SOFTWARE FOR A PARTICLE-SIZE ANALYSER BASED ON IMAGE ANALYSIS TECHNIQUES

Gunther Franz-Martin Berger

A dissertation submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for a degree of Master of Science in Engineering

DECLARATION

I declare that this dissertation is my own unaided work. It is being submitted for the Degree of *Master of Science in Engineering* in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

G.M. Bagge

(Signature of candidate)

17th day of DECEMBER 1985

ABSTRACT

This dissertation describes a technique of obtaining particle-size measurement using 2-dimensional image analysis. Images of coarse material on run-of-mine conveyor-belts may be processed using the methodology described and size-calculation of particles is possible to a high degree of accuracy. The processing involves a histogram modification technique for performing image segmentation, a rule-driven procedure for merging regions and a region-splitting method to resolve particles that are joined together. Images of varying brightness and contrast may be processed without adjustment of any threshold levels, hence lending reliability and robustness to the approach. Particles may also be resolved from very indistinct background features formed by the finer material that is usually present. The work includes the software processing necessary to perform particle recognition on a given image frame providing a foundation on which a real-time instrument may be developed.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the help and assistance rendered by the following persons and institutions :

MINTEK for providing the generous sponsorship and help in obtaining on-site data at the DEELKRAAL gold mine ;

UWTEC for making available the research facilities that were essential to the project ;

Hewlett-Packard for the loaning of the HP-1000 computer as well as the large amount of free service and advice ;

A.S.D. for the use of the DATA-SUD image capture and display unit ;

My supervisor, Professor Rodd, for procuring the necessary equipment as well as for his continued interest and support throughout ;

Professor Bruce Stewart for his role as my MINTEK supervisor ;

and last but definitely not least, my partners Dave Forsyth and Dave Smith whose objective criticism and help was invaluable throughout the duration of the project.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1.0 INTRODUCTION

1.1 PURPOSE OF THE RESEARCH

Computer vision has become a powerful modern problem solving tool, finding application in numerous areas where human vision has otherwise been essential. Recent developments in computer architectures have made it possible to perform the data and code intensive procedures associated with image processing at speeds sufficient for high-speed applications in robot vision, automatic vehicle guidance and character recognition systems, to name only a few.

This dissertation is aimed at the development of techniques in computer vision which may be used to form the basis of a scene-analysis system on which a vision-based particle-size analyser may be established. The proposed methodology covers the processing stages starting with raw pixel information through to the final object recognition and counting while providing an approach useful for applications in other areas concerning computer-vision. The work described in this dissertation is covered by a provisional patent.

1.2 THE PROBLEM

Autogenous milling has become a popular alternative to conventional milling procedures in ore grinding circuits mainly for economic reasons.

In this process the material is milled directly with the larger portion of the feed acting as the grinding medium. This method of milling is, however, more sensitive to the variation in size of the material at the feed-point.

To maintain an even discharge rate and consistent product size, it becomes essential to obtain a measurement of the size distribution of the feed. On-line measurement may be utilised to provide continuous feed-forward control of the plant to compensate for short-term variations in the feed size. It has been discovered that a "neutral" size range exists that is neither crushed by larger material nor takes part in the crushing itself. This results in the mill becoming clogged and seriously degrades throughput. Knowledge of the amount of feed within this critical range will greatly benefit efficient mill-control.

Conventional methods of obtaining size distribution involve off-line sampling by weighing and sieving systems. These methods are accurate for the specific samples that are analysed, but suffer from sampling error between samples unless unreasonably large samples are taken. The fact that the feed is disturbed and that no real-time measurement can be obtained make these methods un-attractive.

Stanley(1975) and Jerez et al(1985) may be consulted for additional background on issues relating to particle size measurement and its importance in mill control.

1.3 IMAGE ANALYSIS AS A SOLUTION

Computer image analysis provides the means to obtain on-line size data using non-intrusive instrumentation. For 2-dimensional computer vision applications a camera (video or solid state) is mounted above the conveying system and a digitised image is produced that may be interpreted using the necessary software techniques. The use of photodetectors is a

crude implementation of image processing in which 1-dimensional information is obtained by scanning the light-intensity variations down a line passing through the centre of the conveyor in the direction of motion. 3-dimensional processing is an extension of the 2-dimensional case and relies on accurate 2-dimensional analysis to build up a 3-dimensional representation of a scene (Luh et al,1985).

Size analysers have been produced that use both 1-dimensional and 2-dimensional processing techniques. Two systems are commercially available and cater for the analysis of coarse particles specifically for mineral processing applications. These systems are now briefly discussed to provide background on current technology in the field of image based particle size analysers.

The first system that will be examined is the ARMCO AUTOMETRICS MSD-95 Material Size Distribution Transmitter (Vignos et al,1979);(Gallagher,1976). This instrument uses 1-dimensional image processing and measures the distribution of chord lengths of particles ie. the distance between dark gaps that separate particles. The measurement is made of the top layer of particles on a conveyor and along a line near the centre of the moving belt. The principle employed is to use low angled lighting to cast shadows that form dark regions between particles along the line of observation. Light intensity patterns along the belt are sensed, via a lens system, by an optical detector. Light intensity observed by the detector is compared to a threshold value and values above this threshold are accepted as corresponding to particles. Chord lengths may then be determined knowing the belt speed and the time the signal remained above the threshold value. It is important to note that the chords depend on where the observation path crosses an object and hence don't necessarily represent the "best" chords useable for size analysis.

Although the simplicity of this design provides on-line processing, its limitations are severe. Since only a single line down the centre of the belt is scanned, no information is available pertaining to the distribution of particles on either side of this line unless a very uniform particle size distribution exists across the breadth of the belt. The

use of chord size as a particle size measurement is itself questionable since it relies on an unchanging shape spectrum to enable statistical deduction of the size distribution to be made. The randomness of material shape and spread on conveyor belts encountered in milling circuits certainly does not favour the use of arbitrary selected chord lengths to obtain a size measure. Even if the largest chord of any particle could be found, the statistics relating it to an actual size measure will still introduce a significant error. Further criticism against this system is that the process of thresholding the received light intensity is absolute and that variations in illumination could cause large errors to appear in the chord-length calculation and hence the size distribution.

A system utilising 2-dimensional image processing techniques was developed by CONTAC Engineering in Chile (Yachner et al,1985). This Z-80 microprocessor based instrument is aimed at compensating for the basic drawback of the ARMCO system, ie. the absence of information on particles lying outside the single scan-line. 2-dimensional image frames are processed at a rate of 10 per second. Each frame goes through a filtering procedure and is binarised to yield only black and white pixels. "Exploration lines" are used to count chord lengths across the width of the picture. Up to 10 lines may be scanned to produce a similar chord-length measure as the ARMCO instrument, but covering the width of the belt. The instrument is still in a prototype stage and needs modification to yield real-time data processing on a host computer.

The CONTAC system closely matches the performence of the ARMCO device (Yachner et al,1985) and does not seem to have any real gain in accuracy. The arbitrary chord-length measure remains a very indirect way of calculating particle size distribution while the use of multiple scan-lines could introduce significant error when large particles fall under more than one scan-line. Fine particles between scan-lines also still elude detection. Simple binarisation of pictures is also a very error-prone method in image analysis, sensitive to threshold level and the joining together of adjacent objects as may be witnessed from the work done by King(1984).(See also Allen(1981)).

The use of limited hardware technology has forced a very simplistic software solution on the designers and has limited the potential of this 2-dimensional image-based system.

The system that is described in this dissertation, although only concerned with the software aspects, overcomes the shortcomings of the methods discussed previously. Use is made of the full versatility of computer vision to provide a complete solution to the particle recognition problem. Implementation is not aimed at a particular hardware system but rather at providing a versatile methodology from which the necessary hardware specifications can be established. Full use of parallel-processing technology will then ensure an accurate on-line instrumentation system.

The processing strategy is based on capturing a 64 gray level image that covers the whole width of the belt with a high speed digital camera. Frames follow each other closely and enable the belt to be monitored continuously. The image in each frame is segmented into small regions using a histogram modification technique. Regions are then merged together in a rule-driven iterative procedure until regions that correspond to physical objects or particles in the captured scene emerge. Errors in the merging process, like for instance the joining of two objects, are corrected and the final size distribution calculated. The surface area exposed to the camera is output as a size measure. This system also enables alternative size measures to be obtained such as for example, calculation of the circumscribing rectangle of a region corresponding to the commonly used "sieve size" measurement of particle size.

The size measurement that is obtained using the methods described in this dissertation is accurate since every pixel in the image contributes information, and not only a few lines. Pixel resolution of the camera determines the system resolution and by changing the lens system various size ranges may be analysed.

The method used to distinguish between particles and background has been designed to compensate for variations in light intensity and makes the accuracy of particle recognition less dependant on fixed threshold lev-

els. Consistently good particle size calculation is achieved despite adverse picture brightness and contrast. The robustness thus incorporated into the design makes the system more reliable and flexible than the systems discussed earlier in this section.

1.4 THE SCOPE OF RESEARCH

The scope of the work presented entails the following:

1. The work concerns the development of all the necessary software to distinguish individual particles in a given 2-dimensional digital image-frame accurately and reliably.

2. The exact type of measurement eg. surface area, maximum chord-length etc. on which to base a size distribution is not of particular concern since an appropriate measure may be implemented once each rock or particle has been individually identified. For the purposes of this dissertation the projected particle surface area exposed to the camera is used as a size measure.

3. The work is aimed at providing a general solution that may be tailored to suit the requirements for a realistic hardware implementation. Here it is important to note that whilst algorithms were developed to favour execution speed, a direct trade-off between speed and accuracy evolved. The system is designed in a modular format to allow certain modules to be removed if it became necessary to upgrade execution time at the cost of overall accuracy.

4. The development facilities limited the work to images of stationary scenes and hence the effects of image-blurring due to belt motion have not been incorporated in the recognition strategy. It is assumed that a suitable frame-grabber would be used on an operational system to reduce these effects to a minimum

1.5 DESCRIPTION OF RESOURCES AND EXPERIMENTAL PROCEDURE

The hardware used to develop software described in the chapters that follow consisted of a FAIRCHILD CCD camera and a DATA-SUD frame-grabbing system. Image data was transmitted to an HP-1000 minicomputer where the necessary processing was done in FORTRAN-77. A video-monitor was linked to the DATA-SUD system to enable the various stages in the processing to be observed. Figure 1 shows the hardware configuration employed in the prototype/development system.

The SPIDER (Subroutine Package for Image Data Enhancement and Recognition) package of image processing software was available for testing a variety of segmentation algorithms described in the literature. It proved to be a stepping stone to the development of dedicated algorithms suitable for the rock-recognition problem. The inadequacies of the commercial software routines could be examined and better methods developed more rapidly than would otherwise have been possible. SPIDER was used as an experimental tool and the software comprising the particle recognition system has no relationship to routines contained in the package.

Image sizes of 128x128 and 350x350 pixels were used for development. The smaller size allowed experimentation with Fourier filtering methods and also reduced the time taken between experimental results. The larger images were used to test the generality of the methods used as they contained more rock-data and effects of picture-edges were also less pronounced. The particular size (350x350) was chosen as this was the largest image that could fit on the available flexible-disks.

Images of rock samples on rubber conveyor material were acquired on the DATA-SUD system and transmitted to the HP-1000. These laboratory images were representative of typical conveyor belt conditions to make the test processing as realistic as possible. Images were also acquired on-site at the Dealkraal gold-mine. Due to poor lighting these images were of inferior quality to those made up in the laboratory but proved to be a good test for the recognition system. All the images used for testing were

taken of stationary scenes since the frame-grabbing equipment was not designed for high acquisition rates.

Results of experimental processing are shown in Chapter 6.

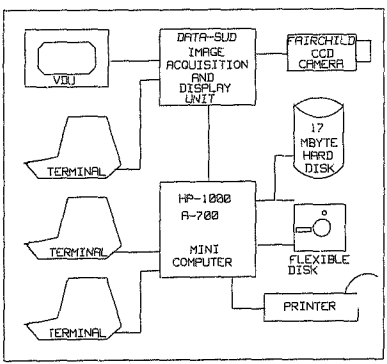


Figure 1. Experimental Hardware Configurstion

1.6 STRUCTURE OF THE DISSERTATION

Chapter 2 gives a brief introduction to concepts involved in image-processing and computer vision. Characteristics of typical conveyor-belt images are discussed with reference to their influence on system design. The inter-action between hardware and software to obtain real-time processing is examined and the criteria that determine the design approach are listed. Finally, a general description of the methodology to achieve particle recognition is given in anticipation of the chapters that follow.

The process used to achieve image segmentation is discussed in Chapter 3. Use of the histogram to achieve compensation for varying image brightness and contrast is outlined. Labelling of a segmented image is explained and examples are shown of processing discussed in the chapter.

Chapter 4 deals with the merging of regions found during image segmentation to form regions that correspond to particles in the image. The first merging operation merges regions according to their gray-level values only. The second merges regions to remove clusters of small regions contained in larger regions using neighbour-descriptions. The third and final merge procedure that is discussed merges regions according to geometrical features. Results of region merging are also presented.

The un-merging of regions using region splitting techniques is discussed in Chapter 5. It is shown how the accuracy of the recognition system may be greatly improved by detecting and correcting merge errors. Region-splitting is demonstrated using examples of processed images.

Chapter 6 shows results of processing applied to both laboratory and real "rock" pictures. It is seen that the system is capable of performing particle recognition under various conditions of lighting. The results are discussed and limitations in the method pointed out.

1.6 STRUCTURE OF THE DISSERTATION

Chapter 2 gives a brief introduction to concepts involved in image-processing and computer vision. Characteristics of typical conveyor-belt images are discussed with reference to their influence on system design. The inter-action between hardware and software to obtain real-time processing is examined and the criteria that determine the design approach are listed. Finally, a general description of the methodology to achieve particle recognition is given in anticipation of the chapters that follow.

The process used to achieve image segmentation is discussed in Chapter 3. Use of the histogram to achieve compensation for varying image brightness and contrast is outlined. Labelling of a segmented image is explained and examples are shown of processing discussed in the chapter.

Chapter 4 deals with the merging of regions found during image segmentation to form regions that correspond to particles in the image. The first merging operation merges regions according to their gray-level values only. The second merges regions to remove clusters of small regions contained in larger regions using neighbour-descriptions. The third and final merge procedure that is discussed merges regions according to geometrical features. Results of region merging are also presented.

The un-merging of regions using region splitting techniques is discussed in Chapter 5. It is shown how the accuracy of the recognition system may be greatly improved by detecting and correcting merge errors. Region-splitting is demonstrated using examples of processed images.

Chapter 6 shows results of processing applied to both laboratory and real "rock" pictures. It is seen that the system is capable of performing particle recognition under various conditions of lighting. The results are discussed and limitations in the method pointed out.

The concluding chapter, Chapter 7, summarises the work done and puts the practicalities involved in implementing the particle recognition system in perspective. Recommendations for future work are given and possible extension of the method to three dimensions is discussed.

2.0 DESIGNING A "ROCK" RECOGNITION SYSTEM

2.1 INTRODUCTION

This chapter discusses the general problem of object recognition and illustrates some of the more commonly used strategies encountered in the literature. Properties of rock images are examined to determine their impact on the approach that should be adopted in solving the particle recognition problem in particular. Attention is given to the inter-action between the hardware and software modules that would make up the size analyser, outlining the trade-off between speed and accuracy. Finally, a description of the complete particle recognition strategy is given to provide an overview of the methods that will be discussed in the more detailed chapters.

2.2 GENERAL OBJECT RECOGNITION

Given a matrix of numbers representing an image of some scene, the recognition problem may be approximated to one of finding and recognising groups of pixels in the image-plane that correspond to objects in the scene. The problem is complicated by the fact that noise may be present in the picture and that objects are normally found in orientations and positions that make interpretation difficult. Objects may occlude each other completely or partly, as is common of scenes containing material conveyed on belts. Variations in lighting of the scene may cause the same scene to be interpreted in different ways because of unwanted shadows, excessive reflection or bad contrast.

The first step in object recognition is to identify local features in an image since they are linked to physical discontinuities in the scene.

Here edges, corners or curves may be identified and spatial variations in texture and intensity analysed. By clustering together groups of pixels that have similar intensity values, an image may be segmented into regions corresponding to parts of the surfaces of objects. This latter technique provides a powerful tool for the analysis of rock scenes and forms the basis of the work in this dissertation.

Hall(1978), Rosenfeld(1982) and Duda and Hart(1978) provide background on important concepts involved in image analysis which supplement the discussion in this section. Rosenfeld(1984) presents a concise summary on methods used for building up scene descriptions in multiple dimensions and is also a useful reference.

Having detected local features, edge and region information may be combined or used separately to build up a scene description. Objects may be described in terms of lines defining their perimeters, a process involving mainly use of edge information. Gu and Huang (1985) show how a perspective scene of a polyhedron may be described by extracting a connected line drawing. On the other hand, interpretation by using region-growing algorithms is also used for object recognition (Brice and Fennema,1977). The use of both edge and region data in conjunction with an expert system to perform scene analysis is described by Nazif and Levin(1984) and provides an interesting low-level segmentation method.

Concentrating on the use of regions to build up a picture description, it is necessary to be able to describe the properties of each region in an image to enable comparisons to be made between regions. Properties may indicate the position of a region in an image, some shape description, texture, reflectance, colour etc. Neighbouring regions may then be merged according to their relative properties to form larger regions until regions result that correspond to objects in the scene. Geometrical relationships between merging regions can also be considered to ensure that the geometry of the resulting regions is reasonable in the context of the objects that are to be identified. For example, when searching for circles in an image, oval-shaped regions may be acceptable but triangular regions not.

The next step after having identified regions corresponding to objects in the scene, is to recognise individual objects on the basis of their region properties (colour, texture, shape) and the relative position of their constituent regions. When considering the side view of a car for example, two circular regions corresponding to its wheels should be found below the region corresponding to its body etc. This is a model matching task and relies on specific knowledge of the properties of the objects that need to be recognised. "Expert" systems may be applied with great effect to perform this type of explicit knowledge reasoning (Rosenfeld 1982 : vol 2).

The properties of scenes on conveyor belts are now discussed as a prelude to the description of the methods used to perform image-analysis for the particle size analyser presented later in this chapter.

2.3 PROPERTIES OF ROCK PICTURES - IMPACT ON SYSTEM DESIGN.

An overview of the properties of the rock-scenes encountered on typical gold-ore carrying conveyors at the Deelkraal gold-mine is presented to give a clearer picture of the recognition problem to be solved.

The on-site investigation yielded the following feed characteristics :

1. The rock material is damp and is covered by a coat of fine particles adhering to the rock-surfaces. The fine coating tends to smooth out rock contours and minimises the reflectance of the surfaces of large rock particles.
   The low reflectance reduces the occurrence of high-lighted facets on individual rocks that could lead to inconsistent shading and hence incorrect segmentation of the rock surfaces.

2. Since the large rocks are covered with fines, rocks and fines exhibit a similar matt-gray hue that makes distinguishing between rocks and fines, on a basis of gray-level difference, impossible.

3. There is a distinct difference between the shade of rocks and fines and the shade of the conveyor belt - hence for any given picture, a binarisation level should exist that would enable the fines and rocks to be distinguished from the background of the conveyor. It was also apparent that shadows cast by larger rocks would be helpful in resolving boundaries of objects on the belt.

4. The feed was fairly evenly distributed on the belt that was investigated. The mechanism of loading the belt however, resulted in clogging of the feed-chutes to the belt and caused infrequent but large piles of material to appear on the conveyor when the chutes were cleared. These piles of material will affect the accuracy of an image measurement system since only the top layer would be analysed. It is felt though, that the sporadic nature of this phenomenon could be compensated for either by eliminating its cause or by introducing an error correction factor in the size distribution calculation.

5. Conveyor motion helps to flatten and spread out clusters of rock. This is important since it shows that the problem of rocks concealing others is not as great as was initially anticipated. It is however important to realise that this was only one type of mill loading system and that conveyors on different plants may be more deeply loaded, hence making accurate size distribution analysis more difficult.

6. Large rocks tend to be more flattened than smaller rocks - an observation important when dealing with statistics relating the projected surface area of rocks to their volume.

7. Rocks in general, lie with their largest surface on the belt thus presenting the contours of the largest projected surface area to an observer situated directly above the belt. Since an important size

criterion in mill-control is the screen size that a particle will pass through, this particular orientation is suitable to enable an accurate measurement to be made of either the surface area or the circumscribing rectangle around the object.

8. Water is used to wash the fine material off the belt and into the mill, thereby cleaning the belt and preventing a build-up of fines that would make particle distinction difficult. The wet belt may cause reflection of the illumination from the belt into the lens system, thus saturating the optical detectors. This may however be compensated for by using polarised filters.

9. The belt speed is approximately 1 m/s on the Deelkraal site. If a 1 metre section of belt is scanned in an image frame, a processing speed of 1 second per frame is required if perfect sampling is to be obtained. Stringent demands are therefore placed on the hardware configuration necessary to perform image analysis in real-time. This also implies that careful consideration be given to the software used for implementing recognition strategies.

2.4 THE HARDWARE/SOFTWARE INTER-RELATIONSHIP

Although the hardware component associated with a working instrument falls outside the scope of this dissertation, (forming as it does a parallel project - Smith(1985)), aspects concerning the hardware configuration are briefly mentioned to provide a feel for the inter-relationship between the software and hardware requirements.

Figure 2 shows the orientation of an image system mounted over a conveyor belt and also shows the relationship between processing time, belt speed and sampling error associated with a 2-dimensional processing system.

It is seen that belt speed (v_{belt}) and the distance along the belt covered by an image scan (l_{image}) determine the time in which a frame may be processed. From Figure 2 a simple equation may be derived relating the length of the belt that is sampled ($l_{sampled}$) to system parameters :

$$l_{sampled} = (l_{image}/(l_{image}+l_{error}) \times l_{belt})$$

substituting $l_{image}+l_{error} = T_{process} \times v_{belt}$

the equation becomes :

$$l_{sampled} = (l_{image}/(T_{process} \cdot v_{belt}) \times l_{belt}) \quad \ldots \quad (1)$$

where

l_{image} is the length of the image frame in the direction of motion ;

l_{belt} is the length of the belt from the loading point to the mill feed-point ;

l_{error} is the length of the belt not processed ;

$T_{process}$ is the time taken to process a frame - $T_{process}$ is a function of l_{image} and b_{image} where

b_{image} is the width of the image and

we assume $b_{image} = b_{belt}$ where
b_{belt} is the width of the belt ;

v_{belt} is the belt speed and

$T_{process} \times v_{belt} \geq l_{image}$ for the case that frames do not overlap.

By re-arranging (1) the fraction of belt sampled ($f_{sampled}$) may be written as :

$$f_{sampled} = l_{image}/(T_{process} \cdot v_{belt}) \quad \ldots(2)$$

$$(f_{sampled} \simeq l_{sampled}/l_{belt})$$

If the accuracy of the recognition system is $A_{recognition}$ % , the overall system *accuracy* ($A_{overall}$) is :

$$A_{overall} = f_{sample} \cdot A_{recognition} \%$$

or

$$A_{overall} = (l_{image}/(T_{process} \cdot v_{belt})). \quad A_{recognition} \% \quad \ldots(3)$$

$T_{process}$ *is a function of* the image size (l_{image} x b_{image}) and the complexity of the software algorithms employed. From experience it has been found that $T_{process}$ is significantly influenced by *image size* according to the amount of serial processing involved as well as the overall complexity of the software.

Since the belt speed v_{belt} is fixed, from (3) it becomes apparent that overall system performance cannot simply be improved by increasing l_{image} unless the *dependency of* $T_{process}$ on the image dimensions can be removed. The only way this may be achieved is by extensive use of parallel processing and by reducing serial work to a minimum. If however, $A_{recognition}$ *depends heavily* on serial processing techniques, $A_{recognition}$ will be degraded if any inherently serial algorithms are removed. Hence a delicate inter-action between hardware and *software exists which* should be carefully considered if real-time measurement of particle size is to be obtained.

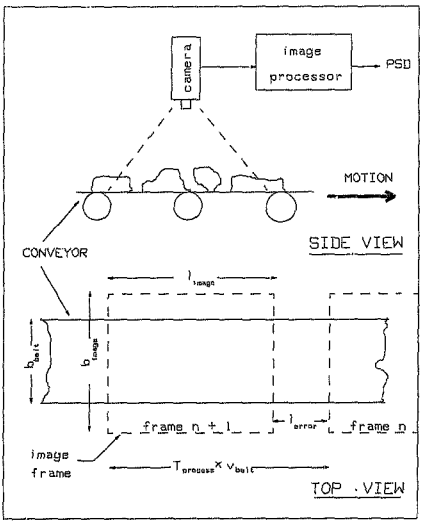


Figure 2. *Sampling of 2-dimensional images*

2.5 THE ROCK-RECOGNITION SYSTEM

Important factors influencing the design approach necessary to produce an effective recognition system have been discussed in the previous sections of this chapter. A global description of the overall rock-recognition strategy is now given in anticipation of more detailed descriptions of the processing modules in the chapters that follow.

Criteria influencing the design are listed below for reference :

1. Image interpretation should be insensitive to variations in brightness and contrast (- the illumination problem).

2. Methods should be as simple as possible to minimise the required processing time.

3. Data storage requirements should be kept as low as possible to minimise hardware cost and complexity.

4. Where possible, methods should favour the use of parallel hardware architectures.

5. System evolution should be modular in the sense that performance may be increased/decreased by adding/removing certain modules. This may be necessary to compensate for restrictions on the available hardware resources.

6. When all the modules are used, the system should be as accurate as possible.

The first step in the recognition procedure is to extract primitive regions that sub-divide the particles in the picture. Correct choice of a segmentation algorithm is vital to ensure that primitive regions are found that have a satisfactory correspondence with prominent features on the image. This is achieved by using a very simple but effective "gray-level

banding" technique where the 64 gray level histogram is reduced to one gray level corresponding to a "background band" and four other gray levels are selected to correspond to four "particle bands". Each particle band contains an equal number of pixels and the pixels belonging to the particle bands are assumed to correspond to particles in the image. By substituting pixels in the image for their band membership gray-levels, segmentation is achieved since neighbouring pixels that occupy the same band are grouped together to form pixel clusters - or regions. This substitution is very fast and provides very rapid image segmentation.

By careful consideration of how the dark band is occupied, a technique was developed that enables pictures of varying contrast to be processed. By examining the histograms of various rock-images, a method was found to establish the furthest extent of the dark band and hence the threshold between objects and background. this method is discussed in more detail in the following chapter.

After the image has been segmented into small regions, each region must be identified by an unique integer to facilitate reference to its location in the image plane and calculation of its geometrical features. This process is termed "labelling" and transforms the banded image consisting of five integer values (corresponding to the five bands) to an image consisting of integer values in the range 1 to n where n is the number of regions in the image. Single pixel regions are also eliminated to reduce the number of regions that have to be processed in the following stage. As the labelling proceeds, the area (size in pixels), band membership and the start-point (see Appendix A) of each region are calculated and entered into region description vectors. By integrating the labelling procedure with the initialisation of the description vectors, the efficiency of the pre-region merging stage is increased. The image segmentation procedure lends itself to implementation in a multi-processor environment (Smith , 1985).

Primitive regions are now merged to form larger regions that eventually correspond to particles in the scene that is being analysed. This is done by finding the neighbouring regions that surround a particular region and

comparing their characteristics to those of the region. A region and its neighbours may thus be merged according to a set of governing rules. The rules used in this merging stage concern only the band membership of the regions involved and were selected by evaluating the effect of various rules on the merge process. A routine has been developed that computes the boundary description of a region, called a chain-code, and also generates a cyclic neighbour string that uniquely describes the occurrence of neighbouring regions around the border. The chain-code provides accurate geometric descriptions of a region's border and may be used for various recognition applications where shape descriptions are useful. The merge process is an iterative procedure that terminates when no more regions can be merged. The number of iterations in which the process terminates is constant, making it possible to save time by avoiding the final iteration in which no merges occur.

The first merging stage results in a merged picture that contains enough information to make a rough particle size estimation possible. A prototype instrument using the methods described above would possibly yield a usable size measure. Its accuracy could then be improved by using the modules described below and by modifying the hardware to cope with the additional time requirement.

The secondary merging stage merges regions according to the geometry of their common borders. The nature of the rules used in the first merging stage allow certain regions to remain un-merged if they border on regions contained in the background band or "dark band". Merging may, however, be necessary between such a region and one of the neighbouring regions not in the dark band. This is achieved by finding merge occurrences where two regions merge to form a new region that has smooth boundaries across the points where the border between the two regions used to exist. Geometric merging will be clarified in Chapter 4. Extensive use is made of the chain-code routine to describe region borders and the intersection points between regions. Geometric merging cleans up regions that exist on the edges of objects and may also have a wide range of applications in areas such as the recognition of machine parts.

The first stage merging process may produce regions that correspond to separate objects that have been merged together. This occurs when the dark band does not extend far enough into the histogram, producing a so called "over-merging" effect where too few pixels have been classified as belonging to the background. Also when rocks lie close together, fines may cause the border between them to become indistinct hence joining them into a single object. To combat this phenomenon, routines have been provided that search for "bottlenecks" and split regions to form separate objects. This is an alternative to boundary erosion techniques commonly found in binary image processing which do not yield acceptable object separation. The use of this un-merging procedure provides a means of correcting errors made in the 5-band segmentation and hence makes the system more robust.

The first two merging stages, ie. gray-level and geometrical merging, are not sufficient to remove clusters of small regions that occur inside large regions. These clusters cannot be removed since they contain one or more regions in the dark band for reasons that will be discussed in Chapter 4. The removal of these regions is important to facilitate the efficient un-merging of regions. A routine has been designed to remove these clusters from the image by generating a neighbour list for each member of a cluster, combining these lists and identifying the regions belonging to the cluster as well as the common surrounding large region. The method is clarified in further detail in Chapter 4.

During all the merging and un-merging procedures discussed above, careful track is kept of the areas of the regions to enable the final size distribution to be calculated. For the purposes of this investigation the size distribution is output in the form of four discrete size ranges and is discussed further in Chapter 6.

2.6 SUMMARY

This chapter has outlined the requirements of an object recognition system against the background of general scene analysis concepts sketched in Section 2.2 . In Section 2.3 properties of rock-images were analysed in terms of their influence on a suitable design approach while the inter-action between hardware and software requirements was discussed in Section 2.4 . Finally, a general outline of the design approach was given in Section 2.5 showing important criteria that were considered in the design and giving a clear overall view of the recognition system.

The chapters which follow will discuss each important stage in the system in more detail highlighting the reasons for selecting certain methods. Program descriptions of each module in the software are presented in Appendix F .

3.0 HISTOGRAM-BASED IMAGE SEGMENTATION

3.1 INTRODUCTION

This chapter describes the first stage in the recognition process, the segmentation of an image into primitive regions. In Section 3.2 an overview of segmentation algorithms is given to provide background on methods that are available in the literature. The 5-band segmentation method used for the particle recognition system is described in Section 3.3 . Reduction from 64 to 5 gray-levels is explained and the use of the 1st gray level to compensate for variations in image brightness discussed. Section 3.4 deals with the procedure involved in labelling regions in the image with unique integers and with the initialisation of the region description vectors.

3.2 IMAGE SEGMENTATION

In order to generate a description of a scene, it is necessary to segment a picture into parts corresponding to objects in the scene. Segmentation is a process of pixel classification in which pixels are classified as belonging to certain regions or objects in the scene. Pixels may for example be classified into dark and light classes to enable light objects to be distinguished from a darker background. Another approach to segmentation is determining whether pixels belong to edges or not thus building up a description of the borders of different objects in an image. Both methods are used in image analysis applications and the choice of either depends on the particular type of pictures that are analysed. The segmenter used in this application is based on the method where pixels are divided into classes according to gray-level. Rosenfeld (1982) pro-

vides a detailed discussion of various segmentation schemes and this reference should be consulted for further background.

The particular segmentation method described in this chapter was developed after segmentation techniques were tested that are available in the SPIDER subroutine package. A method making use of phagocyte and weakness heuristics proposed by Brice and Fennema (1977) was tested and although highly rated in the literature, failed miserably in this application. Apart from producing no meaningful results, execution was very slow (1 hour for processing a 100 x 100 image !).

A split-and-merge algorithm developed by Tanimoto and Pavlidis (1975) was tested, but proved to suffer from slow execution time and sensitivity to input parameters. The method of relative similarity proposed by Yokoya et al (Spider manual,1983) was also evaluated and provided useful results from which the segmenter described in this chapter was developed. The use of this routine itself was not considered, since it demanded 5 image sized workspaces for data manipulation and was sensitive to values of input parameters. Also, no compensation for variations in picture quality could be incorporated directly into the segmentation process - this had to be done using a pre-processing stage to do the required histogram transformation. An additional pre-processing stage incorporating an edge-preserving smoothing operation on the image had to be applied to enable the segmenter to produce meaningful regions. These pre-processing routines increased the time overhead of the software and were hence not desirable.

By investigating the segmentation methods discussed above, the following criteria for obtaining a good segmentation process became apparent :

1. The use of sensitive parameters should be avoided.

2. Data requirements should be kept to a minimum.

3. Speed should be optimised.

4. The method should be robust to variations in picture quality.

5. It must be possible for subsequent merging procedures to successfully generate merged regions that coincide with real objects or features on the image.

6. If possible, the nature of the segmentation method should make image pre-processing unnecessary.

3.3 THE 5-BAND SEGMENTATION METHOD

The segmentation method developed for the particle size analyser is now discussed and it is shown how the requirements listed in section 3.2 are met.

Consider a 64 gray-level picture. From the point of view of identifying rock-particles on the picture, 64 gray levels contain an abundance of information, too much in fact than is required to do effective object extraction. By using only 2 levels, ie. binarising the image, too little information is preserved to give an accurate particle identification. It is hence argued that a certain integer number of levels might exist that will reduce the information content of the picture while still making object recognition possible.

Serra(1982) discusses the use of histogram modification techniques for image segmentation and provides interesting background to the concept of "segmentation by watersheds" as the method of reducing gray-levels is referred to in this reference. Beucher and Lantuejoul (1979) propose the use of similar techniques for contour detection while the work by Meyer(1978 a,b) concerned with the use of "artificial" gray-levels to analyse images of cervical smears is also relevant. Meyer(1978 a) shows that even by reducing the histogram to four levels an effective segmentation may be achieved.

The number of gray-levels necessary for the segmentation of "rock" images was determined by experiment and eventually 5 levels turned out to yield the required performance. The relevance of this level reduction will become clearer as the discussion progresses.

To ensure that pixel information is distributed evenly amongst the chosen levels, a cumulative distribution function is used to yield bands of equal "information" . This ensures that a certain gray-level band does not contain significantly more information than any other as this could lead to a similar loss in resolution as in the case of image binarisation. The first of the five levels is, however, not derived using the cumulative distribution function since it corresponds to the black or or non-object part of the image and is used to compensate for fluctuations in the image histogram as will be seen shortly.

The five gray level bands are derived as follows :

Band 1 contains all the black pixels in the image. Images of conveyor scenes have the characteristic of a dark background with objects being lighter shades of gray, hence the unconditional incorporation of the black pixels into the background band, band 1. Additionally, the next 25 % of the pixels in the image are grouped into the dark band starting with the second darkest pixels. The value of 25 % was found through experimentation and produced consistent image segmentation under varying brightness and contrast. In the case where the first gray level occupies more than 75 % of the image pixels, only the first two gray levels are incorporated into the dark band. This situation arises when only a few objects are present on the belt or when image brightness is very low. Under these conditions it has been found that the first two gray levels correspond acceptably well with picture background. This also implies that if all the pixels in the image occur in the first two gray-levels of the histogram (ie. the dark side), no objects are assumed present.

A facility is also introduced to limit the furthest extent of the dark band into the histogram. The extent of the dark band may be limited by

considering the amount of pixels occurring in the first gray level as this is an indication of picture brightness. For bright pictures, the first gray level will have a smaller occupation, allowing the dark band to extend further into the histogram. For darker pictures, the first gray level will contain more pixels and the dark band could be limited to incorporating say, only 32 out of the 64 gray levels. Limiting the dark band extent was not necessary for pictures that were processed during system testing and it was included to maintain the generality of the approach.

Processing concerned with finding the extent of the dark band is akin to the technique of thresholding an image by examining the histogram structure. For further background information on this aspect of image processing consult Weszka(1978) and Hummel(1979).

Having determined the gray levels corresponding to the dark band, the pixels in the image having the remainder of the gray levels are divided into 4 equal groups using a cumulative distribution function. These groups are the 4 bands that correspond to particles or objects in the image. From now on, these 4 bands will be referred to as the "rock bands". Figure 3 shows the procedure of obtaining the 5 bands more clearly.

To illustrate the manner in which the 5 bands may be used to segment an image, an example is shown :

Assume the bands are occupied as follows :

Band 1 : level 0 to 10 denoted by the integer -1
Band 2 : level 11 to 30 denoted by the integer -2
Band 3 : level 31 to 40 denoted by the integer -3
Band 4 : level 41 to 50 denoted by the integer -4
Band 5 : level 51 to 63 denoted by the integer -5

Note : A gray-level histogram ranging from 0 (black) to 63 (white) is assumed.

Consider the input image :

0	0	30	31	60
5	35	34	36	61
6	40	55	48	60
11	42	45	47	30
13	20	15	16	21

By substituting for the band membership, the resultant segmented image is :

-1	-1	-3	-3	-5
-1	-3	-3	-3	-5
-1	-3	-5	-4	-5
-2	-4	-4	-4	-2
-2	-2	-2	-2	-2

Note : Negative integers are used for labelling since this enables labelling to be done in the *same data space as* the segmented image. Section 3.4 explains the labelling procedure in further detail.

The pixels have thus been clustered together to form regions by the 5-band substitution process. The substitution process is very rapid and hence helps to increase the speed of the overall segmentation procedure. No image pre-processing is required since the banding technique is in itself a type of smoothing operation. An additional requirement amongst those listed in Section 3.2 has thus been met. Figure 4 shows how an actual

conveyor-belt image has been segmented using the 5-band method. It is seen that the input picture suffers from lack of contrast and also has a low brightness. Only 12 out of the possible 64 gray levels in the histogram are occupied. The processed result shows how the 5-band method has enhanced the contrast and has managed to highlight the pertinent features in the image despite the poor brightness level.

At this stage, regions exist as clusters of pixels having the same integer value. It is now necessary to identify each region uniquely by assigning an integer value to all the pixels that form the region. This process is called " region labelling " and is discussed in the next section.

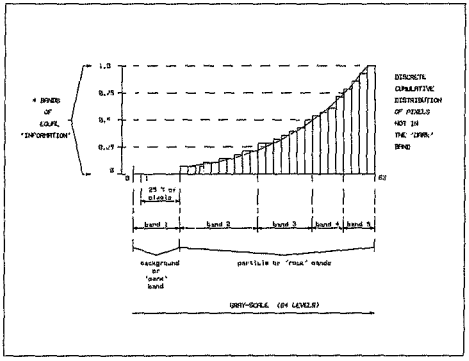


Figure 3. Derivation of the 5 bands used in image segmentation

(a)

(b)

Figure 4. Result of 5-band segmentation: (a) is the input image ; (b) is the processed result . The input image is a scene of particles on a conveyor obtained on-site at the Deelkraal mine

3.4 REGION LABELLING

Labelling of a segmented image is achieved by scanning the image in a raster scan and labelling consecutive occurrences of pixels with the same band membership with a unique integer label. Labelling may be done using either 4 or 8 connected neighbours. 8-connected labelling implies that any given pixel has 8 possible neighbouring pixels to which it may link and hence larger regions are in general produced than with the use of 4-connectedness. 8-connected regions, however, are more stringy in nature and produce regions that are entwined thus yielding rather obscure border geometries. 8-connectedness produces larger and hence fewer regions than 4-connectedness therefore favouring processing speed. The simpler region geometry produced by 4-connected labelling is more advantageous though and was finally chosen as the labelling method. The advantage in 4-connected labelling lies in the fact that long intertwined regions are not formed which in turn results in a more efficient merging process (regions have fewer neighbours) and reduces the occurrences where un-merging of regions is required.

The labelling process overwrites the values of the segmented picture as it proceeds in raster-scan format, hence utilising the same data space and saving memory resources. This is made possible by representing the input segmented image using negative integers and labelling with positive integers. It is therefore also possible to distinguish between pixels that have been labelled and those still having their band membership values which is useful when re-labelling of redundant region segments has to be done. The need for region re-labelling will be discussed shortly. Regions that are encountered consisting of only single pixels are merged into adjacent regions that have the most similar band membership value. This is done to reduce the number of irrelevantly small regions to save processing time in subsequent merging stages.

During labelling, region characteristics are filed in vectors according to region labels. When a new region is found, the x-y co-ordinates where the region is encountered are entered into a vector. These co-ordinates are known as the start-points of a region and are used for referencing the position of the region in the image plane. The 5-band membership of the region is entered into another vector where it will be used for the merging decisions based on band-membership criteria. Regions in band 1 are considered as background regions and regions in the upper 4 bands as particle regions. The area of each region is accumulated in a third vector as each pixel is added to its particular region during labelling. At the end of the labelling process the area of each region is then accurately known.

It frequently happens that a region is initially labelled with more than one label in the upper part of the image and that the parts of the region that have been labelled differently join to form a single region lower down. As two differently labelled parts join, a choice has to be made as to which label should remain. The procedure adopted in this instance is to keep the label that has the smaller integer value and to place the redundant label in a queue for re-use. A back-tracking operation is done to replace all the occurrences of the redundant label in the image with the label chosen to represent the region. Whenever a new region is labelled, the queue containing redundant labels is checked to see if any labels are present for re-use before a new label is utilised. The number of integers used for labelling is thus kept to a minimum and data resources are saved. The use of a similar technique for implementing labelling of binary images in a hardware application is explained by Dinstein et al(1985).

An example of 4-connected labelling is shown below. The links that establish ties to to previous labels are shown to clarify the finer details of the relabelling mechanism.

Input segmented image :

-5 -1 -3 -1 -1

-5 -1 -3 -1 -2

-4 -1 -3 -1 -2

-4 -1 -1 -1 -2

After row 1 is processed we have :

1 2 3 4 → 4

-5 -1 -3 -1 -2

-4 -1 -3 -1 -2

-4 -1 -1 -1 -2

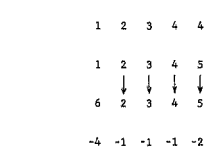
After row 2 :

1 2 3 4 4
↓ ↓ ↓ ↓
1 2 3 4 5

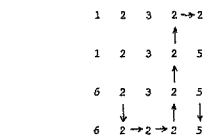
-4 -1 -3 -1 -2

-4 -1 -1 -1 -2

After row 3 :



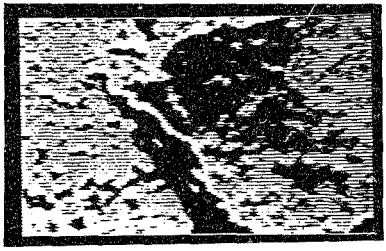
After row 4 :



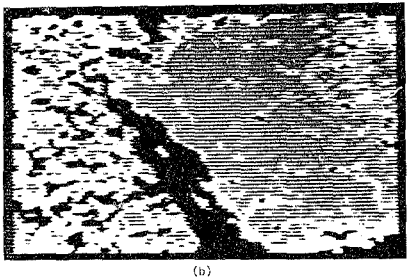
Note how label 4 is replaced by label 2 to correct for ambiguous labelling.

A more detailed description of the region labelling procedure is given in Appendix F in a high level program description language (PDL).

Figure 5 shows what an initially segmented image looks like after labelling has been applied and the border around each region has been highlighted using white pixels. The output image generated after this labelling stage may now be processed by region merging routines to find regions that correspond to the particles that need to be identified.



(a)



(b)

Figure 5. Segmentation and labelling applied to an image: (a) is a segmented image and (b) shows the labelled image with regions outlined in white.The images may appear very similar and this would be due to loss of detail during reproduction.

3.5 SUMMARY

This chapter has dealt with procedures relating to the segmentation of images into primitive regions and the subsequent labelling of these regions. The 5-band *histogram* based segmentation was described and it was explained how the first band could be used to compensate for variations in picture quality. Results of 5-band level reduction were presented to show the improvement in picture contrast and brightness that can be achieved. Region-labelling was discussed with reference to the way in which data-storage requirements are *minimised and region description* vectors are constructed. The next chapter explains the merging of regions to form larger regions that correspond to objects in the scene.

4.0 REGION MERGING

4.1 INTRODUCTION

In the preceding chapter the methods used to achieve image segmentation and labelling were explained. It is now shown how the resulting primitive regions are merged to form representations of objects in the scene. Section 4.2 briefly discusses the format of the region-border descriptions which are essential to the merging procedures discussed later in the chapter. The first merging operation, using only gray level criteria, is explained in Section 4.3 and it is seen that the most significant region merging occurs in this processing stage. Section 4.4 deals with the procedure of merging to remove small region clusters from the image while the final merging stage, based on geometrical criteria, is described in Section 4.5 . Examples of processed images accompany the descriptions in each section.

The procedures that will be discussed are often fairly involved, requiring the incorporation of some of the more detailed aspects into the appendices.

4.2 BORDER DESCRIPTION : CHAIN-CODES AND NEIGHBOURS

When a region is considered for merging, it is necessary to know which other regions in the image surround that particular region. This enables the characteristics of the regions concerned to be compared so that a merge decision can be taken. Such a description of the occurrence of neighbours on a region's border will also be referred to as a "neighbour code" or a "neighbour list". Finding neighbours or generating neighbour

lists is only possible if the border of the region in question is known, hence requiring some border-following procedure to be applied.

Border descriptions of a region may be in the form of a list of co-ordinates that define the border pixels in the image plane, or the border may be defined using a chain-code. A chain-code requires only a single string of numbers where each entry in the string indicates the direction of the next pixel on the border relative to the pixel that was last found. Given the absolute location of the first pixel of the region (or the start-point), the border may be followed exactly using the chain-code information. Chain-codes were chosen for this application since less data space is required for their storage and the geometric merge procedure (discussed in Section 4.5) is made simpler. Appendix B contains a detailed discussion of how the chain-codes are generated using explicit knowledge to make the border searching most efficient. Rosenfeld (1982) provides a thorough discussion of the concepts associated with chain-codes while Tai and Yu(1985) demonstrate the versatility of similar border description strings for performing shape recognition.

Neighbour codes are generated using the method described in Appendix B. Cyclic neighbour codes of a region describe the occurrence of neighbours around the region's border in exact sequence. The number of pixels associated with each neighbour occurrence (including multiple occurrences of the same neighbour) on the border is found, thereby indicating the dominance of a neighbour on the border. Neighbouring regions are also classified as being 4 or 8-connected to the border. Weakly connected neighbours (diagonal or 8-connected neighbours) may then be ignored during merging. The position in the chain-code where each neighbour is found for the first time is also recorded. Finding a certain neighbour on the border is thus faster and this is used in the geometric merge routine. Details of the features that have just been discussed are available in Appendix B together with examples showing the format of the neighbour-description vectors as well as conventions used in chain-code following.

4.3 GRAY-LEVEL MERGING

The 5-band segmentation method results in an image that consists of a large number of regions occupying the upper 4 particle bands (refer to Figure 5). Each band may be considered as being one gray-level in a 5 gray-level image. Since each region has been classified as belonging to one of the five bands (or gray-levels) during the labelling process, it is possible to merge regions using their gray-level values.

Various rules may be established according to which a region may merge to its neighbouring regions. By experimentation it was found that the set of rules for gray-level merging is fairly restricted because of the danger of over-merging resulting in too many regions being joined together which should have remained separate. Over-merging of an image could eventually approximate to the case where straight binarisation of the image was performed, making the initial effort in obtaining a segmented image superfluous. It is again emphasised that the reason for using 5-band segmentation and adopting a merge driven region-growing strategy is to overcome the limitations inherent in conventional binary image-processing techniques.

Only two usable rules emerged that allowed a very good first stage merging based on gray-level features to be achieved. These rules are :

1. A region surrounded only by regions that occupy the particle bands merges to all its neighbours if it also has a particle-band gray-level.

   Here it is attempted to resolve a rock region surrounded by other rock regions into a new rock region because it is assumed that the region and its neighbours all correspond to the same object in the image.

   In developing this rule, it was attempted to allow merging between a rock-region and its rock-neighbours if background neighbours occurred

which together did not occupy more than a certain percentage of the total number of pixels on the border. By allowing this relaxation of the rule, significant over-merging occurred showing how restricted the gray-level merge rules would have to be to allow proper region growth.

2. A region occupying the background band which is surrounded by only 1 neighbour merges to this neighbour if the neighbour occupies one of the particle bands.

   This corresponds to a rock-region with a dark shadow or hole in it. It is assumed that particles don't have holes in them and hence merging is performed to remove the background region. Attempting to allow merging if a background region is surrounded by more than one region in the particle bands results in serious merging errors where for example, a large part of the picture background may be changed into a particle region.

The merging process using the two rules listed above is iterative and for a 5-level image terminates within two iterations. During each iteration, each region is considered for merging and valid merges are recorded by assigning the new label of any merged region to a record in a merge-recording vector which is referenced by the old label of the region. After all regions have been considered, merging physically occurs on the image by substituting old region labels for the corresponding entries in the merge-recording vector. The new regions are then given an opportunity to merge during the next iteration. Subtleties regarding the merge-process are clarified in Appendix C where the use of additional vectors necessary for efficient merge-recording is also explained.

An example of a typical merging procedure is now given to illustrate gray-level merging more clearly.

Consider the segmented image in Figure 6. By inspection we see that only regions 1 and 3 have boundary neighbours that agree with a merge-rule, in this case merge-rule 1. Region 1 has a neighbour-code 5 2 3 4 and all

these neighbours have gray-levels in the range 2 to 5 indicating particle band membership. Region 3 has neighbours 1 2 8 7 4 also occupying the higher gray-levels. All the other regions in the image border on the background region, region 6, and hence can't merge. We thus merge region 1 to all its neighbours and region 3 to all its neighbours according to rule 1. After merging we have the labelled image as shown in the figure. The reason for using label 2 to identify the resulting large region is that the start-point of the large region corresponds to that of the region originally having the label "2". This aspect will be dealt with in more detail in Appendix C. The new gray-value of region 2 is chosen to be any value corresponding to a particle band to enable its identification as a rock region.

Figure 7 shows the application of gray-level merging to a segmented picture. It can be seen that significant region-growing has occurred during merging and that the majority of small regions have been merged into larger regions. When a comparison is made between the merged image and the original scene, it is seen that merged regions agree well with features in the scene. It is at this stage possible to calculate a size distribution using the regions present in the image. Error will be introduced due to over-merging and incorrect segmentation, but the results should still be suitable for a reduced complexity system or for implementing a prototype size analyser. To make such a reduced system viable, the segmentation method would have to create more background (ie. change the 25% pixel occupancy to say 30%) to reduce the occurrences of over merging which could of course also remove useful information from the image. Methods of coping with over-merging of regions will be discussed in Chapter 5 and by applying these correction strategies a more accurate size calculation can be obtained.

A high-level PDL description of the gray-level merging process is given in Appendix F.

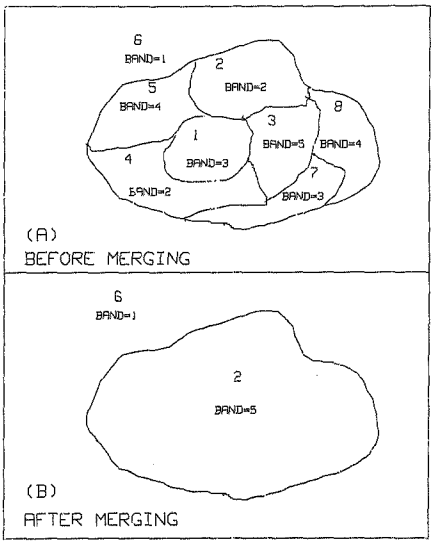
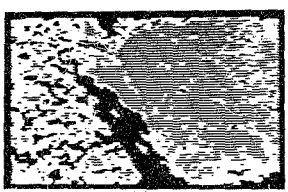


Figure 6. Gray-level merging: (a) is a segmented image before merging is applied. (b) is the merged result.

(a)



(b)

(c)

Figure 7. Results of gray-level merging: (a) shows the input image of a conveyor scene. (b) is the result of segmentation and (c) is the gray-level merged picture.

4.4 MERGING TO REMOVE SMALL CLUSTERS

The gray-level merge rules discussed in Section 4.3 allow regions in the background to be removed from inside regions that belong the particle bands, thereby removing "holes" that occur inside rock regions. It often happens that these holes don't occur on their own but also have small regions attached to them belonging to the particle bands. Since these small particle regions border on the "hole" region belonging to the background band, no merging is allowed. The "hole" region itself borders on more than 1 rock region and hence cannot merge. An alternative process is thus required to remove these regions from larger rock regions. The reason that it is important to remove these regions is that the process of correcting merge errors (described in Chapter 5) requires that regions exist as continuous pixel domains without other regions being included within their perimeters.

Merging is achieved by identifying groups of regions, all within a certain size-limit, that have a common large neighbour that has an area above a second size-limit and also occupies one of the particle bands. The second size-limit is larger than the first to prevent a small enclosed region to be identified as the common neighbour. The process of finding such clusters is described best by considering an example :

Figure 8 shows a large region, region 5, containing a cluster consisting of regions 1, 2, 3 and 4. We assume all the regions in the cluster fall below the size threshold for cluster regions. Region 5 is larger than the size-limit for the surrounding region and also occupies one of the particle bands.

To find the cluster, we start with the neighbour-code of a region that has a size less than the threshold for regions belonging to a cluster, say for example, region 1. It's neighbour code is then 5 3 2.

We identify region 5 as a large region and note that it may be a valid surrounding region. The next step is to find the neighbour-codes of the

remaining neighbours if they all fall below the size threshold for cluster regions, which in this case they do. Hence, we concatenate the new neighbours which do not already exist to the initial neighbour string. Region 3 yields a new neighbour, region 4, which is added to form : 5 3 2 4. Region 2 yields no new neighbours and neither does region 4. Hence the neighbour string terminates with only one large region having been found and *all* the others *part of the cluster*. *All* the entries in the neighbour-string are now merged to yield the required single region shown in the figure. Merging fails if more than 1 large region occurs in the neighbour string, the string exceeds a certain length or a surrounding large region occupies the background band. If no common surrounding region is found, the merging obviously also fails.

Figure 9 shows how an image containing clusters has been processed. It is seen that regions have been effectively "cleaned up" making it possible to apply the un-merging strategy described in chapter 5. Although some of the "holes" seem to consist of only single regions, they in fact consist of clusters which often have regions joined to them that are too small to show any pixels on the *demonstration image* other than the white border pixels that define their perimeters. All the single "holes" would in any case have been removed during gray-level merging.

A PDL description of the cluster-removing routine is presented in Appendix F.

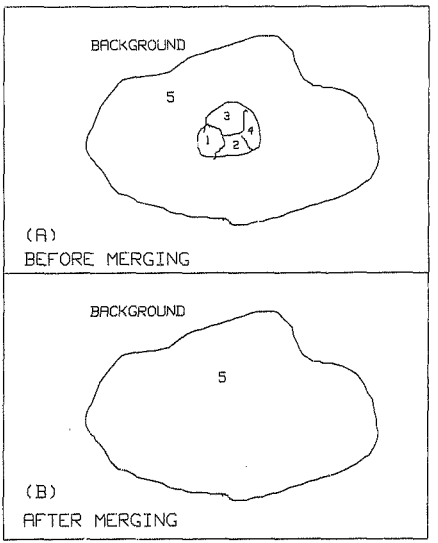
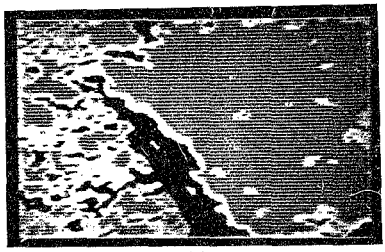
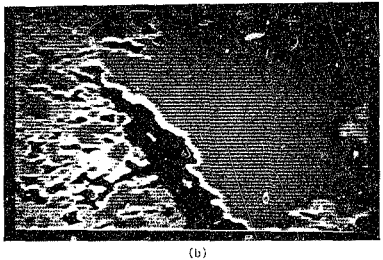


Figure 8. Region-cluster merging: (a) shows a region containing a cluster and (b) is the result after the cluster is merged. (Refer to the text for details).



(a)



(b)

Figure 9. Results of cluster removal from a real image.: (a) shows an image containing regions resulting from gray-level merging - the regions in this image contain region clusters. (b) shows the same image after application of cluster merging.

4.5 GEOMETRIC MERGING

The very limited set of rules used for gray-level merging has lead to the consideration of geometrical properties of regions for identifying further *merge* conditions. Figure 10 shows a common situation that arises when two particle regions cannot be merged to represent a single object using gray-level rules. The nature of the boundaries between the two regions has to be considered if correct merging is to be achieved. In Case 1 shown in the figure we have a smooth boundary resulting from the merge while Case 2 results in a boundary that has a point of inflection. Hence for Case 2, the regions cannot merge.

To perform geometric merging, an exact cyclic neighbour-code is required. Such a code enables the neighbours of a region to be located in terms of their position along the chain-code in exact geometrical sequence and allows multiple occurrences of the same neighbour to be correctly mapped. Geometrical merging can only be achieved if all the occurrences of a neighbour on the border are examined in the correct sequence and the border-geometry conditions are met at all the points in question.

The border-geometry conditions necessary for geometric merging are tested using a set of masks that are placed relative to the points where a region intersects its neighbour. A different mask is used depending on whether the intersection point being tested is the beginning of the neighbour occurrence or whether it is the end of the occurrence. Figure 11 shows the convention followed in using the masks. the first mask is termed the "interception mask" and the second the "detachment mask". These masks impose limits on where pixels belonging to the neighbour region may exist relative to the intersection point in question. If both masks are satisfied, a neighbour may merge to the region in question - in the case of a multiple occurrence of a neighbour, *all* the masks corresponding to each occurrence must be satisfied.

The strategy used to define the masks is to only allow merging if smooth region borders will result. This is incorporated into the masks as is shown in Figure 11 where the areas relative to the interception points where no pixels belonging to the neighbouring region are allowed to exist are defined in terms of the angles α and Θ. For the best performance, both angles turned out to be 45 degrees. The details concerning the different masks that are used and the way the masks are constructed during border searches are discussed in further detail in Appendix D.

Figure 12 shows how geometrical merging is applied to an image that has been through gray-level merging. Geometrical merging is useful in cleaning up regions which exist at the edge of larger regions since these are especially neglected during gray-level merging because they neighbour on the background of the picture. Geometrical merging is not the most important merge procedure used in the recognition strategy since it is limited to follow rigid rules as defined by the mask configurations. The fairly "noisy" structure of region borders at a pixel level make it difficult to define most of the conditions where merging could be effective without also increasing the occurrence of merge errors. For this reason geometrical merging is also more effective if the image is initially labelled using 4-connectedness where regions of less complex geometry usually emerge. In the concluding chapter the relevance of geometrical merging to the particle recognition system will again be considered.

Appendix D contains further examples of geometric merging, showing how geometrically regular objects can be merged because of their well defined and smooth borders. Algorithms associated with geometric merging appear in PDL form in Appendix F.

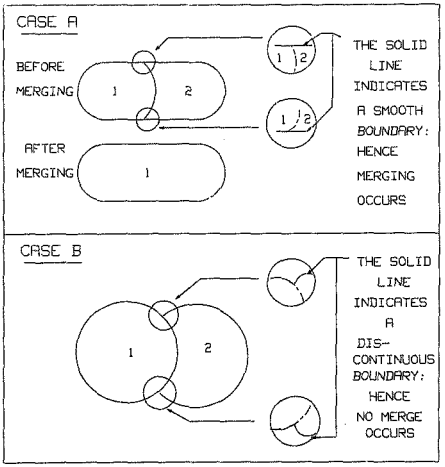


Figure 10. Geometry decisions: Case (a) allows merging ; case (b) does not

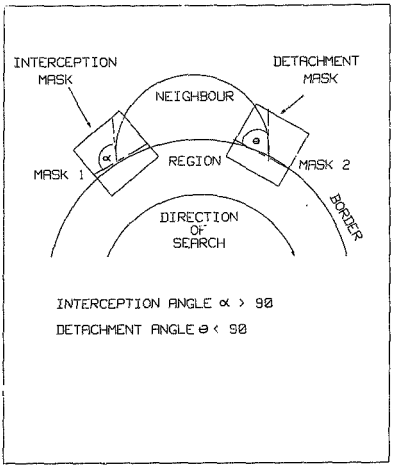
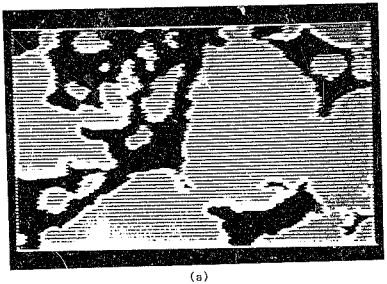
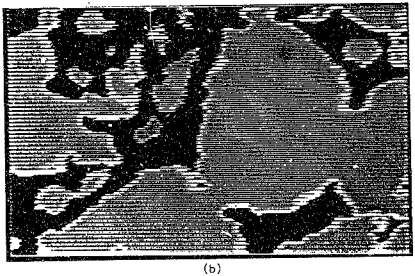


Figure 11. Masks used for geometric merging



(a)



(b)

Figure 12. Results of geometric merging: (a) shows an image merged using gray-level rules. (b) shows the result after geometrical merging : note the cleaning up of small regions on the edges of larger regions.

4.6 SUMMARY

In this chapter it was shown how regions contained in a labelled segmented image are merged using three different merge procedures. The first merge process uses gray-level criteria to determine merge rules for merging. It was seen that this merging stage produces the most significant region-growing to occur and even allows a rough estimation of particle size to be made. The second merge stage introduced in the chapter uses neighbour-lists to merge region clusters to ensure that the larger regions in the image are free of smaller regions situated inside their perimeters. The last merge procedure that was discussed dealt with merging of regions using geometrical features. It was seen that the least amount of merging occurred using this procedure. The following chapter shows how errors made during image-segmentation and region-merging may be corrected using a region-splitting method.

5.0 UN-MERGING OF REGIONS

5.1 INTRODUCTION

A problem which frequently arises in scene analysis is that of distinct components in a scene which have been joined together. This may come about during the thresholding process (especially in binary processing) or during the merging process where regions are merged incorrectly. This chapter discusses a method of region splitting which may be used to separate a single region into constituent parts which in fact represent objects in the scene. It is shown that for rock-pictures the process is fairly simple if we assume that rocks in general have convex geometry.

Work has been done on the problem of separating blobs in pictures and usually the approach has been to apply border-eroding and region shrinking algorithms. These techniques were tested using routines available in the SPIDER package and were found to seriously degrade the scene representation while not really solving the problem of joined regions. For background information on border-eroding methods see Rosenfeld(1982) and for applications in particle sizing Serra(1982).

Section 5.2 compares border shrinking and the region-splitting techniques to provide background to the choice of the region-splitting technique developed for this application. In Section 5.3 the region-splitting technique itself is explained in more detail and in Section 5.4 it is shown how the procedure is accelerated to minimise the necessary execution time.

5.2 UN-MERGING : SPLITTING VS SHRINKING.

Figure 13 shows an example of a region which needs to be un-merged. It is seen that based on the assumption that rocks are convex *in shape*, the region shown does not qualify as being a rock. A frequently used method of solving this problem is to traverse the border of the region a few times, each time removing a layer of border pixels. The border is hence "shrunk" and eventually the two constituent parts will be separated. An obvious limitation to this method is that a region's size is reduced as the shrinking proceeds, making a size calculation less accurate. It is also not possible to define the number of iterations which are required to separate the objects completely since it will differ depending on how tightly they are joined. Shrinking may, however, be applied when counting of regular objects such as spheres or discs is required. Shrinking may be applied followed by an expansion procedure where objects are "grown" without allowing adjacent objects to merge again. This method is an improvement on the first but still relies on the shrinking process to separate "blobs" in the first instance. Serra(1982) discusses the application of such methods for analysing binary images of connected spherical objects.

Region-splitting provides a more versatile approach to the un-merging problem. Here it is attempted to find a "bottleneck" in the region corresponding to where the two objects have been joined. By simply drawing a line across the bottleneck, the two constituent regions may be separated as is shown in Figure 13. No pixels are lost during the splitting and the processing involved also doesn't rely on any iterative process. It should be realised however, that this technique may only be used because of the assumption that rocks have a convex geometry. For objects having narrow protrusions such as say the handle of a hammer, this method will attempt to split the protrusion from the object, ie. separate the handle from the hammer. Fairfield(1983) explains the use of [illegible]d geometric analysis to perform segmentation of blobs into sub [illegible] and is applicable to cases where the simple convex structure assumed here may not be

the only shape description involved. The next section outlines the region-splitting technique in more detail.

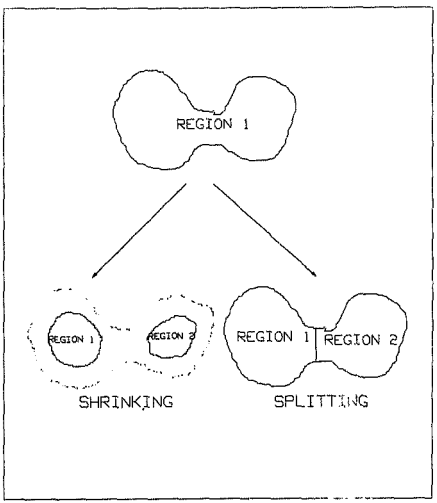


Figure 13. Shrinking vs Splitting: notice the reduction in surface area in the case of shrinking.

5.3 UN-MERGING USING REGION-SPLITTING

Region-splitting is achieved by scanning a region in four directions as is shown in Figure 14. During each scan, the lengths of the chords across the region are recorded. The position of each chord is also recorded eg. for a horizontal scan, the line of the scan (y-ordinate) and the column (x-ordinate) at which the chord starts and ends are recorded. It should be noted that more than one chord may be found on any scan-line depending on the geometry of the region. The software for the rock-recognition system makes provision for 10 chords on any given scan-line and this has proven to be more than adequate.

The length of each chord is compared to a reference length and if a chord is shorter than the reference, it is assumed that a bottleneck exists across the region at the location of that chord.

The reference length may be obtained in various ways. It is for example possible to search for the longest chord found during a certain scan and then to define the reference length as being some fraction of the longest chord. This method was used and apart from being computationally time-consuming also didn't yield accurate splitting for all shapes of regions. It was found that by using the circumscribing rectangle (see Appendix A) as a basis for deriving the reference chord length, better results were produced.

In Case (a) in Figure 14, ie. vertical scanning, the side along the x-axis of the circumscribing rectangle would provide the initial length from which the reference is obtained by dividing the length by some factor. The factor found to yield the best splitting was 5. Hence, for horizontal and vertical scanning, the reference length is 0.2 x (the length of the side of the circumscribing rectangle that lies in the direction of scanning). For scanning in the diagonal directions, the reference length is taken as 0.2 * COS (45) x (the length of the shortest side of the circumscribing rectangle).

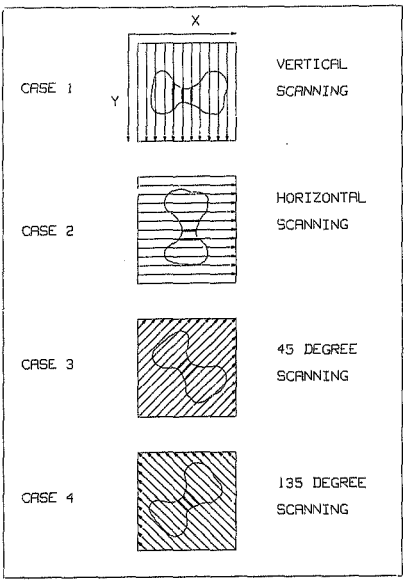


Figure 14. Scan-directions for region-splitting

Having found the chord that corresponds to a bottleneck, it is necessary to find two additional chords, one on either side of the first chord. These chords are used when re-labelling of the split regions is performed and this will be described shortly. The corresponding start and end-points of the three chords should lie close together to ensure that they all define the same bottleneck. The three chords defining the bottleneck will from now on be referred to as the "split-chords". The region may now be re-labelled by propagating a re-labelling wave-front from one of the outer split-chords and substituting old region pixel labels with a new label that is chosen. Each pixel that is substituted contributes to the area of the new sub-region that is being formed. Re-labelling is completed when no more pixel substitutions occur. Figure 15 shows how the re-labelling wave-front is propagated more illustratively.

The use of three split-chords is now explained :
Two chords are used to form a barrier to prevent the re-labelling wave-front to cross to both sides of the region. The third chord is required to start wave-front propagation. The reason for its use as well as a more detailed description of the re-labelling method is described in Appendix E. The two chords forming the barriers are necessary to prevent a diagonal pixel to be crossed at the end-points of the split-chords during re-labelling which is done using an 8-connected re-labelling mask. This is especially necessary when dealing with the diagonal scanning searches where only a double line can stop a wave front from passing through. The two split-chords are initially represented by pixels with the label "0" which prevents the passage of propagation fronts. After a region has been successfully split and re-labelled to produce the two sub-regions, the split-chord pixels are changed from label "0" to the value of the new label being used.

After re-labelling the region to form two separate labelled regions the areas of the two regions are compared to see if an acceptable split was made. To prevent a region to be split into a number of insignificantly small sub-regions, the smaller of the two regions should always be larger than some minimum fraction of the larger. The fraction that is used for the rock-recognition is 1/10 which produces a good coarseness in the

splitting operation. If a region is below the size-limit, the un-merge is un-done by simply replacing the new labels with the old label thus reconstructing the original single region.

The areas of the sub-regions may also be used to detect a particular situation where problems often arise. This is when a region that is being split contains other regions inside its perimeter. Figure 16 shows the situation in question. Region 2 is a region inside region 1 and we see that chords have been found that stretch from region 1 to region 2. A re-labelling wave-front now propagates without finding two separate regions. This problem is reduced by eliminating regions using the cluster merging technique described in Chapter 4 section 4. Such situations however, still arise if regions have included regions that have areas which are larger than the limit specified for the cluster-merging routine mentioned above. This area-limit cannot be made too large since significant merge errors arise if the cluster-elimination is applied to regions that are too big.

The problem is solved by keeping the two chords that form the split on the region. Each split chord is still labelled as "0" hence still preventing further wave-fronts from propagating through them. If further split-chords are found in the same region, the re-labelling wave-fronts will terminate because they cannot pass through the "wall" formed by the first split-chords that still exist. For more complex situations it may be necessary to repeat the above procedure a number of times before splitting is achieved.

The region-splitting technique is time-consuming because of the large amount of re-labelling and wave-front propagation which is required. The following section shows how methods are applied to enable processing to be speeded up.

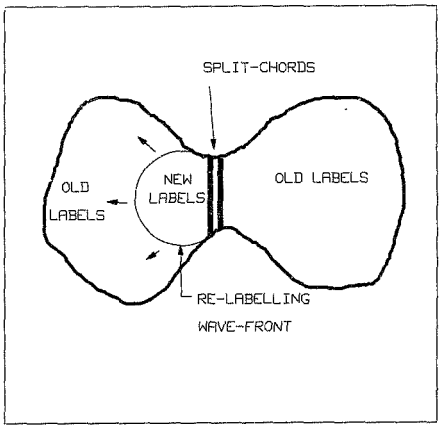


Figure 15. Propagation of the re-*labelling* wave-front

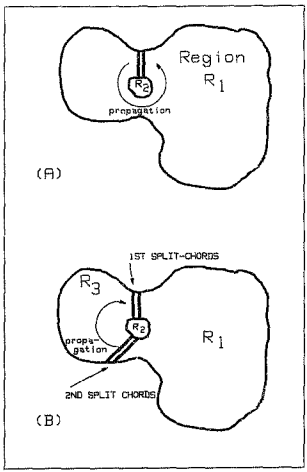


Figure 16. A problem encountered during region-splitting: (a) shows the propagation front that returns back to the split-chords without generating a sub-region. (b) shows how the wall formed by the first split-chords enables successful sub-region labelling to be achieved during a following labelling procedure.

5.4 ACCELERATING THE UN-MERGE PROCESS

Because of the large amount of repetitive computation involved in the un-merging process, a substantial demand on processing-time is made and it is desirable to relieve this time burden to an extent that it becomes feasible to implement the un-merge procedure in a real hardware environment. This section explains some of the methods that have been investigated to aid the acceleration of the region-splitting stage.

Efficient execution may be achieved by careful consideration of the iteration-process in which regions are successively split as well as keeping track of the number of times that sub-regions are unsuccessfully found and re-labelled. The sequence in which the different scans are performed is also important - referring back to Figure 14 we find that the horizontal and vertical scans are quicker than the diagonal scans because diagonal scanning produces twice the number of scan-lines than scanning along the principle axes. We hence apply the faster scans first followed by the slower scans and thereby increase the possibility that a region will be split without requiring further diagonal searching to be used. Time is also saved by not processing all the particle-regions in the image. Regions are only sent through the un-merge process if they are above a certain size. This may be done because merge errors are more significant for the larger regions in the image.

The iteration process allows a region to be repeatedly split until either all the constituent sub-regions that are found are below the size threshold or all the scan-directions have been searched. When a region is split into two sub-regions, a record is kept for each sub-region containing the last scan-line that was used to split them apart. The scan-direction that was used is also known and hence whenever one of the sub-regions is processed again, scanning resumes in the scan-direction that was recorded and at the associated scan-line. Hence redundant scans are never applied to a region making the process more efficient and hence quicker. The method of recording scan-lines is explained in further detail in Appendix E.

The second method that was mentioned at the beginning of this section, that of using knowledge of the number of unsuccessful merge attempts, is now discussed further. If a particular region has not yielded successful splitting (requiring un-doing of the complete re-labelling process) more than a specified number of times, then a mode is entered where only every n^{th} scan-line in the particular search direction being used is processed. By skipping scan-lines, problem areas may be "side-stepped" and the execution speeded up. It is of course possible to skip over possible un-merge situations thus degrading accuracy. In a real-time system it would however be better to accept this error rather than to run short of time.

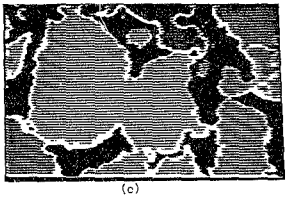
Figure 17 shows how un-merging was applied to a merged rock-region image. The processing shown in the image is the final stage in the recognition process and the areas of the regions shown may now be used to obtain a 2-dimensional particle size distribution. If the circumscribing rectangle of each region is used, a very close approximation to the commonly used sieve-size measurement of particle size may be derived - calculation of the circumscribing rectangle forms part of the-unmerge process and would thus not have to be repeated for regions that have been through the un-merge procedure, hence saving further processing time.

Further examples of un-merging may be found in Chapter 6 in Figures 18 to 25 where it is seen how region-splitting improves the interpretation of a scene and how well merge-errors may be resolved. The black lines in the images show where the problem of small regions contained inside larger regions has occurred and how the split-chords have been used to correct this.

Algorithms describing the region-splitting method are available in PDL form in Appendix F.

(a)

(b)



(c)

Figure 17. Results of region-splitting: (a) shows the input rock-scene , (b) is the merged region image and (c) is the un-merged image.

5.5 SUMMARY

In this chapter it was seen how merge-errors could be recovered using a region-splitting operation. It was shown how the process could be accelerated using various techniques to make searching more efficient and by monitoring the amount of abortive attempts at splitting a region into sub-regions. It was seen that merge-errors are corrected very well and that the accuracy of the recognition system was significantly improved.

The work up to and including this chapter comprises the complete rock-recognition system. The following chapter discusses the performance of the method described in this dissertation with reference to a number of processed conveyor-images.

6.0 EXPERIMENTAL RESULTS

6.1 INTRODUCTION

This chapter discusses the performance of the recognition system which has been described in the previous chapters. Photographs of eight processed images are shown to enable a direct assessment of the accuracy to be made by referring between input images and processed images. An actual size distribution in four ranges is given for each image to enable a comparison to be made to the size distribution computed by the recognition system. Section 6.2 briefly outlines the aspects concerning the test procedure and test images appear in Section 6.3. In Section 6.4 a discussion of the results is given to present an objective view of both the stronger and weaker features of the recognition system. Section 6.5 deals with aspects concerning the execution time required for the rock-counting system.

6.2 BASIS OF PERFORMANCE MEASUREMENT

The results of processing are presented as a series of photographs taken from a monochrome monitor displaying images from computer memory. Each set of photographs shows the input image and the processed image containing regions representing the "recognised" particles. The number of gray-levels present in the input picture is also given to show how the segmentation method copes with pictures of different contrast and brightness. A size distribution has been computed for each input picture by investigating the size of the particles visible in the input image. This distribution is computed based on the surface area of the particles visible in the image - each picture shown is of size 350x350 pixels and hence the area of a particle may be related to a pixel size by comparing

its area to the total area of the photograph. The distribution thus obtained may be compared to the distribution computed by the recognition system and the performance evaluated.

The size-analysis has been limited to provide four ranges of rock-size ie.

- "small" : particles having an area between 0 and 255 pixels
- "medium" : particles having an area between 256 and 1023 pixels
- "big" : particles having an area between 1024 and 4095 pixels
- "large" : particles having an area larger than 4096 pixels

Note: The particular size in pixels defining classification into one of the four size ranges was chosen arbitrarily and not to a particular criterion.

Good comparisons may be made between measured and computed sizes for the classes "large", "big" and "medium". "Small" rocks are however difficult to count by studying a photograph and this part of the distribution has hence not been included in the performance evaluation. The "small" range in the distribution computed by the recognition system may however be used to obtain an estimation of the amount of fine material in the image as will be seen later. The exact relationship between the number of "small" rocks found and the amount of fines (eg. volume measure) will not be investigated in this dissertation. Relating the number of "small" rocks to the amount of fines as well as using the computed surface area of the larger rocks to estimate their physical size are aspects of the size-analysis which fall outside the scope of this work. For more information on use of projected surface area and other properties of irregularly shaped particles to compute a size distribution, consult King(1982), Barbery(1974) and Allen(1981).

Use of the four size ranges was suggested by the sponsors of the project as this was sufficient to implement a mill-control strategy. A technique that would be required to perform rigorous testing of the size-analysis system would be possible only once a prototype is operational. This strategy would entail the passing of material of known size-distribution through the analyser several times, each time re-arranging the material on the belt. The repeatability of the size measure as well as the accuracy of measurement could then be established for various conveyor-belt configurations and types of material. It would therefore also be possible to investigate the effect of the area of the conveyor that is not covered by the sampled image frames on the performance of the system.

By investigating the accuracy of the four-size-range classification a measure of performance may however be obtained based on the images shown in the following section. An error may be calculated for each size range for each picture and an average computed over all eight images shown. The error is computed by dividing the absolute difference between the actual number of particles in a size range and the computed number by the actual number.

Images shown were obtained either from scenes synthesised in the laboratory or from the DEELKRAAL gold-mine. Scenes consist mainly of an assortment of larger rock material imbedded in finer particles to make thorough testing of the particle-finding algorithms possible. The poor quality of the DEELKRAAL images was also a good test for the recognition system since this probably represents the worst condition under which a size analyser could be expected to operate.

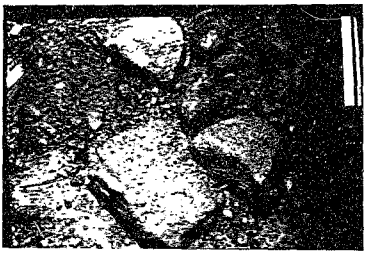
The testing procedure was limited by the number of available images which had to be restricted because of limited available computer storage facilities. Images shown in the next section do however represent typical scenes on conveyors and were randomly selected for processing using the same algorithms. Particular scenes were favoured for presentation only where specific features such as bright patches caused by reflected light were evident to point out difficulties that could arise during use of a size-analyser based on the methods employed here.

The relationship between pixels on the images shown and physical size is approximately one pixel to 4×10^{-6} m^2 Each 350 x 350 picture hence covers an area of approximately 0.5 m^2

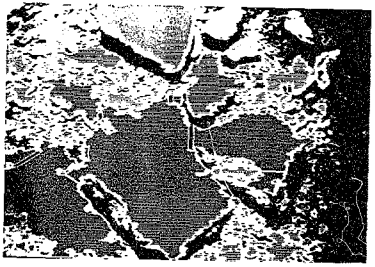
Although the photographs shown are rectangular, they do in fact represent square images since the display monitor reproduces pixels that are elongated instead of square.

The images shown in the next section were all processed using identical algorithms utilising the same parameters to enable a realistic performance evaluation to be made.

6.3 RESULTS



(a)



(b)

Figure 18. Result 1.

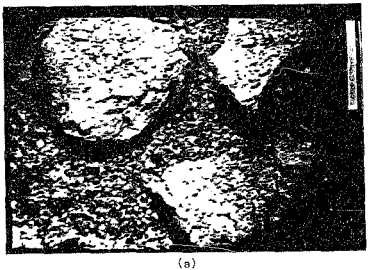
Laboratory image : (a) = input image
(b) = result image

Occupied gray-levels : 0 - 54

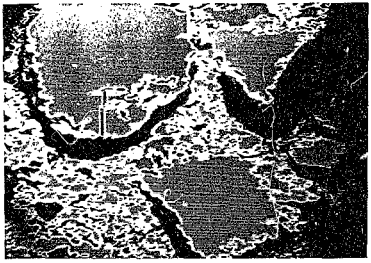
Distribution :

	Computed :	Actual :
LARGE :	4	3
BIG :	5	8
MEDIUM :	16	9
SMALL :	2115	-

Comments : The patch of wet fine material on the bottom-left has reflected the light to cause the additional large region to be found.



(a)



(b)

Figure 19. Result 2.

Laboratory image : (a) = input image
(b) = result image

Occupied gray-levels : 0 - 63 (fully occupied)

Distribution :

	Computed :	Actual :
LARGE :	3	3
BIG :	2	1
MEDIUM :	16	10
SMALL :	2151	-

Comments : More medium rocks are found because of merging of fines and the cluster still present on the large rock in the top-left of the image.

(a)

(b)

Figure 20. Result 3.

Laboratory image : (a) = input image
(b) = result image

Occupied gray-levels : 0 - 47

Distribution :

	Computed :	Actual :
LARGE :	0	0
BIG :	12	6
MEDIUM :	33	20
SMALL :	3646	-

Comments : Reflection of lighting caused too many large regions to be found. Significant over-merging of smaller regions has occurred. Note, however how un-merging still maintains a reasonable interpretation - no "large" regions are found.

(a)

(b)

Figure 21. Result 4.

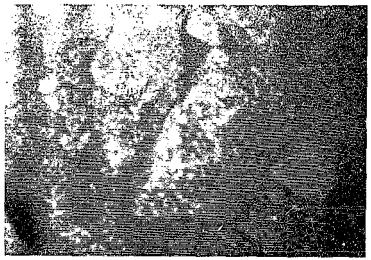
Deelkraal image : (a) = input image
(b) = result image

Occupied gray-levels : 0 - 11 (note the low occupation)

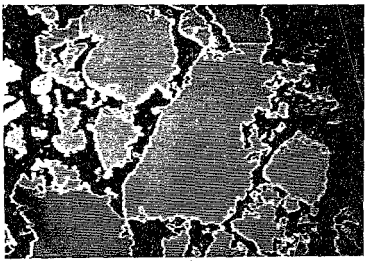
Distribution :

	Computed :	Actual :
LARGE :	4	2
BIG :	7	6
MEDIUM :	20	16
SMALL :	5926	-

Comments : The error in the number of large regions is due to over-merging of fines in the righthand side of the image as well as the under-merging of the *large rock in the* centre to form 2 seperate regions also classified as "*large*". Geometric merging was not capable of merging these two regions to correct the error.



(a)



(b)

Figure 22. Result 5.

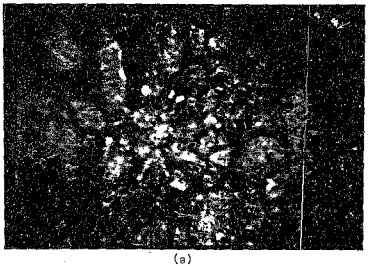
Deelkraal Image : (a) = input image
(b) = result image

Occupied grey-levels : 0 - 14 (note the low occupation)

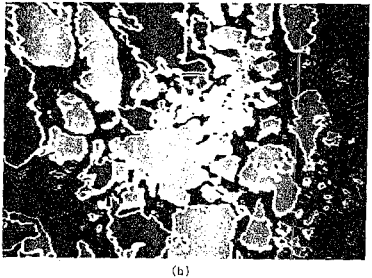
Distribution :

	Computed :	Actual :
LARGE :	3	3
BIG :	4	4
MEDIUM :	15	10 - 15
SMALL :	604	-

Comments : An accurate size measurement was achieved here despite bad contrast and brightness. Un-merging was important in achieving this accuracy.



(a)



(b)

Figure 23. Result 6.

Deelkraal Image : (a) = input image
(b) = result image

Occupied gray-levels : 0 - 13 (note the low occupation)

Distribution :

	Computed :	*Actual* :
LARGE :	3	2
BIG :	7	9
MEDIUM :	25	30
SMALL :	897	-

Comments : Over-merging in the centre was caused by an area of more intense illumination. Interpretation is otherwise acceptable.

(a)

(b)

Figure 24. Result 7.

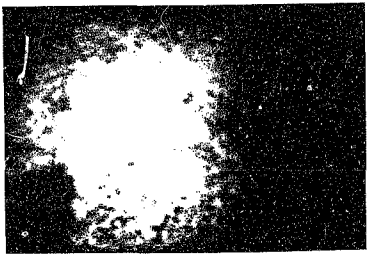
Deelkraal image : (a) = input image
(b) = result image

Occupied gray-levels : 0 - 17 (note the low occupation)

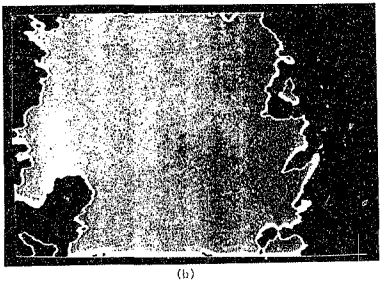
Distribution :

	Computed :	Actual :
LARGE :	3	3
BIG :	4	4
MEDIUM :	18	20
SMALL :	798	

Comments : A very good distribution was obtained here despite poor picture quality.



(a)



(b)

Figure 25. Result 8.

Deelkraal Image : (a) = input image
(b) = result image

Occupied gray-levels : 0 - 51

Distribution :

	Computed :	Actual :
LARGE :	1	1
BIG :	0	0
MEDIUM :	2	0
SMALL :	66	~

Comments : Medium regions result from seperation during 5-band segmentation. Note how well the single large rock has been identified.

6.4 DISCUSSION OF RESULTS

The results of the rock-recognition system shown in the previous section represent a realistic cross-section of the type of images that may be encountered on ore carrying conveyor belts. Images were selected for presentation to illustrate problems that may arise in practice and this has biased the results to indicate somewhat lower accuracy than would be expected under more ideal conditions. Use of four size ranges has resulted in a fairly rigid classification of regions into the various size groups, perhaps discarding a great deal of information that is present pertaining to the more precise size of each region.

For the eight pictures shown, the accuracy was found to be as follows :

- "large" size : 80 %
- "big" size : 60 %
- "medium" size : 70 %

By combining the accuracy of the classification into these size ranges, an overall mean accuracy for the eight images is hence 70 % . It is important to note that this measure represents the accuracy in terms of how a human would interpret the *same 2-dimensional* image and is not a measure based on the actual particles present in the scene. As was discussed in Section 6.2 , an accuracy measure for the "small" class of particles could not be obtained because of the difficulty in counting the small particles in the original images. It is however clear that a close resemblance between the number of small regions and the amount of fine material exists as can be seen from the results.

The resultant accuracy of 70 % should not be interpreted as an absolute performance measure since it only applies to the images shown. Bearing in mind that the pictures shown were chosen to demonstrate some of the problems that arise under unfavorable lighting conditions such as re-

flections and poor contrast, it may be expected that a higher accuracy could be obtained if a larger number of tests are performed under suitable lighting conditions.

The laboratory pictures especially, have yielded good resolution of small particles because of the better picture quality. The images from Dealkraal lack the same resolution of fine material because of their low contrast and brightness. The use of the background band to compensate for poor picture quality during image segmentation has forced darker pixels belonging to the smaller particles to be classified as background. This is evident from the large black areas present on the processed Dealkraal pictures.

When the features of the input images are compared to the regions which are extracted, it is seen that in most cases regions correspond to physical objects in the scene. Bright patches caused by reflection of light are interpreted as rock-regions and this phenomenon would only be eliminated by compensation with suitable lighting techniques or the addition of more complex recognition software possibly comprising of some form of texture analysis combined with the detection of regions consisting of pixels with saturated gray-levels or "maxima".

It is seen that pictures of varying brightness and contrast (evident from the gray-level occupation shown) were processed very well and this is as a direct result of the 5-band segmentation algorithm. The images shown also have the common feature of rocks embedded in fine material thus presenting a very indistinct background from which to distinguish individual particles. The fact that particles have been identified, despite this limitation, is also attributed to the 5-band segmentation method which "creates" the necessary background through use of the background band. The segmentation technique that has been devised for this particular application hence incorporates rigidity into the recognition system and makes it robust against changes in external conditions such as changes in illumination or degradation of the optical path due to the accumulation of dirt.

The importance of the un-merging routine is appreciated when considering the large amount of over-merging which has occurred during processing. If the result-images are studied and the lines resulting from the un-merge procedure are hypothetically removed from the regions, it is seen that if it were not for this region splitting technique, significant merge error would have resulted giving an accuracy similar to that expected from binary image processing methods. It is therefore important that the un-merging procedure be accommodated in a working instrumentation system and that extensive research be done to enable its incorporation into a fast hardware architecture.

6.5 ISSUES RELATING TO PROCESSING-TIME.

The time required to perform processing on a given image using the methods discussed in this dissertation is limited to the execution rate of the computer on which the software is run. For the HP-1000 A700 (configured as an A600) the processing times associated with the two sizes of images used for development and testing are as follows :

SIZE (pixels)	EXECUTION TIME (minutes)
128x128	min. 15 max. 30 ave. 20
350x350	min. 120 max. 210 ave. 150

Processing time is dependent on the number of regions that are found during segmentation and hence differs from image to image. These results apply to test images that were processed in each size and are sufficient for the purposes of finding pertinent trends. Although the processing times achieved are not relevant to real time operation, important con-

clusions may be drawn concerning the relationship between image size and processing time.

From the above results the number of pixels processed per second may be computed for each size and a relationship established. The results are shown below :

SIZE (pixels)	SPEED (pixels/second)		
128x128	max. 18	min. 9	ave. 14
350x350	max. 17	min. 10	ave. 14

It is clear that the number of pixels processed per second is nearly constant for the two sizes of images. This indicates that a linear relationship between processing speed and the image size (in pixels) exists and that the software does not impose any non-linearities on this relationship. It may therefore be concluded that for very small images processing might be achieved in real-time and that since operations performed on regions are localised, it is possible to perform a great deal of parallel processing on an image. By dividing an image into "windows" that are small enough to process in real-time, it is hence possible to process each window in parallel and obtain the necessary processing speed. The linear relationship between speed and image size however also suggests that more powerful serial machines could be used to process larger images in the required time but would probably only be feasible for machines too costly to prove economically viable. For more details on approaches to implement the image-based size-analyser in hardware consult Smith(1985).

6.6 SUMMARY

Results of processed images have been presented in this chapter to give an impression of the performance of the methods developed in this dissertation. It was seen that classification of regions into three size groups "large", "big" and "medium" was done to an accuracy of approximately 70 % for the eight images shown while a good estimation of the amount of fine material could be derived from the "small" size classification. Processing of images with different levels of brightness and contrast was successful and it was evident that rock-recognition is possible despite indistinct background features. The importance of the un-merging procedure to yield accurate size-measurement was also apparent. It was seen that processing-time is directly proportional to the image size in pixels and this suggests that the use of a multi-processor based parallel architecture is feasible to decrease the processing time for large images.

7.0 CONCLUSIONS AND RECOMMENDATIONS

7.1 SUMMARY AND CONCLUSIONS

The particle recognition system described in this dissertation consists of three separate stages, each stage being vital to the success of the overall strategy.

The first step in the recognition process is the segmentation of the input image into primitive regions. These regions could subsequently be used to construct an image consisting of larger regions which correspond to physical objects in the original scene. Primitive regions are classified as belonging to either parts of the background in the image or belonging to parts of objects. An important part of this processing stage is then also to define the threshold that separates the gray-levels which form the background from the gray-levels which correspond to objects. It was explained in Chapter 3 that by reducing the image histogram from 64 to 5 levels a segmentation is produced where the first of the five gray-levels would define background pixels and the remaining four gray-levels define "rock" pixels. By carefully selecting the extent of the background gray-level, i.e. how many gray-levels in the original histogram are mapped into the background gray-level, compensation for variations in picture quality is possible to prevent the classification of unreasonably large portions of the image as either background or particle regions. Hence it is possible to overcome the effects of these variations which would seriously degrade the performance of a system which used only a fixed threshold to distinguish background and objects.

From the results shown in Chapter 6 it is evident that the segmentation method performs good image segmentation for images having varying degrees of brightness and contrast. Images occupying only 12 gray-levels were processed with a similar resolution to images occupying all 64 gray-levels.

The segmentation method incorporating the histogram compensation mechanism forms the foundation on which the accuracy of the entire recognition strategy rests. Fortunately the method of calculating the extent of each of the five gray-levels from the histogram is not complicated and the overall segmentation and region-labelling procedure is very efficient, making the implamentation of this stage of the recognition procedure in a size-analysis instrumentation system feasible.

The second stage in the recognition process is concerned with the merging of primitive regions to form regions which correspond to objects in the scene which is analysed. Merging according to the relative gray-levels of regions and their neighbours is the most dominant merge strategy used since it generates the largest number of merges. As was seen in Chapter 4, this procedure produces regions which may be used to obtain a crude particle size measure. Unfortunately however, this is not considered to be a reliable measure since significant over-merging often results which distorts the interpretation of the original image. A better calculation of a particle size distribution may be obtained at this stage if more background is "forced" into the image during the segmentation process to reduce the occurrences where particles are joined together because of over-merging. It should still be realised though that this would have an effect on accuracy similar to the region shrinking process discussed in Chapter 5. For purposes of developing a prototype instrument it may be informative to use the approach suggested above to get an appreciation for the complexity involved in later development of a system utilising the complete recognition strategy.

A further region-merging stage based on region-border geometry was discussed in Chapter 4. It was seen that a limited amount of merging of small regions on the border of larger regions resulted. Merging of larger regions was ,however, very uncommon because of the irregular shape of the borders of bigger regions at a pixel level. The exclusion of this merge stage would not seriously affect the performance of the recognition system since it would only result in a few more regions being added to the small

size range of particles. The usefulness of *geometric merging* for other applications where scenes of more regularly shaped objects are encountered should however be *realised and further* research into this technique for dealing with geometric descriptions of regions is recommended.

A third merging strategy is used to prepare regions for the final stage in the recognition procedure. This merging strategy concerns the removal of clusters of small regions from inside larger regions classified as "particle" regions. The merging performed here is necessary to improve the efficiency of the region un-merging stage which is seriously hampered by the presence of regions inside the boundaries of other larger regions.

Both the gray-level merging process and the cluster-removal procedure rely on the generation of neighbour-lists of regions *which is mainly a* serial processing operation. A method has however been developed for implementation of these merge-routines *in a multi-processor* environment making their use feasible for real-time application (Smith,1985).

The final stage in the recognition system is a process of correcting *merge-errors* which cause separate objects lying close together to be joined. Regions are scanned in four directions to detect the occurrence of merge errors. When an error is found, a region is split by placing a chord across the region at the position where the error is detected. In Chapter 5 this method was explained and it was pointed out that methods such as recursive region shrinking and growing were not as accurate as the region-splitting approach. In Chapter 6 results were presented that indicated the importance of region-splitting and it was evident that its use is essential in producing an accurate size-distribution calculation.

A restriction on the use of the region-splitting method is however the amount of time *required in scanning* the regions and propagating re-labelling wave-fronts. In Chapter 5 it was shown how processing could *be made more efficient* to reduce the time-overhead associated with region-splitting. The region-splitting technique doubles the execution time of the recognition system even when these techniques are applied making it important to find additional ways of reducing its complexity.

It is believed that the method would have to be adapted to the hardware architecture that is selected for implementation of the size-analyser. The serial algorithms presented in this dissertation will certainly not allow real-time operation and are intended only to prove the viability of the approach of obtaining merge-error correction.

From the results shown in Chapter 6 it was seen that the recognition system is expected to yield an accuracy in excess of 70 % when a size-distribution of three size ranges is calculated. It was also evident that an estimate of the amount of fine material in an image could be given by considering the number of small regions found in the image. The results indicate that the methods developed for the recognition system perform reliably and have combined to provide a robust and flexible solution to the problems associated with the processing of conveyor-belt scenes. Rigorous testing on a large number of images would provide a realistic test of the accuracy of the recognition system which because of a restriction on the computer storage facilities available could not be performed during the formal research.

The objectives that the research was aimed at have been successfully met in that a complete recognition system has been developed to perform interpretation of real 2-dimensional images of coarse particles on conveyor belts. A system has been presented that may be either used in its entirety or may be simplified by removing modules such as the geometric merging or region-splitting stages to improve execution time. This would of course affect the accuracy of the size calculation adversely and would have to be justified by limitations in available hardware technology. It is expected that the complete system would provide performance far superior to that of currently available particle-size-distribution measurement systems mentioned in the first chapter. The complexity of this method is however much greater and a much more sophisticated hardware environment will be required to produce an operational instrumentation system. Although no off-the-shelf hardware is at present available for sophisticated real-time image processing applications, it is believed that the current world-wide research effort into parallel computer architectures for image processing systems will result in off-the-shelf systems becom-

ing available in the near future (Lerner,1985 ; Kuschner and Rosenfeld,1983).

7.2 RECOMMENDATIONS FOR FURTHER WORK

The particle recognition method discussed in this dissertation has been developed without considering effects of image blurring caused by the motion of the particles on the conveyor beneath the camera. The assumption that a very rapid frame-grabbing system would minimise the effects of blurring would have to be investigated and additional image pre-processing routines might have to be added to compensate for blurring if it proved to have significant influence on the accuracy of image interpretation.

Research is required to find the method of lighting best suited to provide optimal conditions for accurate image interpretation. Problems associated with the reflection of lighting off wet surfaces should be addressed as well as the most suitable arrangement of lighting to produce optimum contrast between background and objects as well as between shadows and objects. The type of lighting used (eg. tungsten, neon etc.) should be considered as well as techniques to achieve constant illumination to maintain a constant image brightness.

Once an image acquisition system is available to enable the capturing of moving images and suitable lighting has been established, images may be processed using the developed software to test the accuracy of the recognition system more rigorously. Research may then be conducted to find the size-measure most appropriate for the region information generated by the image system. The size measure, be it surface area, circumscribing rectangle data etc. , may then be used to derive the required size-distribution output. The resolution that would be required for the specific type of material to be measured would have to be ascertained since this would influence the size of the image (in pixels) that would

be needed. Since processing time is directly related to image size (see Chapter 6), the pixel density would determine the complexity of the hardware that would be needed to achieve real-time performance.

Having determined the image size which is required as well as the software routines that would constitute the recognition system, a hardware architecture would have to be chosen in which to implement the instrumentation system. The software will have to be modified to suit the hardware environment which would probably involve re-writing of the serial code for use in a parallel context.

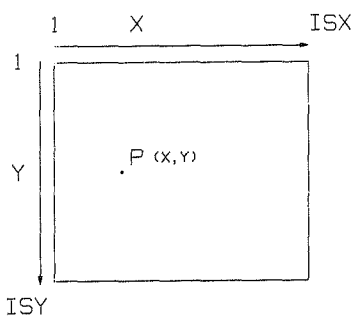
A more ambitious future development would be to consider the extension of the recognition system to process 3-dimensional scenes. By using three cameras situated along orthogonal axes, three views of a scene may be obtained making it possible to extract volume information of objects. The 2-dimensional processing described in this dissertation may be applied to each view and by carefully aligning the pixels in each image a correlation may be done to build up a 3-dimensional description in terms of volume-regions.An interesting development in this field is presented by Luh et al (1985) where three cameras are used to provide a collision-avoidance system for industrial robots. The hardware required for the 2-dimensional system would have to be triplicated and software added to achieve correlation of the region information produced by each 2-dimensional recognition system. It is believed that the hardware would not need to be significantly upgraded to realise the 3-dimensional system and that such processing would be feasible if it was found that the accuracy of the 2-dimensional system was hampered by the lack of depth information.

APPENDIXES

APPENDIX A. CONVENTIONS

The *following conventions* regarding the image co-ordinate system and region descriptions are used in the work described in this dissertation :

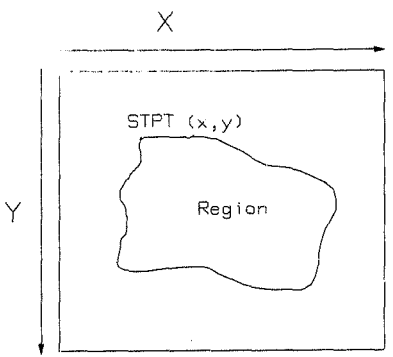
Co-ordinate system :



P(x,y)=pixel at co-ordinates (x,y)
ISX = length of image in *x-direction (no. of* pixel columns)
ISY = length of image in y-direction (no. of pixel rows)

Figure 26. The image co-ordinate system

The start-point of a region :



STPT(x,y) = start-point
= 1st pixel with the region id encountered during a standard x-y raster scan starting at (1,1) in the positive x-direction.

Figure 27. Definition of the start-point of *a region*

Circumscribing rectangle :

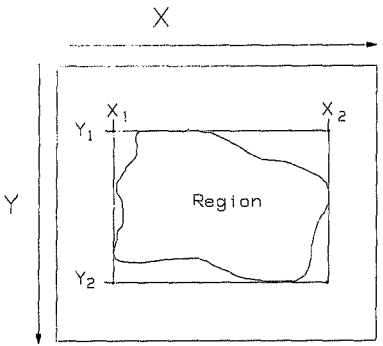


Figure 28. The circumscribing rectangle: The 4 *values* y1, y2, x1 and x2 uniquely define a rectangle that circumscribes a region. Region searches such as finding chords used in region *splitting* (*Chapter* 5) may be accelerated knowing the circumscribing rectangle.

APPENDIX B. GENERATION OF CHAIN-CODES AND NEIGHBOUR-CODES

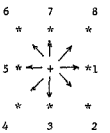
This appendix digresses on the technique used to follow region borders and generate chain-codes as well as cyclic neighbour-strings. The work presented here follows on from the discussion of border descriptions in Chapter 4.

B.1 CHAIN-CODES

Chain-codes are formed by starting at the start-point of a region and tracking the boundary in a clockwise sense until the start-point is reached again (there are special cases where termination isn't achieved by consideration of the start-point alone - this will be explained later).

Each code in the chain indicates the position of the next pixel on the border with respect to the last pixel that was found. Two different kinds of chain-codes are possible depending on whether 4 or 8 connectedness is used. A *4-connected chain*-code allows a pixel on the border to be found in one of only 4 directions relative to the previous border pixel, while an *8-connected code allows one* of 8 directions. The latter gives a more accurate description of the border and is therefor used in the recognition system.

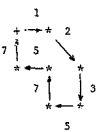
The convention followed for obtaining an 8-connected code is shown below :



where :

'+' = reference pixel
'*' = neighbouring pixel
1-8 = integers defining the direction of each neighbour relative to the reference pixel.

The boundary



(pixel '+' is the start-point)

would be represented by the code 1 2 3 5 7 5 7.

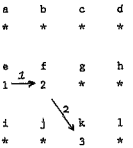
Starting with the absolute co-ordinates of the pixel marked '+', this boundary could then be re-constructed by using the above chain-code.

A feature of the chain-code follower developed here, makes use of explicit knowledge to increase the speed and efficiency of the process of finding successive links in the chain. Since boundaries are always tracked clockwise, it is possible to establish an optimum search-pattern to find the next boundary pixel. This search pattern is also essential to enable neighbouring region pixels to be found in the correct order so as not to

produce incorrect neighbour sequences. The search-pattern is described below and its application to neighbour-finding in the next section.

The first step in obtaining a logical search-pattern is to search pixels *in a clockwise sense*. This follows from the convention followed in respect of the direction that is used to traverse a region's border - in this case we follow the border *in a clockwise sense* and hence also apply the pixel search in a clockwise direction. Since we know the last pixel that was found, that the previous search was clockwise and we know the direction of the link to the last pixel, we also know which pixels must have been searched to reach the last pixel. For each of the eight possible chain-codes, there thus exists a unique direction that points to the first pixel that must be checked in establishing the next code.

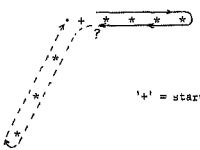
Consider the following situation :



To get from pixel 1 to pixel 2 we say start at pixel (a). Searching in a clockwise sense we then reach (b). Finally, we reach (f) which is the next pixel on the border. The corresponding chain-code is 1. Starting at pixel (f), we now search for the next border-point. Since we know the last code is 1, we also infer that (a) and (b) must have been searched and that the next place to start is (c). We then search (c), (g) and the next link, 2, is found at (k). Similarly, from (k) we start the search at (h) and so on. Thus knowing the previous code in the chain, we know where to search next, hence eliminating searching any pixel more than once. The search position for each previous code (1-8) is given in Figure 29.

Using this search-pattern it is possible to follow region borders that have very strange shapes consisting of only single lines of pixels. The border-following routine available in the SPIDER package could not follow such regions to find closed chain-codes terminating at the start-point.

Although, in most cases, the chain-code terminates at the start-point to give a correct border representation, *it sometimes happens* that something like the following geometry arises :



Here we see that the border consists of only single pixels in a line resulting from a very "stringy" region. The code *follows* the border to the right and returns back to the start-point. Termination at this point is incorrect as we have not yet *incorporated* the lower left part of the region. We get around this problem by finding the last link in the chain-code first. Before we start the chain-code search, we consider the first pixel of the region, ie. the start-point :

```
f   g   h
*   *   *

a       e
*   +   *        '+' = start-point

b   c   d
*   *   *
```

From the definition of the start-point, the region cannot exist at pixels (f), (g), (h) and (a). To find the last pixel on the border, we *now search* in an opposite sense to that used for boundary following, ie. we start at pixel (b) and search sequentially (c), (d) and (e). Unless the region consists of only one pixel, we must find the boundary at one of these four locations. Having found a valid pixel, we then find the link from this pixel to the start-point and hence the last link of the eventual chain-code. Whenever we reach the start-point during a chain-construction, we then check to see whether the link used to reach the start-point corresponds to the correct last link in the chain. If the link is incorrect, the code is merely continued past the start-point until the correct link reaches the start-point.

The chain-code generated using the above methodology is completely cyclic since the last link is the link back to the start-point. for n border points, this code will have n links or entries. The following section describes the use of the chain-code algorithm to find border neighbour-lists.

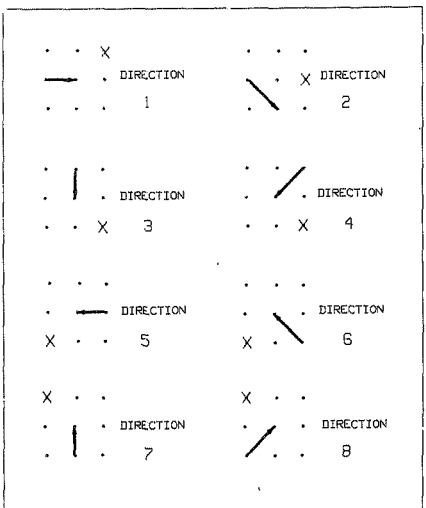


Figure 29. Search-start pixels for each previous link in the chain. "x" is the start-pixel corresponding to the direction indicated by the arrow.

B.2 NEIGHBOUR-CODES

Generation of a list of neighbours as they occur around the border of a region is closely linked to the border-following algorithm that was discussed in the previous section. Neighbour-finding is a simple extension to the *chain-code generator which is in practice* implemented by setting a software switch for either enabling only the chain-code follower or the chain-code with the neighbour-finder option.

When we consider the search-pattern used for border-following, we find that the pixels that are searched to find the *next links in the chain* all occur outside the region's border. Since the information we are after is in fact the identity of the pixels immediately outside the border, all we have to do to construct a neighbour description is to note down the pixels that are searched which do not belong to the region itself.

The strict directional sense of the search-pattern enables us to find *neighbours* in an exact sequence. We may find for instance, that some neighbours occur more than once. If we don't require such an exact representation it is very simple to pack the neighbour-string and delete multiple occurrences. It may, however, sometimes be useful to know the exact description as this makes it possible to merge regions using only *neighbour-strings without ever referring* back to the labelled image. Such merging though, is only possible if no additional geometrical features are to be used for merging (eg. geometrical merging or un-merging using chords as explained in Chapter 5) and hence has not been used in this application. For the sake of completeness, the routine used in the software does generate a cyclic neighbour-*string which is condensed* before being used by the relevant merging procedures.

To illustrate the points made above, consider the following example :

a	b	c	d	e
2	2	3	4	5
f	g	h	i	j
10	1	1	1	5
k	l	m	n	o
9	9	1	1	5
p	q	r	s	t
8	5	5	7	6

Here we have a labelled image and we consider finding the neighbour-code of region 1. (It may be necessary to refer to the previous section for details of the search-pattern which will be referred to here).

Pixel (g) designates the start-point of region 1. Starting with this pixel, we now want to find the first neighbour. A subtle point enters into the procedure at this stage : we have to start finding neighbours by considering the final link in the border chain-code. The last link goes from pixel (m) to (g) and in reaching (g) searches pixels (q) and (l), leaving (k) and (f) unsearched. Because we know which search-pattern is associated with the last link (from Figure 29), we therefore start the neighbour search from pixel (k) and proceed through pixels (f), (a), (b) and (c) till we reach (h) which belongs to the next border pixel. Our neighbour code so far is hence 9, 10, 2, 2, 3. From pixel (h) we obtain the neighbour at (d), which is 4, and so we proceed until the complete code is found. For region 1, the neighbour-code is hence :
9 10 2 2 3 4 5 5 5 6 7 5 5 9.

An additional feature that may be incorporated into the neighbour-code is the recording of the occurrences of frame-edges. This allows us to see whether a region borders on the edge of the image and may be used to prevent the processing of such regions to compensate for discontinuity

effects on the border. We incorporate this data into the neighbour-code by entering a "-1" into the vector when a frame-edge is encountered. The discussion that follows will however not consider such a case.

To keep the neighbour-code length as short as possible, sequential occurrences of the same neighbour are ignored. The number of times a neighbour occurs in succession is however, entered into a vector to give an indication of its occupancy on the border. Our neighbour-code may thus be specified by the following vectors :

neighbour number :	1	2	3	4	5	6	7	8	9	10
neighbour-vector	9	10	2	3	4	5	6	7	5	9
occurrence-vector	1	1	2	1	1	3	1	1	2	1

We see that region 2 occurs twice, region 5 three times at the first occurrence and twice at the second etc.

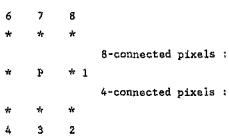
Because the neighbour-code is cyclic, the first and the tenth neighbour are in fact next to each other and since they are both represented by the same label, label 9, it is only necessary to record one occurrence. The tenth neighbour is deleted from the list and the occurrence vector for the first neighbour is incremented accordingly.

We thus have the following cyclic neighbour-code representation :

neighbour number	1	2	3	4	5	6	7	8	9
neighbour-vector	9	10	2	3	4	5	6	7	5
occurrence-vector	2	1	2	1	1	3	1	1	2

Additional useful information may be added to aid the merging process. It is possible to generate a vector that indicates whether a neighbour

is connected to the border in a 4- or 8-connected way. This only applies to neighbours that have *only single pixel occurrences* on the border since a multiple pixel occurrence will always be 4-connected. we again consider the 8-primary directions according to which a pixel may join to another pixel :



Directions 1, 3, 5 and 7 indicate 4-connected links, while 2, 4, 6 and 8 are links found only in 8-connectedness. A region will be joined in 8-connectedness only if the pixel was linked by an 8-connected link and if the next pixel in the sequential search was not on the border of the region whose neighbours *are being found. A region* that is linked in an 8-connected sense doesn't have a strong representation on the border allowing it to be ignored by the merging procedures of the particle recognition system. In our example the only 8-connected region is region 6 at pixel (t).

To facilitate the rapid location of a particular neighbour on the border of a region without starting the neighbour search afresh, a fourth vector is added to the neighbour description. This vector contains the number of the link in the chain-code where the neighbour is found for the first time. (A different number is found for each multiple occurrence).The geometric merge-routine discussed in Chapter 5 makes use of this information *to locate neighbours efficiently.*

The chain-code for our example is as follows :

link number : 1 2 3 4 5

link code : 1 1 3 5 6

It should be noted that the start-point corresponds to link 3 since the last link in the chain-code *always points* to the start-point.

The full neighbour-description incorporating the four description vectors is then as follows :

neighbour number	1	2	3	4	5	6	7	8	9
neighbour-vector	9	10	2	3	4	5	6	7	5
occurrence-vector	2	1	2	1	1	3	1	1	2
connectedness-vector	4	4	4	4	4	4	8	4	4
position-vector	4	5	5	5	1	2	3	3	3

We see that region 5 occurs twice in the neighbour-code. If we are not concerned with the accurate cyclic neighbour occurrence, it is possible to *pack* the neighbour-code by deleting say, the second occurrence of region 5. This results in a shortened neighbour list which only gives an idea of which neighbours occur and makes the *processing associated* with merging quicker.

APPENDIX C. THE MERGE-PROCESS

The merge procedures discussed in Chapter 4 all make use of the merging process that is discussed in this appendix. It is shown how *merges are* recorded using a series of vectors and how iteration may be achieved using these vectors.

The merge-process that is outlined here, is independent of the rules that *affect merging. Given a* list of regions that need to be merged to some other region, each region in the list will be merged to this region using the procedure described below. The *merge-process will* be explained using an example. Consider the group of regions shown in Figure 30 (a).
Assume that we need to merge region 1 to its neighbours 3, 4, 2 and 7.

We define a vector, called the merge-log-vector which is indexed by the label of a region and initially contains the label of the region. For this picture we would thus have :

Index (region label) :	1	2	3	4	5	6	7
Merge-log-vector :	1	2	3	4	5	6	7

We also specify a second vector, the merge-vector, which indicates whether a region has been merged or not. Initially this vector has only "0" entries. When a region is merged, its corresponding entry changes to "1". For our example the two vectors that have been defined so far are as follows :

Index (Region label) :	1	2	3	4	5	6	7
Merge-log-vector :	1	2	3	4	5	6	7
Merge-vector :	0	0	0	0	0	0	0

A third vector, called the sequence-vector, is defined which is used for iteration purposes. Its function is to keep track of which regions actually exist on the picture. Initially, this vector will have all 7 regions as entries :

Index :	1	2	3	4	5	6	7
Sequence-vector :	1	2	3	4	5	6	7

To make it possible to find the position of a region in the sequence-vector, it is required to keep the pointer to that position in another vector, called the pointer-vector. Initially this vector will contain the following data :

Index (Region label) :	1	2	3	4	5	6	7
Pointer-vector :	1	2	3	4	5	6	7

We now start merging by considering region 1 and region 3, say. Before we merge, we must establish which region has the start-point of the region that will result. Clearly in this case, region 3 will have the same start-point as the combination of region 1 merged to region 3. Region 3 is hence known as the **merge-master** The resulting merged region will then be labelled with label 3 instead of label 1.

After this merge the merge vectors become :

Index (region label) :	1	2	3	4	5	6	7
Merge-log-vector :	3	2	3	4	5	6	7
Merge-vector :	1	0	1	0	0	0	0

Note: The image itself is not re-labelled.

When we now consider the merging of region 2 to region 1, we see that we cannot compare the start-point of 2 to that of 1 since 1 no longer belongs to itself but belongs to 3 which has a superior start-point. We hence consider the start-point of the region that corresponds to the merge-log-vector entry referenced by region 1. Since this entry is 3 and region 3 has a better start-point than 2, the merge-master remains 3 and the merge vectors become :

Index (region label) :	1	2	3	4	5	6	7
Merge-log-vector :	3	3	3	4	5	6	7
Merge-vector :	1	1	1	0	0	0	0

The next merge we consider is that between region 1 and region 4. We check the merge-log entry corresponding to region 1 and find region 3. We see that region 4 has a dominant start-point in this case and hence is the new merge-master, This means that the merges where 3 was the the merge-master must now be replaced by 4. We achieve this by checking the merge-vector referenced by label 3 and seeing that a "1" appears indicating that region 3 belongs to a merged system. We then simply check the complete merge-log-vector and replace all entries of "3" with "4" hence noting the merge.

The merge vectors then become :

Index (region label) :	1	2	3	4	5	6	7
Merge-log-vector :	4	4	4	4	5	6	7
Merge-vector :	1	1	1	1	0	0	0

We then merge region 7 using the same method and obtain :

Index (region label) :	1	2	3	4	5	6	7
Merge-log-vector :	4	4	4	4	5	6	7
Merge-vector :	1	1	1	1	0	0	1

Note: *In general, when we consider two regions for merging, we always use the start-points of the region's* <u>indirect</u> *identity found in the merge-log-vector.*

We merge using the indirect identity as was shown when we merged region 4 to region 1, but actually manipulated the label "3" because it was the merge-master of the region group to which region 1 belonged. We may thus merge two regions and find that they both belong to merged groups and hence must replace the merge-master of the one group with the label of the merge-master of the other group if the second group has the more dominant master

We now assume that the group of regions that has been merged in our example so far correspond to one iteration.

We remove all the entries in the sequence-vector that correspond to regions no longer having their own label in the merge-log-vector. We do this by checking the merge-log-vector and when we find a case where the index

doesn't correspond with the entry, we access the corresponding entry in the pointer-vector, retrieve the pointer to the sequence-vector and delete the region in question from the sequence-vector by setting the entry to "0". After all the entries in the merge-log-vector have been checked, the "0" entries are removed from the sequence-vector and the sequence-vector is packed. The pointer-vector is then updated so that the remaining regions have the correct index to the sequence-vector. Pointer-vector entries of regions that no longer exist are set to "0".

The pointer-vector and sequence-vector for our example become the following :

Index :	1 2 3
Sequence-vector :	4 5 6

Index (region label) :	1 2 3 4 5 6 7
Pointer-vector :	0 0 0 1 2 3 0

The actual image is relabelled by indexing the merge-log-vector with every label on the image and substituting the pixels with the new region identities. Figure 30 (b) shows the new image that has been created. It is seen that the sequence-vector contains only the regions that now exist and may thus be used to investigate merges for a successive iteration.

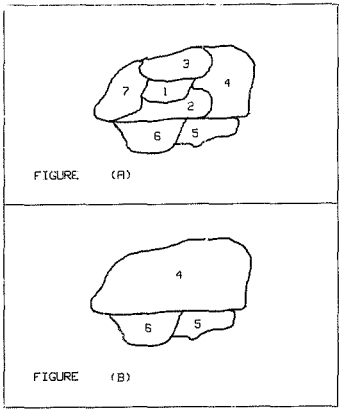


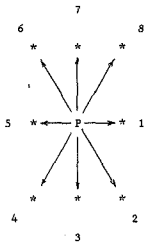
Figure 30. Merge example: (a) shows a labelled region-cluster - the background has not been labelled in this example. (b) is the region configuration after the first iteration.

APPENDIX D. MASKS USED FOR GEOMETRIC MERGING

In Chapter 4 section 4.5 the geometric merge-procedure was described in broad outline. Details of the *interception* and detachment masks are described in this appendix. An example is shown of how the masks are applied to an image containing regions of more regular geometry than those encountered in rock-pictures.

The concepts described in connection with chain-codes are used again in dealing with the creation of masks used for geometric merging. Masks are *established using* "difference" chain-codes rather than normal codes to enable masks to be created relative to the link in the border chain-code at which the neighbour is intercepted. *Different masks are* used depending on the link that was used to point at the border-pixel where either the neighbour was found for the first time (ie. the case of interception) or the neighbour was last encountered (ie. the case of detachment).

We recall from Appendix B the convention regarding chain-codes :



If the last link was 1, 3, 5,or 7 one set of masks must be used and for a link of 2, 4, 6 or 8 a second set is used. The reason for this will become clearer later on.

We now consider a last link that points to a neighbour interception pixel as is shown below. We assume the region under consideration has label 1 and the two neighbours have labels 2 and 3 respectively. The case of intercepting region 3 is considered :

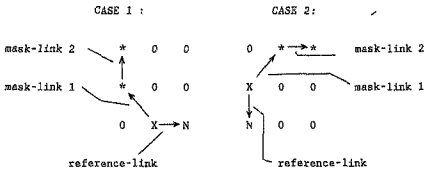
a	b	c	d	e
2	2	2	3	3
f	g	h	i	j
2	2	3	3	3
k	l	m	n	o
1	1→	1	1	3
p	q	r	s	t
1	1	1	1	1

Pixel (m) corresponds to the pixel at which region 3 is first encountered. The arrow indicates the direction of the last link in the border chain ie. direction 1. Relative to this link we may now establish a mask to check that this interception is valid, ie. the pixel sequence (k), (l), (h), (d) should represent a smooth line. The definition of "smooth" in this instance is that the angle between the line (k), (l), (m), (n) and the line (k), (l), (h), (d) should not be greater than 45 degrees. The mask is established by starting from pixel (l) and defining certain pixels relative to the link joining (l) and (m).

Pixels are defined by adding an offset to the link in question to establish a new link which points to the first pixel in the mask. If we add 5 we get the link-direction 1+5 = 6 which when taken as referenced from

pixel (1) establishes a new link pointing at pixel (f). At pixel (f) we establish the condition that region 3 may not exist there for an interception to be valid. This condition is satisfied and we check the next pixel by adding 1 to the previous link which yields the link-direction 7. Taking this direction referenced from (f) we reach pixel (a) which is again checked to see if region 3 exists there. If it doesn't, the next pixel in the mask is defined and so on until the complete mask has been searched.

A "difference" code may be established to define mask-pixels relative to a chain-code link. In the example above the difference-code that is necessary to check pixels at (f) and (a) is : 5 1 . This difference-code indicates the offset that should be added to a link pointing at a certain pixel to find the next pixel in the mask. The mask used in the example is repeated below together with the same mask relative to another initial reference link pointing in another direction :



difference code to establish mask pixels = 5 1 .

ie. mask-link 1 = reference-link + 5

mask-link 2 = mask-link 1 + 1

key : X = mask-reference pixel

N = pixel to which reference-link points

* = pixels constituting the mask

0 = "don't care" pixels

We see that in case 2 the mask has been rotated relative to that in case 1 but has remained in the same position relative to the reference-link by using the same difference-code (5 1) in each case.

It is seen that for the directions 1, 3, 5, and 7 the same mask will be found relative to the reference link using this method. Different masks are required for directions 2, 4, 6 and 8 since the 45 degree rotation changes horizontal and vertical lines of pixels in the mask to diagonal lines which need to be thickened in order for the test conditions to remain consistent.

The masks used for the geometrical merge routine a[illegible] below :

INTERCEPTION MASKS :

Convention :

0 = pixels not checked

X = pixels where the region must exist

* = pixels where the neighbour cannot exist

F = pixel where the neighbour is initially encountered

⟶ = reference-link

Case 1 : reference links of 1, 3, 5, 7, :

```
0  0  0  *  *
*  *  *  *  0
0  *  0  0  0
0  0  X⟶F  0
0  X  X  0  0
```

Case 2 : reference links of 2, 4, 6, 8 :

```
*  *  0  0  0
0  *  *  0  0
X  0  *  *  *
0  X  X  0  0
0  0  X  F  0
```

DETACHMENT MASKS :

Convention : N = *pixel* where *next* neighbour region is found for the first time

Case 1 : reference-links of 1, 3, 5, 7 :

```
*  *  0  0
0  *  *  0
0  0  *  *
X  X  X→N
```

Case 2 : reference-links of 2, 4, 6, 8 :

```
0  0  0  *
X  0  0  *
0  X  0  *
0  0  X  0
          ↘
0  0  0  N
```

These masks were derived by careful consideration of the methods used in *chain-code generation* and neighbour finding to reduce the number of pixels that need to be checked. Merging was tested on geometrically regular shaped regions to ensure that all the masks worked according to specification. Some of the geometries tested are shown in Figure 31.

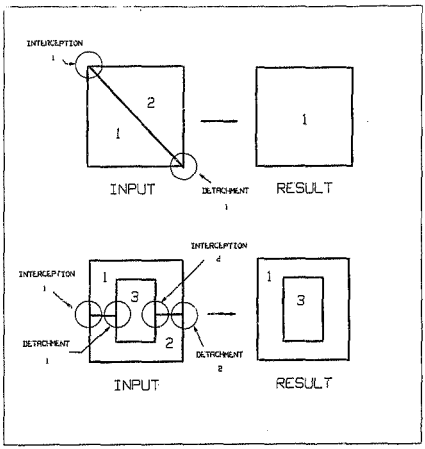


Figure 31. Merges performed to test the geometrical masks: (a) shows a situation where the diagonal masks could be tested and (b) where the orthogonal masks were checked. In (b) the situation is shown where four tests are needed since region 1 occurs twice on the boundary of region 2 (and vice versa).

APPENDIX E. FURTHER DETAILS ON REGION-SPLITTING

Further details regarding the region-splitting method discussed in Chapter 5 are presented in this appendix. Region re-labelling as well as recording of scan-lines to enable efficient iteration of the un-merging process are explained more fully.

E.1 REGION RE-LABELLING

When a region is split into two sub-regions during the un-merge process, it is necessary to label one sub-region with a new integer label to enable recording of its area as well as facilitate subsequent further splitting of that sub-region. Use of three split-chords to achieve this was briefly discussed in Chapter 5. The re-labelling process using these chords is now dealt with more fully showing details of the label propagation mechanism.

The two split-chords that are placed across the "bottleneck" in the region consist of lines of pixels having the label "0" to distinguish them from any region labels that exist on the image (region labels are always greater than zero). This forms a barrier to prevent a re-labelling wave-front from crossing to the sub-region that retains the old region label. When the first chord is placed, a check is made to see whether the chord itself is not placed in a position where if re-labelling were attempted, there would not be sufficient room to propagate labels. This is done by placing an imaginary "line" across the centre and at right angles to the chord and checking whether pixels belonging to the region that is to be split exist on both sides of the chord along this line. If this check succeeds, the second split-chord is placed next to the first to complete the "barrier". The third chord is not marked using "0" pixel labels and is only used to check that a free chord exists alongside the first and

on the side from which the re-labelling wave-front is to propagate. The diagram below illustrates the situation discussed above by means of an example :

a	b	c	d	e	f
1	1	1	1	1	1
g	h	i	j	k	l
2	1	1	1	1	2
m	n	o	p	q	r
2	2	0	0	2	2
s	t	u	v	w	x
2	2	0	0	2	2
y	z	A	B	C	D
2	2	0	0	2	2
E	F	G	H	I	J
2	3	3	3	2	2
K	L	M	N	O	P
3	3	3	3	3	3

Region 2 is the region that is to be split. The two split-chords are defined by

chord 1 : pixels o u A

chord 2 : pixels p v B

The check-line would exist at s t u v w x and since region 2 initially occupies all these pixels, the check passes. The third split-chord then exists at pixels n t and z and is the line from which the propagation wave is started. The above case of course, applies to a vertical split across

region 2. A re-labelling wave-front may be started travelling leftwards from the third *split-chord* but it was found that an alternative strategy produced quicker re-labelling especially if the neck of the "bottleneck" was very narrow. The idea is to extend a line of new pixel labels from against the *first split-chord* leftwards towards the first region boundary and then to propagate new labels upwards and downwards. This method is favoured since the initial wave-front of new labels is usually larger and re-labelling is hence faster. After extension of the first re-labelled line the situation shown previously becomes the following :

```
1 1 1 1 1 1

2 2 1 1 1 2

2 2 0 0 2 2

4 4 0 0 2 2

2 2 0 0 2 2

2 2 3 3 3 2

3 3 3 3 3 3
```

We have used label 4 as being the label *of the new* sub-region of the original region 2. The wave-front of label 4 is now propagated upwards using a mask defined as follows :

```
M M M
M X M
```

"x" is a pixel in the row of new labels that was previously established. The pixels in a re-labelled row are investigated one by one using the mask

shown above. Their neighbouring pixels that coincide with "M" entries in the mask are replaced with the new label if they have the label of the old region. The structure of the mask also enables the re-labelling wave-front to propagate to the left and right hence allowing re-labelling of regions that have more complex geometry than the case of our example. After the mask has been used by every pixel in a line, the line that is investigated is moved upwards and the process repeated on the pixels have just previously been re-labelled. When the mask reaches the top of the region or no more re-labelling occurs, a similar mask is used and re-labelling is done in the downward direction.
The mask is shown below :

Re-labelling continues downwards until the bottom of the region is reached or until no more re-labelling is performed. The process of upward and downward labelling proceeds until a labelling scan in a certain direction yields no more substitutions.

To complete the re-labelling, the initial two chords of "0" label pixels are also re-labelled with the new label. After re-labelling the example is then as is shown below :

1 1 1 1 1 1

4 4 1 1 1 2

4 4 4 4 2 2

4 4 4 4 2 2

4 4 4 4 2 2

4 4 3 3 3 2

3 3 3 3 3 3

A new start-point is calculated for the old region and for the new region and the new areas are also found. *In our example* the area of r 2 changes from 24 to 8 and region 4 gets assigned an area of 16. *these regions may now* once again be considered for merging by co the horizontal scanning and then using the remaining scans ie. horizontal and the two diagonal scans.

The direction used for label propagation depend on which region-scan is used. Figure 32 shows the direction associated with each scan.

This concludes the discussion on region re-labelling. The next section explains how region-scanning may be optimised by knowing which scan-lines have already been used.

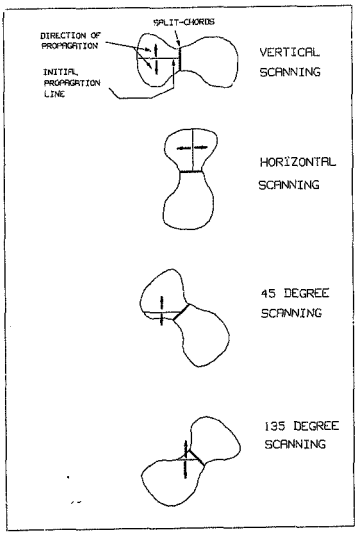


Figure 32. Re-labelling directions.

E.2 OPTIMISING REGION SCANNING

Sub-regions that are formed when a region is split must themselves be investigated to see if they need to be divided. It is now shown that it is not always necessary to apply scanning in all directions to each sub-region again.

To illustrate the concepts that will be developed more clearly, we consider an example of a region that is to be split. Figure 33 (case A) shows a region that is to be split in the vertical direction. Scanning proceeds in an ordered fashion from left to right and at scan-line x we decide to split region A to form regions A and B. If we subsequently investigate the splitting of region B we see that we don't need to search B in the vertical direction since this was done previously when scanning region A for the first time. Hence we proceed directly to the next scan. Thus by recording the scan-line at which the split was achieved, we can see whether we require further searching using a certain scan and if we do, where to start searching. The situation of having to continue a scan is shown in case B in Figure 33 where scanning of region B commences at scan-line X1 and another split is found at scan-line X2. Only the scan-lines between scan-line X1 and scan-line X2 are searched instead of having to search the complete region B from the beginning. Hence for any region it holds in general that scanning resumes at the last scan-line of the scan-direction used to form the split from which the region was derived. If the scan-line happens to coincide with the furthest extremity of the region in the search-direction, the next scan-direction is chosen starting with the first scan-line that penetrates the region.

The recording of scan-lines is simple in the cases of vertical and horizontal scanning since we only record the X or Y scan-line respectively at which the previous split occurred. The mechanism of recording a diagonal scan-line is more complicated because no direct index exists with which to uniquely define a diagonal scan-line in the orthogonal co-ordinate system of the image plane. The recording of diagonal scan-lines will hence be discussed below :

Consider the situation shown in Figure 34. In Case 1 we see the situation shown for a + 45 degree scan. The region is contained in its circumscribing rectangle which defines the square in which scan-lines are searched. The rectangle is defined by the X and Y limits Y_top, Y_bottom, X_right and X_left. During scanning, scan-lines are referenced depending on whether they intersect the lefthand side or the lower side of the rectangle. Hence scan-line 1 in Case 1 will be referenced by the co-ordinates (X_left,Y1) and scan-line 2 by (X2,Y_bottom). Since we know the direction of scanning, this information is all we need to construct scan-lines within the reference frame of the circumscribing rectangle. Scan-lines are thus started from from the top of the lefthand side of the rectangle proceeding downwards. When the lower side is reached, scan-lines resume along the bottom side rightwards until the rightmost side is reached. Because the circumscribing rectangle changes after splitting into sub-regions (each sub-region gets a new rectangle), the scan-lines that were used during splitting need to be referenced according to their absolute position in the image co-ordinate system. This is achieved by calculating the intersection of the scan-lines with the leftmost side of the image ie. the Y-axis of the image. From Case 1 in Figure 33 we see that scan-line 1 is hence indexed by the intersection at Y_index1 and scan-line 2 at Y_index2. The Y intersections are the only values that need to be recorded to enable a 45 degree scan-line to be reconstructed and referenced back to the new circumscribing rectangle of a sub-region.

Case 2 in the figure shows the similar situation associated with 135 degree scanning relative to the positive X-axis. In Case 1 the index value on the Y-axis will always be positive but in Case 2 it could be positive or negative. Notice that in 135 degree scanning the reference axes on the circumscribing rectangle are different to allow continuous scan-line construction starting at the leftmost limit of the lower side working rightwards towards the right-hand side and then up the right-hand side towards the upper side of the rectangle.

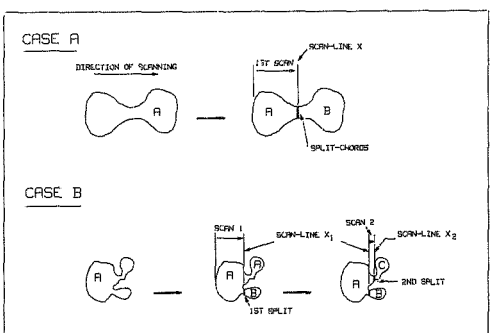


Figure 33. Optimising region scanning: The details regarding this figure are explained in the text.

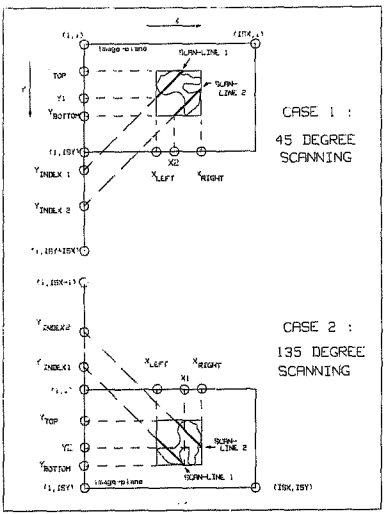


Figure 34. Recording of diagonal scan-lines.: Case A is for +45 scanning and case B for +135 scanning. Details are discussed in the text.

APPENDIX F. HIGH-LEVEL DESCRIPTION OF THE ROCK-RECOGNITION SOFTWARE

A PDL description of the software described in this dissertation is presented in this appendix. The objective is to provide a high-level description of each module and to illustrate how the modules tie together. Where required, sufficient detail is presented to make the specific functional behaviour of a routine clear to enable continuation of certain concepts that were introduced in the main chapters and appendixes of the dissertation. The actual code used to implement the system is available as a full FORTRAN-77 listing from the Department of Electrical Engineering, University of the Witwatersrand.

F.1 MAIN PROGRAM

NAME : Program FPZ5

PURPOSE :

This is the main program segment used to unfile the input image from disk and call the various routines that perform the different stages in the recognition process. The final size-distribution is calculated at the end of the main program and a result image filed on disk.

INPUT :

1. Filename defining the image to be processed.

OUTPUT :

1. Particle size distribution in four ranges.
2. File on disk showing the interpretation of the input image using highlighted regions

CALLS :

1. Image Labelling Sub-Program
2. Gray-Level Merge Subroutine
3. Region-Cluster Merge Sub-Program
4. Geometric Merge Sub-Program
5. Un-Merge Sub-Program : Horizontal And Vertical

6. Un-Merge Sub-Program : Diagonal

DESCRIPTION :

BEGIN DATA DEFINITION :

Image : *Image matrix of size* (ISX,ISY)

ISX : Number of columns in the image

ISY : Number of rows in the image

* Region description vectors : *

area_vector : Vector containing region areas

graylevel_vector : Vector containing region gray-levels

startpoint_vector : Vector containing region start-points

* merge vectors : *

merge_vector : Indicates merged regions

merge_log_vector : Contains merge substitution labels

sequence_vector : Vector containing region labels that exist on the image

pointer_vector : A vector indicating the position of a region in the sequence_vector

* appendix C explains the use of these vectors for merging regions *

END DATA DEFINITION

```
BEGIN PROCEDURE :

    GET (From disk) :
        Input Image  * 64 gray-level rock picture *
    END GET

    DO :
        * Construct the image histogram *
    END DO

    DO :
        * Compute the extent of the dark-band for the 5-level
          segmentation method *

        * Calculate the cumulative distribution for the remainder
          of the histogram  *

        * Compute the remaining 4 histogram bands (particle bands)
          for the 5-level method *
    END DO

    DO :
        * Substitute image pixels for their new 5-band values *

        * The image is now reduced to 5 gray-levels.
          The bands are represented by the following pixel values :
                Band 1 =  0          Band 4 = -32
                Band 2 = -16         Band 5 = -48
                Band 3 = -20
          The image now consists of the 5 integer values above. *
    END DO

    * we now label the 5-band image to form identified regions
      and find the initial region description vectors : *

    CALL ( Image Labelling Sub-Program ) :
```

```
     INPUTS  :
               1. 5-level image
               2. Connectedness to be used (4 or 8)

     OUTPUTS :
               1. Number_of_regions
               2. Region description vectors :
                     area_vector
                     graylevel_vector
                     startpoint_vector
               3. Labelled image
END CALL

DO :
     * Initialise merge vectors : *
               merge_vector
               merge_log_vector
               sequence_vector
               pointer_vector
     * see appendix C for details *
END DO

* Now merge according to gray-level : *

CALL (Gray-Level Merge Subroutine) :
     INPUTS  :
               1. Labelled image
               2. Region description vectors
               3. Merge vectors
               4. Number_of_regions
               5. Maximum number of iterations to be
                  performed
     OUTPUTS :
               1. New labelled image
               2. New region description vectors
               3. New merge vectors
```

```
                4. new number_of_regions
END CALL

* Now merge to remove region clusters : *

CALL (Region-Cluster Merge Sub-program) :
     INPUTS  :
                1. Labelled image
                2. Region description vectors
                3. Merge vectors
                4. Number_of_regions
                5. Maximum size allowed for cluster regions
                6. Maximum number of regions in a cluster
                7. Minimum size allowed for a region that
                   contains a cluster
     OUTPUTS :
                1. New labelled image
                2. New region description vectors
                3. New merge vectors
                4. New number_of_regions
END CALL :

* Now merge using geometry : *

CALL (Geometric Merge Sub-Program) :
     INPUTS  :
                1. Labelled image
                2. Region description vectors
                3. Merge vectors
                4. Number_of_regions
                5. Minimum size of regions considered
                6. Minimum size of neighbours
                7. Number of merge iterations
     OUTPUTS :
                1. New labelled image
                2. New region description vectors
```

```
                3. New merge vectors
                4. New number_of_regions
END CALL

* Un-merging begins : *

DO :
     * Initialise the iteration vector : *

   * iteration_vector(1,-) = region_id                              *
   * iteration_vector(2,-) = last scan line (vertical search)      *
   * iteration_vector(3,-) = last scan line (horizontal search)    *
   * iteration_vector(4,-) = last scan line index ( +45 search)    *
   * iteration_vector(5,-) = last scan line index ( +135 search) *

   index=1
     REPEAT UNTIL (index > 200):
          iteration_vector(1,index) = 0
          iteration_vector(2,index) = 1
          iteration_vector(3,index) = 1
          iteration_vector(4,index) = 1000
          iteration_vector(5,index) = 1000
          index=index+1
     END REPEAT
END DO

DO :
     * Enter rock-region id's that have area larger than
       the limit for un-merging into iteration_vector(1,-) *

     iteration_vector_size = * number of regions in
                               iteration_vector*
END DO

REPEAT UNTIL (No more regions in the iteration_vector) :
```

```
* fetch a region label from iteration_vector(1,-) *

CALL (Un-Merge Sub-Program : Horizontal and Vertical) :
     INPUTS  :
               1. Label of region to be split
               2. Region description vectors
               3. Chord-length factor * see App. E *
               4. Last scan-line (horizontal)
               5. Last scan-line (vertical)
               6. Number_of_regions
               7. Labelled image

     OUTPUTS :
               1. New labelled image
               2. New number_of_regions
               3. Label of new sub_region
               4. Last scan-line (horizontal)
               5. Last scan-line (vertical)
               6. New region description vectors
               7. Circumscribing rectangle co-ordinates
                  * for use by diagonal un-merge program *
END CALL

IF (Region was not split) THEN :

     CALL (Un-Merge Sub-Program : Diagonal splitting) :
          INPUTS  :
                    1. Region label
                    2. Region description vectors
                    3. Chord length factor
                    4. Last scan-line index (+45 )
                    5. Last scan-line index (+135 )
                    6. Number_of_regions
                    7. Labelled image
                    8. Circumscribing rectangle
                       co-ordinates
```

```
                OUTPUTS :
                        1. New labelled image
                        2. New number_of_regions
                        3. Label of new sub-region
                        4. Last scan-line index (+45 )
                        5. Last scan-line index (+135 )
                        6. New region description vectors
            END CALL
        END IF

        IF (region was split) THEN :
            * remove sub-region labels from iteration_vector(1,-)
              if their areas are less than the threshold for
              un-merging *

            * add sub-region labels to iteration_vector(1,-) if
              their areas are above the threshold :
                also add the last scan-lines into
                iteration_vector(2,-),..,iteration_vector(5,-).
        ELSE :

            * remove region label from iteration-vector(1,-) *
        END IF

        DO :
            * pack iteration_vector to delete "0" entries *
        END DO

END REPEAT

DO :
    * create an image showing the regions corresponding to
      particles by substituting region labels on the image
      for the region gray-levels.
                rock regions : gray-level > 15
                background   : gray-level < 16 ;
```

```
        Outline each region with a white border  *
    END DO

    PUT (to disk) :
                * file the region image *
    END PUT

    DO :
        * compute size distribution in 4 ranges using the
          area_vector *

        * distribution is as follows :
              large rocks   : area > 4095 pixels
              big rocks     : area > 1023 pixels
              medium rocks  : area > 255  pixels
              small rocks   : area > 0    pixels  *
    END DO

    PUT (to printer) :
                * size distribution *
    END PUT

END PROCEDURE
```

F.2 IMAGE LABELLING SUB-PROGRAM

F.2.1 MAIN ROUTINE

NAME : PROGRAM PXGL

PURPOSE :

Labels regions consisting of pixels with the same gray-level with unique integers. The input image consists of negative integers to enable labelling on the input image matrix without the need for additional workspace. During labelling, region description vectors (area, gray-level and start-points) are constructed.

METHOD : As discussed in chapter 3

CALLS :

1. RELABEL subroutine
2. QUEUE subroutine

CALLED BY :

1. MAIN PROGRAM

INPUTS :

1. Gray-level image consisting of negative integers
2. Connectedness : 4 or 8

OUTPUTS :

1. Labelled image

2. Number of regions

3. Description vectors :
   area_vector
   graylevel_vector
   startpoint_vector

DESCRIPTION :

BEGIN DATA DEFINITION :

pixel_mask_vectors : define which neighbours of a pixel already have labels and which have not yet been labelled. Different mask-vectors are used depending on the connectedness specified and on the position of the pixel in the image. Different masks are used for pixels in the first row, first column, last column etc.

* region description vectors : *

area_vector : vector of region areas
graylevel_vector : vector of region gray-level's
startpoint_vector : vector of region start-points

END DATA DEFINITION

BEGIN PROCEDURE :

DO :
* process first pixel at co-ordinates (1,1) *

```
        * assign label "1" to the first pixel *

        * initialise the description vectors : *
            area_vector(1) = 1
            startpoint_vector(1) = 1 ; 1
            greylevel_vector(1) = absolute value of
                                  the first pixels
                                  gray-value
END DO

REPEAT UNTIL (all pixels in row 1 are processed) :

    DEFINE PROCEDURE (PIXEL_LABELLING) :

        * fetch the appropriate pixel_mask_vector *

        * process the pixel_mask_vector neighbours
          in raster scan sequence : the first pixels
          will correspond to already labelled pixels
          followed by the un-labelled pixels      *

        REPEAT UNTIL (all neighbours in the mask are
                      processed or the pixel is
                      labelled) :

            IF (a labelled neighbour has the same
                gray-level as the pixel) THEN :

                area_vector(neighbour_label)=
                  area_vector(neighbour_label)+1
            END IF

            IF (an un-labelled neighbour has the
                same gray-level as the pixel) THEN :

                * get the next region-label -
```

```
        ( new_label )                    *

        area_vector(new_label)=1
        graylevel_vector(new_label)= absolute
                                     value of
                                     the pixel
                                     gray-level

        startpoint_vector = Co-ordinates of the
                            pixel

        * label the pixel with new_label *
    END IF

    IF (no neighbour match ie. single pixel
        region) THEN :

        * find the neighbour with the most
          similar gray-value                 *

        IF (similar neighbour is labelled) THEN :

            area_vector(neighbour_label) =
              aresvector(neighbour_label) + 1

            * label the pixel with the neighbour
              label  *
        END IF

        IF (similar neighbour is un-labelled) THEN :

            * get the next region label -
              (new_label)   *

            area_vector(new_label) = 1
            graylevel_vector(new_Label) = absolute
```

```
                                                    neighbour
                                                    gray-value
                    startpoint_vector = co-ordinates of
                                        pixel

                    * label pixel with new_label *
                END IF
            END IF

        END REPEAT

    END PROCEDURE DEFINITION (PIXEL_LABELLING)

END REPEAT

    REPEAT UNTIL (last row is reached) :

        DO :
            * process first pixel in each row *

            * fetch the appropriate pixel_mask_vector *

            * process the pixel_mask_vector in raster
              scan format *

            EXECUTE DEFINED PROCEDURE (PIXEL_LABELLING)

            DEFINE PROCEDURE (LABEL_CHECK) :

                * after labelling a pixel check whether
                  a labelled neighbour exists having a
                  different label but the same gray-value
                  ie. an ambiguous label  *

                IF (ambiguous label exists) THEN :
                  actual_label = * the smaller label *
```

```
                redundant_label = * the larger label *

                * now substitute redundant_label for
                  actual_label on the image : *

                CALL (Relabel Subroutine)
                  INPUTS :
                           1. startpoint_vector
                           2. actual_label
                           3. redundant_label
                           4. image being labelled
                  OUTPUTS :
                           1. image being labelled
                           2. startpoint_vector
                END CALL

                * area_vector(actual-label) =
                     area_vector(redundant_label)
                   +   area_vector(actual_label)
                  area_vector(redundant_label) = 0

                * place the redundant_label in
                  a queue for re-use  *

              END IF

        END PROCEDURE DEFINITION (LABEL_CHECK)

    END DO (first pixel in each row)

    REPEAT UNTIL (last pixel in row is reached) :

        * fetch appropriate pixel_mask_vector *

        * process mask in raster scan          *
```

```
            EXECUTE DEFINED PROCEDURE (PIXEL_LABELLING)

            EXECUTE DEFINED PROCEDURE (LABEL_CHECK)

        END REPEAT

        DO (process last pixel in each row) :

            * fetch appropriate pixel_mask_vector *

            * process in raster scan              *

            EXECUTE DEFINED PROCEDURE (PIXEL_LABELLING)

            EXECUTE DEFINED PROCEDURE (LABEL_CHECK)

        END DO

    END REPEAT (all except last row completed)

    DO (process first pixel in last row) :

        * fetch pixel_mask-vector   *

        * process in raster scan    *

        EXECUTE DEFINED PROCEDURE (PIXEL_LABELLING)

        EXECUTE DEFINED PROCEDURE (LABEL_CHECK)

    END DO

    REPEAT UNTIL (last pixel is reached) :

        * fetch pixel_mask_vector  *
```

```
            * process in raster scan   *

            EXECUTE DEFINED PROCEDURE (PIXEL_LABELLING)

            EXECUTE DEFINED PROCEDURE (LABEL_CHECK)

        END REPEAT (only last pixel left)

        DO (process last pixel) :

            * fetch pixel_mask_vector  *

            * process in raster scan   *

            EXECUTE DEFINED PROCEDURE (PIXEL_LABELLING)

            EXECUTE DEFINED PROCEDURE (LABEL_CHECK)

        END DO

END PROCEDURE
```

F.2.2 RELABEL SUBROUTINE

NAME : Subroutine Relabel

PURPOSE :

Relabels redundant regions during image labelling. Modifies the startpoint_vector accordingly.

METHOD :

Starts in the row containing the startpoint of a region and substitutes the pixel values with a new label value using a raster scan that terminates when no substitutions occur within a certain row.

CALLED BY : Image Labelling Sub-Program

INPUTS :

1. startpoint-vector
2. actual_label
3. redundant_label
4. image being labelled

OUTPUTS :

1. image being re-labelled
2. startpoint_vector

DESCRIPTION :

BEGIN PROCEDURE :

```
        * find the start-point of the region defined by redundant_label *

        * set reference position to the start-point *

        REPEAT UNTIL (a complete row is processed without a
                      substitution) :

            IF(pixel = redundant_label) THEN :

                pixel = actual_label

            END IF

        END REPEAT

        * determine whether actual_label or redundant_label has the
          dominant start-point *

        startpoint_vector(actual_label) = co_ordinates of the dominant
                                          start-point

END PROCEDURE :
```

F.3 GRAY-LEVEL MERGE SUBROUTINE

NAME : Subroutine Main_merger

PURPOSE :

This subroutine performs gray-level merging of regions

INPUTS

1. Labelled image
2. Region description vectors
3. Merge vectors
4. Number of regions
5. Number of merge iterations
6. Gray-level defining the start of the first rock band (normally = 16)

OUTPUTS :

1. New labelled image
2. New region description vectors
3. New merge vectors
4. New number of regions

CALLED BY: Main Program

CALLS :

1. Subroutine Chain/Neighbour

2. Subroutine Merge_process

3. Subroutine Resort

DESCRIPTION :

```
BEGIN DATA DEFINITION :

   * region description vectors : *

        area_vector       : vector containing region areas
        graylevel_vector  : vector containing region gray-levels
        startpoint_vector : vector containing region start-points

    * merge vectors : *

        merge_vector      : indicates merged regions
        merge_log_vector  : contains merge substitution labels
        sequence_vector   : contains the regions that exist
                            on the image
        pointer_vector    : Indicates the position of each
                            region found during segmentation
                            in the sequence_vector

END DATA DEFINITION

BEGIN PROCEDURE :

   REPEAT UNTIL (specified number of iterations is reached
                 or merging terminates) :

        REPEAT UNTIL (all regions in the sequence_vector
                      are processed) :
```

```
DO :

     * find the next region to process from the
       sequence_vector  *

     * find the start-points of this region
       from the startpoint_vector   *

END DO

* now find the neighbours :  *

CALL (Chain/Neighbour Code Subroutine) :

     INPUTS :
               1. Labelled image
               2. Label of the region being processed
               3. Startpoint_vector
               4. Chain-code/neighbour-code option
                  (select neighbour-code calculation)
     OUTPUTS :
               1. Neighbour description vectors
                  ( refer to subroutine description )
               2. Number of neighbours in the code

END CALL

DO :
     * classify the region in terms of its 5-band
       membership *

     CASE OF :

          1.  region gray-level = background  :
              classify region as "background"
```

```
            2.  region gray-level = bands 2 to 5  :
                classify region as "rock"

        END CASE

END DO

DO :
     * find the 5-band membership of each neighbour
       using the greylevel_vector *

     * find the percentage of neighbours that are
       rocks and the percentage that are background
       ( edge labels are assumed to belong to the
        background) *

END DO

DO  (apply the merge rules) :

     CASE OF :

          1.  region is background  :

              IF (all neighbours are rocks) THEN :
                IF (there is only 1 neighbour) THEN :
                  merge_type = 2 * hole inside a
                               rock region *
                ELSE :
                  merge_type = 7 * no merge *

                END IF
              ELSE :
                merge_.ype = 7 * no merge *

              END IF
```

```
                END IF

            2.  region is rock  :

                IF (all the neighbours are rocks) THEN :

                  merge_type = 6 * rock surrounded by only
                                   rocks *

                ELSE :

                  merge_type = 7 * no merge *

                END IF

        END CASE

    END DO

    IF (merge_type = 2 or merge_type = 6) THEN :

        * merge the region to its neighbours : *

        DO :
            * find the indirect identity of the region
              from the merge_log_vector *

            merge_master = indirect region

            merge_master_startpoint = start-point of
                                      indirect region

        END DO

        * remove all edge labels from the neighbour string *
```

```
REPEAT UNTIL (all neighbours in the neighbour-string
              have been processed) :

     * fetch a neighbour from the string *

     * find its indirect identity from the
       merge_log_vector *

     * find the start-point of the indirect neighbour
       from the startpoint_vector *

     IF (the start-point of the indirect neighbour
         is more dominant than that of the
         merge_master) THEN :

       merge_master = id of indirect neighbour

       merge_master_startpoint = start-point of
                                 indirect neighbour

     END IF

END REPEAT

REPEAT UNTIL (all neighbours in the neighbour list
              are processed) :

     * fetch the next neighbour *

     merge_region = neighbour id

     IF (merge_region is not the merge_master and its
         indirect identity is not the merge_master) THEN :

          CALL (Merge_Process Subroutine) :
```

```
                INPUTS :
                        1. merge_region
                        2. merge_master
                        3. merge_type
                        4. merge vectors
                        5. region description vectors

                OUTPUTS :
                        1. new merge vectors
                        2. new description vectors

            END CALL

        END IF

    END REPEAT

    DO (merge the region) :

        merge_region = region id

        IF (region isn't the merge_master and the indirect
            region isn't the merge_master) THEN :

            CALL (Merge_process Subroutine)
                INPUTS  : * as for the previous call *

                OUTPUTS : * as for the previous call

            END CALL

        END IF

    END DO

END REPEAT (all regions in sequence_vector have been processed)
```

```
        IF (merges were recorded) THEN :

            * compute new labelled image by substituting
              pixels with their indirect labels in the
              merge_log_vector *

            * set all merge_vector entries to zero *

            * re-order the sequence_vector *

            CALL (Resort Subroutine) :

                INPUTS  :
                        1. region description vectors
                        2. merge vectors
                        3. number of regions in sequence_vector

                OUTPUTS :
                        1. merge vectors
                        2. new number of regions in sequence_vector
                           ( = number_of_regions )

            END CALL

        END IF

    END REPEAT (main iteration loop)

END PROCEDURE
```

F.4 MERGE_PROCESS SUBROUTINE

NAME : Subroutine Merge_process

PURPOSE : Recording of merge occurrences in the merge vectors

INPUTS :

1. merge_region
2. merge_master
3. merge_type
4. merge vectors
5. region description vectors

OUTPUTS :

1. new merge vectors
2. new region description vectors

CALLED BY :

1. Gray-level Merge Subroutine
2. Region-cluster Merge Subroutine
3. Geom_tric Merge Subroutine

DESCRIPTION :

```
BEGIN DATA DEFINITION :

        * merge vectors ; *

            merge_vector          : indicates merged regions
            merge_log_vector      : indirect region id's
            sequence_vector       : regions existing on the image
            pointer_vector        : position of regions in sequence_vector

        * region description vectors : *

            area_vector       : region areas
            graylevel_vector  : region gray-levels
            startpoint_vector: region start-points

        * general : *

            merge_region      : id of region that is to be merged
                                to the merge_master

            merge_master      : region with the dominant start_point

            merge_type        : Rule used to establish the merge

END DATA DEFINITION

BEGIN PROCEDURE :

    IF (merge_vector(merge_region) = 1) THEN :

        * merge_region belongs to a merged group *

        merge_region is "multipla"

    ELSE
```

```
    * merge_region hasn't been previously merged *

    merge_region is "single"

END IF

CASE OF :

    1.  merge_type = 2 or 6  :

        graylevel_vector(merge_master) = 48
        * (resulting region is also a rock region) *

    2. * further cases may be added if more merge_types are
        defined *

END CASE

DO :
    area_vector(merge_master) = area_vector(merge_master)
                              +  area_vector(merge_region)

    area_vector(merge_region) = 0

    merge_vector(merge_master) = 1
    *  record that merge_master is part of a
      merged group) *

    IF(merge_region is single) THEN :

        merge_log_vector(merge_region) = merge_master

        merge_vector(merge_region) = 1

    ELSE
```

```
            * change all occurrences of merge_region in
              merge_log_vector to merge_master *

        END IF

    END DO

END PROCEDURE
```

F.5 RESORT SUBROUTINE

NAME : Subroutine Resort

PURPOSE :

This subroutine re-orders the sequence_vector after any merge iteration and deletes the occurrences of regions that have been merged from the sequence_vector.

INPUTS :

1. region description vectors
2. merge vectors
3. number of regions in sequence vector

OUTPUTS :

1. merge vectors
2. new number of regions in the sequence_vector

CALLED BY :

1. Gray-level Merge Subroutine
2. Cluster-region Merge Subroutine
3. Geometric Merge Sub-Program

DESCRIPTION :

BEGIN DATA DEFINITION :

```
        * merge vectors : *

            merge_vector          : indicates merged regions
            merge_log_vector      : indirect region id's
            sequence_vector       : regions existing on the image
            pointer_vector        : position of regions in sequence_vector

        * region description vectors : *

            area_vector           : region areas
            graylevel_vector      : region gray-levels
            startpoint_vector     : region start_points

END DATA DEFINITION

BEGIN PROCEDURE :

    REPEAT UNTIL (all entries in the sequence_vector have been
                  processed) :

        * fetch a region (region_id) from the sequence_vector   *

        IF (area_vector(region_id) = 0 OR
            pointer_vector(region_id) < = 0 AND
            merge_log_vector(region_id) <> region_id) THEN :

            * knock region_id out of sequence_vector *

        END IF

    END REPEAT

    DO :

        * pack sequence_vector *
```

```
        * find the new number of regions in the sequence_vector *

    END DO

END PROCEDURE
```

F.6 CHAIN/NEIGHBOUR CODE SUBROUTINE

NAME : Subroutine Chain

PURPOSE :

Generates a chain-code description of a region's border. A cyclic neighbour string is also generated. The neighbour string includes edge labels denoted by "-1". Vector nhbtype_vector classifies neighbours as being either 4 - or 8-connected to the border. Record of the first occurrence of a neighbour along the chain-code is kept in nhbpos_vector. The total number of pixels belonging to a neighbour (at each occurrence) is stored in nhbr_vector(2,-). nhbr_vector(1,-) contains the labels of the neighbour regions and chain_vector contains the border chain code.

METHOD :

As described in Appendix A and Chapter 4.

INPUTS :

1. Labelled image
2. label of region to be processed
3. startpoint_vector
4. chain-code/neighbour-code option : 1 = chain 2 = chain + neighbour

OUTPUTS :

1. *neighbour*-code vectors
2. chain-code vector

3. length of neighbour-code

4. length of chain-code

CALLED BY :

1. Grey-level Merge Subroutine

2. Region-cluster Merge Subroutine

3. Geometric Merge Subroutine

DESCRIPTION :

BEGIN DATA DEFINITION :

* neighbour-code vectors : *

nhbr_vector(1,-)	: neighbour labels
nhbr_vector(2,-)	: number of pixels per occurrence
nhbtype_vector	: connectedness of neighbours
nhbpos_vector	: *position of neighbours in the chain-code*

* chain-code vector : *

chain_vector	: vector of *chain-code links*

* general : *

search_start_vector	: data-vector containing the pixel that needs to be searched next corresponding to the previous link in the chain-code
nhb_counter	: counter to accumulate the

```
                                        number of neighb_ rs

        code_counter                    : counts links in the chain-code

END DATA DEFINITION

BEGIN PROCEDURE :

    DO :

        * find the region's startpoint from startpoint_vector *

        * initialise all neighbour and chain vectors to zero  *

    END DO

    IF (region is not a single pixel region) THEN :

        DO :
            * start at the region start-point and find the
              leftmost boundary pixel of the region by
              searching anti-clockwise the directions
              4, 3, 2 and 1 (see Appendix B) *

            * find the last link of the code by recording the
              direction 180 degrees from the direction in which
              the leftmost pixel was found *

            nbb_counter = 0

            code_counter = 0

        END DO

        IF ( Select = 2 * neighbours + chain-code * ) THEN :
```

```
      * find the neighbours of the start-point pixel.
        Start in the direction obtained from the
        search_start_vector - index the search_start_vector
        with the direction of the leftmost border pixel
        found above. *

      * increment nbb_counter for each new neighbour *

      * enter the neighbour labels into
        nnbr_vector(1,nbb_counter). Complete the entries
        in vectors  nbbr_vector(2,nbb_counter),
                    nbbtype_vector(nbb_counter) and
                    nbbpos_vector(nbb_counter)
        for each neighbour.  *

END IF

DO :

      * find the first link in the chain-code by starting
        at the start-point and searching clockwise beginning
        with the pixel in direction 1. *

      * enter this link into the chain_vector *

      * change reference pixel to the pixel pointed
        at by the first link *

      code_counter = 1

END DO

REPEAT UNTIL (the start-point is reached AND the link
              pointing to the start-point is the
              last_link) :
```

pixels

searchvector_x(11,3,x) = number of chords in scan-column x

* do next column to the right *

END REPEAT

rhlimit = x * the right-hand side of the circumscribing rectangle has been found - note that this is a column not containing any reg_label pixels *

* start at column to the left of start-point column : *

REPEAT UNTIL (a column is processed where the region does not occur) :

* assume column_pointer = x *

* repeat the processing done for the previous scan *

* find the next column to the left *

END REPEAT

lhlimit = x * the left-hand side of the rectangle has been found *

* now find chords across the region in the horizontal direction : *

* set the column_pointer to index lhlimit *

* set the row_pointer to reference the row in which the start-point occurs *

REPEAT UNTIL (a row is reached where no reg_label pixels occur) :

```
            * start with the reference pixel *

            * find the direction of the first pixel
              searched using the search_start_vector
              refarenced by the previous chain link *

            * search clockwise until the next border
              pixel is reached *

            IF (select = 2 * neighbours + chain *) THEN :

              * increment nhb_counter for each new neighbour
                that is found *

              * add each neighbour pixel that is found during
                the search to the neighbour vectors. Occurrences
                of the same neighbour only increments
                nhbr_vector(2,nhb_counter). Complete the
                other neighbour vectors appropriately. *

              * pixels on the edges of the image are recorded
                as "-1" in nhbr_vector(1,nhb_counter) *

            END IF

            code_counter = code_counter + 1

            * enter the new link into the chain_vector *

            * move to the next reference pixel *

        END REPEAT

    ELSE (region consists of only one pixel) :

        code_counter = 0
```

```
        IF (select = 2  * neighbours + chain *) THEN :

            * find all the neighbouring pixels using a
              sequential search around the start-point's
              8 neighbours.     *

        END IF

    END IF

    IF (select = 2  * neighbour + chain *) THEN :

        IF (the last neighbour in the list is the
            same as the first)  THEN :

            * delete the last neighbour *

            * incorporate the relevant data concerning
              the last neighbour into that of the first *

            nhb_counter = nhb_counter - 1

            * the neighbour-code is now cyclic *

        END IF

    END IF

END PROCEDURE
```

F.7 REGION-CLUSTER MERGE SUBROUTINE

NAME : Subroutine Clean

PURPOSE :

Removes clusters of small regions from inside large regions.

METHOD : Described in Chapter 4

INPUTS :

1. Labelled image
2. Region description vectors
3. Merge vectors
4. Number_of_regions
5. Maximum size of regions in a cluster
6. Maximum number of regions in a cluster
7. Minimum size of a region that contains a cluster

OUTPUTS :

1. New labelled image
2. New region description vectors
3. New merge vectors
4. New number_of_regions

CALLED BY : Main Program

CALLS :

1. Chain/Neighbour Code Subroutine
2. Merge_process Subroutine
3. Resort Subroutine

DESCRIPTION :

BEGIN DATA DEFINITION :

* region description vectors : *

area_vector	: region areas
grayleval_vector	: region gray-levels
startpoint_vector	: region start-points

* merge vectors : *

merge_vector	: indicates merged regions
merge_log_vector	: indirect region id's
sequence_vector	: regions that exist on the image
pointer_vector	: position of regions in sequence_vector

* general : *

big_limit	: minimum size of surrounding region
cluster_size_limit	: maximum size of a cluster region
cluster_limit	: maximum number of regions in a cluster
cluster_list	: vector used to accumulate regions belonging to a cluster

```
        large_region        : id of the region that surrounds
                              the cluster

END DATA DESCRIPTION

BEGIN PROCEDURE :

    REPEAT UNTIL (all regions in the sequence_vector
                  have been processed) :

        * find the next region from the sequence_vector *

        * find its size from the area_vector *

        IF (region's size is less than cluster_size_limit) THEN :

            CALL (Chain/Neighbour Code Subroutine) :
                INPUTS  :
                          1. labelled image
                          2. label of region
                          3. startpoint_vector
                          4. select neighbour calculation
                OUTPUTS :
                          1. neighbour vectors
                          2. number of neighbours found

            END CALL

            REPEAT UNTIL (all neighbours in the neighbour list
                          have been processed) :

                * fetch a neighbour *

                IF (neighbour is an edge label) THEN :
```

```
  * abort : do next region in sequence_vector *

END IF

IF (area of neighbour is greater than
    big_limit) THEN :

  IF (neighbour is a rock-region) THEN :

    IF (no other large region has yet been
        found) THEN :

      large_region = neighbour

      * enter neighbour into cluster_list *

      IF (cluster_list exceeds cluster_limit) THEN :

        * abort : do next region in sequence_vector *

      END IF

    ELSE :

      * abort : do next region in sequence_vector *

    END IF

  ELSE :

    * abort : do next region in sequence_vector *

  END IF

ELSE :
```

```
        IF (area of neighbour is less than
            cluster_size_limit) THEN :

          * enter neighbour into cluster_list *

          IF (cluster_list exceeds cluster_limit) THEN :

            * abort : do next region in sequence_vector *

          END IF

        ELSE :

          * abort : do next region in sequence_vector *

        END IF

      END IF

  END REPEAT

  REPEAT UNTIL (all regions in cluster_list are processed) :

      * fetch region from cluster_list *

      IF (region is not the large_region) THEN :

        CALL (Chain/Neighbour Code Subroutine) :
          INPUTS  : as previous call

          OUTPUTS : as previous call
        END CALL

        REPEAT UNTIL (all new neighbours have been
                      processed) :
```

```
* fetch a new neighbour *

IF (new neighbour is not the same as the
    sequence_vector region being processed) THEN :

  IF (the new neighbour is an
      edge label) THEN :

    * abort : do next region in sequence_vector *

  END IF

  IF (area of new neighbour is greater
      than big_limit) THEN :

    IF (another large region exists) THEN :

      * abort : do next region in
        sequence_vector *

    ELSE :

      IF (new neighbour is a background
          region) THEN :

        * abort : do next region in
          sequence_vector *

      END IF

      large_region = new neighbour

    END IF

  ELSE :
```

```
                IF (area of new neighbour is greater than
                    cluster_size_limit) THEN :

                  * abort : do next region in sequence_vector *

                END IF

              IF (new neighbour does not yet exist in
                  the cluster_list) THEN :

                IF (cluster_list is not larger than
                    cluster_limit) THEN :

                  * add new neighbour to list *

                ELSE :

                  * abort : do next region in sequence_vector *

                END IF

              END IF

            END IF

          END REPEAT (new neighbour processing complete)

        END IF

    END REPEAT (regions in cluster_list done)

    IF (no large region was found) THEN :

        * abort : do next region in sequence_vector *

    END IF
```

```
            * find the merge_master in the cluster_list
              - ( details of this procedure are the same
              as for the grey_level merge subroutine ) *

            * call merge_process subroutine to merge
              all the regions in the cluster_list
              as well as the sequence_vector region
              being processed - (the same procedure is
              once again used as for grey_level merging *

        END IF (region size limit)

    END REPEAT (all regions in the sequence_vector have been processed)

    * compute the new labelled image *

    * clear the merge_vector *

    * call Resort Subroutine to find the new sequence_vector *

END PROCEDURE
```

F.8 GEOMETRIC MERGE SUB-PROGRAM

NAME : Program Geos

PURPOSE : Merges regions according to geometric rules.

METHOD : Described in Chapter 4 and Appendix D.

INPUTS :

1. Labelled image
2. Region description vectors
3. Merge vectors
4. Number_of_regions
5. Minimum size of regions considered for merging
6. Minimum size of neighbours to allow merging
7. Number of merge iterations

OUTPUTS :

1. New labelled image
2. New region description vectors
3. New merge vectors
4. New number_of_regions

CALLED BY : Main Program

9. Gu, W. and Huang, T. Connected line drawing extraction from a perspective view of a polyhedron, IEEE Trans. on P.A.M.I., no 4, July 1985

10. Hall, E. Computer Image Processing and Recognition, New York : Academic Press, 1976

11. Hummel, R. Histogram modification techniques, Computer Graphics and Image Processing, vol 4, 1975

12. Jerez, J. C., Toro, H., Vc Borries, G. H., Automatic control of semi-autogenous grinding at Los Bronces, IFAC Symposium on Automation for Mineral Resource Development, Brisbane : Queensland, Australia, July 1985

13. King, R. Determination of the distribution of size of irregularly shaped particles from measurements on sections or projected areas, Powder Tech., 32, 1982

14. King, R. Measurement of particle size distribution by image analyser, Powder Tech., 39,1984, pp 279-289

15. Kushner,T.R , Rosenfeld, A., Interprocessor requirements for parallel image processing, IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management Proceedings, IEEE Computer Society Press, California : USA, October 1983 , p 177-183

16. Lerner, E. J. Parallel Processing gets down to business, High Technology, July 1985, p 20-28

17. Luh, J. and Klaasen, J. A three dimensional vision by off-shelf system with multi-cameras, IEEE Trans. on P.A.M.I., vol 7 no 1, January 1985, pp 35-45

```
        ccl                 : length of chain-code

    * neighbour vectors : *

        nhbr_vector(1,1..nbl)    : neighbour list
        nhbr_vector(2,1..nbl)    : occurrence of neighbours
        nhbpos-vector(1..nbl)    : position of neighbours in chain-code
        nhbtype_vector(1..nbl)   : connectedness of neighbours
        nbl                      : total number of neighbours

    * general : *

        link                : link in chain-code at which geometry
                              is investigated

        region_size_limit   : smallest region that is processed

        nhb_size_limit      : smallest neighbour that is processed

        occurrence_limit    : minimum number of pixels necessary
                              for a neighbour occurrence before
                              merging is considered

END DATA DEFINITION

BEGIN PROCEDURE :

    REPEAT UNTIL (specified number of iterations are complete
                  or no further merges occur) :

        REPEAT UNTIL (all regions in the sequence_vector
                      have been processed) :

            * fetch a region from the sequence_vector *
```

```
IF (the region's area is less than region_size_limit
    or it is a background region) THEN :

  * abort : do next region *

END IF

* now find the region's neighbours : *

CALL (Chain/Neighbour Code Subroutine) :
   INPUTS  :
             1. labelled image
             2. label of region
             3. startpoint_vector
             4. Chain/neighbour select : neighbour option

   OUTPUTS :
             1. neighbour description vectors
             2. chain_vector
             3. length of neighbour-code (nbl)
             4. length of chain-code (ccl)

END CALL

* neighbours and border chain-code computed *

* now investigate neighbours : *

REPEAT UNTIL (all neighbours are processed) :

  * fetch a neighbour *

  * check merge_vector to see if this neighbour
    has been rejected during a previous occurrence
    in the neighbour-code *
```

```
IF (region has been rejected before or neighbour
    is a background region or its area is less
    than nhb_size_limit or occurrence of neighbour's
    pixels is less than occurrence_limit) THEN :

     reject region using an entry in the merge_vector
     - add 10 to the merge_vector entry corresponding
     to the neighbour label *

   * abort : investigate next neighbour *

END IF

* neighbour is ready for geometry check : *

* set reference to the region start-point *

* find the position of the neighbour interception
  link from nhbpos_vector *

* follow the region border using the chain_vector
  - stop at the neighbour interception link *

* find the interception mask to be used :
  4-connected mask for links 1, 3, 5 and 7 ;
  8-connected mask for links 2, 4, 6 and 8.

* apply the mask using the difference chain
  code in the appropriate interception_mask
  vector *

IF (the conditions at every mask point are
    not satisfied) THEN :

   * reject the neighbour : add 10 to its
     merge_vector entry *
```

```
   * abort : do next neighbour *

END IF

* interception has succeeded - now check detachment : *

* find the link in the chain-code where the next neighbour
  occurs *

* move to this link *

* find the detachment mask :
  4-connected mask for link 1, 3, 5 and 7 ;
  8-connected mask for link 2, 4, 6 and 8 *

* apply the detachment mask - compute the
  position according to the difference cha
  in the appropriate detachment_mask vector *

IF (the mask conditions are not met) THEN :

   * reject the neighbour as before *

   * abort : do next neighbour *

END IF

* neighbour has passed the mask tests *

* check whether all the occurrences have been tested *

IF (other occurrences of the neighbour still have not
    been tested) THEN :

   * abort : do next neighbour *
```

```
END IF

* region may now merge to its neighbour : *

* find the merge_master *

IF (region is not the merge_master) THEN :

   merge_region = region id

   merge_type = 6
   CALL (Merge_process Subroutine) :
        INPUTS  :
                  1. merge_region
                  2. merge_master
                  3. merge_type
                  4. merge vectors
                  5. region description vectors

        OUTPUTS :
                  1. new merge vectors
                  2. new region description vectors

   END CALL

END IF

IF (neighbour isn't the merge_master) THEN :

   merge_region = neighbour id
   merge_type = 6

   CALL (Merge-process Subroutine)

        INPUTS  : * as before *
```

```
                    OUTPUTS : * as before *

                  END CALL

                END IF

                * merge complete *

              END REPEAT (all neighbours processed)

              * reset all entries in merge_vector that are larger
                than 10 by subtracting 10 *

          END REPEAT (all regions processed) :

          * reset merge_vector to "0" *

          * compute new region image *

          * call Resort Subroutine to find new sequence_vector *

      END REPEAT (iterations completed)

END PROCEDURE
```

F.9 UN-MERGE SUB-PROGRAM

F.9.1 PROGRAM 1 : VERTICAL AND HORIZONTAL SPLITTING

NAME : Program UNM

PURPOSE : Performs region splitting to eliminate merge errors.

METHOD : As described in Chapter 5 and Appendix E

CALLED BY : *Main Program*

INPUTS :

1. Label of region to be processed
2. Region description vectors
3. Chord_length_factor
4. Last scan-line (horizontal) - startchord_y
5. Last scan-line (vertical) - startchord_x
6. Number_of_regions
7. Labelled image

OUTPUTS :

1. New labelled image
2. New number_of_regions

3. Last scan-line (horizontal) - startchord_y

4. Last scan-line (vertical) - startchord_x

5. New region description vectors

6. Circumscribing rectangle co-ordinates

DESCRIPTION :

BEGIN DATA DEFINITION :

* region description vectors : *

area_vector : region areas
graylevel_vector : region gray-levels
startpoint_vector : region start-points

* general : *

reg_label : label of region to be processed

new_label : label of new sub-region

chord_length_factor : fraction used to define the maximum chord-length for which splitting occurs

startchord_x : column in image where region reg_label was previously split during vertical processing

startchord_y : row in image where region reg_label was previously split during horizontal processing

isy : number of rows in the image

isx : number of columns in the image

* circumscribing rectangle co-ordinates : *

toplimit : y-ordinate defining the upper side of the circumscribing rectangle

botlimit : y-ordinate defining the lower side of the rectangle

lhlimit : x-ordinate defining the left side

rhlimit : x-ordinate defining the right side

* chord construction vectors : *

searchvector_x(1..11,1..3,1..isx) : accumulates chords in the vertical scan

* definition of records : *

(1..10,-,x) : allows for 10 chords along scan-column x

(11,3,x) : accumulates the number of chords found in scan_column x

(c,1,x) : row in which chord c was found within scan-column x

(c,2,x) : row defining the end of chord c

(c,3,x) : length of chord c

searchvector_y(1..11,1..3,1..isy) : accumulates chords in the horizontal scan

* definition of records : *

* same as for searchvector_x : replace "column" with "row" and "x" with "y"

END DATA DEFINITION

BEGIN PROCEDURE :

* start by finding all the chords across the region in the vertical direction : *

* set the column_pointer to reference the column containing the start-point of the region *

REPEAT UNTIL (a column is processed which has no pixels belonging to the region) :

* assume column_pointer = x *

* find all the chords in the vertical scan-line downwards - enter the relevant data as follows :

searchvector_x(chordnumber,1,x) = first row where chord "chordnumber" occurs

searchvector_x(chordnumber,2,x) = row where last pixel of chord "chordnumber" occurs

searchvector_x(chordnumber,3,x) = length of chord "chordnumber" in

pixels

searchvector_x(11,3,x) = number of chords
in scan-column x

* do next column to the right *

END REPEAT

rhlimit = x * the right-hand side of the circumscribing
rectangle has been found - note that this is
a column not containing any reg_label pixels *

* start at column to the left of start-point column : *

REPEAT UNTIL (a column is processed where the region does
not occur) :

* assume column_pointer = x *

* repeat the processing done for the previous scan *

* find the next column to the left *

END REPEAT

lhlimit = x * the left-hand side of the reccangle has been found *

* now find chords across the region in the horizontal direction : *

* set the column_pointer to index lhlimit *

* set the row_pointer to reference the row in which the start-point
occurs *

REPEAT UNTIL (a row is reached where no reg_label pixels occur) :

pixels

searchvector_x(11,3,x) = number of chords in scan-column x

* do next column to the right *

END REPEAT

rhlimit = x * the right-hand side of the circumscribing rectangle has been found - note that this is a column not containing any reg_label pixels *

* start at column to the left of start-point column : *

REPEAT UNTIL (a column is processed where the region does not occur) :

* assume column_pointer = x *

* repeat the processing done for the previous scan *

* find the next column to the left *

END REPEAT

lhlimit = x * the left-hand side of the rectangle has been found *

* now find chords across the region in the horizontal direction : *

* set the column_pointer to index lhlimit *

* set the row_pointer to reference the row in which the start-point occurs *

REPEAT UNTIL (a row is reached where no reg_label pixels occur) :

```
    * assume y is the row_pointer *

    * find all the chords in the horizontal scan from left to right *

    * enter the data as follows :

        searchvector_y(chordnumber,1,y) = first column where chord
                                          "chordnumber" is found

        searchvector_y(chordnumber,2,y) = last column where chord
                                          "chordnumber" is found

        searchvector_y(chordnumber,3,y) = length of chord in pixels

        searchvector_y(11,3,y)          = number of chords found

    * process the next row below *

END REPEAT

botlimit = y * the bottom of the circumscribing rectangle has been
           found *

toplimit = * row containing the start-point *

* the circumscribing rectangle is now completely defined by
    botlimit, toplimit, lhlimit, rhlimit  *

* now define the maximum chord-size allowed for a split to occur : *

splitlimit = (botlimit-toplimit)/chordlength_factor

* now find the column from which splitting should begin : *

startchord = lhlimit
```

```
IF (startchord_x is greater than startchord) THEN :

     startchord = startchord_x

END IF

IF (startchord_x equals rhlimit) THEN :

     * abort : do horizontal chord checking *

END IF

* move the column_pointer to startchord  *

REPEAT UNTIL (column_pointer reaches rhlimit or a split occurs
              successfully) :

     REPEAT UNTIL (all chords in searchvector_x(-,-,column_pointer)
                   have been processed or a successful split occurs) :

          * fetch the next chord from searchvector_x *

          IF (searchvector_x(chordnumber,3,column_pointer) equals
              splitlimit) THEN :

               * the length of the chord favours merging *

               * now find chords having start and end points
                 within 3 pixels of the start and end points
                 of the chord being considered. One chord must
                 be found in the columns on either side of
                 that referenced by column_pointer *

               IF (chords cannot be found) THEN :
```

```
  * abort : do next chord *
END IF

* define the original chord as "splitchord" *

* define the chord on the left as "matchchord 1" *

* define the chord on the right as "matchchord 2" *

* substitute the pixels in the image corresponding
  to splitchord and matchchord 2 with label "0" *

* the number of pixels substituted is stored
  in variable "temp_size" *

* find a label with which to re-label the new
  sub-region to be formed : define this label
  as "new_label" *

* extend a chord leftwards starting with the
  middle of matchchord 1. This chord consists
  of new_label pixels and terminates when the
  region border is encountered. Assign the
  number of substituted pixels to "new_size" *

* propagate a re-labelling wave-front starting
  at the first re-labelled chord upwards
  until toplimit is reached *

* propagate the re-labelling front downwards
  until botlim is reached *

REPEAT UNTIL (no more re-labelling occurs) :

  * propagate the re-labelling wave-front
    upwards and downwards *
```

```
  * accumulate the number of re-label
    occurrences in new_size *

END REPEAT

IF (new_size equals temp_size) THEN :

  * a "doughnut" structure exists ie.
    the re-labelling has not managed to
    split the region *

  * leave the "0" value chords and re-label
    the new_label values back to the
    original labels *

  * abort : process the next chord *

END IF

new_size = new_size + temp_size

IF (new_size is too small or too large) THEN :

  * undo the un-merge *

  * replace the "0" value chords with the
    original label *

  * replace the new_label pixels with the
    original label *

  IF (new_label replacement has been done
      more than 10 times) THEN :

    * increase the column_pointer increment
```

```
                    from 1 to 10 to accelerate the
                    processing *

                  END IF

                  * abort : do next chord *

                END IF

                * the split is now successful *

                * replace the "0" value chords with new_label *

                * find the new start-points of both the old region
                  and the new region *

                * modify the region description vectors accordingly : *

                  area_vector(reg_label) = old_size - new_size

                  area_vector(new_region)= new_size

                  graylevel_vector(new_region) =
                  graylevel_vector(reg_label)


            END REPEAT (chords)

        END REPEAT (columns)

        lastchord_x = column_pointer * record the last scan_line *

        IF (a split occurred) THEN :

            * return to main program *
```

```
END IF

* now do horizontal search *

* define the maximum chord-size allowed for splitting : *

splitlimit = (rhlimit - lhlimit)/chordlength_factor

* find the row from which splitting should begin : *

  startchord = toplimit

  IF (startchord_y is larger than startchord) THEN :

    startchord = startchord_y

  END IF

  IF (startchord = botlim) THEN

     * abort : return to main program to do
              diagonal splitting *

  END IF

* set row_pointer to index startchord *

REPEAT UNTIL(row_pointer reaches botlimit or a successful
             split occurs) :

     * REPEAT THE PROCEDURE USED FOR VERTICAL SPLITTING -
         substitute searchvector_x for searchvector_y,
                  x for y and rightwards for downwards
                                                  etc. *

END REPEAT
```

```
        lastchord_y = row-pointer * record the last scan-line *

END PROCEDURE
```

F.9.2 PROGRAM 2 : DIAGONAL SPLITTING

NAME : Program UN45

PURPOSE : Performs diagonal region-splitting

METHOD : As discussed in Chapter 5 and Appendix E

INPUTS :

1. Region label
2. Region description vectors
3. Chordlength_factor
4. Last scan-line index (45 degree splitting)
5. Last scan-line index (135 degree splitting)
6. Number_of_regions
7. Labelled image
8. Circumscribing rectangle Co-ordinates

OUTPUTS :

1. New labelled image
2. New number_of_regions
3. Label of new sub-region
4. Last scan-line index (45 degree splitting)

5. Last scan-line index (135 degree splitting)

6. New region description vectors

CALLED BY : Main Program

DESCRIPTION :

The method of un-merging is similar to that outlined in Appendix F 9.1. Only the direction in which chords are computed is different as well as additional complexity regarding the referencing of scan-lines as was explained in Appendix E. Because of the similarity of the processing to horizontal and vertical splitting, a PDL description of diagonal splitting is not presented here. Particular details concerning diagonal splitting will become clear when consulting the FORTRAN listing.

REFERENCES

1. *Allen*, T. Particle Size Measurement, Chapman and Hall, 1981

2. Barbery, G. Determination of particle size distribution from measurement of sections, Powder Tech., 39, *1984*, *pp* 279-289

3. Beucher, S. and Lantuejoul, Ch., Use of watersheds in contour detection, International Workshop on Image Processing, Rennes, France, 1979

4. Brice, C. and *Fennema*, C. Scene analysis using regions, (in Computer Methods in Image Analysis, Aggrawala, Duda and Rosenfeld (eds.), Wiley, 1977)

5. Dinstein, I., Yen, D., Flickner, M., *Handling memory* overflow in connected *component* labelling applications, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 7 no 1, January 1985

6. Duda, R. and Hart, P. Pattern *Classification and* Scene Analysis, New York, Wiley, 1978

7. Fairfield, J. Segmenting blobs into subregions, IEEE Transactions on Systems, Man and Cybernetics, Vol 13 no 3, May/June 1983

8. *Gallagher*, E. Opto-Electronic Coarse Particle Size Analyser for Industrial Measurement, Ph.D. Thesis, University of Queensland, Australia, December 1976

18. Meyer, F.(a) Iterative image transformation for an automatic screening of cervical smears, Journal of Histochemistry and Cytochemistry, 27 no 1, 1978, pp 128-135

19. Meyer, F. (b) Contrast feature extraction (in Chermaut, J. (ed), Quantitive analysis of microstructures in material sciences, biology and medicine, Practical Metallography, Riederer Verlag, 1978)

20. Nazif, A and Levine, M. Low level image segmentation : an expert system, IEEE Trans. on P.A.M.I., vol 5, September 1984, pp 555-577

21. Rosenfeld, A. and Kak, A., Digital Picture Processing, Academic Press, 1982

22. Rosenfeld, A. Computer vision : signals, segments and structures, IEEE A.S.S.P. Magazine, January 1984

23. Serra, J. Image Analysis and Mathematical Morphology, Academic Press, 1982

24. Smith, D. Imaging Systems Capable of Real Time Size Distribution Analysis, M.Sc. Dissertation, University of the Witwatersrand, Johannesburg, December 1985

25. SPIDER MANUAL, Joint Systems Development Corporation, Tokyo : Japan, 1983

26. Stanley, G. G., Considerations in the application of autogenous milling, Journal of the South African Institute of Mining and Metallurgy, October 1975

27. Tanimoto, S. and Pavlidis, T. A hierarchical data structure for picture processing, Computer Graphics and Image Processing, vol 4, 1975, pp 104-199

28. Tsai, W. and Yu, S. Attributed string matching with merging for shape recognition, IEEE Transactions on P.A.M.I., vol 7 no 4, July 1985

29. Vignos, J. et al. On line instrument for coarse particle *size distribution measurement*, paper presented at SME/AIME Fall Meeting, Tucson Arizona : USA, October 1979.

30. Weszka, J. A survey of threshold selection techniques, *Computer Graphics and Image Processing*, vol 7, 1978, pp259-256

31. Yachner,L., Gonzales, C. , Von Borries, G., Nobile, R., *Coarse particle size distribution analyser*, IFAC Symposium on Automation for Mineral Resource Development, Brisbane : Queensland, Australia, July 1985