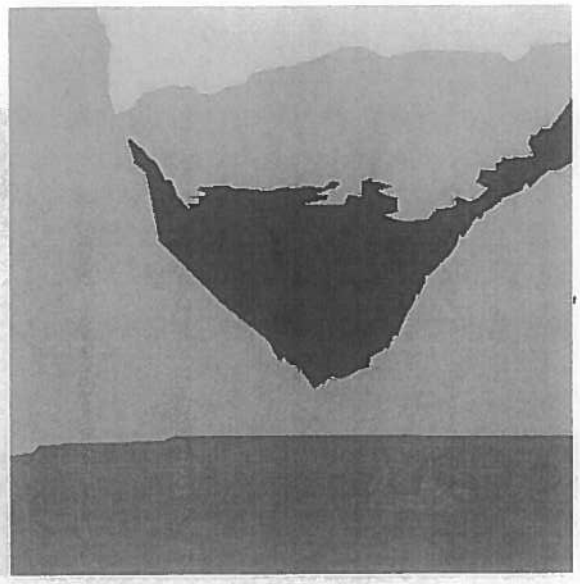
k	Colour Time (ms)	Greyscale Time (ms)	Mean Time (ms)
1	3536.16	3581.76	3556.57
2	3626.24	3632.88	3616.72
3	3745.51	3787.95	3762.68
4	3840.96	3870.59	3848.03
5	3950.58	3973.80	3952.34
6	4065.16	4071.42	4053.59
7	4192.43	4215.90	4193.41
8	4305.35	4342.90	4316.56
9	4420.38	4433.36	4412.48
10	4523.65	4583.54	4550.94
11	4631.63	4728.15	4686.29
12	4748.20	4842.53	4800.73
13	4863.17	5006.83	4952.68
14	4944.46	5129.20	5064.83
15	5061.73	5207.39	5151.96
16	5150.81	5353.08	5283.66
17	5231.84	5486.51	5404.13
18	5316.27	5627.90	5531.46
19	5429.34	5736.34	5640.56
20	5812.22	5560.22	5597.60

A.4 5 Hand Segmentations

Test Images 1 to 5 and thier defined hand segmentations.







A.5 Results of the WSCM Tests

WSCM Values The WSCM values for different runs of Two Phase K-Means and Region Growing. Note the Two Phase K-Means was run with non spatial merging of 1000 and spatial merging of 2000 with ID filtering set to radius 2 pixels and threshold 0%. Each was repeated 20 times and averaged.

Method	Image 1	Image 2	Image 3	Image 4	Image 5
KMeans $k = 6$	0.77	0.40	0.36	0.55	0.45
KMeans $k = 12$	0.72	0.48	0.53	0.60	0.48
KMeans $k = 18$	0.62	0.45	0.60	0.63	0.45
Reg 50	0.70	0.73	0.93	0.75	0.72
Reg 100	0.95	0.65	0.66	0.52	0.38
Reg 200	0.49	0.43	0.18	0.18	0.20
Reg 50 SNN	0.69	0.69	0.93	0.74	0.68
Reg 100 SNN	0.94	0.53	0.66	0.50	0.36
Reg 200	0.48	0.36	0.19	0.19	0.20

Segments Produced

Note integer values are left as, all others are given to 2dp.

Method	Image 1	Image 2	Image 3	Image 4	Image 5
KMeans $k = 6$	5.95	8.70	10.95	11.35	8.40
KMeans $k = 12$	8.40	17	26	18.45	11.15
KMeans $k = 18$	10.15	20.55	39.60	22.50	12.20
Reg 50	2146	5819	5306	2067	4377
Reg 100	629	785	1396	353	1157
Reg 200	95	63	539	118	620
Reg 50 SNN	1080	2457	3528	1099	2644
Reg 100 SNN	267	264	1008	273	693
Reg 200	34	30	426	85	303

Testing Crystal Segmenter WSCM values

For comparison the crystal segmenter was set to the same number of segments produced by each procedure and run upon each image to gain a basis for comparison. The WSCM values are give below:

Method	Image 1	Image 2	Image 3	Image 4	Image 5
KMeans $k = 6$	0.35	0.21	0.31	0.15	0.34
KMeans $k = 12$	0.47	0.32	0.48	0.21	0.34
KMeans $k = 18$	0.49	0.35	0.57	0.21	0.38
Reg 50	0.93	0.91	0.94	0.76	0.94
Reg 100	0.88	0.71	0.90	0.55	0.88
Reg 200	0.76	0.42	0.84	0.41	0.84
Reg 50 SNN	0.91	0.79	0.93	0.67	0.92
Reg 100 SNN	0.83	0.57	0.89	0.51	0.85
Reg 200	0.63	0.38	0.84	0.36	0.79

Relative WSCM values

These values are the WSCM values produced by each process divided by the WSCM produced by the crystal segmenter to produce a comparable value.

Method	Image 1	Image 2	Image 3	Image 4	Image 5
KMeans $k = 6$	2.20	1.88	1.16	3.56	1.31
KMeans $k = 12$	1.54	1.51	1.10	2.86	1.40
KMeans $k = 18$	1.27	1.30	1.05	2.98	1.19
Reg 50	0.75	0.81	0.99	0.98	0.78
Reg 100	1.08	0.92	0.73	0.95	0.43
Reg 200	0.65	1.02	0.22	0.45	0.24
Reg 50 SNN	0.76	0.88	1.00	1.10	0.74
Reg 100 SNN	1.14	0.93	0.75	0.98	0.43
Reg 200	0.77	0.95	0.23	0.54	0.25

A.6 Results of the Hierarchical Speed Tests

The results of the runs of the Hierarchical Two Phase K-Means and Graph Based Segmenter upon the HIPR2[7] image set.

Technique	Image Size	Col Time (ms)	Grey Time (ms)	Mean Time(ms)
HTPK-Means $k = 6$	64	99.8	65.61	73.45
	128	307.19	276.79	282.88
	256	1023.34	1166.94	1128.23
	512	4366.46	5305.65	5061.8
HTPK-Means $k = 12$	64	269.24	202.22	217.29
	128	449.83	409.41	417.35
	256	1607.75	1622.27	1612.31
	512	7467.05	8097.34	7915.48
Grapher 100 SNN	64	91.51	58.65	66.2
	128	291.61	249.26	258.3
	256	1418.63	2146.16	1965.16
	512	17047.03	12738.66	13708.36
Grapher 200 SNN	64	73.49	54.54	58.81
	128	260.39	233.4	238.86
	256	1228.75	1144.86	1160.14
	512	6588.54	6195.46	6263.72

Appendix B. Code Appendix

B.1 KMeans

```
package javaseg.segmenter;

import javaseg.image.*;
import javaseg.image.color.*;
import javaseg.filter.*;

import java.util.*;

public class KMeans extends Segmenter implements ReSegmenter{

  //The integer relating to the mean a pixel is allocated to
  public int[] id;
  //The colour and position values of the means
  protected double[][] mean;
  protected double[][] oldmean;
  //The number of points in this mean
  protected int[] size;
  protected int k, iterations, idFilterSize, meanSeparationDistance;
  protected boolean segmentColor = true;
  protected boolean filterIds = true;
  protected double idFilterThreshold = 0.5d;
  protected int meanSetupType = 0;
  public KMeans(ImageData image, TreeMap settings){
    super(image,settings);
  }

  public KMeans(ImageData image) {
    super(image);
  }

  public KMeans(){
    super();
  }

  public KMeans(int x, int y){
    this(new LinearImageArray(x,y));
  }

  public String getName(){
    return "Basic KMeans";
  }

  public TreeMap getDefaultSettings(){
    TreeMap set = super.getDefaultSettings();
    set.put("ITERATIONS","3");
    set.put("K","6");
    set.put("SEGMENT COLOR","true");
    set.put("ID FILTER","true");
    set.put("ID FILTER SIZE","7");
    set.put("ID FILTER THRESHOLD","0.5");
    set.put("MEANS SETUP (sep/rnd)","sep");
    set.put("MEAN SEPARATION DIST","100");
    set.put("COLOR COLLAPSE THRESHOLD;","4000");

    return set;
```

```
}

public void loadSettings(TreeMap settings){
  super.loadSettings(settings);
  iterations=getSettingInt("ITERATIONS");
  k = getSettingInt("K");
  segmentColor=getSettingBoolean("SEGMENT COLOR");
  idFilterThreshold=getSettingDouble("ID FILTER THRESHOLD");
  filterIds = getSettingBoolean("ID FILTER");
  idFilterSize = getSettingInt("ID FILTER SIZE");
  String setup = getSettingString("MEANS SETUP (sep/rnd)");
  if(setup.equals("sep"))
    meanSetupType=1;
  else if(setup.equals("rnd"))
    meanSetupType=0;
  meanSeparationDistance= getSettingInt("MEAN SEPARATION DIST");
}

public void setImageData(ImageData image){
  this.image=image;
}

protected void setUpVars(){
  mean=new double[k][5];
  oldmean=new double[k][5];
  size=new int[k];
  segments = new Vector();
  for(int i=0;i<k;i++){
    segments.add(new PlainColorSegment());
  }
  //Note id setup moved from setImageData
  setUpIds(image.getWidth()*image.getHeight());
}

private void setUpIds(int max){
  id = new int[max];
  for(int i=0;i<id.length;i++){
    id[i]=-1;
  }
}

protected void setUpMeans(){
  switch(meanSetupType){
    case 0:
      setUpRandomMeans();
      break;
    case 1:
      setUpSeparatedMeans();
      break;
  }
}

protected void setUpRandomMeans(){
  for(int i=0;i<mean.length;i++){
    mean[i][0]=(int)(Math.random()*256);
    mean[i][1]=(int)(Math.random()*256);
    mean[i][2]=(int)(Math.random()*256);
    mean[i][3]=(int)(Math.random()*image.getWidth());
    mean[i][4]=(int)(Math.random()*image.getHeight());
    for(int j=0;j<5;j++){
      oldmean[i][j]=mean[i][j];
    }
  }
}
```

```
protected void setUpSeparatedMeans(){
  int attempts = 100;
  for(int i=0;i<mean.length;i++){
    setRandomImagePointMean(i);
  }
  boolean done = false;
  while(--attempts>=0){
    done = true;
    for(int i=0;i<mean.length;i++){
      double dist = 1000;
      double meanid = -1;
      for(int j=0;j<mean.length;j++){
        if((j!=i) && getDistance(mean[i],mean[j])<meanSeparationDistance){
          setRandomImagePointMean(i);
          done=false;
        }
      }
    }
    if(done) break;
  }
}

private void setRandomImagePointMean(int m){
  int x = (int)(Math.random()*image.getWidth());
  int y = (int)(Math.random()*image.getHeight());
  int[] c = image.getColor(x,y);
  mean[m][0]=c[0];
  mean[m][1]=c[1];
  mean[m][2]=c[2];
  mean[m][3]=x;
  mean[m][4]=y;
}

protected void colorCollapse(int colorCollapseThreshold){
  for(int i=0;i<(mean.length-1);i++){
    if(size[i]!=0){
      for (int j = i + 1; j < mean.length; j++) {
        if (getDistance(mean[i], mean[j]) < colorCollapseThreshold)
          collapseMeans(i, j);
      }
    }
  }
}

protected void spatialColorCollapse(int colorCollapseThreshold){
  int width = image.getWidth();
  for(int i=id.length-1;--i>=0;){
    if( id[i]!=-1 && id[i+1]!=-1 && //Check that neither of the values are not assigned
        id[i]!=id[i+1] &&  //If the two adgacent IDs are not equal (otherwise there is no point)
        ((i+1)%width!=0) && // and we are not at the edge of a row
        (getDistance(mean[id[i]],mean[id[i+1]])<colorCollapseThreshold)){
//and the distance between them is sufficiently small
      collapseMeans(id[i],id[i+1]);
    }
    if( (i+width)<id.length &&
        id[i]!=-1 && id[i+width]!=-1 &&
        id[i]!=id[i+width] &&
        (getDistance(mean[id[i]],mean[id[i+width]])<colorCollapseThreshold)){
      collapseMeans(id[i],id[i+width]);
    }
  }
}

private void collapseMeans(int i, int j){
  for(int a=0;a<5;a++){
```

```
      mean[i][a]=(mean[i][a]+mean[j][a])/2;
    }
    size[i]+=size[j];
    size[j]=0;
    replaceID(j,i);
  }

  private void replaceID(int replace, int with){
    for(int i=id.length;--i>=0;){
      if(id[i]==replace) id[i]=with;
    }
  }

  public void resegment(Segment s){
    this.loadSettings(settings);
    setUpVars();
    setUpMeans();
    setUpIds(s.getSize());

    while(iterations>0){
      iterations--;
      iterate(s);
      updateOldMeans();
    }

    for(int i=0;i<id.length;i++){
      PlainColorSegment stemp = (PlainColorSegment) segments.elementAt(id[i]);
      int[] xy = s.getPixel(i);
      stemp.addPixel(xy[0],xy[1],image.getColor(xy[0],xy[1]));
      stemp.color=new int[]{(int)mean[id[i]][0],
(int)mean[id[i]][1],(int)mean[id[i]][2]};
    }

  }

  private void updateOldMeans(){
    double diff = 0;
    for(int i=0;i<k;i++){
      diff+=ColorComparator.euclidianNDistance(mean[i],oldmean[i]);
      for(int j=0;j<5;j++){
        oldmean[i][j]=mean[i][j];
      }
    }
    //if(diff<50){
       //System.out.println("Convergence below 50 detected, early finish.");
       //break;
    //}
  }
  protected void iterate(Segment s){
    int[] xy = new int[2];
    int[] c = new int[3];
    int m;

    for(int i=0;i<id.length;i++){
      xy = s.getPixel(i);
      c = image.getColor(xy);
      m = findClosest(c,xy);
      updateMean(m,c,xy,i);
    }

  }


  public void segment(){
```

```
    System.out.println("KMeans seg called");
    setUpVars();
    setUpMeans();
    while(iterations>0){
      iterations--;
      iterate();
      updateOldMeans();
    }

    if(filterIds)
      id = IDFilter.linearIdModeThresholdFilter(id,image,
k,idFilterSize,idFilterThreshold);
    colorCollapse(4000);

    writeIdsToSegments();
  }

  protected void iterate(){
    int[] xy = new int[2];
    int[] c = new int[3];
    int m;
    for(int i=0;i<id.length;i++){
      xy = image.getXYPosition(i);
      c = image.getColor(xy);
      m = findClosest(c,xy);
      updateMean(m,c,xy,i);
    }
  }

  //Given a particular colour and posistion value returns the integer relating
  // to the mean it most closely resembles.
  protected int findClosest(int[] c, int[] xy){
    double[] point = new double[]{c[0],c[1],c[2],xy[0],xy[1]};
    double dist = getDistance(point,mean[0]);
    int pos= 0;

    for(int i=1;i<mean.length;i++){
      double d2 = getDistance(point,mean[i]);
      if(d2<dist){
        dist=d2;
        pos=i;
      }
    }
    return pos;
  }

  protected double getDistance(double[] a, double[] b){
    if(segmentColor){
      return ColorComparator.sumOfSquares(a,b);
    }else{
      double x = a[3]-b[3];
      double y = a[4]-b[4];
      return x*x+y*y;
    }
  }

  protected void updateMean(int m, int[] c, int[] xy, int linpos){
    int[] point = new int[]{c[0],c[1],c[2],xy[0],xy[1]};
    if(size[m]==0){ //If this is the first element allocated to the mean
      for(int i=0;i<5;i++){
        mean[m][i]=point[i]; //Set the value of the mean to that of the point
      }
      size[m]=1;
    }else{ //Otherwise
      for(int i=0;i<5;i++){
```

```
        mean[m][i] = (double) (size[m] * mean[m][i] + point[i]) /
            (double) (size[m] + 1); //Update new mean
      }
      size[m]++; //Increment the new id size
    }
    int pointid=id[linpos];
    if(pointid!=-1){ //If the point was allocated to another id
      for(int i=0;i<5;i++){
        mean[pointid][i]=(double)((size[pointid]+1)*mean[pointid][i]-point[i])
/(double)size[pointid]; //Update old mean
      }
      size[pointid]--; //Decrement the old id size
    }
    id[linpos]=m;//Update the id
  }

  public void print(String s){
    if(Math.random()<0.01)
      System.out.println(s);
  }

  protected void writeSegmentsToIds(){
    setUpIds(image.getWidth()*image.getHeight());
    int width = image.getWidth();
    for(int i=segments.size();--i>=0;){
      Segment s = (Segment)segments.elementAt(i);
      for(int j=s.getSize();--j>=0;){
        int[] pix = s.getPixel(j);
        id[pix[0]+pix[1]*width]=i;
      }
    }
  }

  protected void writeSegmentsToMeans(){
    addSetting("K",""+segments.size());
    loadSettings();
    mean=new double[k][5];
    oldmean=new double[k][5];
    size=new int[k];
    for(int i=segments.size();--i>=0;){
      Segment s = (Segment)segments.elementAt(i);
      for(int j=s.getSize();--j>=0;){
        int[] pix = s.getPixel(j);
        int[] c = image.getColor(pix);
        mean[i][0]+=c[0]; mean[i][1]+=c[1]; mean[i][2]+=c[2];
        mean[i][3]+=pix[0]; mean[i][4]+=pix[1];
        size[i]++;
      }
      mean[i][0]/=size[i]; mean[i][1]/=size[i];
      mean[i][2]/=size[i]; mean[i][3]/=size[i];
      mean[i][4]/=size[i];
    }
  }

  protected void writeIdsToSegments(){
    segments = new Vector();
    System.out.println("Addin "+k+" fresh segments");
    for(int i=0;i<k;i++){
      segments.add(new PlainColorSegment());
    }
    for(int i=0;i<id.length;i++){
      if(id[i]!=-1){
        PlainColorSegment s = (PlainColorSegment) segments.elementAt(id[i]);
        s.addPixel(image.getXYPosition(i)[0], image.getXYPosition(i)[1],
image.getColor(image.getXYPosition(i)[0], image.getXYPosition(i)[1]));
```

```
        s.color = new int[] {(int) mean[id[i]][0], (int) mean[id[i]][1], (int) mean[id[i]][2]};
      }
    }
  }

  protected void scrubEmptySegments(){
    Vector out = new Vector();
    for(int i=segments.size();--i>=0;){
      Segment s = (Segment)segments.elementAt(i);
      if(s.getSize()!=0){
        out.add(s);
      }
    }
    segments = out;
  }

}
```

B.2 TwoPhaseKMeans

```
package javaseg.segmenter;

import javaseg.image.*;

import java.util.*;

public class TwoPhaseKMeans extends KMeans{
  int spacialCollapseThresh;

  public TwoPhaseKMeans() {
    super();
  }
  public TwoPhaseKMeans(ImageData i){
    super(i);
  }
  public TwoPhaseKMeans(ImageData i, TreeMap settings){
    super(i,settings);
  }
  public String getName(){
    return "Two Phase KMeans";
  }
  public TreeMap getDefaultSettings(){
    TreeMap set = super.getDefaultSettings();
    set.put("SPACIAL COLOR COLLAPSE","2000");
    return set;
  }
  public void loadSettings(TreeMap settings){
    super.loadSettings(settings);
    spacialCollapseThresh=getSettingInt("SPACIAL COLOR COLLAPSE");

  }
  public void resegment(Segment s){
    addSetting("SEGMENT COLOR", "true");
    super.resegment(s);
    secondPhase();
  }

  public void segment(){
    //System.out.println("Phase 1:");
    addSetting("SEGMENT COLOR", "true");
    super.segment();
    //System.out.println("Phase 2:");
```

```
    secondPhase();
  }

  protected void secondPhase(){
    Vector colorsegs = this.getSegments();
    Vector finalsegs = new Vector();
    Vector tempsegs = new Vector();
    addSetting("SEGMENT COLOR", "false");

    for (int i = 0; i < colorsegs.size(); i++) {
      Segment s = (Segment) colorsegs.elementAt(i);
      super.resegment(s);
      finalsegs.addAll(getSegments());
      segments = new Vector();
    }
    addSetting("SEGMENT COLOR", "true");
    segments.addAll(finalsegs);
    processSegments();
  }

  private void processSegments(){
    this.scrubEmptySegments();
    if(spacialCollapseThresh!=0){
      this.writeSegmentsToIds();
      this.writeSegmentsToMeans();
      this.spacialColorCollapse(spacialCollapseThresh);
      this.writeIdsToSegments();
      this.scrubEmptySegments();
    }
  }
}
```

B.3 HierarchicalTwoPhaseKMeans

```
package javaseg.segmenter;

import javaseg.image.*;
import javaseg.graph.*;
import javaseg.filter.*;
import javaseg.gui.*;

import java.awt.*;
import java.awt.image.*;
import java.util.*;

public class HierarchicalTwoPhaseKMeans extends TwoPhaseKMeans{
  SegmentTreeNode tree;
  ImageData[] pyramid;
  TwoPhaseKMeans[] kmeans;
  int levels;

  public HierarchicalTwoPhaseKMeans(){
  }
  public HierarchicalTwoPhaseKMeans(ImageData im, TreeMap set) {
    super(im);
    loadSettings(set);
    setImageData(im);
  }

  public String getName(){
    return "Hierarchical Two Phase KMeans";
  }
```

```
public void setImageData(ImageData in){
  super.setImageData(in);
  pyramid = Sampler.getUpsideDownSmoothedPyramid(in,levels);
  tree  = new SegmentTreeNode(getFullSegment(),0);
  kmeans = new TwoPhaseKMeans[levels];
  for(int i=0;i<levels;i++){
    kmeans[i]=new TwoPhaseKMeans(pyramid[i],settings);
  }
}

public TreeMap getDefaultSettings(){
  TreeMap set = super.getDefaultSettings();
  set.put("LEVELS","3");
  return set;
}

public void loadSettings(TreeMap settings){
   super.loadSettings(settings);
   levels=getSettingInt("LEVELS");
 }
public void segment(){
  //System.out.println("HTPKMeans running..");
  int d=100;
  processNode(tree,d);
  //System.out.println("Actually processing as completed here!");
};

public Image[] getImagePyramid(){

  Image[] imm = new Image[levels+1];
  System.out.println(tree);
  for(int i=0;i<levels+1;i++){
    imm[i]=new BufferedImage(512,512,BufferedImage.TYPE_INT_ARGB);
    segments = tree.getDescendants(i);
    SegmentPainter.paintFalseColor(imm[i].getGraphics(),segments);
  }
  return imm;
}

protected void processNode(SegmentTreeNode node, int maxdepth){

  //System.out.println("Processing seg ["+node.getSegment().getSize()+"]");
  if(node.depth==maxdepth) return;
  // Stop recursing if the predecribed depth has been reached
  if(node.depth==levels) return;
  // Or if the stop condition has been met
  if(stopCondition(node)) return;

  kmeans[node.depth].addSetting("K",""+k);
  //kmeans[node.depth].segment();
  kmeans[node.depth].resegment(node.getSegment());
  Vector segs = kmeans[node.depth].getSegments();
  //System.out.println("    Children produced: "+segs.size());
  if(node.depth==(levels-1)){
    for(int i=0;i<segs.size();i++){
      node.addChild((Segment)segs.elementAt(i));
    }
  }else{
    for (int i = 0; i < segs.size(); i++) {
      node.addChild(upSizeSegment( (Segment) segs.elementAt(i),
                                   node.depth + 1));

    }
  }
  for(int i=0;i<segs.size();i++){
```

```
      processNode(node.getChild(i), maxdepth);
    }
  }

  protected boolean stopCondition(SegmentTreeNode t){
    //Simple size check to start;
    if(t.getNumberOfChildren()==1){
      //System.out.println("Stopping due to single child class");
      return true;
    }
    int pixels = pyramid[t.depth].getWidth();
    pixels*=pixels;

    if(t.getSegment().getSize()<pixels/(k*k/5)){
      //System.out.println("Cutting due to size condition!");
      return true;
    }
    return false;
  }

  protected Segment upSizeSegment(Segment s, int depth){
    PlainColorSegment out = new PlainColorSegment();
    double[] cav = new double[3];
    for(int i=0;i<s.getSize();i++){
      int[] px = s.getPixel(i);
      px[0]*=2;
      px[1]*=2;
      int[] c = pyramid[depth].getColor(px);
      cav[0]+=c[0];cav[1]+=c[1];cav[2]+=c[2];
      out.addPixel(px[0],px[1],c);
      c = pyramid[depth].getColor(px[0]+1,px[1]);
      cav[0]+=c[0];cav[1]+=c[1];cav[2]+=c[2];
      out.addPixel(px[0]+1,px[1],c);
      c = pyramid[depth].getColor(px[0],px[1]+1);
       cav[0]+=c[0];cav[1]+=c[1];cav[2]+=c[2];
      out.addPixel(px[0],px[1]+1,c);
      c = pyramid[depth].getColor(px[0]+1,px[1]+1);
      cav[0]+=c[0];cav[1]+=c[1];cav[2]+=c[2];
      out.addPixel(px[0]+1,px[1]+1,c);
      //out.addPixel(px[0],px[1],c);
      //out.addPixel(px[0]+1,px[1],c);
      //out.addPixel(px[0],px[1]+1,c);
      //out.addPixel(px[0]+1,px[1]+1,c);
    }
    cav[0]/=s.getSize();cav[1]/=s.getSize();cav[2]/=s.getSize();
    cav[0]/=4;cav[1]/=4;cav[2]/=4;
    out.setColor((int)cav[0],(int)cav[1],(int)cav[2]);
    return out;
  }
  private Segment getFullSegment(){
    PlainColorSegment s = new PlainColorSegment();
    for(int x=0;x<pyramid[0].getWidth();x++){
      for(int y=0;y<pyramid[0].getHeight();y++){
        s.addPixel(x,y,pyramid[0].getColor(x,y));
      }
    }
    return s;
  }
}
```

B.4 RegionGrower

```
package javaseg.segmenter;

import java.util.*;
import javaseg.image.*;
import javaseg.image.color.*;
import javaseg.graph.*;
import javaseg.filter.*;

//SRegionGrower using mod 3 selection of positions in array

public class RegionGrower extends Segmenter{
  public int[][] id;
  public EdgeGraph graph = new EdgeGraph();
  int threshold;
  int count = 0;
  int pos = 0;
  boolean snn = false;

  public RegionGrower(){
    super();
  }

  public RegionGrower(ImageData i){
    super(i);
  }

  public RegionGrower(ImageData i, TreeMap settings){
    super(i, settings);
  }

  public String getName(){
    return "Region Grower: Mod 3 selection";
  }

  public TreeMap getDefaultSettings(){
    TreeMap set = super.getDefaultSettings();
    set.put("THRESHOLD","100");
    set.put("SNN FILTER","false");
    return set;
  }
  public void loadSettings(TreeMap set){
    super.loadSettings(set);
    threshold = getSettingInt("THRESHOLD");
    snn = getSettingBoolean("SNN FILTER");
  }

  public void setImageData(ImageData i){
    if(snn){
     super.setImageData(EdgePreserving.symetricNearestNeighbour(i));
    }else{
      super.setImageData(i);
    }
    segments=new Vector();
    id = new int[image.getWidth()][image.getHeight()];
    for(int x=0;x<image.getWidth();x++){
      for(int y=0;y<image.getHeight();y++){
        id[x][y]=-1;
      }
    }
  }

  public void segment(){
```

```
    segments = new Vector();
    int seg=1;
    while(count<(image.getWidth()*image.getHeight())){
      segments.add(doOneSegment(seg++));
    }
  }

  protected void incrementSeedPixel(){
    int xpos= getX();
    int ypos = getY();
    while(id[xpos][ypos]!=-1){
      //pos+=177147;
      pos+=59049;
      if (pos >= id.length*id[0].length) {
        pos%=id.length*id[0].length;
      }
      xpos= getX(); ypos = getY();
    }
  }
  private int getX(){
    return pos%image.getWidth();
  }
  private int getY(){
    return pos/image.getWidth();
  }

  protected Segment getNewSegment(int segid){
    return new BasicSegment(segid,threshold);
  }

  protected Segment doOneSegment(int segid){

    Segment seg = getNewSegment(segid);
    Vector agenda = new Vector();
    incrementSeedPixel();
    int[] pix = new int[]{getX(),getY()};
    agenda.add(pix);
    while(!agenda.isEmpty()){
      pix = (int[])agenda.remove(0);
      if(seg.addPixel(pix[0],pix[1],image.getColor(pix[0],pix[1]))){

        count++;
        id[pix[0]][pix[1]]=segid;
        agenda.add(new int[]{pix[0]-1,pix[1]});
        agenda.add(new int[]{pix[0]+1,pix[1]});
        agenda.add(new int[]{pix[0],pix[1]-1});
        agenda.add(new int[]{pix[0],pix[1]+1});
      }
    }
    return seg;
  }

  private class BasicSegment extends PlainColorSegment{
    int maxdiff;
    int segid;
    public BasicSegment(int id, int threshold){
      maxdiff=threshold;
      segid=id;
    }
    public boolean addPixel(int x, int y, int[] c){
      if( x<0 || y<0 || x>=image.getWidth() || y>=image.getHeight()){
        graph.addEdge(new int[] {0, segid});
        return false;
      }
```

```
      if(id[x][y]!=-1){
        graph.addEdge(new int[] {segid, id[x][y]});
        return false;
      }

      if(pixels.size()==0){
        color=c;
        pixels.add(new int[]{x,y});
        return true;
      }
      if(ColorComparator.euclidianDistance(color,c)>maxdiff){
       return false;
      }
      pixels.add(new int[]{x,y});
      return true;
     }

  }//End of Basic Segment class
}
```

B.5 AveragingRegionGrower

```
package javaseg.segmenter;

import javaseg.image.*;
import java.util.*;
import javaseg.image.color.*;

public class AveragingRegionGrower extends RegionGrower{
  public AveragingRegionGrower(ImageData i, TreeMap settings) {
    super(i,settings);
  }
  public AveragingRegionGrower(ImageData i){
    super(i);
  }
  public AveragingRegionGrower(){
    super();
  }
  protected Segment getNewSegment(int segid){
    return new AveragingSegment(segid,threshold);
  }

  public String getName(){
    return "Averaging Region Grower: w/Mod 3 and SNN";
  }

  private class AveragingSegment extends PlainColorSegment{
      int maxdiff;
      int segid;
      double[] col = new double[3];

      public AveragingSegment(int id, int threshold){
        maxdiff=threshold;
        segid=id;
      }
      public boolean addPixel(int x, int y, int[] c){
        //If the location is outside of the image add a RAG entry for the 0 region
        if( x<0 || y<0 || x>=image.getWidth() || y>=image.getHeight()){
          graph.addEdge(new int[] {0, segid});
          return false;
        }
```

```
        //If the pixel is allready part of another region then add a RAG entry to the
        // id of that region.
        if(id[x][y]!=-1){
          graph.addEdge(new int[] {segid, id[x][y]});
          return false;
        }

        //If this is the first pixel we have seen then set the average colour of this
        // region to that of this first pixel
        if(pixels.size()==0){
          col[0]=c[0];
          col[1]=c[1];
          col[2]=c[2];
          pixels.add(new int[]{x,y});
          return true;
        }

        //Otherwise if the pixels colour is not sufficiently similar then return false.
        color[0]=(int)col[0];
        color[1]=(int)col[1];
        color[2]=(int)col[2];
        if(ColorComparator.euclidianDistance(color,c)>maxdiff){
         return false;
       }

       //Otherwise add the pixel to the region
       pixels.add(new int[]{x,y});

       //Then update the mean colour
       col[0]*=(double)(pixels.size()-1);
       col[0]+=c[0];
       col[0]/=(double)pixels.size();
       col[1]*=(double)(pixels.size()-1);
       col[1]+=c[1];
       col[1]/=(double)pixels.size();
       col[2]*=(double)(pixels.size()-1);
       col[2]+=c[2];
       col[2]/=(double)pixels.size();

       return true;
      }

    }//End of Basic Segment class
}
```

Bibliography

[1] R. Fisher. http://www.inf.ed.ac.uk/teaching/modules/av/. *Advanced Vision Course*, 2004.

[2] R. Fisher[Ed]. http://homepages.inf.ed.ac.uk/rbf/cvonline/. *CVonline: On-Line Compendium of Computer Vision [Online]*, 2004.

[3] K. S. Fu and J. K. Mui. A survey on image segmentation. *Pattern Recognition Vol 13*, pages 3–16, 1981.

[4] M. Vetterli H. Radha, R. Leonardi and B. Naylor. Binary space partitioning (bsp) tree representation of images. *Journal of Visual Communication and Image Representation*, Sep 1991.

[5] T. Balch J. Bruce and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pittsburgh, PA 15213, 2000. School of Computer Science, Carnegie Mellon University.

[6] A. Ikonomopoulos M. Kunt and M. Kocker. Second generation image coding techniques. *Proceedings IEEE, vol. 73*, Apr 1985.

[7] A. Walker E. Wolfart R. Fisher, S. Perkins. http://homepages.inf.ed.ac.uk/rbf/hipr2/. *The Hypermedia Image Processing Reference (HIPR2)*, 2004.