Massey University Library
Thesis Copyright Form

Title of thesis: DNA Sequence Reading by Image Processing

(1) (a) I give permission for my thesis to be made available to readers in Massey University Library under conditions determined by the Librarian.

(b) I do not wish my thesis to be made available to readers without my written consent for ... months.

(2) (a) I agree that my thesis, or a copy, may be sent to another institution under conditions determined by the Librarian.

(b) I do not wish my thesis, or a copy, to be sent to another institution without my written consent for ... months.

(3) (a) I agree that my thesis may be copied for Library use.

(b) I do not wish my thesis to be copied for Library use for ... months.

Signed . Fan Baozhen FANBZ

Date 30, 3, 94

The copyright of this thesis belongs to the author. Readers must sign their name in the space below to show that they recognise this. They are asked to add their permanent address.

NAME AND ADDRESS DATE

DNA Sequence Reading by Image Processing

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Science

in Computer Science at

Massey University

Supervised by

Dr Donald Bailey

and

Dr John Hudson

Fan Baozhen

1993

To my mother, father
and sisters

Abstract

The research described in this thesis is the development of the DNA sequence reading system.

Macromolecular sequences of DNA are the encoded form of the genetic information of all living organisms. DNA sequencing has therefore played a significant role in the elucidation of biological systems. DNA sequence reading is a part of DNA sequencing. This project is for reading DNA sequences directly from DNA sequencing gel autoradiographs within a general purpose image processing system.

The DNA sequence reading software is developed based on the waterfall software development approach combined with exploratory programming. Requirement analysis, software design, detailed design, implementation, system testing and maintenance are the basic development stages. The feedback from implementation and system testing to detailed design is much stronger in image processing than a lot of other software development.

After an image is captured from a gel autoradiograph, the background of the image is normalised and the contrast is enhanced. The captured image consists several lane sets of

bands. Each of the lane set represents one part of a DNA sequence. The lane sets are separated automatically into subimages to be read individually. The gap lines between the lane sets are detected for separation. The geometric distortions are corrected by finding the boundaries of the lane set in the subimage. The left boundary of the lane set is used to straighten lane set and the right boundary is used to warp the lane set into a standard width. If separation of the lane sets or geometry correction is unsuccessful by automatical processing, manual selection is used. After the band features are enhanced, the individual bands are extracted and the positions of the bands are determined. The band positions are then converted into the order of the DNA sequence. Different part of a sequence from subsequences are merged into a longer sequence.

In most of the cases, the individual lane sets in a captured image are able to be separated automatically. Manual processing is necessary to handle the cases where the lane sets are too close.

The system may reach an accuracy of 98% if the bands are clear. Manual checking and correcting the detected bands helps to obtain a reliable sequence. If a lane set on the autoradiograph is indistinct or bands are too close it may reduce the accuracy, in extreme cases to the point where it is unreadable. For a 512×512 image captured from a gel autoradiograph, preprocessing takes 90 seconds, processing each subimage takes 40 seconds on a 33Hz 486 PC. If processing a 430×350 mm autoradiograph with 16 lane sets, assuming 6 images are required, it takes about 40 minutes.

Preface

As I complete my study for my master's degree, I want to call to mind my mother and father and sisters, and my whole family who have encouraged and supported me so much over the past years. I am very grateful to my family, who have helped me to persist in my studies when I often felt too old to continue. I feel very relieved and thankful that after so many years of what seems like wasted time and many struggles I have now completed my work.

When I was a small girl I loved studying very much, and was inspired by my parents, and by my sisters. However, circumstances were very difficult from my youth onwards, as a result of the environment at that time.

My parents had no opportunity to get an advanced education when they grew up in China. So they placed their hope on their children. Indeed their greatest desire was that their children would get a good education. As their daughters, we sought not to disappoint their expectation. At school we all got high grades in our work. My eldest sister, Guizhen, and my second sister, Guimei, passed the entrance examinations and were both admitted to university. My third sister, Guie, delayed going to middle school because she had been competing in wushu, a sport in which she became outstandingly

successful and the champion for the whole of China. After middle school, she too went to university. I am the fourth in the family, and I was admitted to one of the best middle schools in Qingdao city after taking the entrance examinations.

My sisters set a fine example for me, inspiring me subconsciously to go on from school to university, and then after my undergraduate study to undertake postgraduate research, and then do research or teach in a university. It all seemed so straight-forward. But the Cultural Revolution in China commenced just as I was completing my three years of study at middle school. So overnight the dream of studying in university evaporated. High school study is for a period of three years between middle school and tertiary study in China, and I felt sure I could go there. However, because my father had once been a capitalist I was not permitted to attend, even though my school marks were always excellent. When I discovered that I would not be permitted to study in high school I was broken-hearted. In my dreams I would imagine that I was at school, and when I woke up, I would cry because I would never be allowed to take a seat in the lecture theatre.

Later, I went to the countryside and worked in a factory. All the "golden time" of my youth had come to nothing. It was ten years later when once again people were admitted to university by sitting the entrance examinations. However, I had attended no classes at high school level, while others had attended for three years. My sisters encouraged me to take the entrance exams anyway, and with their help and my own study I got very good marks much to my surprise. Even then I encountered prejudices against admitting me to university, and I almost missed out on gaining entrance. But I was stubborn, and I clung onto the gleam of hope and refused to succumb to the opposition. So finally I entered my university career, ten years late. My university study in China was the essential preparation to overseas study.

New Zealand is the first foreign country I have ever visited. As a mature student, I found study overseas much more difficult than for others in the same course. Language is one problem, but I have also experienced a serious ailment in my back, and although I do not look ill, I have constantly had to struggle with it. Nobody else has any idea just how acute this little problem has been. I spent a great deal of time trying to control the pain, so that I could concentrate better on my work. The suffering reduced my ability to work and to live, and this made me lose confidence. The worry and distress made my condition worse, and several times I almost abandoned my study. Yet my family's desire that I succeed, and my own long-cherished desire from when I was a child, kept me going, preventing me from giving up. It may have been very hard, but I am determined to go forward and upward.

God has taken care of me. Just before completing my MSc I again almost lost confidence about my future direction. Just at this point the pain in my back began to reduce. So the hope of what lies in the future has returned to me. The desire to climb forward and upward has returned to my heart. I am still my stubborn self, and I will not give up on my aspirations.

During the difficult experiences of my life, I have always had the complete support of my whole family to get me over dangers and difficulties. My mother has always been my guide through life. Whenever difficulty or danger has come near, I have always remembered when I was a child lying in my cradle. She would pick me up and place me securely in her arms. In a way this is what she still does when I am in difficulties. She has always protected and helped her children and encouraged them to go forward and upward. I wish my mother good health, much happiness and long life.

During the two years when I was working on my masters in New Zealand, two members of my family have died, my honoured father and my third sister Guie. They were so far away across the oceans that they could not farewell me before they went to heaven. The pictures of two roses in the Introduction of my thesis are a special tribute to my father and my sister Guie.

When I left home to come to New Zealand, my father wept. He had a premonition that he would never see his youngest daughter again. But I do not believe that my father and sister Guie have left the world. I dreamed about my father last night. I know from this that my father is still alive. So I have a good reason to do what my family want and what they hope for me. I can feel that God is with me, and peace and happiness are present with me, for that is what my sister Guie prayed for and wished as blessing for me in her will. So I want my father, my sister Guie and all my family to smile upon me as I live here.

This thesis is therefore a gift to my father and mother as a thanksgiving for the way they brought me up, and also to my sisters in thanksgiving for their care and love. It is also a memorial to my father and my sister Guie.

Acknowledgments

My all family, my parents, my sisters, my brother-inlaws, my nieces and my nephews, have encouraged and supported me over the past years when I study in NZ. Many thanks to everyone in my family.

Many thanks to my supervisor Dr Donald Bailey, for his valuable guidance throughout this project, for his helping me to learn image analysis techniques, for his patience in trying to understand my Chinese way of thinking and talking, especially when checking my thesis and conference papers, for his encouragement and support when I felt like giving up since my father and sister died.

I would like to thank Dr Nick Ellison for providing invaluable technical information on DNA sequencing and for the provision of the autoradiographs used to develop the software.

I would also like to thank Professor Mark Apperley and Dr John Hudson for their encouragement and support and to thank Terry Cunniffe for being a nice neighbour and helping set up memorial for my father with David's help.

CONTENTS

Chapter 1

Introduction

Digital image processing has been a rapidly evolving field during the last 30 years with a growing range of applications in a broad spectrum of science and engineering disciplines [Jain, 1989]. This growth is coupled with improvements in the processing speed, image display and storage capabilities of computers and cost effectiveness of the related signal processing devices and computers [Pratt, 1978]. Image processing is a broad subject of interdisciplinary study and research in such diverse fields as computer and information science, statistics, physics, astronomy, chemistry, biology, psychology, medicine, geology, engineering and so on [Bailey, 1985].

DNA sequence reading is one application of image processing in fundamental genetic and cellular analysis. Genetic and cellular analysis is an important part of biological, agricultural and medical research [Bodmer, 1987]. The goal of this project is to incorporate automatic DNA sequence reading capability from a gel autoradiograph within a general purpose image processing system.

The genetic information of a living organism is encoded by the DNA contained within every living cell of that organism. DNA sequences are a representation of the genetic structure of DNA molecules. After the DNA reactions are run on an electrophoresis gel

the DNA sequence can be read from the gel autoradiograph. A DNA sequence reading system is developed for reading sequences directly from DNA sequencing gel autoradiographs by image processing techniques. A number of subsequences in an image are captured from a gel autoradiograph. Each subsequence must be read separately. Most of the subsequences may be separated automatically. If necessary, manual processing may be used to separate the subsequences. The subsequence boundaries are extracted and are used to correct for geometric distortions. Individual bands are extracted and the band positions are detected. The order of the DNA sequence is then determined from the sequence of band positions. Different parts of a sequence from different subimages are merged into a longer sequence. The algorithms of the DNA sequence reading system are developed and implemented using VIPS (Vision Image Processing System), version 4.1, which currently runs on an PC under Windows, an Apple Macintosh and a DEC Micro VAX.

Chapter 2 includes a brief survey of image processing and introduces the Vision Image Processing System (VIPS). Some of the image processing concepts are explained which are used in the following chapters on DNA sequence reading software development. These are image acquisition, feature enhancement, linear convolutional filters, intensity histograms, image segmentation, thresholding and line profiles. VIPS is used as the software development and implementation environment for this project. The hardware components required by VIPS and software features of VIPS are described.

In chapter 3, the DNA sequencing application is described more fully and an image processing software development model, based on the waterfall software development approach combined with exploratory programming, is presented. Each stage of the model (requirement analysis, software design, detailed design and implementation, system testing and maintenance) is described as it relates to DNA sequence reading system development. A general model for image processing and the final system function module structure are given in section 3.4. The control module algorithms of DNA sequence reading system are described in section 3.5.

Chapter 4 describes the algorithms for the image processing module of the DNA sequence reading system in detail. An image must be acquired from a DNA sequence gel autoradiograph. Contrast enhancement increases the faint bands in the DNA sequence images. Several subsequences may be in the same captured image and each subsequence must be read separately. The gap lines between subsequences are detected, which are used to obtain separate subimages for each subsequence. Geometric distortions often occur on gel autoradiographs. In order to successfully process most of

the gel autoradiographs, geometry correction is necessary. The subsequence boundaries are extracted and are used to correct for geometric distortions. The left boundary is used to shear the image to straighten left side and the right boundary is used to trapezoid warp the right side. The individual bands are extracted and the positions of the bands are then detected. The band positions are sorted into a list and then converted into a DNA sequence. Different parts of a sequence are merged into a longer sequence.

Chapter 5 discusses the results of the DNA sequence reading system. Automatic separation of lane sets and geometry warping are successful on most of the images captured. If the boundaries of a band lane set are not clear, manual selection of the lane set and manual geometry correction are used. The accuracy of the system depends on the clarity of the bands on the gel autoradiograph. If the bands are clear, the accuracy of automatic processing may reach 98%. The output is checked and any errors may be corrected manually to obtain an acceptable accuracy. Timing is carried out on a 33MHz 486 PC. Automatically processing a 430 × 350 mm autoradiograph with 16 lane sets (see Figure 3.1-1) takes about 40 minutes.

Chapter 6 summarises the system and the thesis and gives suggestions for future work.

The mathematical expression of the VIPS operations used in this thesis are summarised in Appendix I. Appendix II gives VIPS programs for the DNA sequence reading system. Several new VIPS commands were developed for DNA sequence reading: **STRAIGHTEN**, **PARALL**, **SEQUENCE**, **POINTS**, **SORT** and **JOIN**. The C programs for these commands are given in Appendix III.

During the research for the project the following papers were published:

Fan B., Bailey D.G. and Hudson J., Image processing in DNA Sequence Reading, 7th NZ Image Processing Workshop, pp 117-122, Christchuch NZ, (8, 1992).

Fan B. and Bailey D.G.,Algorithms for DNA Sequence Reading by Image Processing, NZ Computer Science Research Students' Conference, pp 109-116, Hamilton NZ, (10, 1992).

Fan B. and Bailey D.G., Algorithms for DNA Sequence Reading by Image Processing, NZ Journal of Computing, pp 47-56, Palmerston North NZ, (5, 1993).

Fan B. and Bailey D.G., Preprocessing Algorithms for Automatic DNA Sequence Reading, TENCON' 93 Computing, pp 998-1001, Beijing China, (10, 1993).

Chapter 2

Image processing and VIPS

Visual imagery is one of the most important sensory inputs to the human perceptual system [Kasturi and Trivedi, 1990]. However, many of the activities which need to be done in modern science are repetitive and mundane, so people have attempted to automate such activities. Since vision is so important, we have also attempted to automate vision. Many theoretical and technological breakthroughs are required before we could emulate the visual functions of human beings [Jain, 1989].

Image processing, as a technology, is a step toward automating visual tasks. Image processing and analysis are used primarily to augment human vision and to extract numerical information from images by applying series of mathematical operations. In practical image processing, there are two principal application areas. The first is the improvement of pictorial information for human interpretation. This is also called computer assisted vision and is primarily used to augment human vision and to extract information from images. The second is the processing of scene data for autonomous machine perception, mainly for inspection and manipulation, which is known as machine vision [Gongalez and Wintz, 1987].

The DNA sequence reading system described in this thesis is an application of computer assisted vision. In section 2.1, some concepts and terminology of image processing techniques used in the following chapters are explained.

Vision Image Processing System (VIPS) is a tool for investigating the applicability and practicability of image processing techniques to industrial and scientific problems. VIPS is used as the software development environment for developing the DNA sequence reading system. The hardware required and the software features of VIPS are described in section 2.2.

2.1 Image Processing

Some concepts and terminology of image processing techniques, which will be used in the following chapters, are explained in this section. These are image acquisition, feature enhancement, linear convolutional filters, intensity histograms, image segmentation, thresholding and line profiles.

Image acquisition. Images can be acquired by a sensor system specially designed to view a scene and provide a digital representation of that scene. The continuous two dimensional intensity pattern is quantised, or sampled, both in position and in intensity giving a digital image. The sampling rate is the number of pixels (picture elements) per unit area. This must be larger than twice the maximum frequency to preserve the useful information in an image without aliasing [Gongalez and Wintz, 1987].

We may consider a digital image as a two dimensional array f(x,y), whose row and column indices identify a point in the image and the corresponding matrix element identifies the grey level or colour at that point.

Feature enhancement. Image enhancement is one of the principal image processing techniques. Images are processed to enhance or extract features which are important for the particular application. Filters, multiple image combination and histogram modification are used for enhancement.

A spatial domain filter calculates the value of each pixel of the output image as some function of the values corresponding pixel and its neighbours in the input image. Different functions enhance or detect different features. This neighbourhood can be thought of as a moving window, where the window is moved from pixel to pixel as each output value is calculated. Figure 2.1-1 illustrates a (3×3) moving window centred at (x, y) in an image.

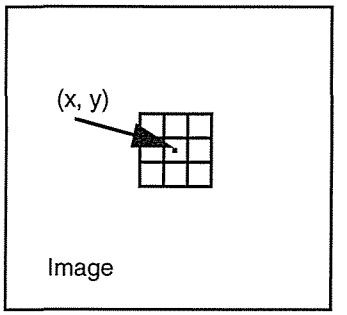


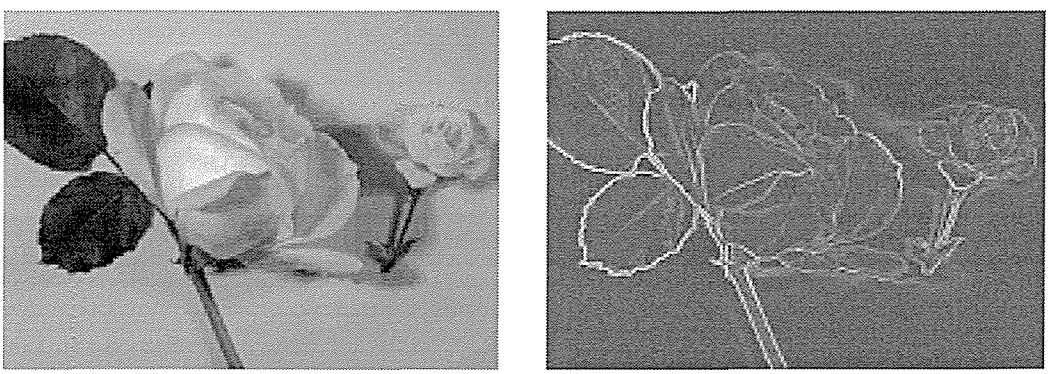
Figure 2.1-1 A 3×3 moving window centred at (x, y) in an image.

The general moving window filter with an m×n window can be represented mathematically as

$$p_{out}(x,y) = f(p_{in}(x+i, y+j), \quad \forall i \in \{-\tfrac{m}{2},..,\tfrac{m}{2}\}, j \in \{-\tfrac{n}{2},..,\tfrac{n}{2}\}) \qquad \text{(Eq. 2.1-1)}$$

where p_{out} is the output image, $f(\ )$ is the filter function which is evaluated at each position of the window in the image, p_{in} is the input image [Bailey, 1993].

Figure 2.1-2 gives an example of edge enhancement. The (a) is the original image, (b) is an edge detected image using a Sobel filter.



a) b)

Figure 2.1-2 Edge enhancement
a) original image, b) edge detection by Sobel filter.

Linear convolutional filters. Linear convolutional filters are one class of filters, which use a weighted average or linear combination of the pixel values within the window. The filter function $f(\ )$ in Eq.2.1-1 can be represented by

$$p_{out}(x,y) = \frac{1}{scale}\left|\sum_{i=-\frac{m}{2}}^{\frac{m}{2}}\sum_{j=-\frac{n}{2}}^{\frac{n}{2}} w(i,j)\,p_{in}(x+i,y+j)\right| \qquad \text{(Eq. 2.1-2)}$$

where $w(i,j)$ are the fixed kernel weights corresponding to each position within the moving window. The kernel weights are chosen to detect a particular property of interest in an image. Figure 2.1-3 gives noise smoothing and different edge detection filter kernel weights [Bailey et al, 1984].

$$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \quad \frac{1}{3}\begin{bmatrix}1&1&1\\0&0&0\\-1&-1&-1\end{bmatrix} \quad \frac{1}{3}\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix} \quad \frac{1}{4}\begin{bmatrix}0&1&2\\-1&0&1\\-2&-1&0\end{bmatrix}$$

noise smoothing | horizontal edge detection | vertical edge detection | diagonal edge detection

Figure 2.1-3 Kernel weights of smoothing and detection.

Intensity histograms. The grey level histogram is a function which summarizes the grey level content of an image. The function shows, for each grey level, the number of pixels in the image that have that grey level. The abscissa is grey level and the ordinate is frequency of occurrence (number of pixels) [Castleman, 1979]. Intensity histograms may be used to enhance images. Figure 2.1-4 gives a histogram of an image. Figure 2.1-5 gives a contrast enhanced image and the histogram, which expanded the range of grey levels linearly .

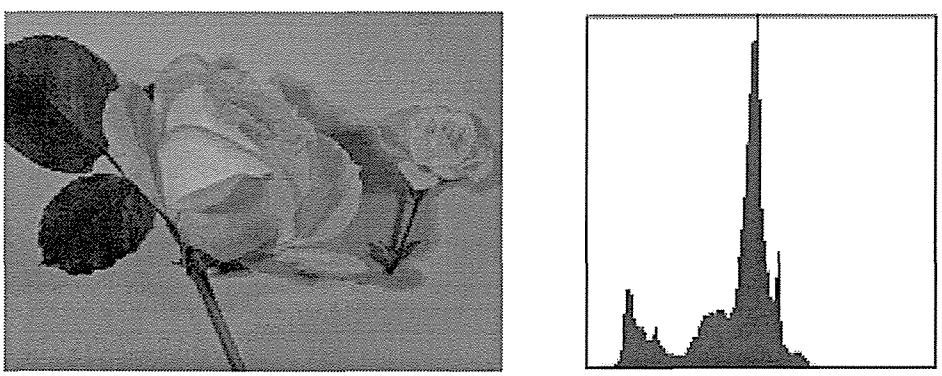


Figure 2.1-4 An image and the intensity histogram.

The intensity histogram may also be used for image segmentation.

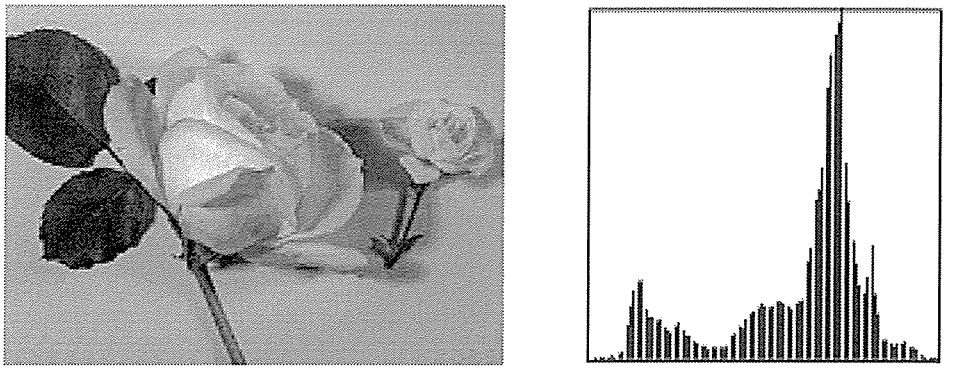


Figure 2.1-5 The contrast stretched image and the histogram.

Image segmentation. Image segmentation is the splitting of an image into its meaningful constituent parts, or regions which have a common property [Gongalez and Wintz, 1987]. Edge detection, boundary tracking, thresholding and region growing are the principal image segmentation approaches [Bailey, 1991]. Image thresholding is the most commonly used method of image segmentation. It splits an image into component regions based on the pixel value at each point. Simple thresholding uses a single threshold level. All pixels less than this threshold are categorised as belonging to one class, while those above the threshold are categorised as belonging to the second class. Figure 2.1-6 gives an example of image segmentation using thresholding. The threshold level is selected by examining the histogram (see Figure 2.1-5) to segment the background and the object. The threshold is chosen at 155 which is just to the left of the main peak, separating the background and the object.

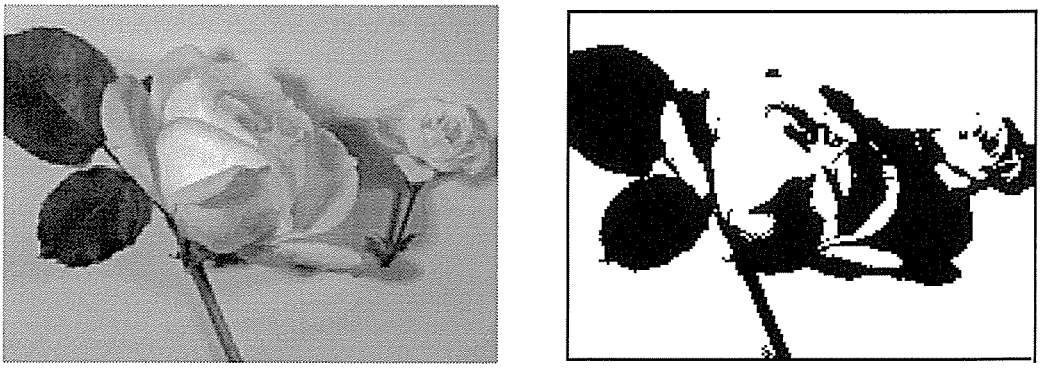


Figure 2.1-6 Image segmentation.

Figure 2.1-7 gives an example of edge detection by thresholding. These pixels with level less than 30 are classified as not edge and set to white. The remaining pixels (greater than 30) are classified as edge pixels, and are set to black.

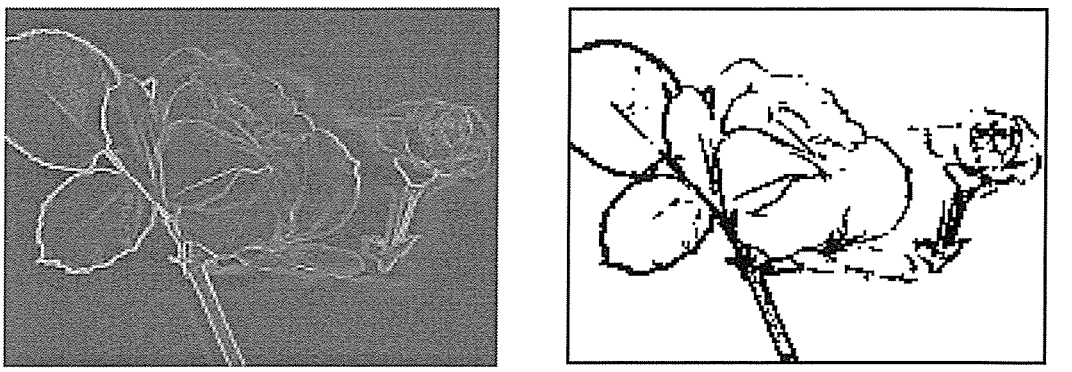


Figure 2.1-7 Image thresholding.

Line profile. The intensity line profile of a image is used to analyse the image features at a defined line. Sharpness of edges, contrast, noise and thresholding range may be determined by analysing the intensities along a line in an image. Figure 2.1-8 shows a profile along the line of the image. The sharpness of edges can be seen from the slope of the lines. The contrast of the image can be seen from the profile shape. The noise can be seen clearly on flat part.

Figure 2.1-8 An image with line profile.

These image processing techniques, along with others, can be combined to form image processing algorithms.

Vision Image Processing System (VIPS) is a tool for developing image processing algorithms. The DNA sequence reading system is developed and implemented using VIPS.

2.2 Vision image processing system (VIPS)

VIPS is designed to support general-purpose image processing. It is a tool for investigating the applicability and practicability of image processing techniques to industrial and scientific problems. It runs on several different computing platforms, MicroVAX, IBM compatible PC and Macintosh with a common command line interface used on all. Sequences of VIPS command lines may be combined together into program files. The main function of VIPS is to provide an environment for the development of image processing algorithms. Therefore, the system is highly interactive and has available hundreds of general purpose image processing operations and functions. VIPS is written in portable C. New commands and new functions may be developed and added to the system by users if necessary [Image Analysis Unit, 1992]. VIPS is developed by my supervisor, Dr. Donald G. Bailey, at the Image Analysis Unit of Massey University.

VIPS hardware requirements. The required hardware components of VIPS are similar even when VIPS is implemented on different computing platforms. Image sensor, frame grabber, image display, computer and image processing software storage are the indispensable components required for image processing. The components of VIPS hardware required are shown in Figure 2.2-1.

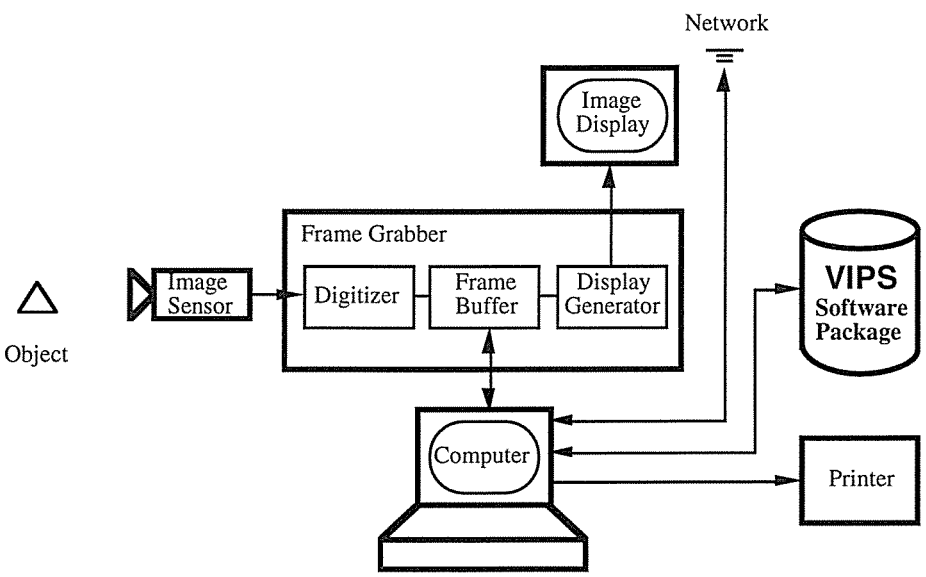


Figure 2.2-1 Components of VIPS hardware required.

An image sensor is required for image acquisition which converts an image into an analog signal suitable for input into a frame grabber. The most common method for capturing images is to use a solid state video camera.

The frame grabber provides the interface between the computer, and the camera and display. A frame grabber typically involves three functional parts: a digitiser, a frame buffer and a display generator. The analog signal produced by the camera is transformed to a digital format. The digital image data are stored into the frame buffer, where they can be read by the CPU in the computer. The function of the display generator is to convert the stored digital information into an analog video signal, and output this signal to a video monitor or image display, where the image can be viewed.

Image display is indispensable for checking the result images. A monochrome or colour video monitor is used to display analog video signal images, or images are displayed directly in another window on the computer terminal.

A general purpose computer provides versatility as well as ease of VIPS application programming. Currently, a microVAX under VMS, an IBM compatible PC under Windows or a Macintosh are used as platforms for VIPS and VIPS applications. At least 2M bytes of RAM is required for data throughput on a general image processing project. The memory requirements depend on the number of images required, which depends on the intended application.

A printer is used to hard copy resultant images and output files if necessary.

Network may be used to get images to be processed from remote computers or to convey results to remote computers.

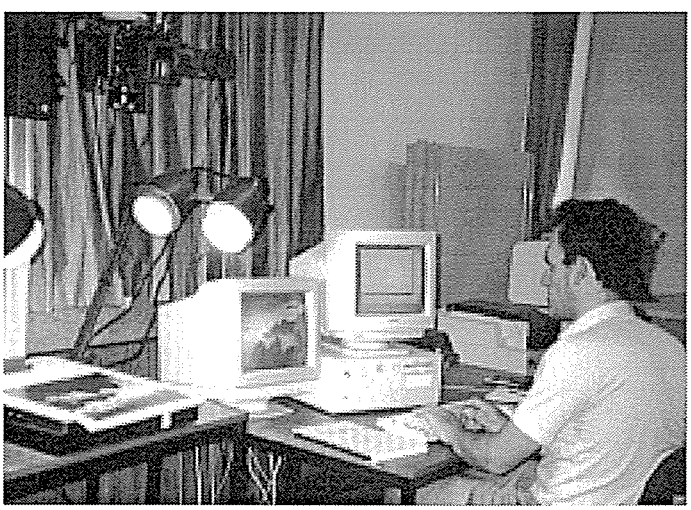


Figure 2.2-2 Vision Image Processing System on a PC.

Figure 2.2-2 shows VIPS running on a PC with the hardware required.

VIPS software. VIPS software package is command based. Operations within VIPS are invoked by entering commands in response to a prompt. The VIPS window on the screen shows VIPS command lines, user prompt and text input and output. The VIPS kernel part includes command parser, command table and variable table. Each command and function code includes command definition and command procedure. Display driver and utilities are also provided for use by commands. Figure 2.2-3 gives the VIPS structure.

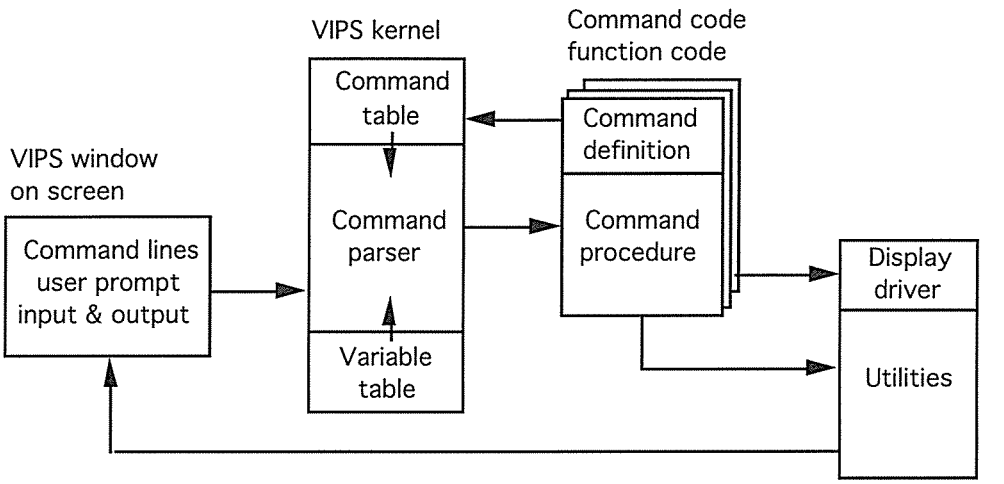


Figure 2.2-3 VIPS structure.

VIPS is written in portable C. Hundreds of VIPS commands are available. New commands may be added by the user by providing a command definition and a command procedure. New functions may be added as well.

The general VIPS command format is as below:

```
VIPS V4: COMMAND parameter1 [parameter2 ...] [/option1 ...]
```

In a VIPS application, command lines can be combined in a text file, as a program. As an example, a rotate program is given below which captures a (512×512) image from the display, then rotates the image and displays the rotated images repeatedly.

```
PROGRAM                                  !rotate.vip
   DISPLAY  image
   FOR  n = 1  4                         !repeat 4 times
      ROTATE  image                      !rotate 90 degrees
      DISPLAY  image
   END
END
```

Chapter 3

DNA Sequencing Software Development

DNA sequences are a representation of the genetic structure of DNA molecules. The generation and analysis of DNA sequence data, DNA sequencing, has played a significant role in the elucidation of biological systems. DNA sequence reading is a part of DNA sequencing, and provides a bridge between the generation and the analysis of DNA sequence data.

DNA sequences may be read manually by trained technical staff. However, although the reading process is straight forward, it is tedious, and therefore prone to errors. After reading, the sequence must then be transcribed into a computer for entry into or comparison with a sequence database. This transcription step is also error prone. The very nature of the problem: being well defined and very repetitive makes it amenable to image analysis techniques. Although several commercial packages are available for DNA sequence reading, these are either limited in what they do (for example an operator manually points to each band using a digital-pad) or expensive (for example. laser sequence reading is part of the whole sequencing machine) and highly specialised (all the software does is DNA sequencing). The goal of this project is to incorporate

automatic DNA sequence reading capability from a gel autoradiograph within a general purpose image processing system.

In this chapter, the DNA sequencing process is described, an image processing software development model is presented and each stage in the model is described with processes of DNA sequence reading system development. A general model for image processing and the final system function module structure are given in section 3.4. The control flow diagrams of control modules in the DNA sequence reading system are described in section 3.5.

3.1 DNA sequencing

The genetic information of a living organism is encoded by the DNA contained within every living cell of that organism. DNA itself consists of a chain of nucleotide residues derived from the bases adenine, cytosine, guanine and thymine. Thus the genetic code can be determined by reading the sequence of bases within the DNA, a process referred to as DNA sequencing.

The current methodology of DNA sequencing has its origins in a variety of different fields of nucleic acid enzymology and chemistry. DNA sequencing relies on the incorporation of radiolabelled nucleotide residues into the DNA molecule to be sequenced and the generation of populations of radiolabelled oligonucleotides that begin from a fixed point and terminate randomly at a fixed residue within that DNA molecule. The population of radiolabelled oligonucleotides is separated on the basis of size by gel electrophoresis and the DNA sequence is determined by the examination of an autoradiographic image of the separated oligonucleotides [Hindley, 1983].

The DNA sequence itself is determined by reading the sequence of bands in the four lanes on the autoradiograghic image of the sequencing gel starting at the bottom of the gel and working up, where each lane represents one of the four bases, adenine (A), cytosine (C), guanine(G) and thymine (T). The correct sequence is obtained by noting the most intense band at each level, developing a feel for the spacing [Daivies, 1982].The sequence read from gel autoradiogragh either is used directly, or form the input to a software package which obtains a consensus sequence from overlapping sequences [Elder and Southern, 1987].

Automatic reading of DNA sequencing gel autoradiographs is the purpose of the project mentioned in the thesis.

Figure 3.1-1 gives an example of DNA sequence gel autoradiograph.

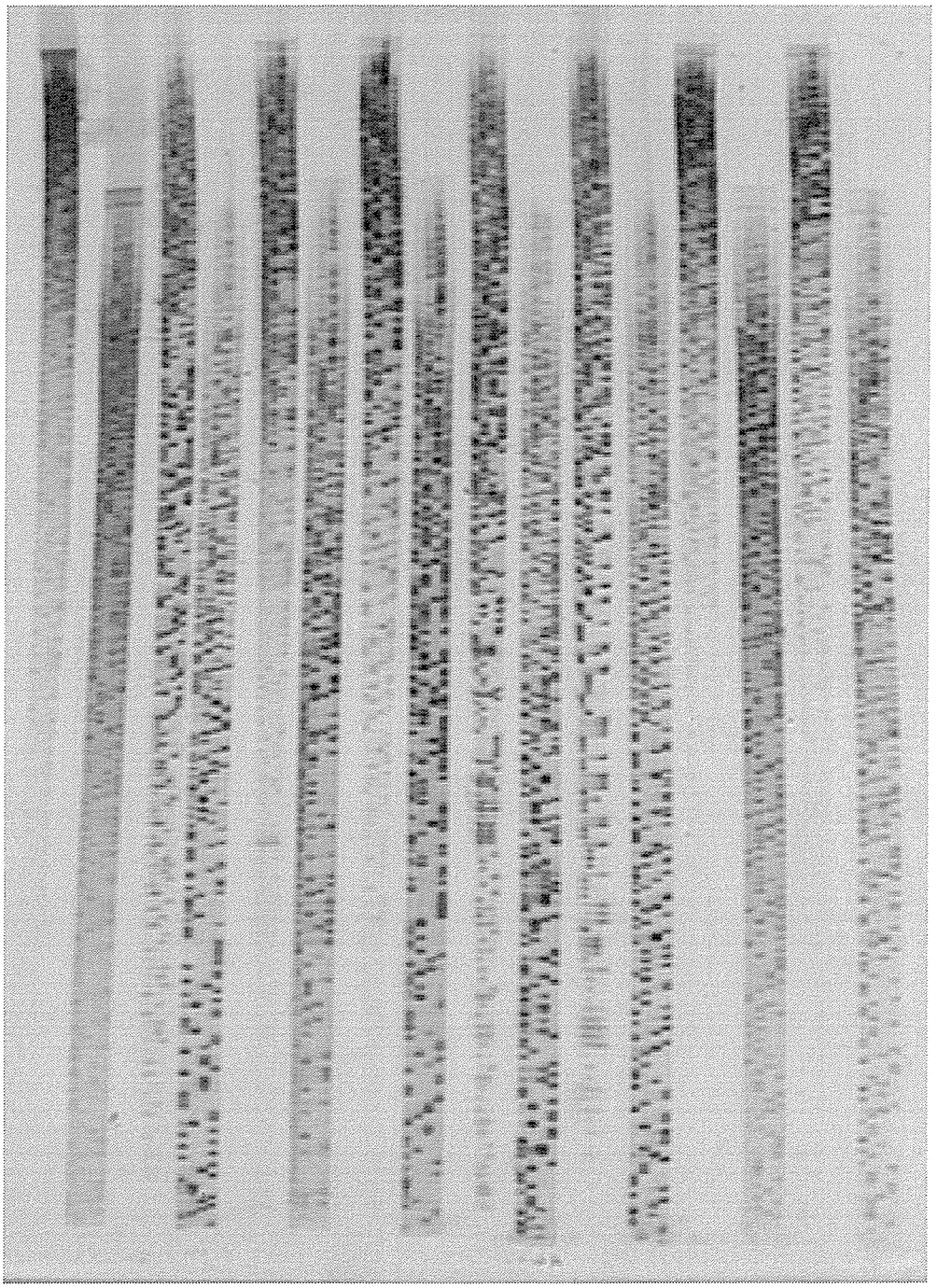


Figure 3.1-1 DNA sequence autoradiograph.

3.2 An Image Processing Software Development approach

Image processing software development is inherently iterative. Since both hardware and techniques are still limited, applying image processing to a particular application relies strongly on checking intermediate results during the development process. The standard waterfall model of software development [Sommerville, 1989] fits most of the stages in image processing software development. However the feedback from implementation and system testing to detailed design is much stronger in image processing than a lot of other software development.

An approach based on the waterfall model combined with exploratory programming were used to develop the DNA sequence reading system. Figure 3.2-1 outlines this approach.

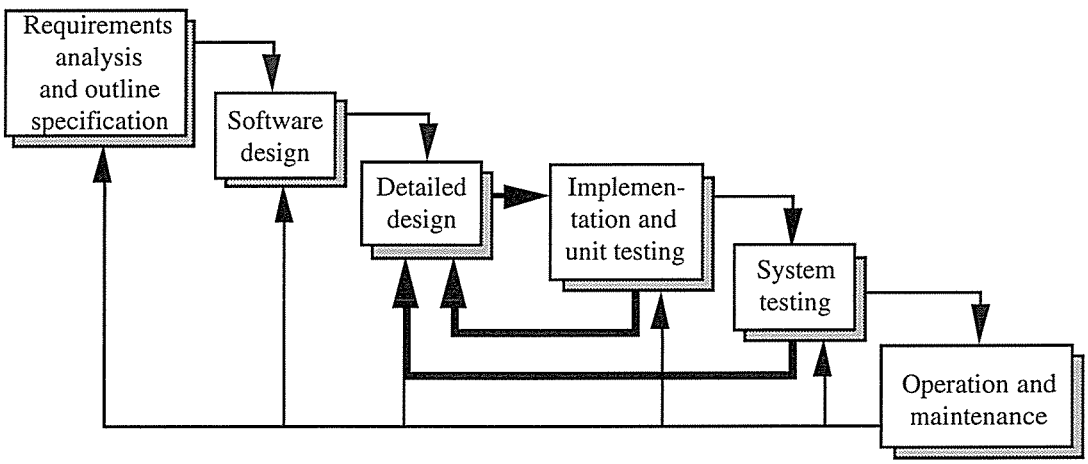


Figure 3.2-1 Image processing software development model.

The general requirements analysis stage includes feasibility study, requirement definition and specification, and then system specification. The feasibility study specially relies on image processing hardware and image processing techniques. The detailed requirement cannot be defined at this stage, only outline specification is possible. A general model for image processing software design can be used in a small system design. Image acquisition, image preprocessing, feature extraction and post data processing are the top modules in the structure diagram of any image processing software. Different applications require different middle modules in the structure diagram. Detailed design can be defined as selecting the individual image operations. Implementation can be defined as supplying the series of operations with parameters. Detailed design depends on implementation. A depth-first search is used to select a

suitable operation path (the algorithm) from a operation tree for each image processing function module. Implementation generates preliminary programs and then decides the operation parameters, instead of deriving the program from low-level specification. System testing demonstrates the adequacy of the system rather than the correctness of the programs. When a larger number of images are tested, various complicating factors and special cases may be introduced, so that the detailed design and implementation must be refined repeatedly. The development stages overlap and feed information to each other.

The following sections will give detailed descriptions of each stages in the image processing software development model with the DNA sequence reading system development processes.

3.3 Requirement analysis

It is important to make a distinction between user need and software requirement. Users may need a software system to support their task, but the problems to be solved must be collected and analysed. A software requirement is a property that the software system must satisfy. Feasibility, requirement definition and specification, and system specification make up general requirement analysis.

The feasibility study of a image processing software relies on image processing hardware and image processing techniques mastered by people and available to the developers. The detailed requirements are hard to define but an outline specification is possible. A prototype of exploratory programming is often required if it is a new application area to the software developer.

In developing a system for DNA sequence reading, VIPS was chosen as the development environment. Exploratory programming is used to determine the feasibility of using image processing techniques and to outline the possible accuracy. A straightforward part of a DNA sequence autoradiograph is captured into a sample image (see Figure 3.3-1). After thresholding the image, most of the bands on the DNA sequence image are segmented. It is possible to increase the readability of the image by preprocessing and enhancing the band features. The positions of the centre of gravities of the segmented bands may be used to get the sequence of the bands.

The accuracy is very important in the system. The accuracy should be at least as good as that from manually reading the bands. Uncertainty codes may be used, but missing or false bands are not acceptable. It is possible to increase the accuracy by image

preprocessing, but it is hard to affirm how well the process will be at this stage. A manual correction step may be added at the end of the algorithm to make the system more acceptable.

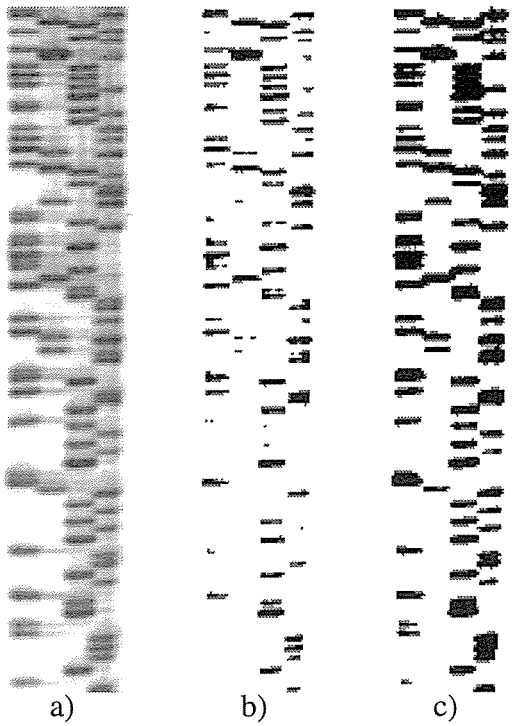


Figure 3.3-1 Exploratory programming results
a) a sample DNA sequence;
b) segmented image with threshold 110;
c) segmented image with threshold 150.

Besides general image processing operations, DNA sequence reading may require individual operations, for example, sorting band positions into a DNA sequence. VIPS is extendable, allowing new operation commands to be added for an application system. So, the project of DNA sequence reading is created.

3.4 Software design

Design is a creative process. Software design is the process of representing the functions of each software system in a manner which may readily be transformed to one or more programs. Image processing software design requires that the software developers have strong experience and extensive background in image processing. A general model of image processing software design may be used to form the top modules of a small image processing application. Figure 3.4-1 gives a general model of image processing software design [Bailey, 1988].

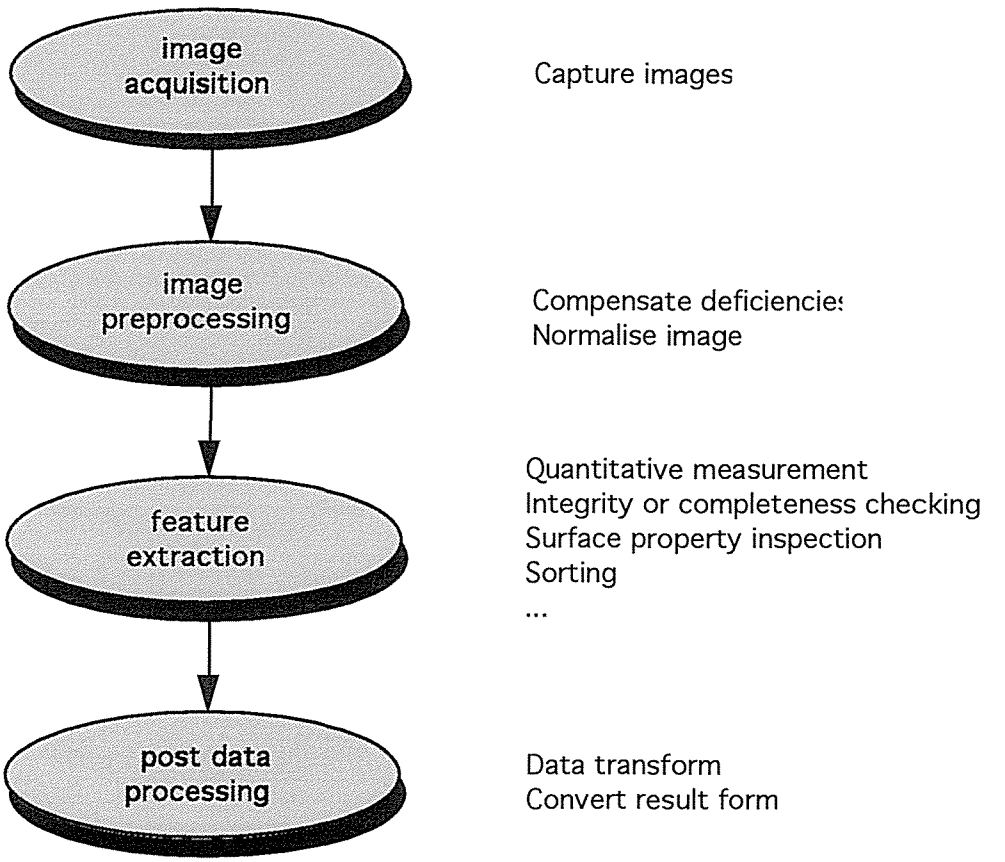


Figure 3.4-1 A general model of image processing software design.

The initial DNA sequence reading system top module structure is given in Figure 3.4-2. The DNA sequence images must be captured from a DNA sequence autoradiograph. Contrast enhancement is the main part of preprocessing, which enhances the flat bands in the image. The information to be extracted is the position and order of the bands in the image. For this, the bands are filtered and detected. The purpose of the system is to obtain the DNA sequence. The positions of the bands are scanned and then sorted. The order of the bands is transformed into the order of the bases in the DNA sequence.

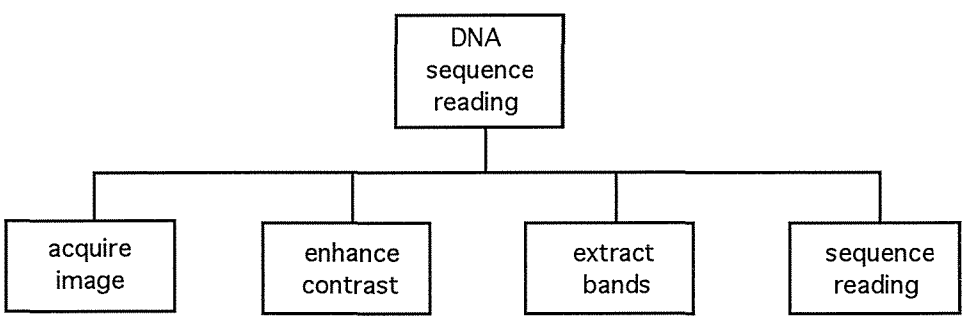


Figure 3.4-2 DNA sequence reading system top module structure.

After more images were tested, an automatically process of all subsequences in a captured image is added into the system. The final system function module structure is refined as Figure 3.4-3.

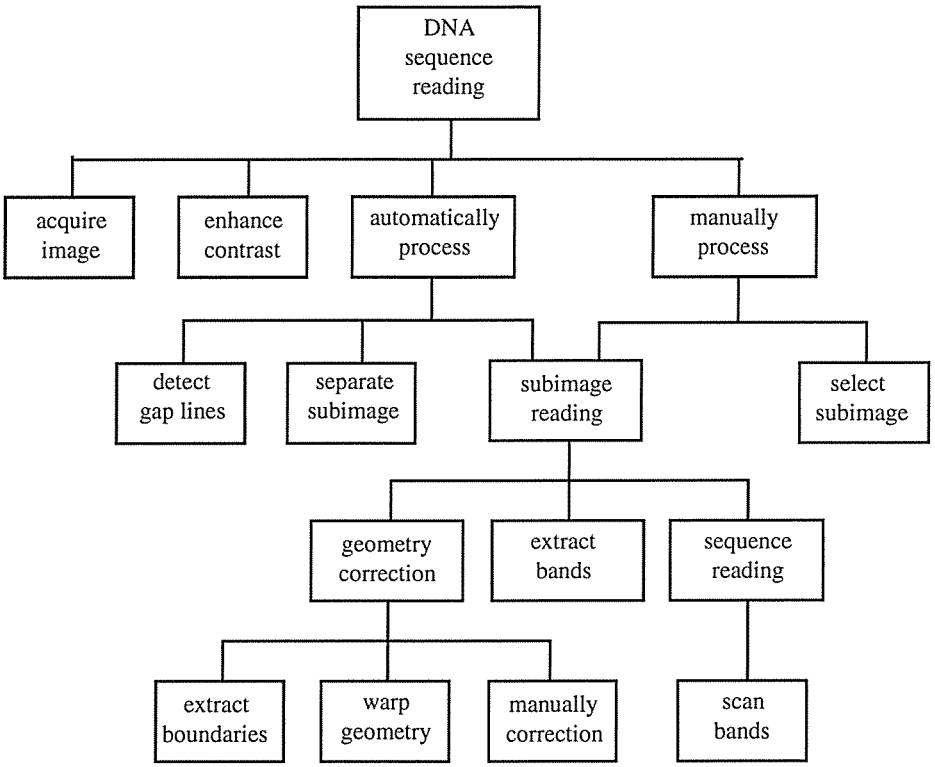


Figure 3.4-3. DNA sequence reading system function module structure.

3.5 Detailed design and implementation

Detailed design and implementation are closely related to problem solving. Each function module is designed into an algorithm. In an image processing context, this involves developing an algorithm for each function module to obtain the desired resultant image or data from a given image. The detailed design depends on implementation.

There is little or no underlying theory that may be used to determine the sequence of image processing operations through which a desired solution image can be obtained from a given image. In practice, an operation sequence (an algorithm) is frequently decided by trying out operations and checking the resultant images [Bailey, 1988]. Detailed design may be defined as selecting the operations while implementation may be defined as determining suitable parameter values. The operation selection relies on implementation. In other words, algorithm development is largely a heuristic process.

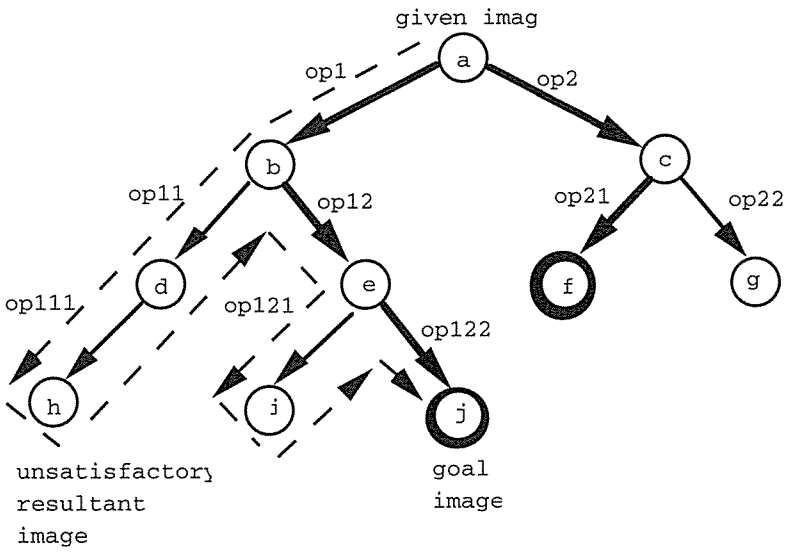


Figure 3.5-1 Searching an image processing operation path.

There may be several operation paths for solving the same problem. An operation search tree for an image processing function module may be created by a experienced image processing algorithm designer. Searching for a path that gives satisfactory results is the purpose of detailed design. Figure 3.5-1 represents a simplified image processing operation search tree with depth-first search. The root of the tree is a given image to be processed. The median nodes are median resultant images. The arc between two nodes is a single image processing operation (or embedded algorithm). The leaves of the tree are resultant images. Some resultant images are not successful. The goal images are satisfactory (node j, f).

The control module designs of the DNA sequence reading system are described briefly below. The image processing module algorithms will be given in Chapter 4.

DNA sequence reading module. The *DNA sequence reading* module is the main system control module. The image to be processed may be either captured directly from a gel autoradiograph by VIPS through a camera, or loaded from image data files which have been captured previously. The user is offered the selection of the acquisition mode, or to exit from the system. When loading from a file, the input image file name is checked until the file name is acceptable. As preprocessing, the *enhance contrast* module is called to make faint bands readable and enhance the contrast between band lane sets and the inter-set spaces. Since each lane set represents a different part of the DNA sequence, the lane sets must be separated into different subimages and then be processed individually. Most of lane sets may be separated

automatically. If the image can not be separated well, for example if the space between the lane sets is too small, the subimages may be selected manually. The process type is selected by user. The control flow diagram (CFD) of the system control module is given in Figure 3.5-2.

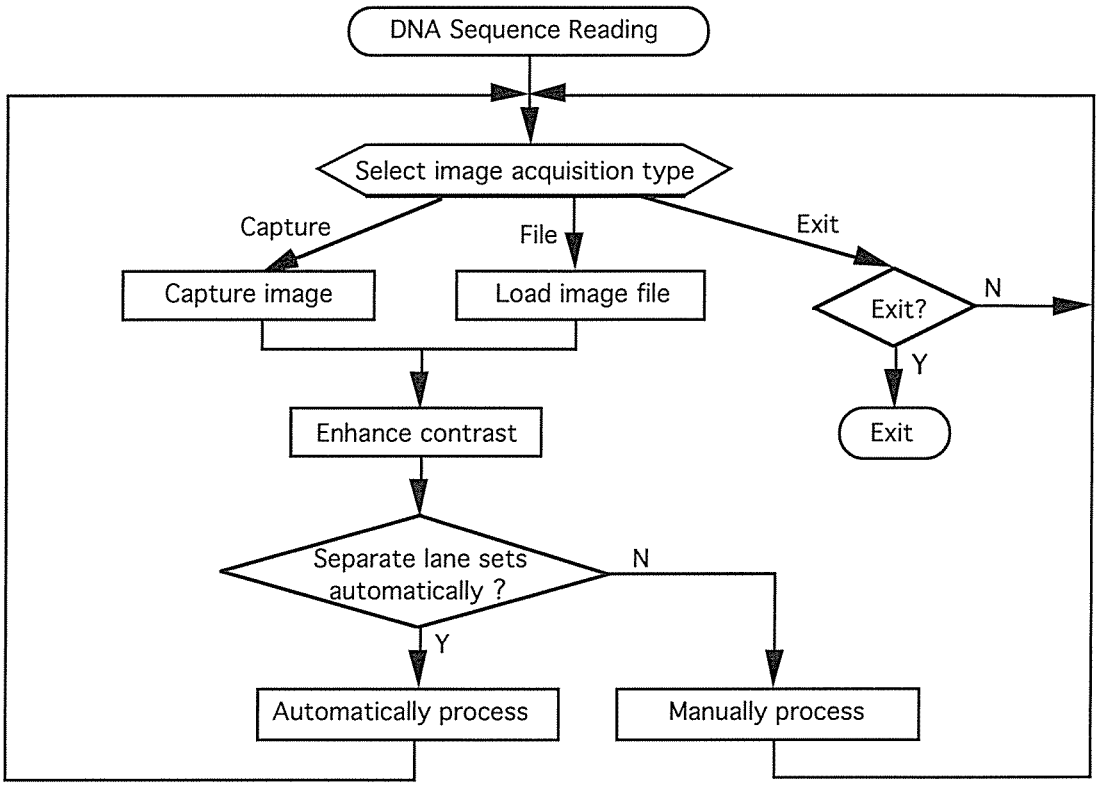


Figure 3.5-2 System control module CFD.

Automatically process module. The *automatically process* module separates each subimage automatically before processing each subsequence. *Detect gap lines* module detects the spaces between lane sets and then obtains the separation key points, the leftmost point and the rightmost point of each gap centre line. The *separate subimage* module successively extracts subimages using the separation key points of the gap line. If the lane set is not extracted satisfactorily, the user may select the lane set manually for the subimage. The *subimage reading* module processes each subimage to give the corresponding DNA sequence. Figure 3.5-3 gives the *automatically process* module CFD.

Manually process module. The *manually process* module is necessary to handle the cases where the image cannot be separated automatically. For example, if the DNA sequencing reactions are loaded into the gel without gaps between each set of four sequencing reactions, the system cannot locate separation points, preventing automatic lane set separation. The *manually process* module prompts user to successively select

each lane set manually before reading the subsequence. This is repeated until all the required subsequences are processed. Figure 3.5-4 gives the *manually process* module CFD.

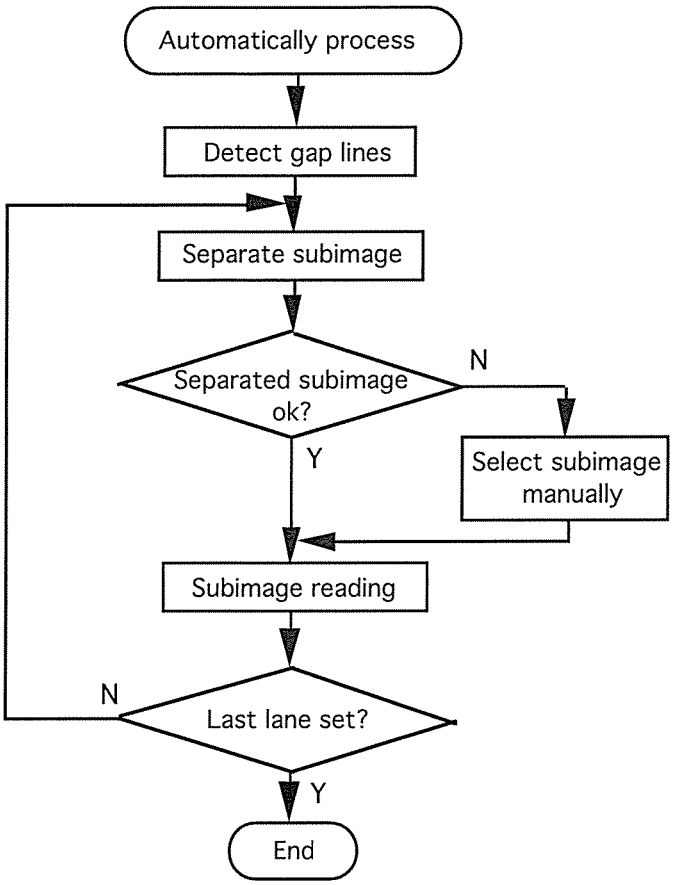


Figure 3.5-3 Automatically process module CFD.

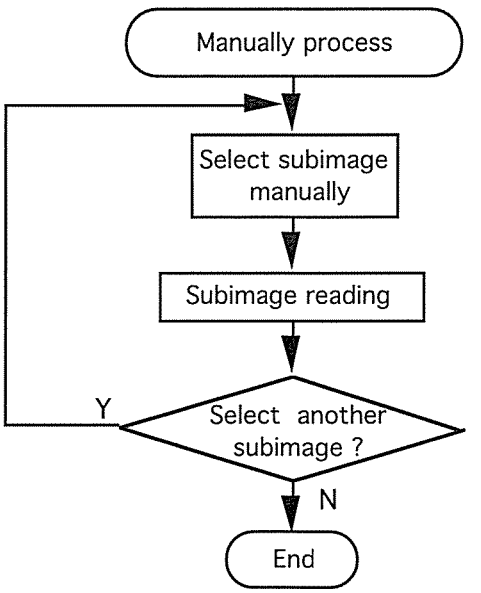


Figure 3.5-4 Manually process module CFD.

Subimage reading module. The *subimage reading* module is called by both *automatically process* and *manually process* modules. The *subimage reading* module processes a subimage containing one lane set which represents one subsequence. The *geometry correction* module corrects any irregularities in the geometry of the lane set, which call the *extract boundaries* module and the *warp geometry* module. The *extract boundaries* module determines the left and right boundaries of the lane set as required for geometry warping. The *warp geometry* module uses the left boundary of the lane set to straighten the set, and the right boundary of the lane set to align the bands between the lanes with a predefined width. The *extract bands* module removes the background, enhances the band features and then obtains the band positions by band detection. The *scan bands* module scans the band positions to give the DNA subsequence and then joins subsequences into a longer sequence. Figure 3.5-5 gives the *subimage reading* module CFD.

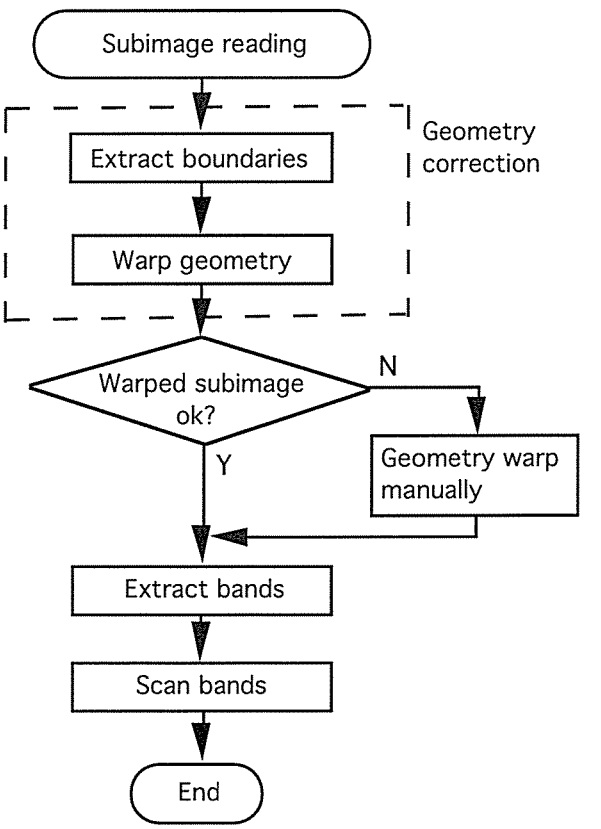


Figure 3.5-5 Subimage reading module CFD.

3.6 System testing and maintenance

System testing integrates and tests the programs as a complete system to ensure that the software requirements have been met. Image processing system testing demonstrates the adequacy of system for the application rather than the correctness of the programs.

The initial exploratory programming processes a single, or a small range of representative images. A larger range of images must be tested to verify that the system is adequate. Various complicating factors and special cases may be introduced, and the system modified to handle these. If necessary, the execution time may be reduced by modifying the algorithm to reduce or eliminate processing bottlenecks. The information obtained is fed back to previous phases, especially the detailed design and implementation phases. The algorithms must be refined repeatedly and the operation parameters must be adjusted again and again until the system is satisfactory. It is possible that more function modules (and algorithms) are needed to overcome the deficiencies.

After initial detailed design and implementation of the DNA sequence reading system, more DNA sequence images had been captured and tested. It was found that the band lane sets on the images are often much darker than the gaps between the lane sets. Because each lane set will be separated into individual subimage to be read respectively, it is possible to separate each lane set automatically instead of having to select each lane set manually. The gap line detection module and lane set separation module are added into the system to accomplish this. In the case of some unsuccessful images, the manual processes are still kept in the system as an option. If the designer has good skill and experience on similar tasks, the automatic lane set separation process should have been considered at requirement analysis stage.

Maintenance is the longest life-cycle phase. The system is put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life-cycle, improving the implementation of system units and enhancing the system's services as new requirements are perceived. Maintenance of image processing system is often needed for more special cases of images which have not been covered in system testing.

Figure 3.6-1 shows the DNA sequence reading system which is running on a Macintosh.

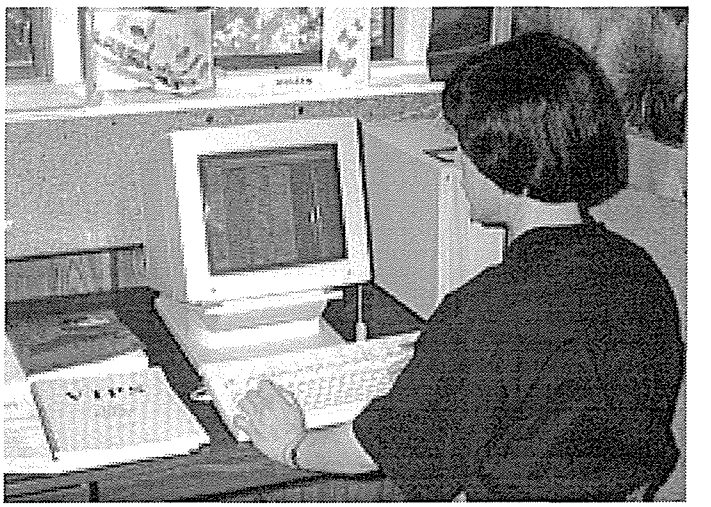


Figure 3.6-1 DNA sequence reading system runs on Macintosh.

Chapter 4

DNA Sequence Reading Algorithms

The DNA sequence reading system is developed using a combination of exploratory programming and the waterfall model. The system software has a modular structure. The system function module structure and the design of each control module in the function module structure have been described in chapter 3.

In this chapter, the algorithms of image processing modules in the function module structure of the DNA sequence reading system are described in detail. The data flow diagram (DFD) of each module is given with explanation of each processing step in separate sections. Expressions for the associated mathematical operations and the corresponding VIPS command lines are given in appendixes I and II.

4.1 Image acquisition

Gel autoradiographs are commonly used for DNA sequencing in genetics research (see Figure 3.1-1). An image is captured from an autoradiograph with sufficient resolution to give good contrast between adjacent bands. The impulse noise in the image is smoothed and the normalisation of the background is carried out. The image to be

processed may be captured directly by VIPS through a camera or loaded from image data files which have been captured previously. Figure 4.1-1 is the *acquire image* module data flow diagram.

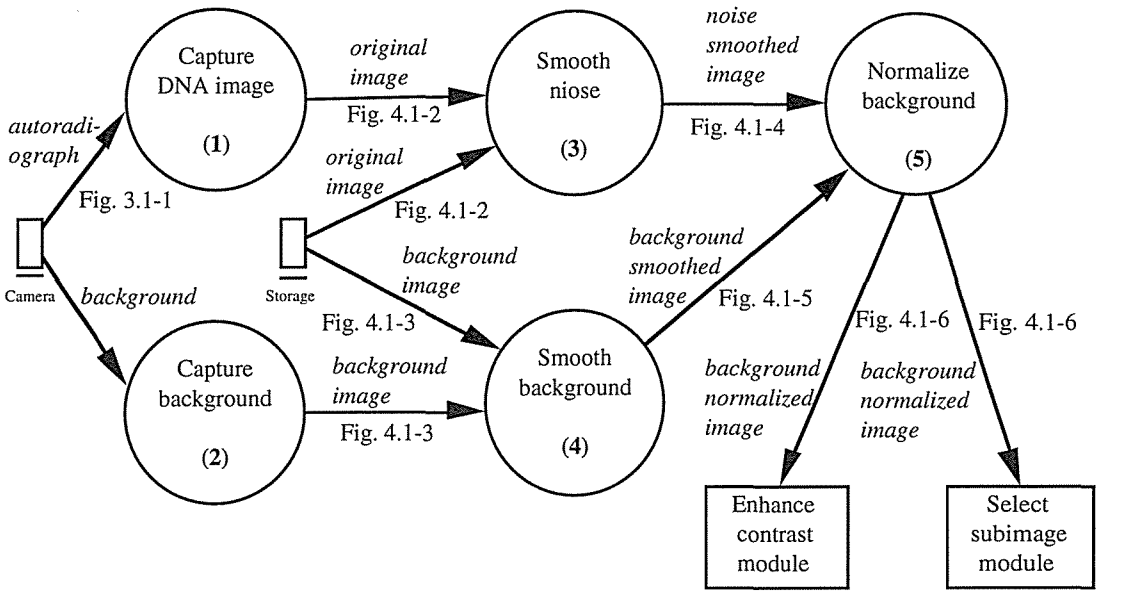


Figure 4.1-1 Acquire image module DFD.

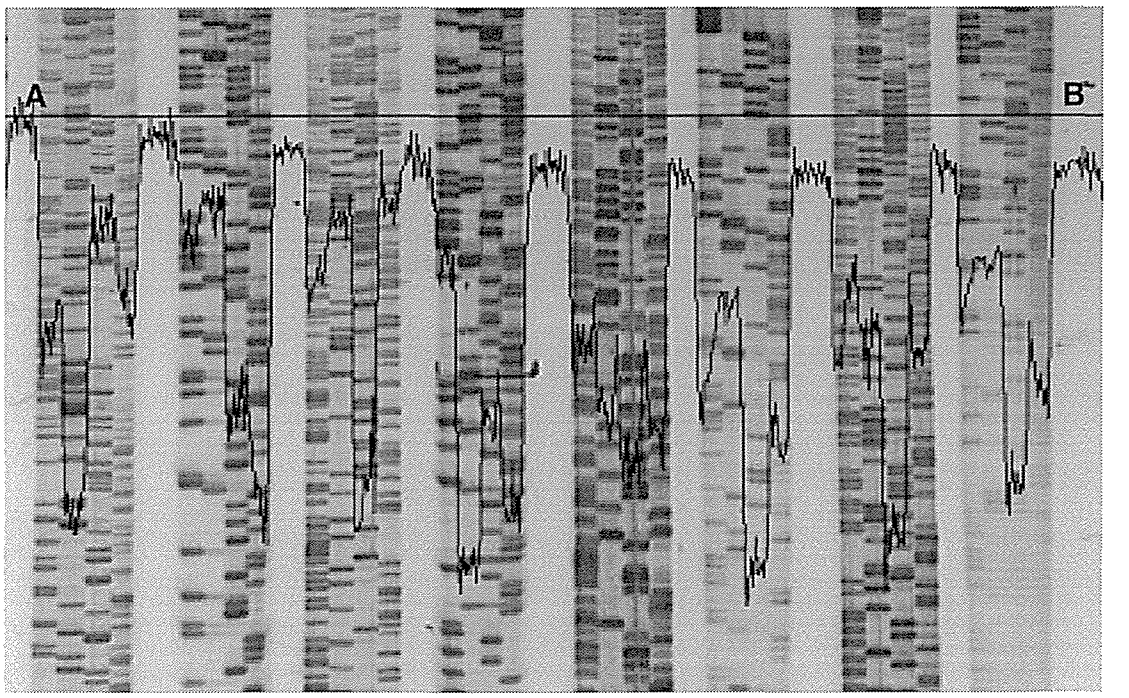


Figure 4.1-2 A captured DNA sequence image with profile.

(1) Capture DNA image. The autoradiograph is placed on a light box to give good contrast. A transmission image of the autoradiograph is acquired through a solid state

video camera (see Figure 2.2-2). The camera limits the image resolution to 512×512 picture elements (pixels). A large gel autoradiograph therefore contains more information than can be processed readily in a single image. For this reason, several overlapping images are obtained of the autoradiograph, with the resulting subsequences merged after processing. The size of each subimage is limited by sampling considerations.

The smallest features of interest in the image are the bands in the lanes. The Whittaker-Shannon sampling theorem [Gonzalez and Wintz, 1987]) requires that adjacent bands have a minimum spacing of 2 pixels (one pixel for the band and one for the gap between the bands). This gives a best case lower limit on the resolution — if the bands are half way between pixels the contrast will be very low. A band spacing of at least 3 pixels is required to detect the individual bands reliably. However, the bands are not spaced evenly along the length of the gel. The spacing is maximum at one end, and decreases to a minimum at the other end. In practise, the bands get so close together that the sequence past a certain point is not even readable manually. Therefore, the image size is determined by the spacing at that readable limit.

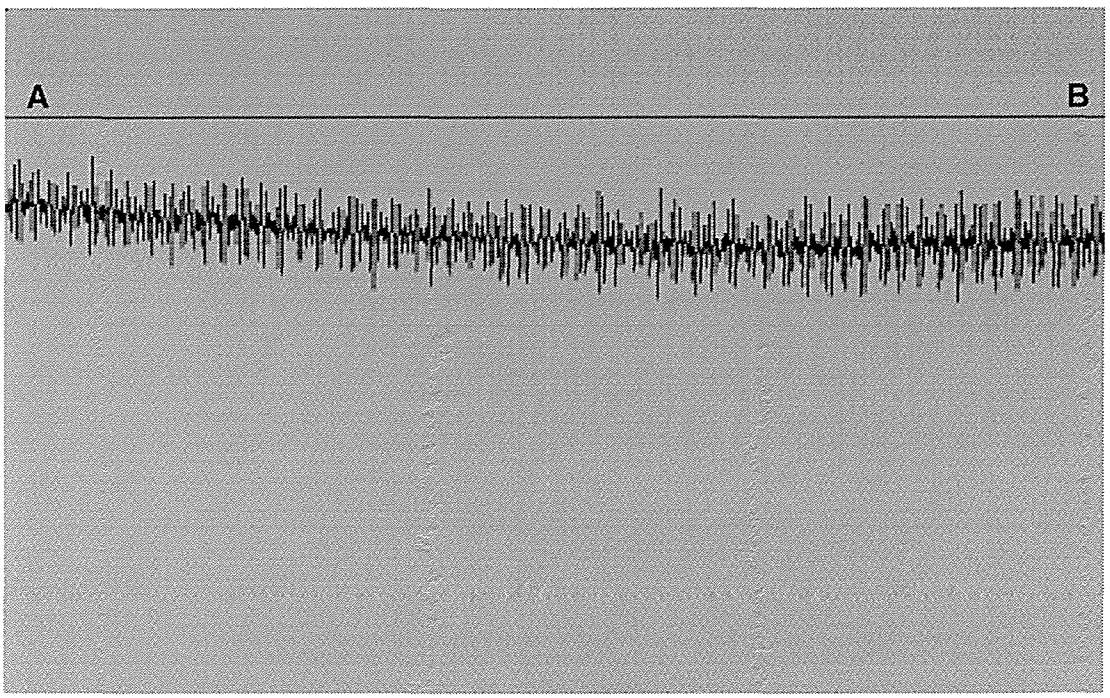


Figure 4.1-3 A captured background image with line profile.

A typical autoradiograph with a readable area of 250×350 mm may require 6 or more separate images to achieve the required resolution. Figure 4.1-2 shows a typical images captured from an autoradiograph (Figure 3.1-1) and will be used through out this

chapter as an example for most of the image processing steps. The intensity profile along the line A-B is plotted to show the features of the image, which will be used to compare the processed image.

(2) Capture background. Normalisation of the background reduces the variation of the background which is caused from the light box and camera lens. A background image is required for normalising the background. The background image is captured at the same environment as DNA sequence image only without the autoradiograph on the light box. Figure 4.1-3 shows the background image of Figure 4.1-2. The intensity profile along the line A-B shows variation of the background and noise on a captured image.

(3) Smooth noise. The high frequency noise in the captured images may be removed by a lowpass filter before further processing. A local average smooth operation is used to filter the noise in the image. The average smooth operation is a special linear convolutional filter. The weights of the average smooth operation in a moving window (see Figure 2.1-1) are all one. This enables the expression of local average smooth operation, **`BOX AVERAGE`**, to be simplified from **`FILTER LINEAR`** (see Eq.AI-4).

A small window size 2×3 is used to keep the small band features of the image while removing most of the noise. Figure 4.1-4 shows the image, with the profile along line A-B, after smoothing. It can be seen that the line profile shape is much smoother than that in Figure 4.1-2.

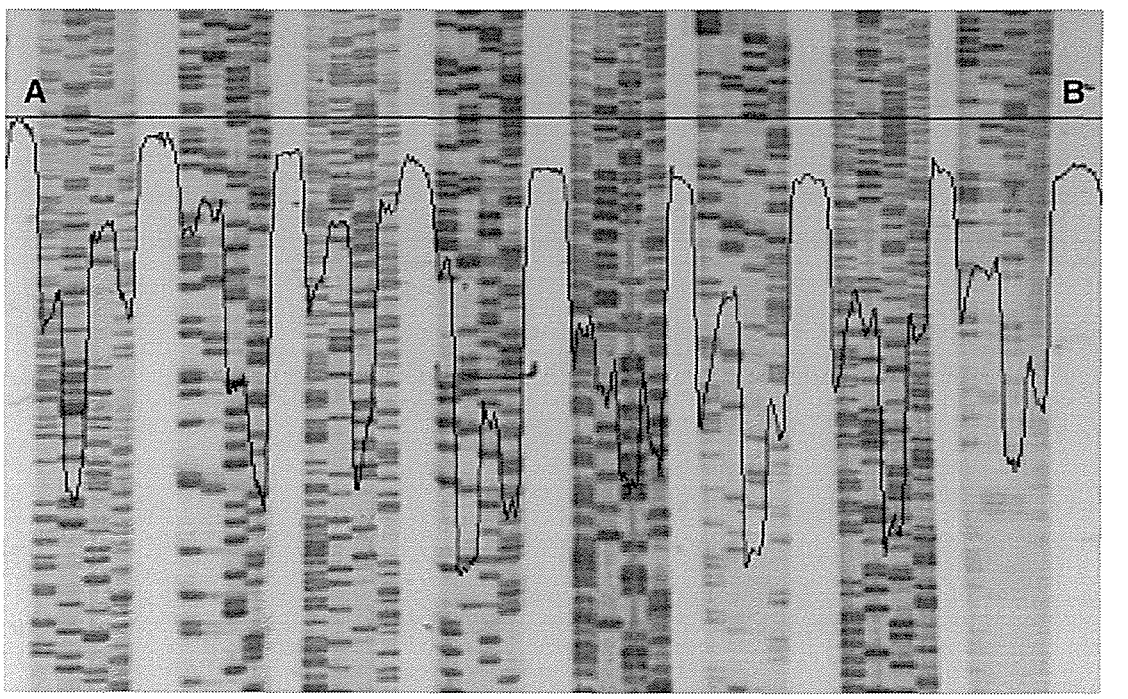


Figure 4.1-4 The impulse noise smoothed image.

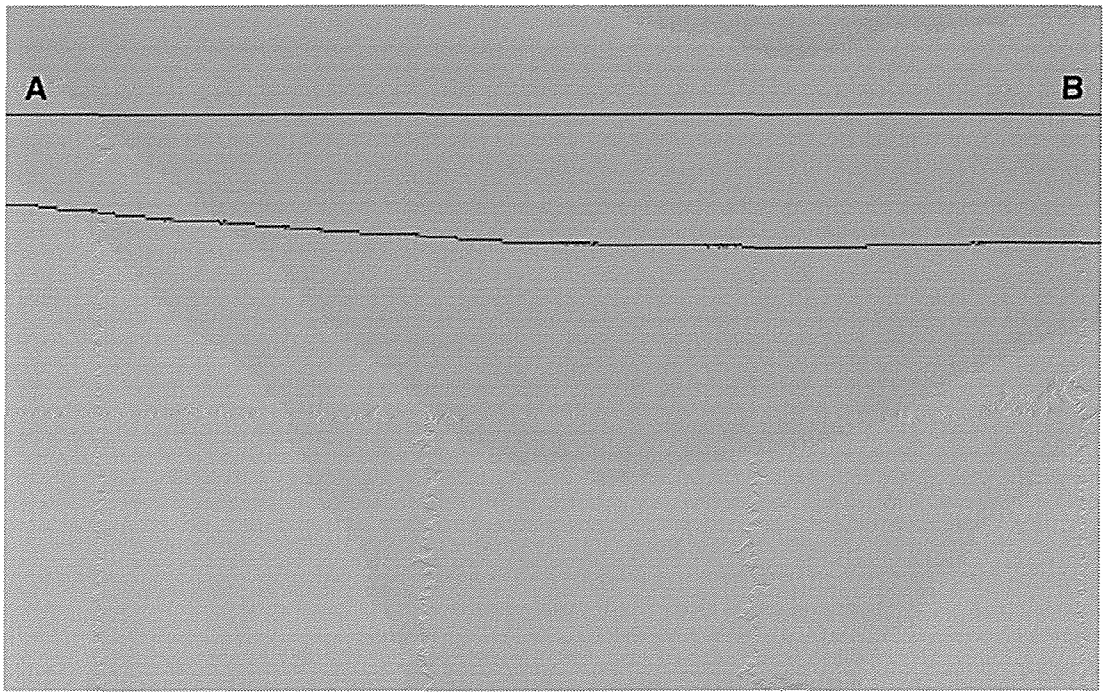


Figure 4.1-5 Noise smoothed background image.

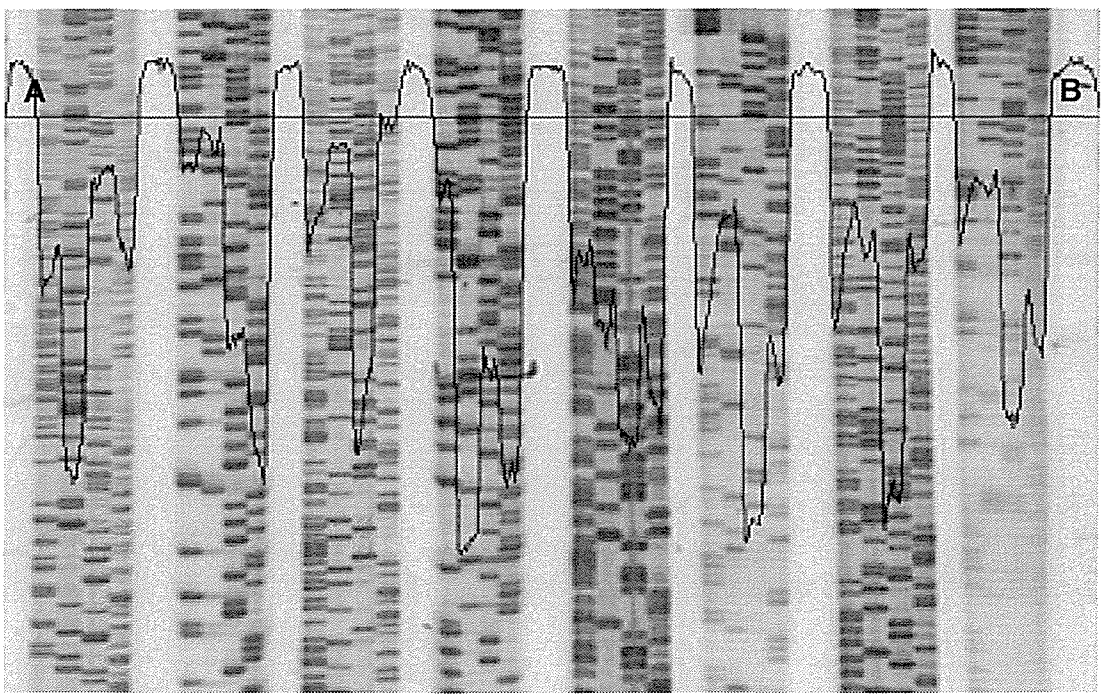


Figure 4.1-6 Background normalised image.

(4) Smooth background. The background image also needs to be smoothed. A larger window size is used, since the background image contains no fine detail of interest and

a larger window gives more smoothing. Figure 4.1-5 shows the background image, with the profile along the line A-B, after smoothing with a 15×15 window. After noise smoothing, the profile shape in Figure 4.1-5 is much smoother than that in Figure 4.1-3.

(5) Background normalise. The background of object image may be variable because of uneven illumination from the light box and camera lens. A division operation is used to normalise the image with respect to the background. Division is a point operation on two images, which divides the object image by the background image and then multiplies by a constant (Eq.AI-14). Each pixel value is calculated from the corresponding pixel values in the input images. In this application, to prevent saturation the multiplier constant k is set to 200. Figure 4.1-6 shows the background normalised image with profile at position A-B, from Figure 4.1-4 and Figure 4.1-5. The intensity of the spaces between the lane sets (the background) is normalised into similar grey levels.

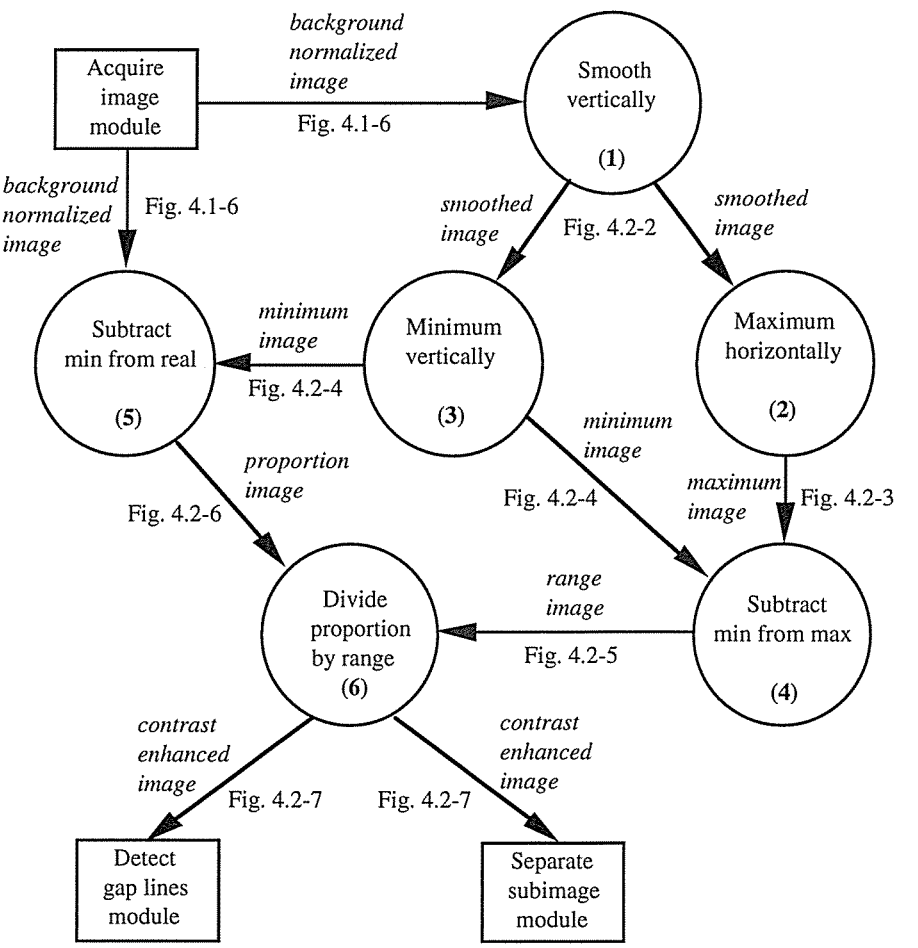


Figure 4.2-1 Enhance contrast module DFD.

4.2 Contrast enhancement

Some of the bands in the image are quite faint, and are not able to be detected reliably. A contrast enhancement step is required to make more faint bands readable. Also contrast enhancement improves reliability of gap position detection (section 4.3) and boundary extraction (section 4.5). A local linear intensity stretch operation **BOX STRETCH** (Eq.AI-7) may be used to enhance the contrast.

In DNA sequence images, the widths of the lane sets may vary from one image to another, and the length of the space between bands is also variable. If the same window size is used on an image with wider lane sets or larger empty spaces, noise may be stretched into a grey level band which will interfere with thresholding in the segmentation step, or with the lane set boundaries in the geometry correction step. If the same moving window size is used on an image with narrower lane sets, or the window size is too big, some bands will not be enhanced well.

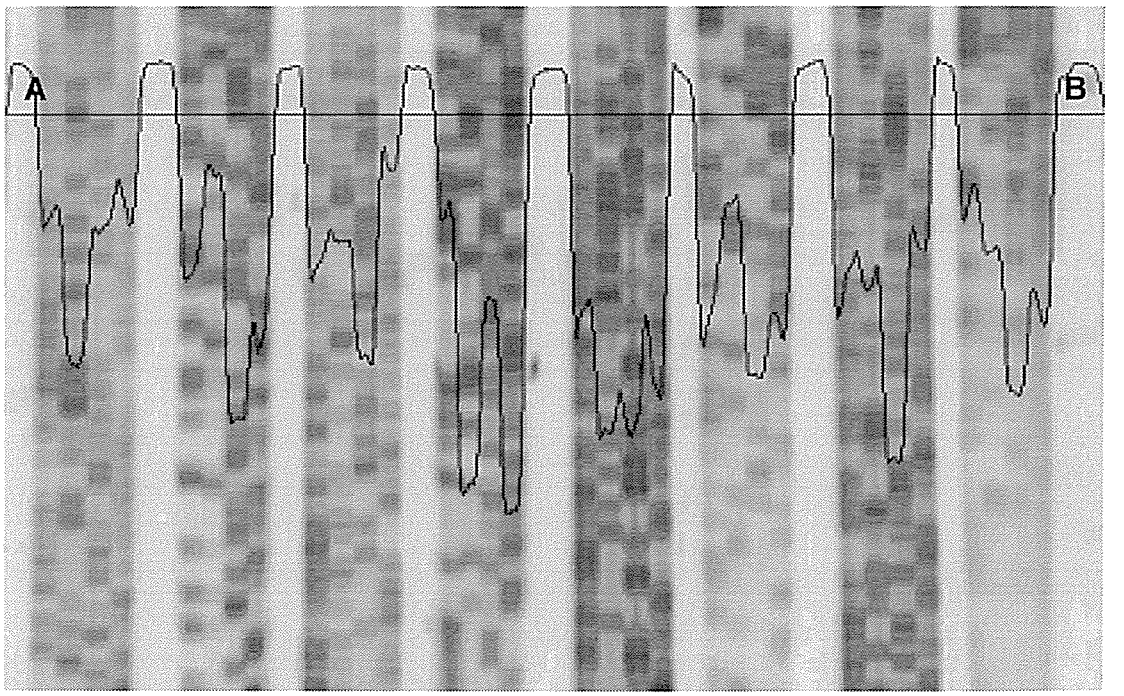


Figure 4.2-2 Vertically smoothed image.

A series of operations are used to enhance contrast without being limited by the band lane set width. The series of operations for contrast enhancement work on the same basis as **BOX STRETCH**, except that the maximum and minimum values are detected using different windows. A maximum image is obtained by **BOX MAXIMUM** using a

horizontal moving window, and a minimum image is obtained by **BOX MINIMUM** using a vertical moving window. Subtraction of the minimum image from the maximum image gives a stretch range image. Subtraction of the minimum image from the original image gives the proportion image. The proportion image is then divided by the range image to get a contrast enhanced image. Figure 4.2-1 is the *enhance contrast* module DFD.

(1) Vertically smooth. An average smoothing operation **BOX AVERAGE** is used to further smooth out any noise which may be presented in the image before finding the minima and maxima. A window size of 15×3 smooths the bands vertically with minimal interference across the boundaries of the lane sets. Figure 4.2-2 shows the vertically smoothed image from Figure 4.1-6.

(2) Horizontal maximum. A maximum image is required to get a stretching range image. Each pixel value of the maximum image is determined by the maximum pixel value in the moving window (Eq.AI-5).

A horizontal 1×80 window is used to obtain the horizontal maximum image. In fact, the maximum image gives the intensities of the gaps between the lane sets. The width of the window was selected at 80, assuming that at least five lane sets are captured in the image. This ensures that the gap pixels will be included in the window even for centre pixels of the lane sets. Figure 4.2-3 shows a maximum image from Figure 4.2-2.

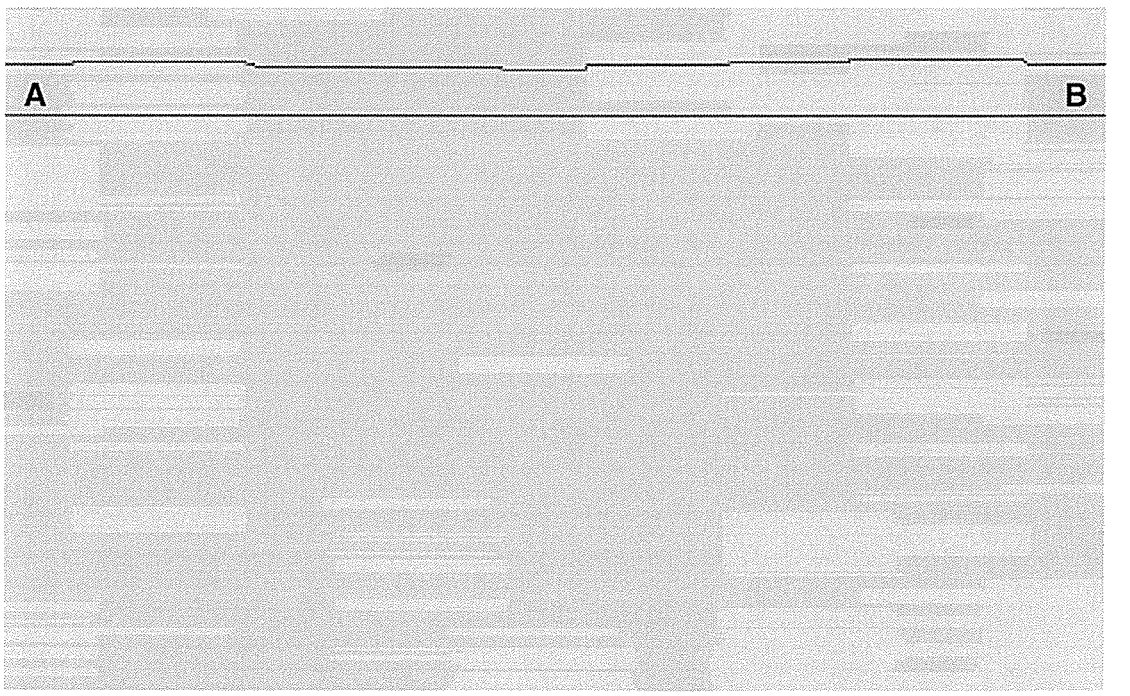


Figure 4.2-3 Maximum image.

(3) Vertical minimum. A minimum image is also required to get the stretching range image. Each pixel value of the minimum image is determined by the minimum pixel value in the moving window (Eq.AI-6).

A vertical moving window is used to extend the bands along the length of the lane. It is possible that no band exists in a very long distance in some lanes. For this reason, a large window size 400×1 is used to ensure that bands are always extended with lanes. Figure 4.2-4 shows the minimum image obtained from Figure 4.2-2.

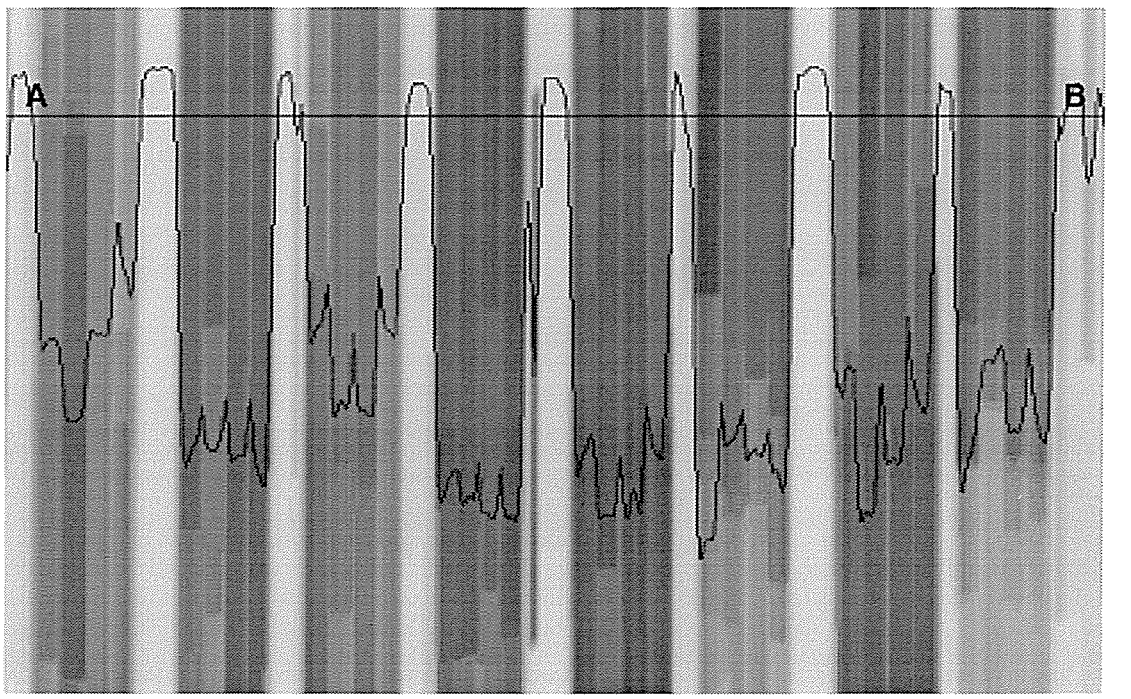


Figure 4.2-4 Minimum image from Fig.4.2-2.

(4) Range image. The range image for contrast enhancement is obtained by subtracting the minimum image from the maximum image (Eq.AI-12).

If resultant value is negative it is set as zero. Figure 4.2-5 shows the range image obtained from Figure 4.2-3 and Figure 4.2-4

(5) Proportion image. The proportion image is obtained by subtracting (Eq.AI-12) the minimum image from the original image, which gets the proportion of each pixel value in the image between the maximum image and the minimum image.

Each pixel value of the proportion image then has 5 added to prevent excessive enhancement between the lane sets where the range is small. This effectively prevents

enhancement of any noise in the gaps. The proportion image is shown in Figure 4.2-6 which is obtained from Figure 4.2-4 and Figure 4.1-5.

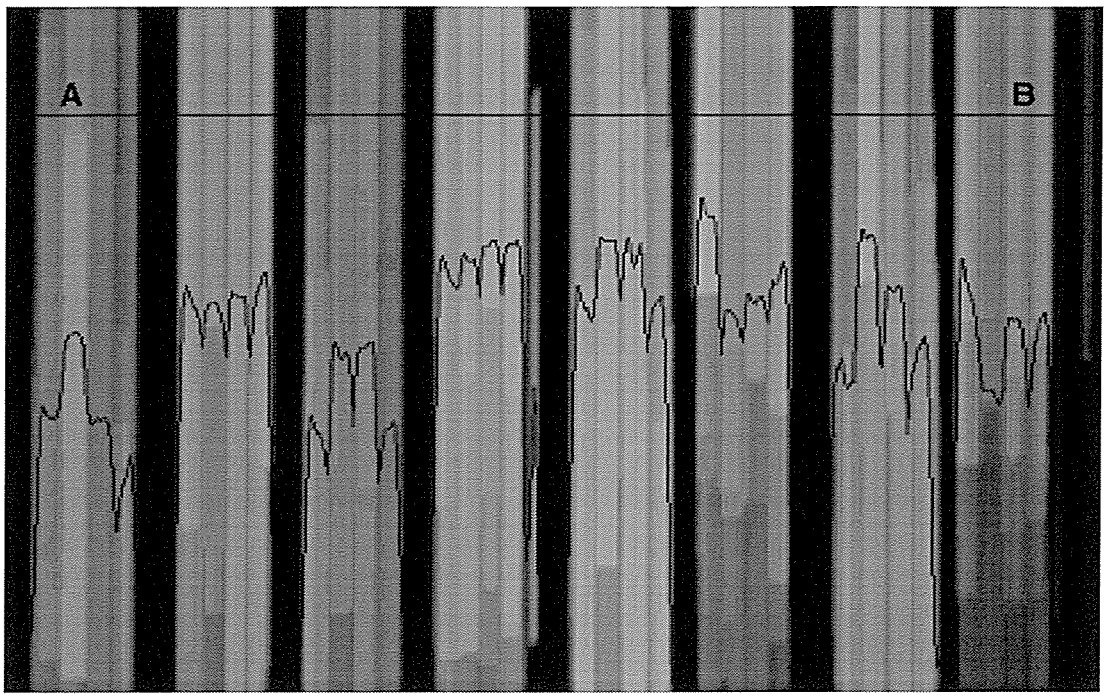


Figure 4.2-5 Range image from Fig.4.2-3 and Fig.4.2-4.

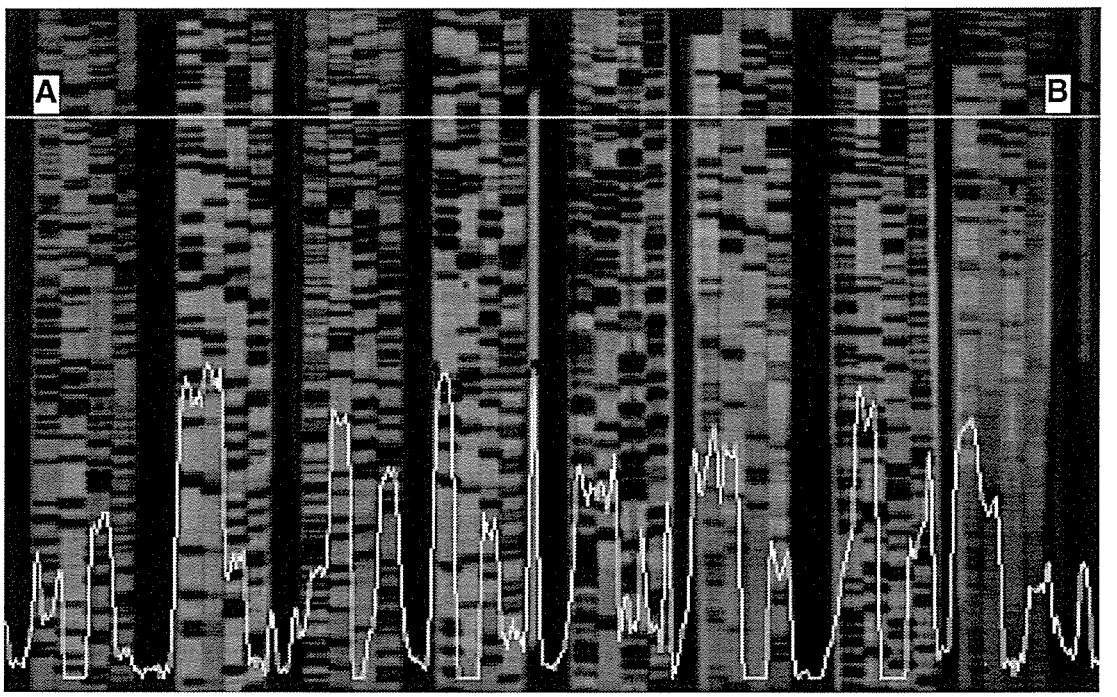


Figure 4.2-6 Proportion image from Fig.4.2-4 & Fig.4.1-6.

(6) Contrast stretch. Then contrast stretching is performed by dividing (Eq.AI-14) the proportion image by the range image. The contrast enhanced image is shown in Figure 4.2-7 with profile of line A-B. The amplitude of the profile is larger than that of Figure 4.1-6. The gaps between lane sets are whiter and the bands are blacker.

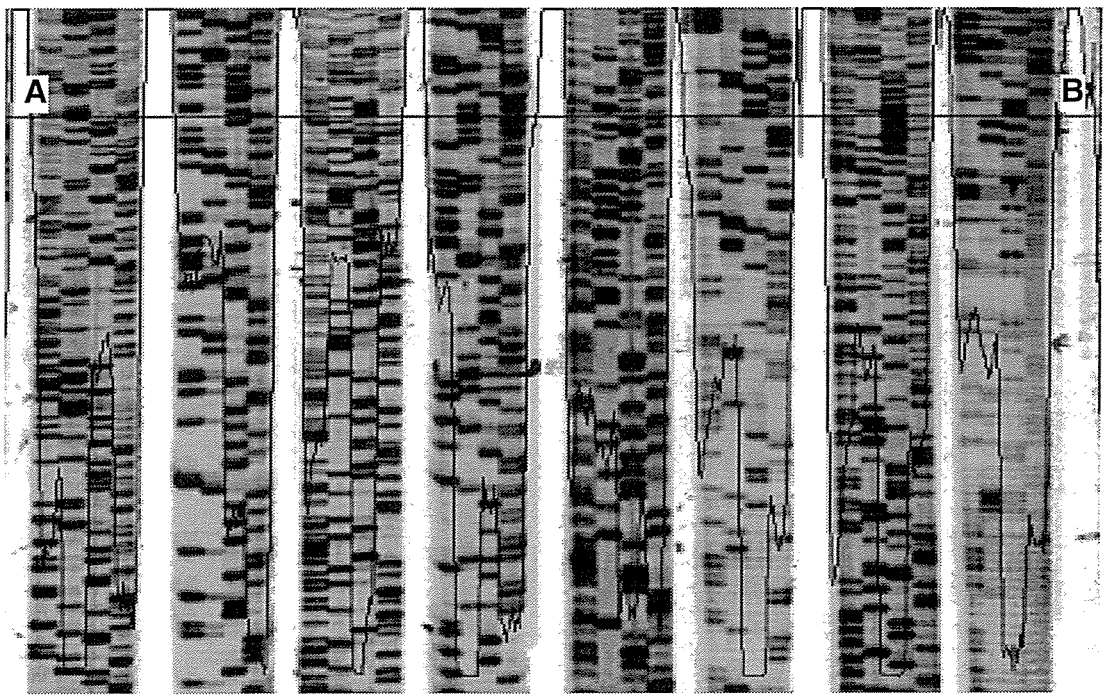


Figure 4.2-7 Contrast enhanced image with profile.

4.3 Gap line detection

There are number of lane sets in a captured image, each of which represents a part of a DNA sequence. The different sets may be from different parts of the same DNA sequence or from different DNA sequences, depending on the samples loaded into the reaction gel. The lane sets must be separated into subimages and then be read individually. If there are gaps or spaces between the lane sets, the gap centre lines can be detected for automatic separation of the lane sets. Smoothing the image vertically makes the lane set boundaries clearer. The smoothed image is then thresholded to obtain a gap image. The gap centre lines can be obtained by thinning the gap areas to a single pixel wide. The gap centre lines then are coded and the separation points detected. The gap line detection module DFD is given in Figure 4.3-1.

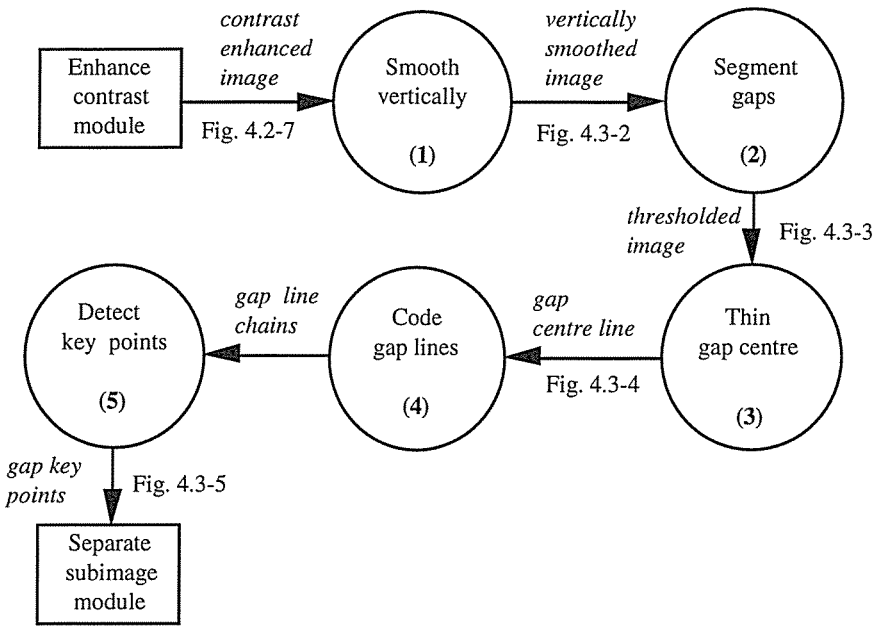


Figure 4.3-1 Detect gap lines module DFD.

(1) Vertically smooth. Gap line detection extracts the vertical features of the image. A vertical smoothing operation (Eq.AI-4) keeps the vertical features of the lanes and masks the effect of the individual bands. This makes the lane set boundaries clearer for segmentation of the gaps. A 40×3 window is used for this smoothing. Figure 4.3-2 shows the minimum image from Figure 4.2-7.

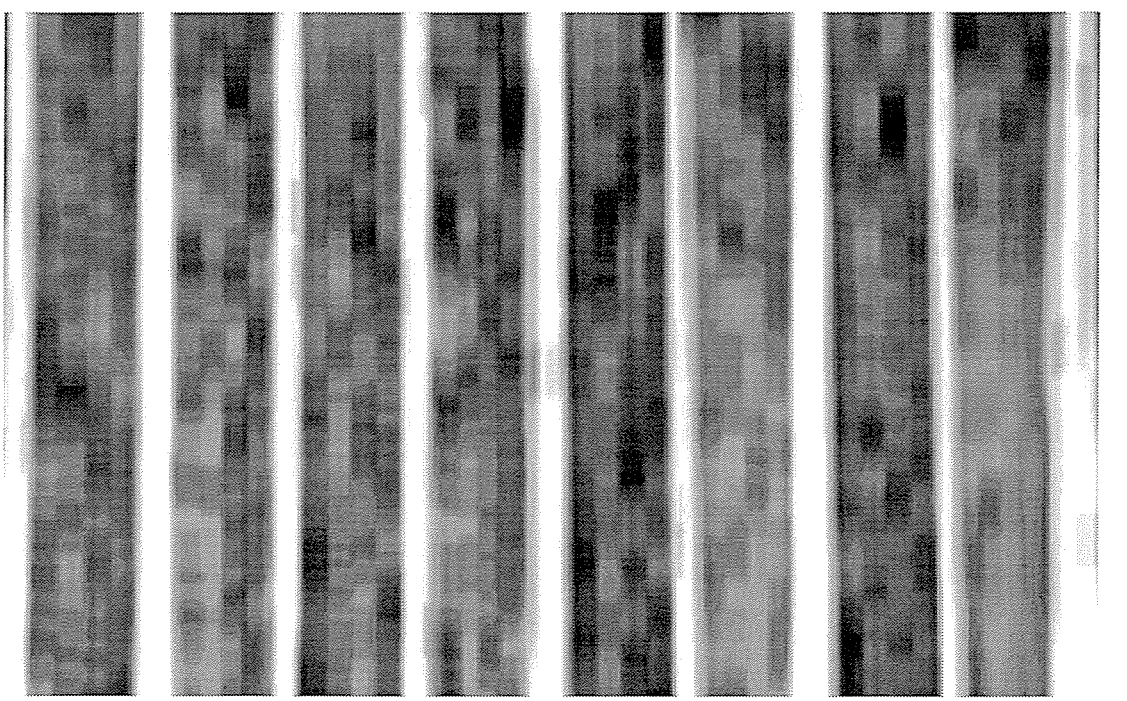


Figure 4.3-2 Vertically smoothed image.

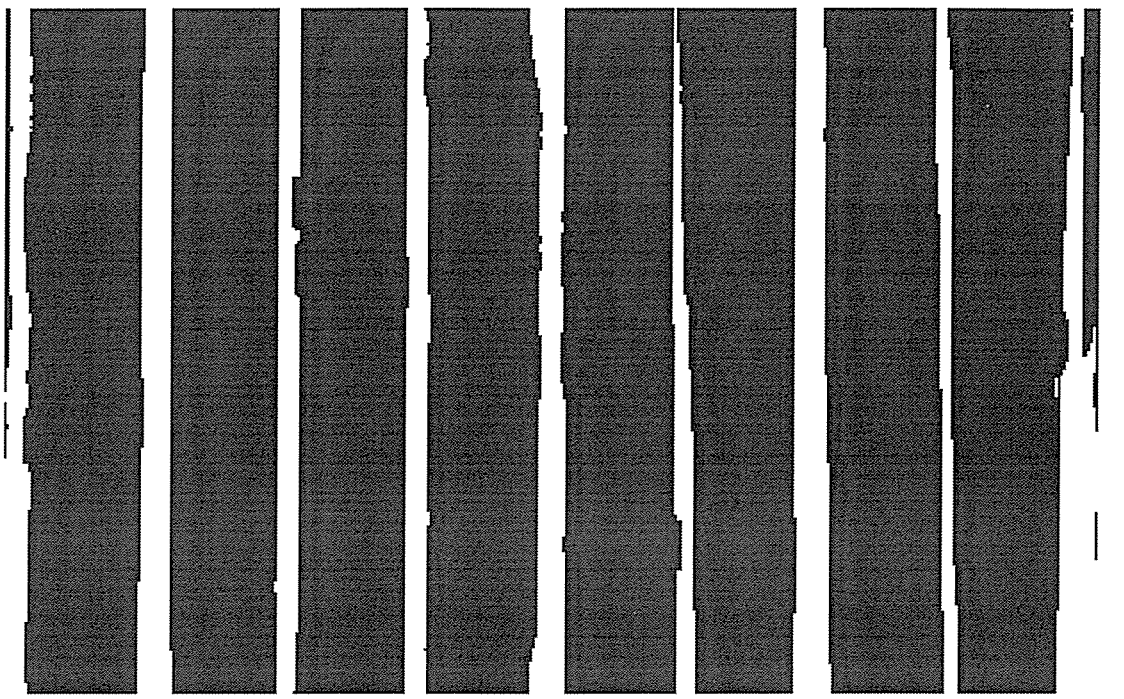


Figure 4.3-3 Gap segmentation.

(2) Segment gaps. The gaps are then segmented by thresholding (Eq.AI-15). The threshold range is set as 235 to 255. In order to obtain smoother gaps, the average smoothing operation and thresholding are repeated. Figure 4.3-3 shows the gap image of Figure 4.3-2.

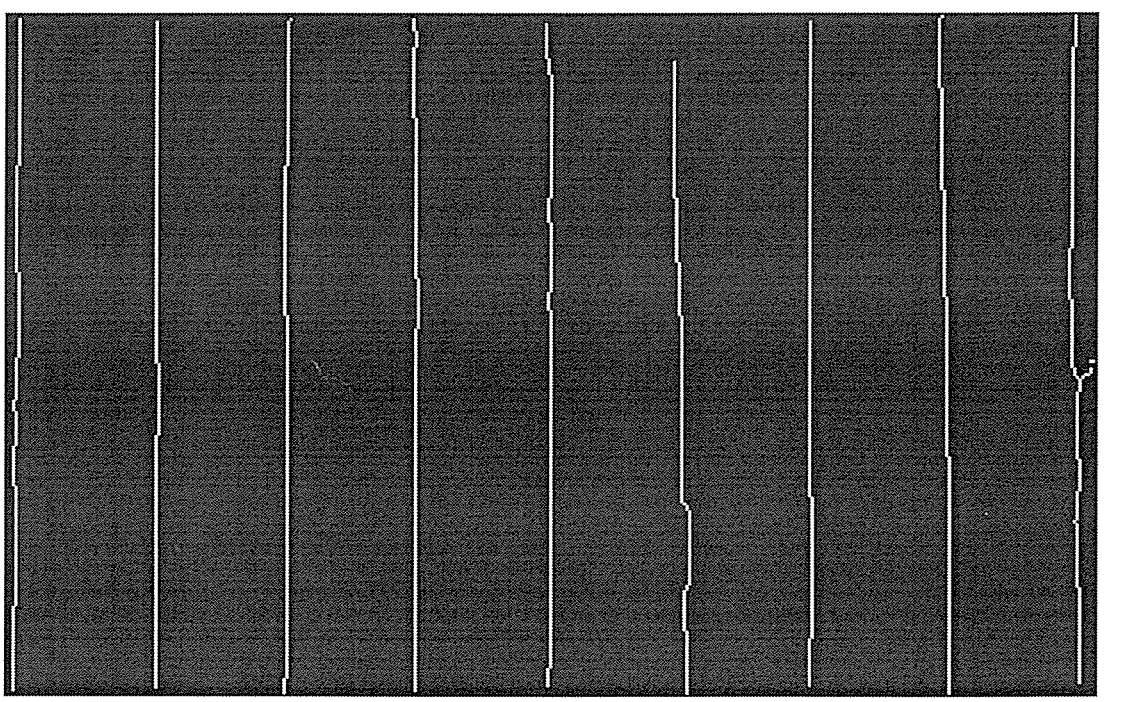


Figure 4.3-4 Gap centre lines.

(3) Thin gap centre. The purpose of gap line detection is to get the gap position for lane set separation. For this purpose, each gap is then thinned into a single pixel wide skeleton . The image is skeletonised by first labelling pixels with their distance from the background. The set of pixels which are symmetrically placed with respect to the background are detected. This set is given a linear structure by removing pixels in such a way that the original topology is preserved [Arcelli and Di Baja 1985]. Figure 4.3-4 shows the gap centre line image which is thinned from Figure 4.3-3.

(4) Code gap lines. The gap lines are then coded to get the pixel position chains. The gap lines are coded by scanning the image until the first pixel of each line is found. The line is then tracked, being converted into a chain code with all positions of the line pixels. Each gap line is tracked separately [Freeman 1974].

(5) Detect key points. The leftmost and the rightmost pixels of each gap centre line are detected. Figure 4.3-5 gives the gap key points of Figure 4.3-4, which are the leftmost pixel and the rightmost pixel of each gap centre line. False line chains or more than one chain for each gap may occur. False position points are detected and removed by distance and length checking. If two position points are very close the one with longer chain is kept.

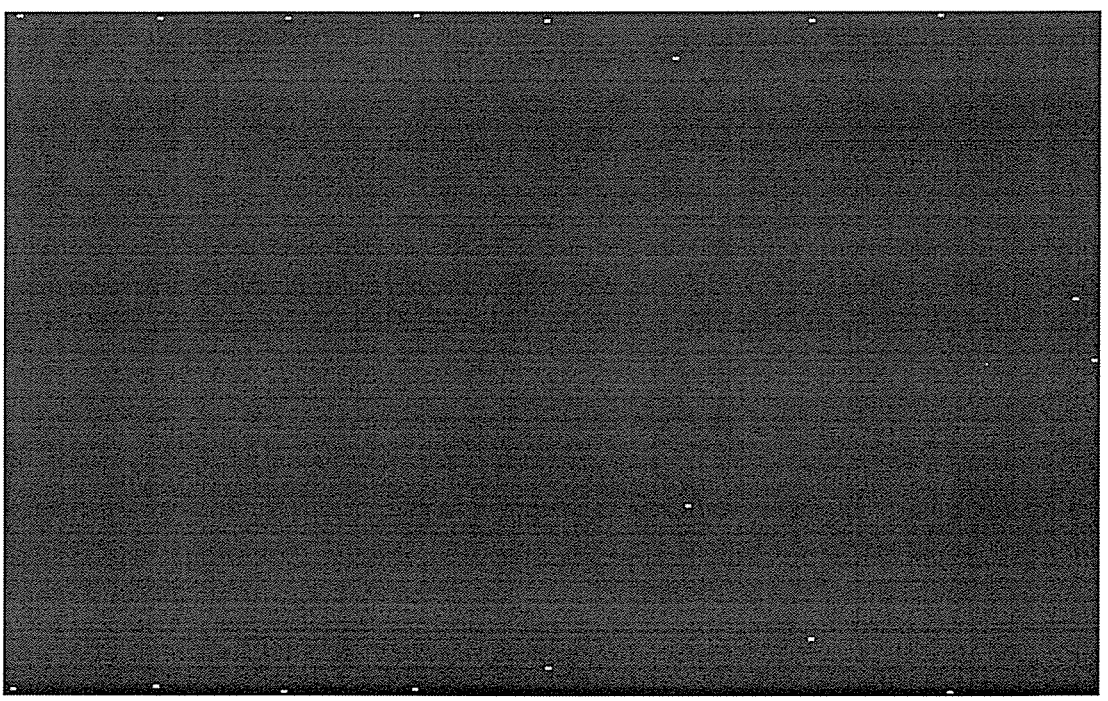


Figure 4.3-5 Gap key points.

4.4 Subimage separation

The lane sets in a DNA sequence image must be split into separate subimages for individual processing. The detect gap line module detects the key points of each gap centre line. The leftmost point of the gap to the left of a lane set is used as the left boundary. Similarly the rightmost point of the gap to the right of a lane set is used as the right boundary. This prevents losing of the image if the lanes are not exactly vertical. All of the pixel data between the left and right boundaries are copied into a separate subimage. Figure 4.4-1 is the separate subimage module DFD.

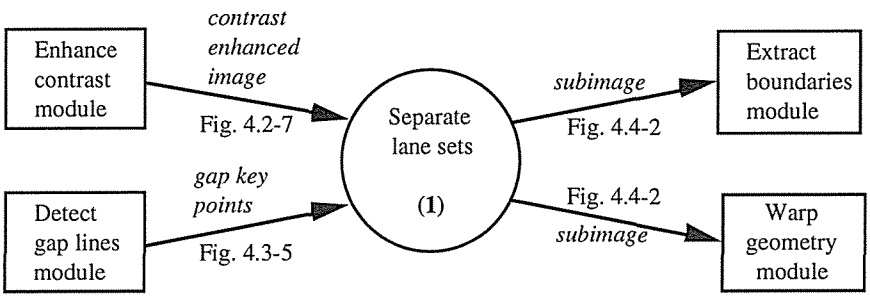


Figure 4.4-1 Separate subimage module DFD.

Figure 4.4-2 shows one of the lane sets of the example image, which is separated from Figure 4.2-7.

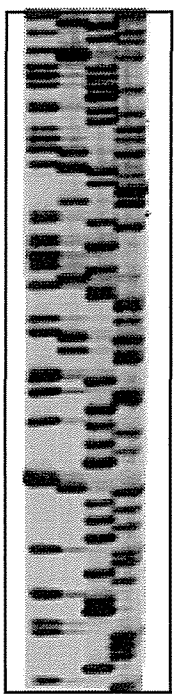


Figure 4.4-2 Single lane set subimage.

4.5 Boundary extraction

Geometric distortions often occur on gel autoradiographs because of variations in the conditions during electrophoresis. In order to successfully process most of the gel autoradiographs, geometry correction is necessary. Most of the distortions may be corrected by finding the boundaries of the lane set, and making these vertical. To find the lane set boundaries, the vertical edges of the band lane are enhanced by vertical smoothing and then detected by a linear convolution filter. The local maxima along each row are detected, and false edges are eliminated. A distance image is used to distinguish between the lane set boundaries and individual lane boundaries within the set. The boundaries are then converted to a series of line segments, represented as a list of key points, which are used to straighten the lanes.

Boundary extraction module DFD is given in Figure 4.5-1.

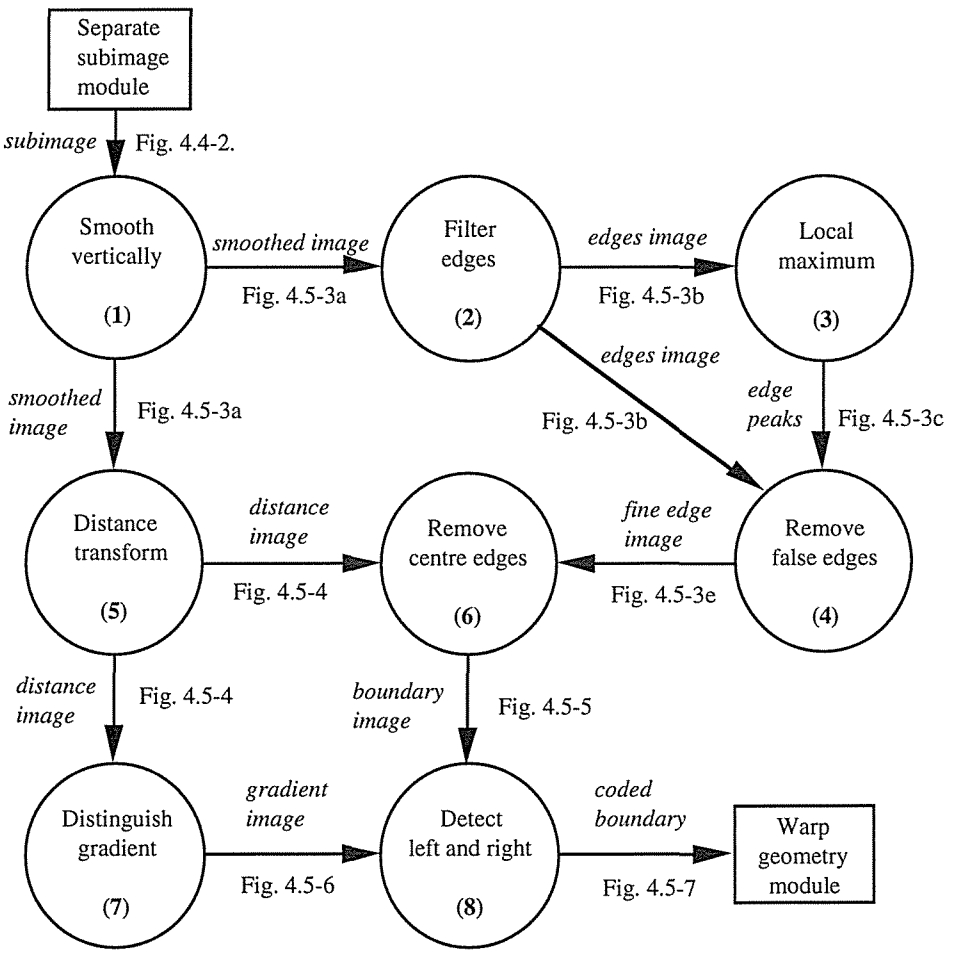


Figure 4.5-1 Extract boundary module DFD.

(1) Vertically smooth. The first step is to smooth (Eq.AI-4) the image vertically using a 40×3 moving window. This makes the vertical features (the lanes) in the image more visible as in Figure 4.5-3a.

(2) Filter edges. A linear convolution filter (Eq.AI-2) extracts the vertical edges of the lane sets. A 3×3 window is used with the kernel weights in Figure 4.5-2. The detected edges as are shown in Figure 4.5-3b.

$$\frac{1}{9}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Figure 4.5-2 The filter weights for extracting vertical edges.

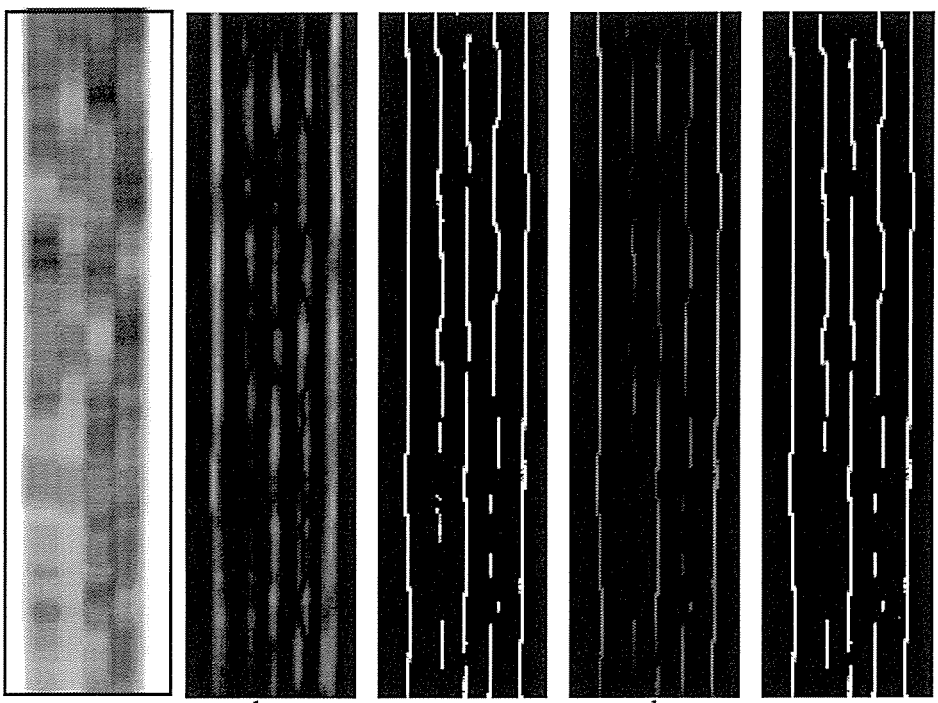


Figure 4.5-3 Edges extraction
a) vertically smoothed image; b) edges image; c) edge peaks;
d) false edges removed image; e) fine edge image.

(3) Local maximum. After further smoothing a local maximum detection filter, **BOX EXTREME** (Eq.AI-9), detects the maximum pixels. These correspond to points of maximum gradient, along the edges of the individual lanes. A 1×15 window is used to

detect maxima corresponding to each of the detected edges along each row. Figure 4.5-3c shows the location of the edge peaks.

(4) Remove false edges. Each detected edge is labelled with the edge strength by logically **AND**ing (Eq.AI-13) the results from the previous two steps (see Figure 4.5-3c).

The image is then thresholded (Eq.AI-15), so that only the significant edges are kept. The false edges removed image is given in Figure 4.5-3e. (In this example, there were not many false edges.)

(5) Distance transform. Only the left and right boundaries of the lane set are required for geometry correction. Approximate boundaries may be found by thresholding the vertically smoothed image (Figure 4.5-3a). These boundaries are shown in Figure 4.5-4a.

The boundary image is then distance coded to determine the distance of each pixel from the boundaries. The distance transform expression, **DISTANCE** is given in Eq.AI-18. Figure 4.5-4b gives the distance coded image of Figure 4.5-4a.

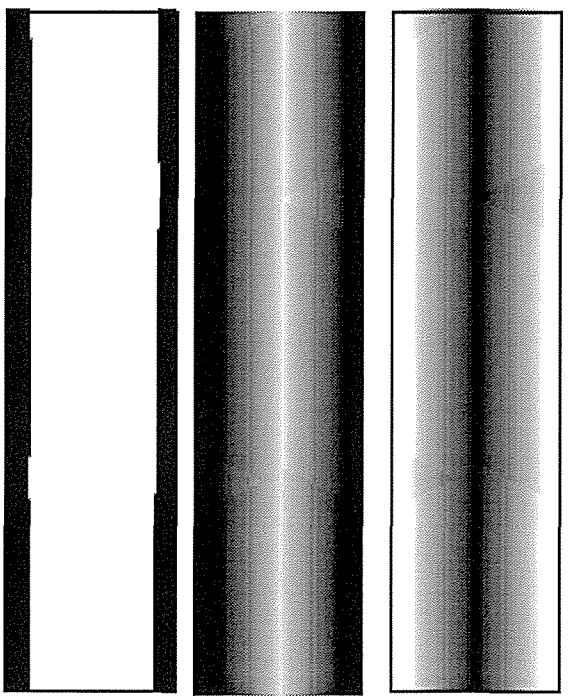


Figure 4.5-4 Distance transform
a) Approximate boundaries; b) Distance image ; c) Inverted distance image.

The distance image is then inverted, **INVERT** (Eq.AI-17), to make the central region darker for reducing the strength of the edges within the lane set. Figure 4.5-4c shows the inverted distance image.

(6) Remove centre edges. The inverted distance image **AND**ed (Eq.AI-13) with the edge image to code the edges with their distance. This reduces the strength of the edges within the lane set (see Figure 4.5-5c). The image is then thresholded (Eq.AI-15) to extract only the lane set boundaries as shown in Figure 4.5-5d.

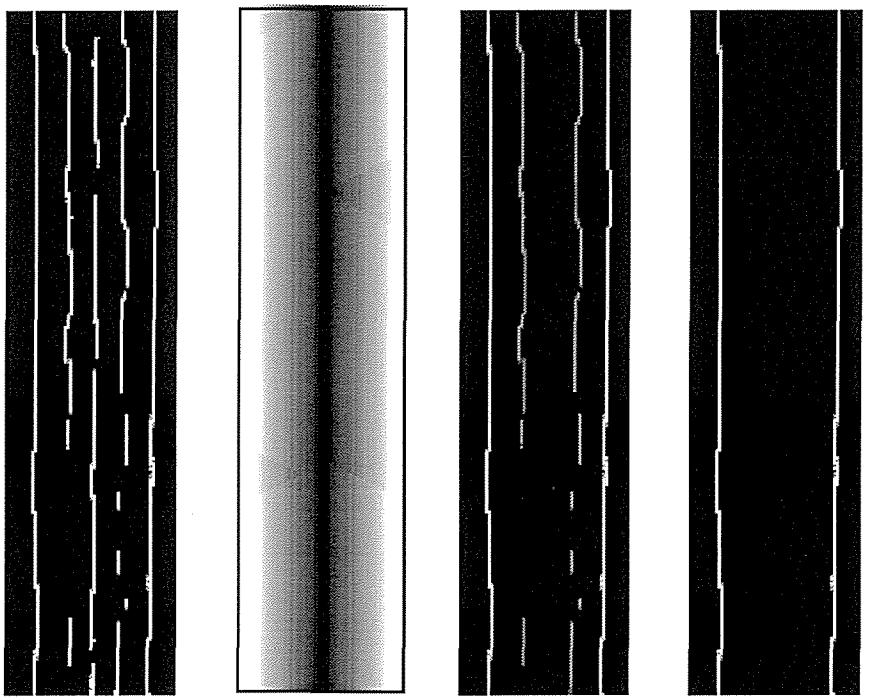
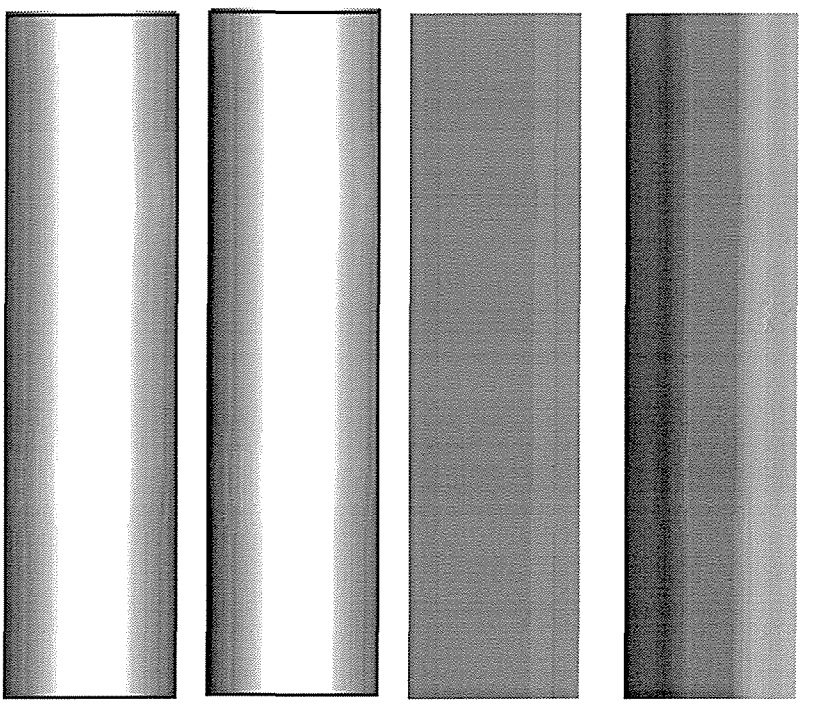


Figure 4.5-5 Remove centre edges
a)edges image; b)distance image; c)centre strength reduced image; d)boundary image.

(7) Distinguish left and right . The left boundary of the lane set is used to straighten left side of the lane set and the right boundary is used to stretch the right side to make the lanes within the set. For this reason, the left and right boundaries must be distinguished. Distinguishing the left and right boundaries by positions alone is not reliable since there may be false boundaries. Since the left boundary consists of positive step and the right boundary consists of a negative step, the left and right boundaries may be distinguished on the basis of the sign of the edge gradient in that region.

Thresholding the distance image and smoothing it obtain the approximate lane set area around the boundaries (Figure 4.5-6a). Shifting one column, **COPY** (Eq.AI-19), and then subtracting (Eq.AI-12) the two images with one column difference obtain a gradient image (Figure 4.5-6c).

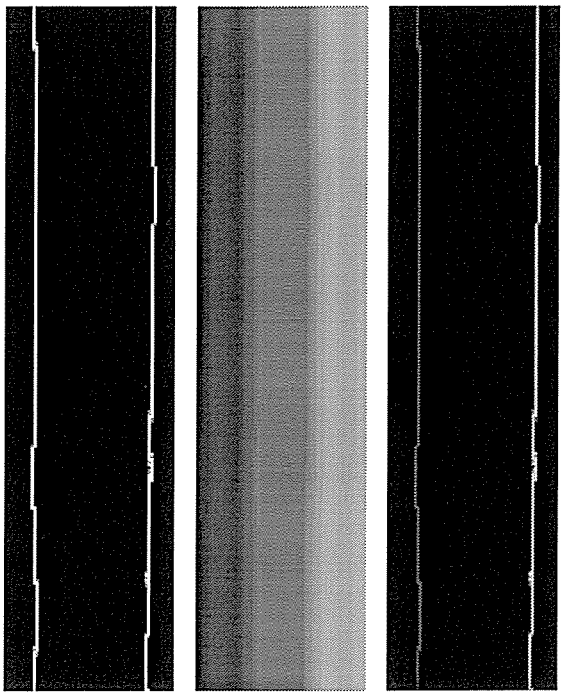
The gradient image is then expanded linearly (Eq.AI-16) to stretch the contrast range. The contrast range are set as 108 and 148. The gradient image is shown in Figure 4.5-6d.



a) b) c) d)

Figure 4.5-6 Gradient extraction
a) smoothed boundary area; b) shifted image from (a);
c) variation image; d) gradient image.

(8) Detect left and right boundaries. Since the left edge pixel values in the gradient image are about 108 (set in **EXPAND** operation) and the right edge pixel values are about 148, **AND**ing the gradient image with the boundary image (Eq.AI-13) obtains a coded boundary image with a darker left boundary and a lighter right boundary (Figure 4.5-7).



a) b) c)

Figure 4.5-7 Coding boundaries
a) the boundary image; b) gradient image; c) coded boundary image.

4.6 Geometry warping

Geometric distortions must be corrected before the band order can be read reasonably. With the knowledge that each band lane runs roughly parallel to its neighbours, that the lanes are of approximately equal width and are approximately equally spaced, and that any changes take place in a continuous manner [Elder & Southem, 1987], it is possible to make lane set boundaries vertical. The extracted boundaries are converted into series of key points. Besides the first and the last row, every big change (three columns) in the boundary is defined as a key point. The left boundary points are used to shear the image to straighten left side and the right boundary points are used to trapezoid warp the right side to make the lane set into a standard width. Since the extracted boundaries may be broken into several parts, the key points of the boundaries are refined before straightening. Figure 4.6-1 gives the warp geometry module DFD.

(1) Convert left boundary. The coded boundaries (Figure 4.5-6) have different intensities. The pixel values of left boundary are less than 128 meanwhile the pixel values of right boundary are greater than 128. Thresholding (AI-15) between levels 1 and 127 detects the left boundary, which is then coded as a chain. The first chain is

extracted if more than one chain is coded. The extracted chain is then converted into a series of points.

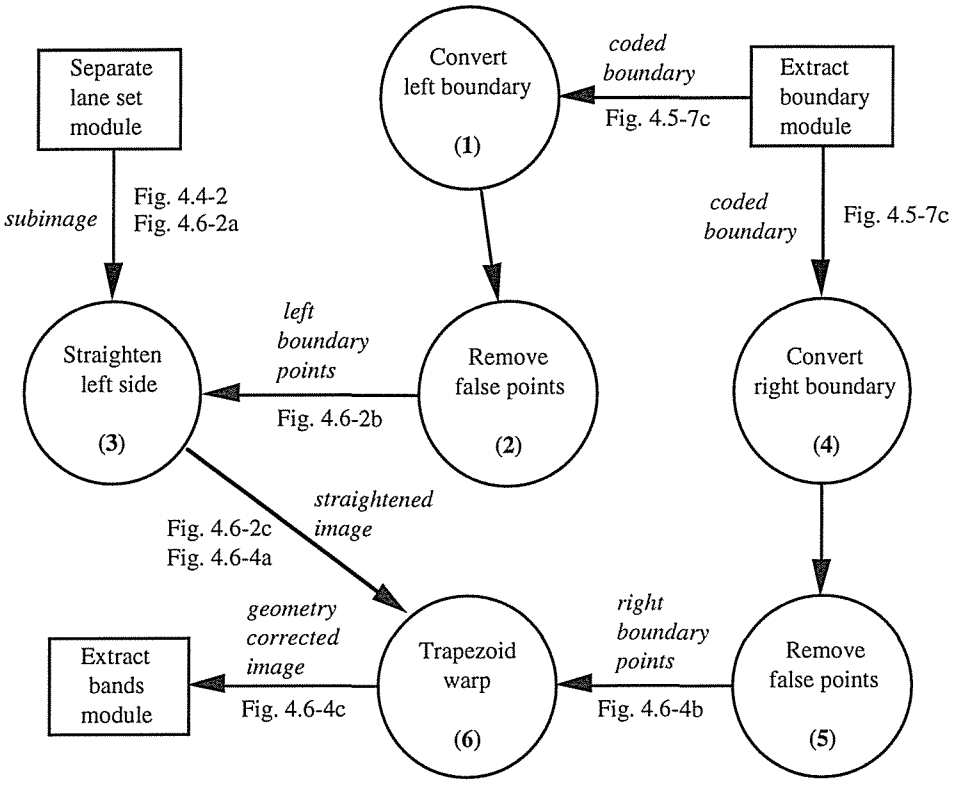


Figure 4.6-1 Warp geometry module DFD.

(2) Remove left false points. Since part of the next subsequence or noise may be extracted into the boundary image, false boundaries may be converted. If false boundaries occur in the boundary image or the boundary is broken into several parts after extracting, more than one chain may be coded. The series of points are refined. The false points are removed according to the positions and the length of the chain. If two chains are coded and the vertical positions are close, these are possibly broken parts from the boundary. If one is above another the two parts are joined together. If the two chains are parallel the shorter one is removed. Figure 4.6-2b gives the left boundary key points.

(3) Straighten left side. The left side of the lane set is straightened vertically by using the left boundary key points to shear the image horizontally, to put the key points into column 0 of the image. Figure 4.6-3 illustrates the schematic of shifting left hand side between two key points.

All pixels in a given row are moved by the same amount:

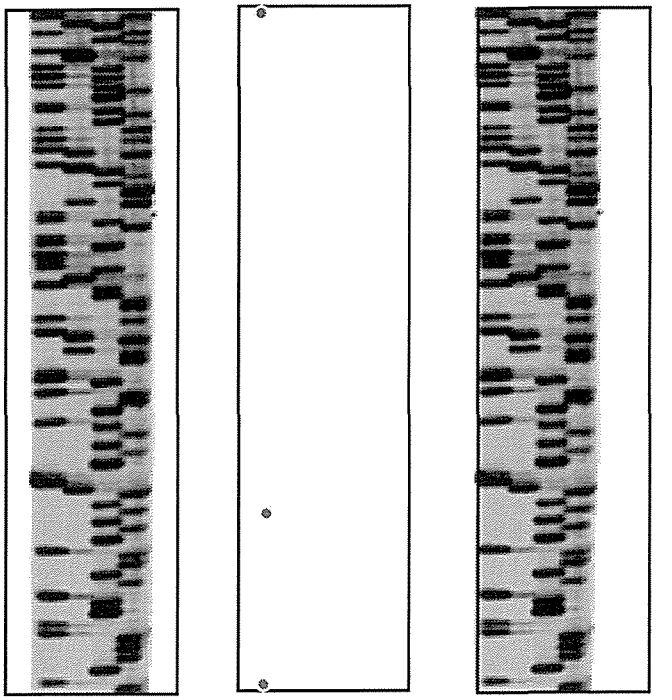


Figure 4.6-2 Left side straightening
a) subimage; b) left boundary points; c) straightened image.

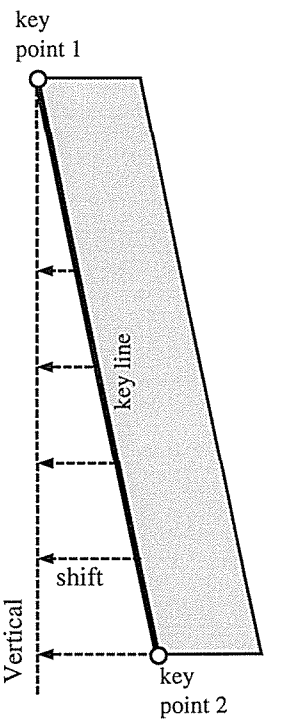


Figure 4.6-3 Shift left side correction schematic.

$$\text{shift} = \frac{\text{key point 2 col} - \text{key point 1 col}}{\text{key point 2 row} - \text{key point 1 row}}(\text{row} - \text{key point 1 row}) + \text{key point 1 col}$$

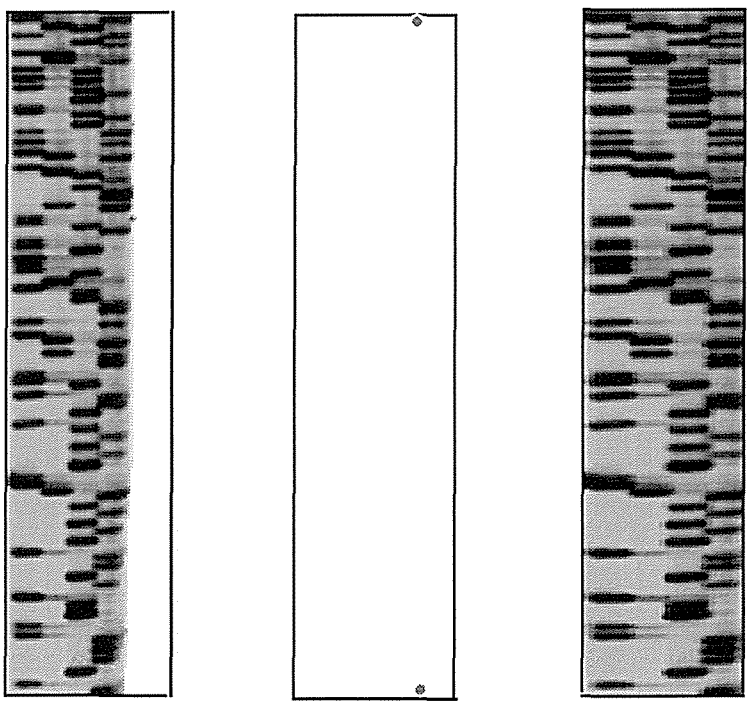
Shifting each part between each two key points into column 0 of the image straightens the left side (see Figure 4.6-2c).

The **STRAIGHTEN** VIPS command was specially written for this application, with the C program given in Appendix III.

(4) Convert right boundary. The pixel values of right boundary are larger than 128. The image is thresholded between 128 to 255, then chain coded and converted to points as before (the same as the left boundary).

(5) Remove right false points. Detection of right false points is similar to the left side. The broken chain parts are joined and any false chains are removed. Figure 4.6-4b shows the right boundary key points image.

(6) Trapezoid warp. Lanes may not be the same width over their complete length. Lanes can be made parallel by warping the image from a trapezoid. Figure 4.6-5 illustrates the stretching of the right side between two key points.



a) b) c)

Figure 4.6-4 Right side stretching
a) straightened image; b) right boundary points; c) geometry corrected image.

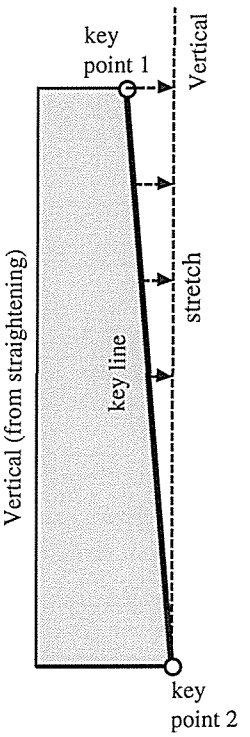


Figure 4.6-5 Stretch right hand side schematic.

The whole row is stretched as:

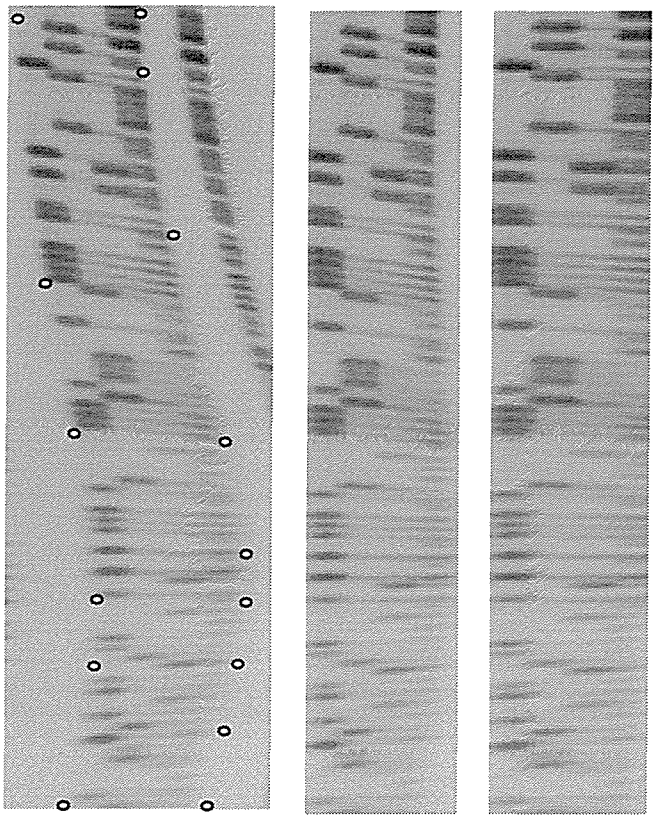
$$\text{stretch} = \frac{\text{key point 2 col} - \text{key point 1 col}}{\text{key point 2 row} - \text{key point 1 row}}(\text{row} - \text{key point 1 row})$$

The stretching is performed by duplicating certain pixels as the row is copied. Figure 4.6-4c shows the right side trapezoid warped image.

The **PARALLEL** command was written for this purpose, and C program is given in Appendix III.

Figure 4.6-6 shows an example of boundary correction by manually selecting left and right side key points.

A similar procedure may be used if necessary to make the bands horizontal by shearing and stretching the image vertically. Figure 4.6-7 gives an example of using this method to correct for "smiling".



a. b. c.

Figure 4.6-6 Geometry correction

a)geometrically distorted image; b)straightened image; c)geometry corrected image.

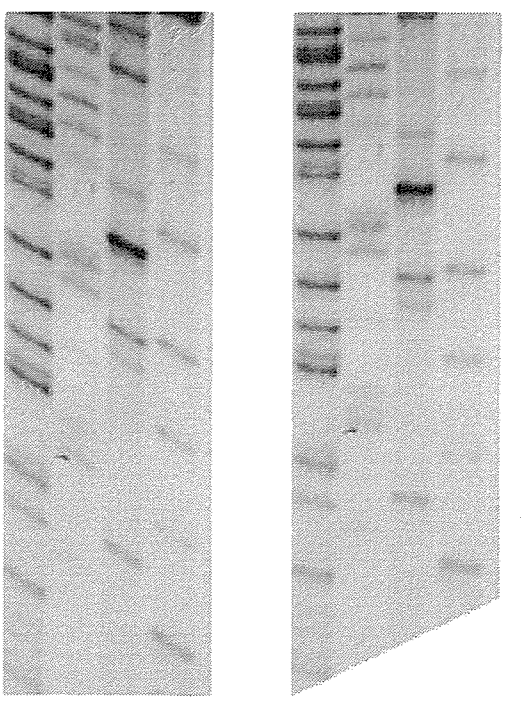


Figure 4.6-7 Geometry correction of skewed bands.

4.7 Band extraction

After the geometry has been corrected in the separate subimages of each lane set, the next phase of the processing is to extract the individual bands. Contrast enhancement improves the readability of the faint bands and the background is clipped to reduce noise. Keeping only the centre of each lane is useful to avoid problems when bands from different lanes overlap. Horizontal smoothing along the length of the bands reduces the noise and enhances the separation where the bands are very close. Following this, the bands are enhanced by filtering. The maximum pixel in each column of the band is extended through the whole band to make detection more reliable. The bands are then detected by thresholding. Figure 4.7-1 gives the extract bands module DFD.

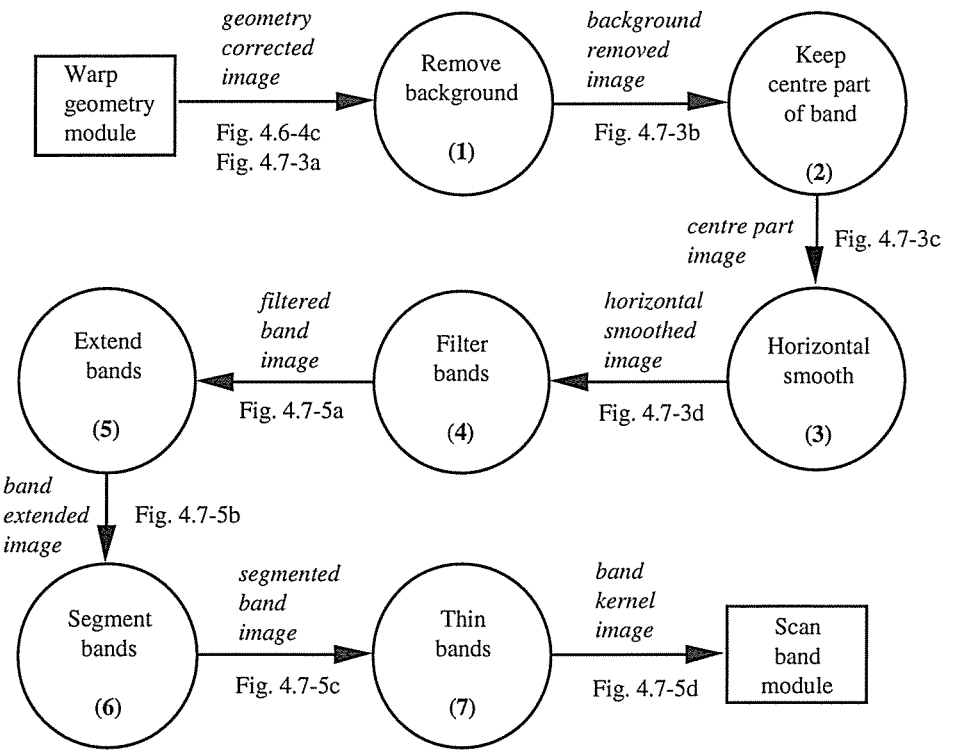


Figure 4.7-1 Extract bands module DFD.

(1) Remove background. The background does not contain any information of interest, and variations in contrast in the background and shadows cause problems in later processing. In the histogram of the band image (Figure 4.7-2), the sharp peak on the right corresponds to the light background, the broad peak corresponds to the shadows and the sharp peak on the left corresponds to the bands. Since the background is lighter than the bands, the background may be removed by clipping. *Add*ing a

constant increases the background to 255 (Eq.AI-11). If the resultant pixel value exceeds 255 then it is clipped at 255. The band image is then expanded linearly (Eq.AI-16) to stretch the contrast range of the lower intensities. The background removed image is given in Figure 4.7-3b.

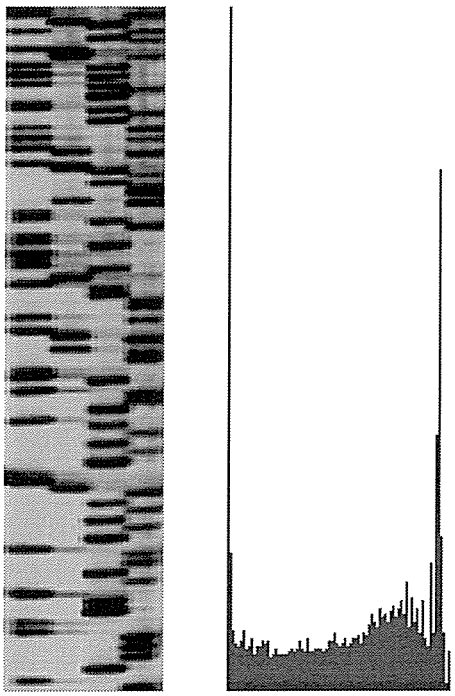
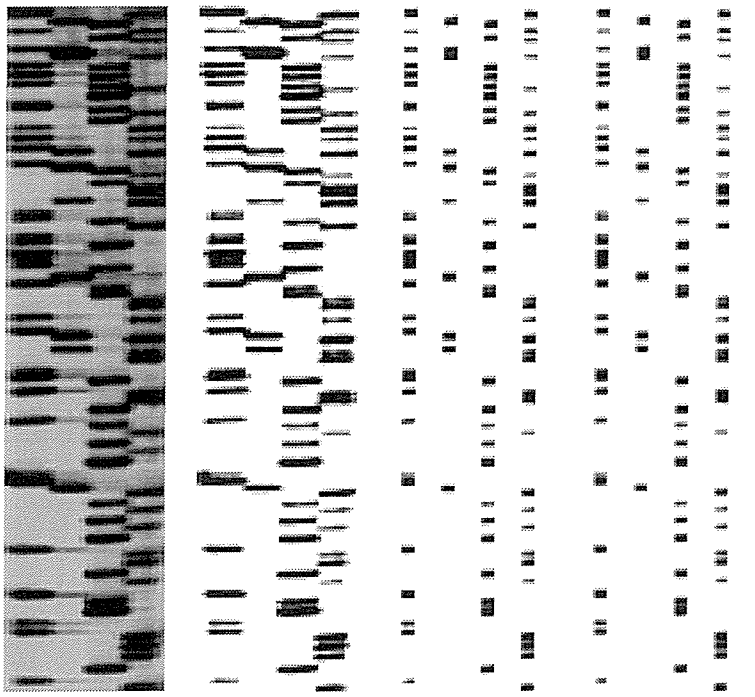


Figure 4.7-2 Band image and the histogram.

(2) Keep centre part of band lane. Only the centre of each lane is kept to avoid problems when bands from different lanes overlap. A subroutine is used to clear to the background level strips between the band lanes.

(3) Horizontal smooth. The next step is to select only the bands. Smoothing (Eq.AI-4) along the length of the bands before feature extraction reduces the noise and enables some close bands to be separated. Such a smoothing step increases the accuracy and extends the readable range of the DNA sequence from the gel autoradiograph. Smoothing is accomplished by using a moving average with a 2×5 window. Again only the centre of each lane is kept. Figure 4.7-3d shows the smoothed image.

(4) Filter bands. A 3×3 linear convolutional filter is then used to enhance the bands. For each output pixel, the linear filter takes a weighted average of the pixel values in the neighbourhood of that pixel in the input image. The weights used are shown in Figure 4.7-4.



a) b) c) d)

Figure 4.7-3 Background removing
a) geometry corrected image; b) background removed image;
c) centre parts of lanes; d) horizontal smoothed image.

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ -3 & -3 & -3 \end{bmatrix}$$

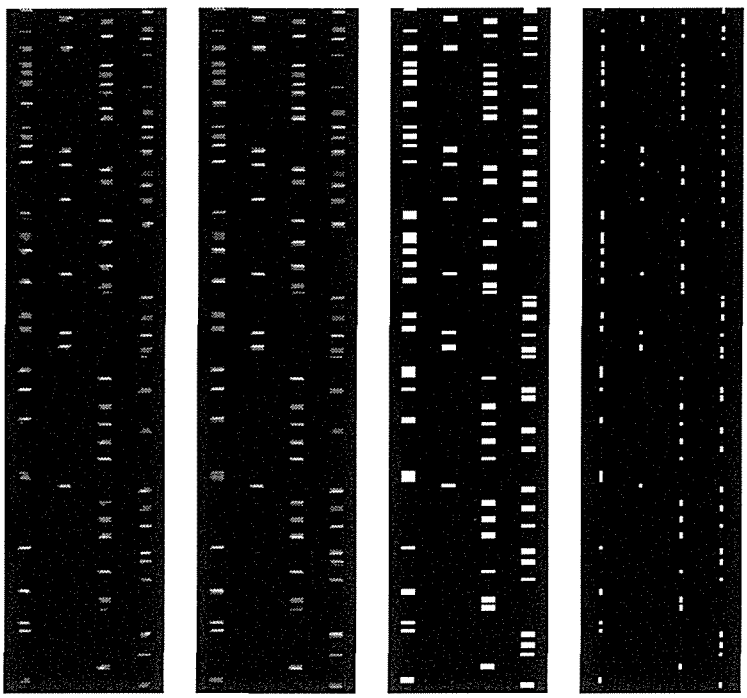
Figure 4.7-4 The filter weights for extracting the bands.

This filter combines horizontal smoothing with vertical edge detection in one operation. Only the positive value is used for the output; negative values are set to zero (Eq.AI-3). This keeps only one edge of each band.

The filtered image is expanded linearly (Eq.AI-16) to stretch the contrast range. A non-linear edge enhancement filter, **BOX ENHANCE** (Eq.AI-8), further enhances the detected edges and separates close bands using a 3×13 window. Figure 4.7-5a shows a filtered and enhanced image.

(5) Extend bands. The lightest pixel in each row of the band is then extended through the whole band to make detection more reliable. A maximum operation (Eq.AI-5) with

1×24 window is used for extension. Again only the centre of the lane is kept. Figure 4.7-5b shows the band extended image.



a) b) c) d)

Figure 4.7-5 Band filtering
a) band filtered image; b) band extended image;
c) band segmented image; d) kernel band positions.

(6) Segment bands. The image is then thresholded (Eq.AI-15) to detect the bands (Figure 4.7-5c).

(7) Thin bands. Only the band kernel is kept for position scanning. The band kernel is also used to obtain the width of the band, which may detect combined bands. A subroutine is used to clear the background. Figure 4.7-5d shows the band kernel image.

4.8 Band scanning

The individual band kernels are scanned to determine the order of the bands within the four lanes, and hence the order of the bases in the DNA sequence. The band positions are detected and then sorted into a list, going upwards from the bottom of the image. The list is then converted into a DNA sequence. Two or more close bands may be extracted as a single band. If a band is wider than a preset width, it is split into two bands. Detected bands are then drawn on the image. The user may correct the sequence

by adding missing bands or deleting false bands. A lower case letter is used in the sequence to indicate that the result is uncertain. Figure 4.8-1 gives the scan bands module DFD.

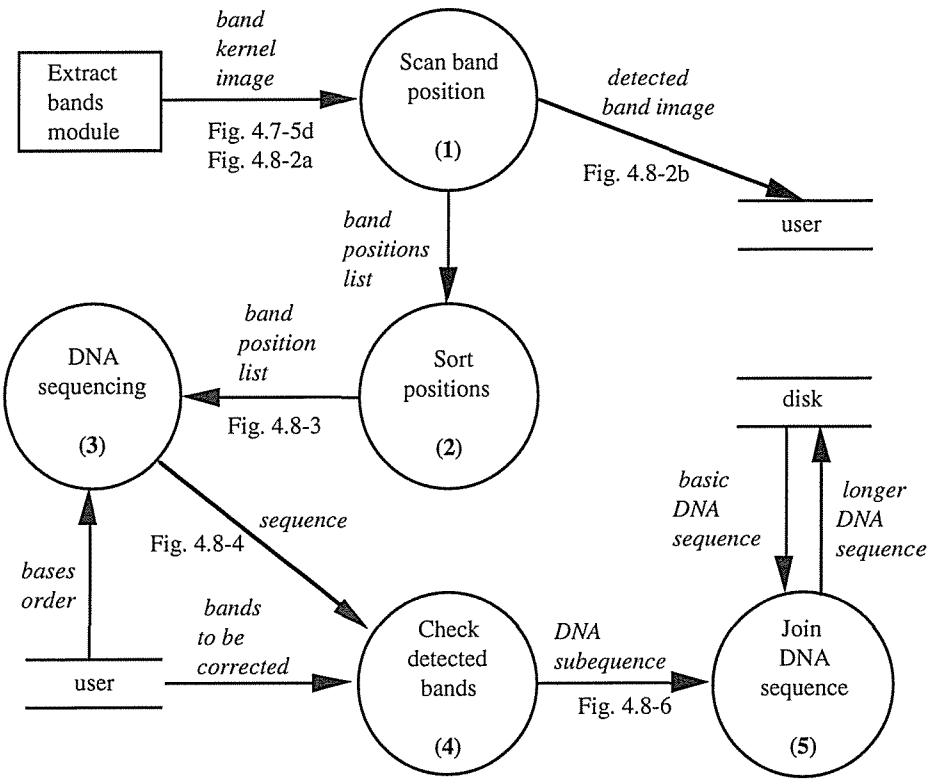
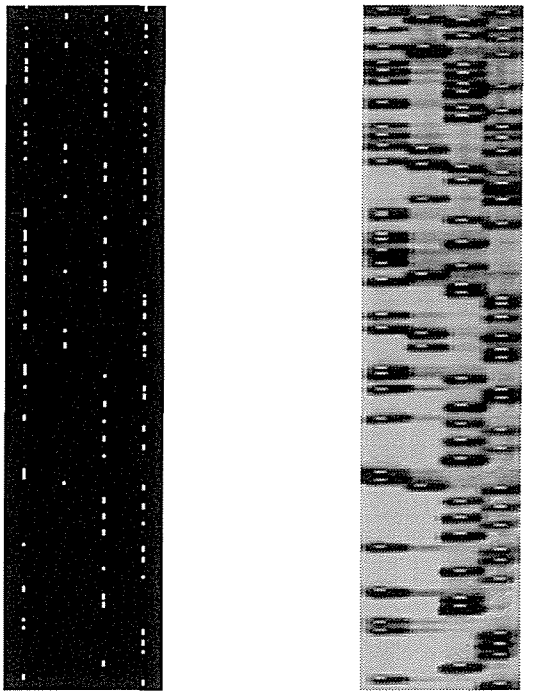


Figure 4.8-1 Scan bands module DFD.

(1) Scan band positions. The band kernel image is coded into chains (one chain for each band). For each chain, the position is detected and put into a list. The detected bands are then drawn onto the image to show the user which bands have been detected. It is possible that some close bands or combined bands are detected as one. The missing bands are detected by the width of the extracted bands. The band spacing varies along the lane. Figure 4.8-2b shows the scanned bands and detected missing bands.

(2) Sort positions. DNA sequences are normally read from the bottom of the sequence up. The position list is therefore sorted into this order. Figure 4.8-3 gives the sorted position list for this example. In the sorted list, the first number of the pair is the row within the image of the band. The second number is the column in the image which depends on the lane in which the band is found.

The **SORT** command was written specially for this application, and the C program is given in Appendix III.



a) b)

Figure 4.8-2 Band scanning
a) band kernel; b) detected bands.

{ (506,87) (502,12) (492,62) (483,87) (476,87) (467,87) (465,12) (459,12) (448,62) (442,62)
(435,12) (427,87) (421,62) (413,87) (406,87) (403,12) (394,62) (387,87) (382,62) (374,87)
(369,62) (360,87) (357,37) (350,12) (349,12) (337,62) (330,87) (324,62) (316,87) (311,62)
(307,12) (298,62) (292,87) (286,87) (285,12) (277,62) (274,12) (270,12) (261,87) (256,87)
(253,37) (245,87) (243,37) (240,12) (232,87) (230,12) (222,87) (217,87) (213,62) (208,62)
(204,12) (199,37) (193,62) (191,12) (181,12) (175,62) (173,12) (170,12) (161,87) (159,62)
(156,12) (153,12) (144,87) (143,37) (132,87) (130,62) (124,87) (120,62) (117,37) (115,12)
(107,87) (106,37) (103,12) (97,87) (96,12) (89,87) (89,12) (82,62) (78,87) (75,62) (71,12)
(63,62) (59,87) (57,62) (56,12) (50,62) (48,12) (43,62) (42,12) (35,87) (30,12) (29,37)
(23,87) (21,62) (16,12) (16,87) (10,62) (7,37) (3,87) (1,12) (0,87) } }

Figure 4.8-3 Band position list.

(3) DNA sequencing. Each lane in the gel autoradiograph represents one of the four bases **A**denine, **C**ytosine, **G**uanine or **T**hymine (abbreviated as **A**, **C**, **G** and **T**) which make up the DNA. The lane order may be defined by the user (or is assumed to be **T**, **C**, **G**, **A** by default). The series of bands corresponds to the relative positions of the different bases along the DNA strand. The column number of each band is used to classify it as belonging to one of the four bases. This results in the base sequence of the section of DNA represented by the image.

Ambiguities may arise on a sequencing gel and during sequence reading. Several bands may be bunched together in the same lane on the gel autoradiograph. In this case it is

not always possible to resolve all of the bands [Brown, 1984]. An uncertainty code can be used to indicate that there may be one or more extra bands undetected. Sometimes bands may be detected in the same position in different lanes. In such cases, it is uncertain which band is correct. A similar problem occurs when the bands are very close in the image. Uncertainty codes increase the readable range of the DNA sequence on the gel autoradiograph by indicating possible errors in the output. If necessary, such uncertainties may be resolved by the user. A lower case letter is used to indicate that the result is uncertain.

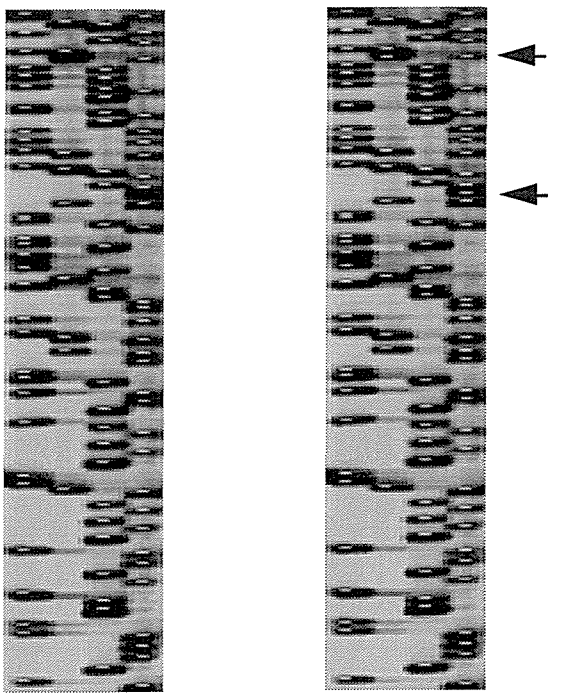
The DNA base order and the format of sequence result file may be defined by user. The default base order is "TCGA" and the default width of output file is 50 bases, organised in groups of ten.

The DNA sequence read from Figure 4.8-2a is shown in Figure 4.8-4. Note that there are two uncertain codes in this example.

```
ATGAAATTGG  TAGAATGAGA  GACTTGAGAG  TGAATGTTAA  CACTATAAGG
TCGTTGTTAG  TTACAGAGCT  ACTATAtGAG  TGAGTGTGTA  TCAGTaGCAT
A
```

Figure 4.8-4 The DNA sequence of the above example.

The C program for the **SEQUENCE** command is given in Appendix III.



a) b)

Figure 4.8-5 Detected band correction
a) detected bands; b) corrected bands.

(4) Check detected bands. The accuracy of DNA sequence is very important in DNA sequencing. The results should be at least as accurate as that from manually reading the bands. It is impossible that the image processing algorithms can read all possible images without error. Therefore it is necessary to cheek the detected bands. From the detected band image, user may select the bands to be corrected using a mouse, adding missing bands or deleting false bands. Meanwhile the uncertain bands must be confirmed before joining into a longer sequence. Figure 4.8-5b shows the final image and Figure 4.8-6 gives the final DNA subsequence.

```
ATGAAATTGG  TAGAATGAGA  GACTTGAGAG  TGAATGTTAA  CACTATAAGG
TCGTTGTTAG  TTACAAGAGC  TACTATATGA  GTGAGTGTGT  CATCAGTAGC
ATA
```

Figure 4.8-6 Final DNA sequence.

(5) Join DNA sequence. Resolution requirements (described in section 4.1) mean that the complete length of each lane set is not able to be contained in a single image. This means that the sequence data obtained from each subimage represents only part of the sequence, and the subsequences from several images need to be joined to get the complete sequence of each lane set.

Also, varying the gel running times makes different parts of the DNA sequence readable. Therefore it is possible to get a very long DNA sequence by joining different parts of the sequence that are obtained from different gel running times.

The DNA sequence joining procedure is given as follows and the detailed C program of **JOIN** command is given in Appendix III.

Sequence joining procedure:

```
    put first part of the sequence in the result sequence;
    take the first band of the second part;
    for each band in the same base in the first part
        check that the remaining bands in first part match the second part;
        if all remaining bands match
            copy remainder of second part to result sequence;
            finished match - exit;
        end if;
    end for;
end procedure.
```

This procedure can be repeated to join multiple parts together. A DNA sequence joining example is shown in Figure 4.8-7.

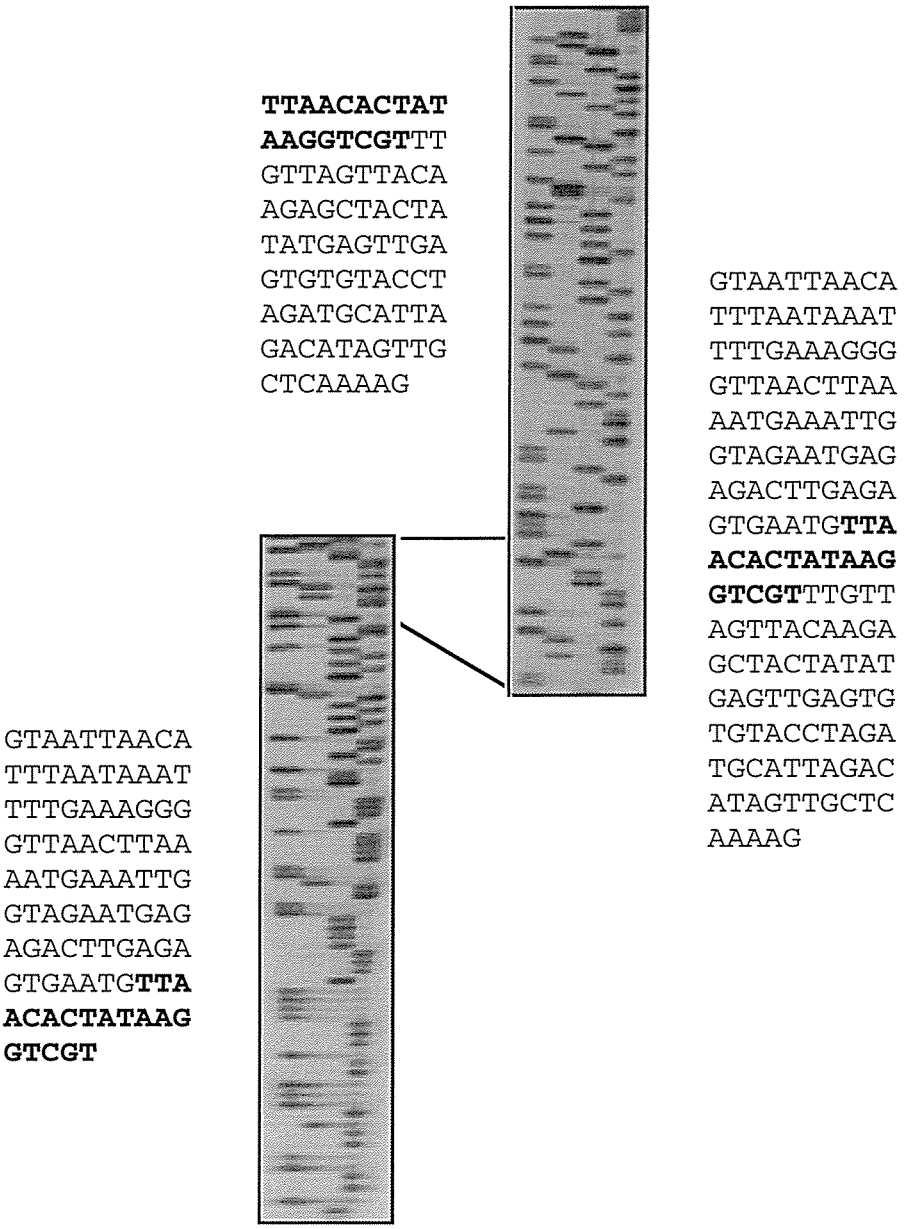


Figure 4.8-7 Sequence joining (from bottom up).

Chapter 5

Results

The DNA sequence reading system has been tested on several gel autoradiographs made by different geneticists. A larger range of gel autoradiographs needs to be tested before general use.

Automatic separation of lane sets and geometry warping are successful on most of the images captured. The lane set separation and geometry warping require that the edges of the subsequence must not touch each other, or anything dark. If the boundaries of a band lane set are not clear, manual selection of the subimage and manual geometry correction are used. Some unreliable subsequences at the bottom of the sequence may need further geometry correction to get an acceptable sequence.

The accuracy of the system depends on clarity of the bands on the gel autoradiograph and the clarity mainly depends on reactions running on the gel. If the bands are clear, the accuracy of automatic processing may be 98% or better. The accuracy needs to be at least as good as that from manually reading the bands. The output is then checked and any errors may be corrected manually to obtain an acceptable accuracy, especially if the bands are not clear enough. If the bands are too close, the accuracy will be reduced. The top part of a gel autoradiograph, where the bands are more closely

spaced, may be captured by zooming, increasing the accuracy and widening readable range of the autoradiograph. If only part of a subimage is readable, the unreadable part may be removed. Some unreliable subimages at the bottom of the sequence may be useful after being checked and corrected, which also broadens the readable range of the autoradiograph.

5.1 Subimage separation and geometry warping

Subimage separation and geometry warping are successful on most of the images tested. Gaps between band lane sets are required for automatic subimage separation and geometry warping. If the boundaries of a band lane set are not clear, manual selection of the lane set and manual geometry correction are required.

All of the subimage examples shown so far in this chapter have been successfully separated automatically. If there is no gap between lane sets (subsequences), the subsequences can not be automatically separated. For example, subsequences b and c in Figure 5.1-1 have no gap between them. Also, if the lane sets are very oblique they may not be separated properly. In those cases manually selection of a subimage is used.

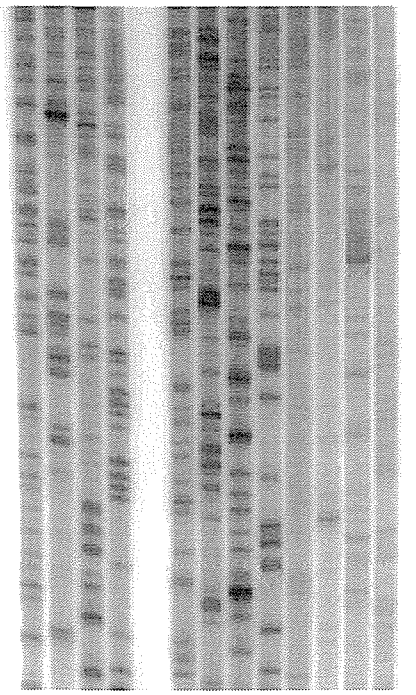


Figure 5.1-1 No gaps between subsequence b) and c).

If there is no gap between band lane sets, the boundaries of the two subsequences cannot be detected and so geometry warping can not be done automatically. In the

above example, the boundaries between subsequence b and c can not be detected. Boundary key points must be selected manually for geometry warping.

If the boundary is not clear, for example there are extra marks beside a boundary, these may interfere with boundary detection and geometry warping. The subimage may be corrected either from the unsuccessfully warped result or from the original separated subimage. Figure 5.1-2a shows an example of unsuccessful geometry warping. There is a pen mark beside the edge (see Figure 5.1-2a1), which is recognised as the boundary (see a2). The unsuccessfully warped result is then corrected manually into a3.

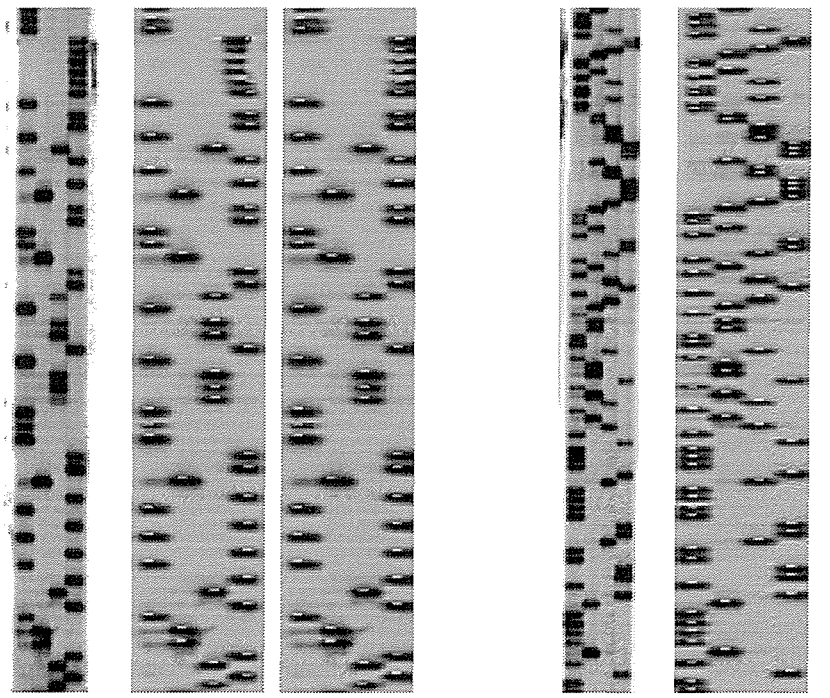
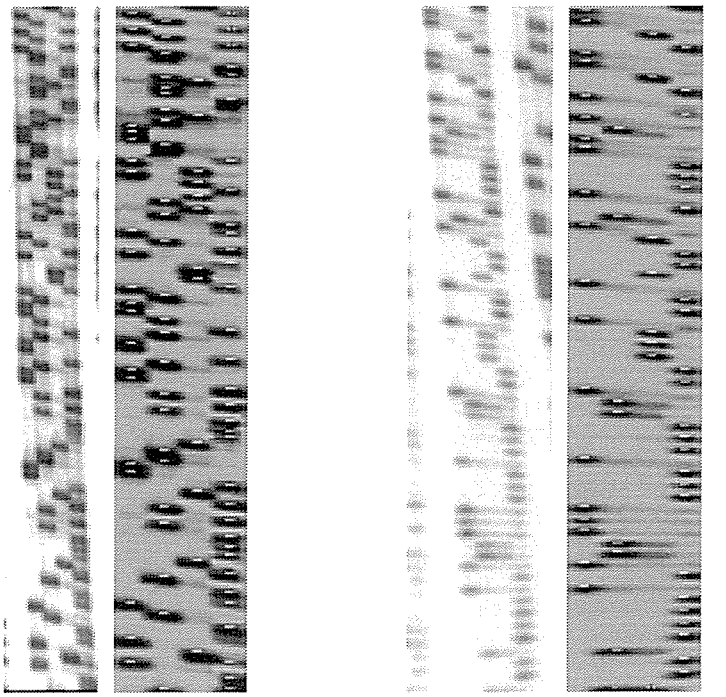


Figure 5.1-2 Geometry warping requirement of clear boundaries.

If there is a space between the marks and sequence edge, the mark will be ignored, and the image will be warped successfully. Figure 5.1-2b shows a successful example in which a gap is between the edge and a mark is ignored.

If the lanes are very oblique or the bands are very faint, geometry warping may fail. Manually correction is needed if it does. Figure 5.1-3a is a successful example of automatically geometry warping. Figure 5.1-3b is an example of manual geometry correction. The band lanes are oblique and the bands are faint so the boundaries are not detected correctly. Manually selection of boundary key points is needed. The corrected image in Figure 5.1-3b shows the result of manual geometry correction.



a) b)

Figure 5.1-3 Oblique lane examples.

5.2 Accuracy

The clarity of the bands on the gel autoradiograph is a very important factor of the accuracy. If the bands are clear, the accuracy may reach 98% or better. For example, the error ratio of clear band subimage 2), 4), 5), 6) in Figure 5.2-2 and subimage 3 in Figure 5.2-3 is less than 2%. The reading errors of automatically processing these examples are listed in Figure 5.2-1.

Examples	Sub no	Detected bands	False bands	Missing bands	Uncertain bands	Error ratio	Comments
Figure 5.2-2	1	41					bottom unreliable
	2	59	0	0	0	0%	clear
	3	46					bottom unreliable
	4	59	0	0	0	0%	clear
	5	57	0	1	0	2%	clear
	6	60	1	0	0	2%	clear
Figure 5.2-3	1	101	6	2	2	8%	not clear
	2	60					bottom unreliable
	3	94	0	2	0	2%	clear
	4						bottom unreliable
	5	98	2	3	0	5%	not clear
	6	55	3	1	0	7%	faint
	7	85	0	4	0	5%	some combined bands
Figure 5.2-4	1	163	4	10	14	9%	top close
Figure 5.2-5	2	38	0	4	4	10%	top unreadable

Figure 5.2-1 Results of examples.

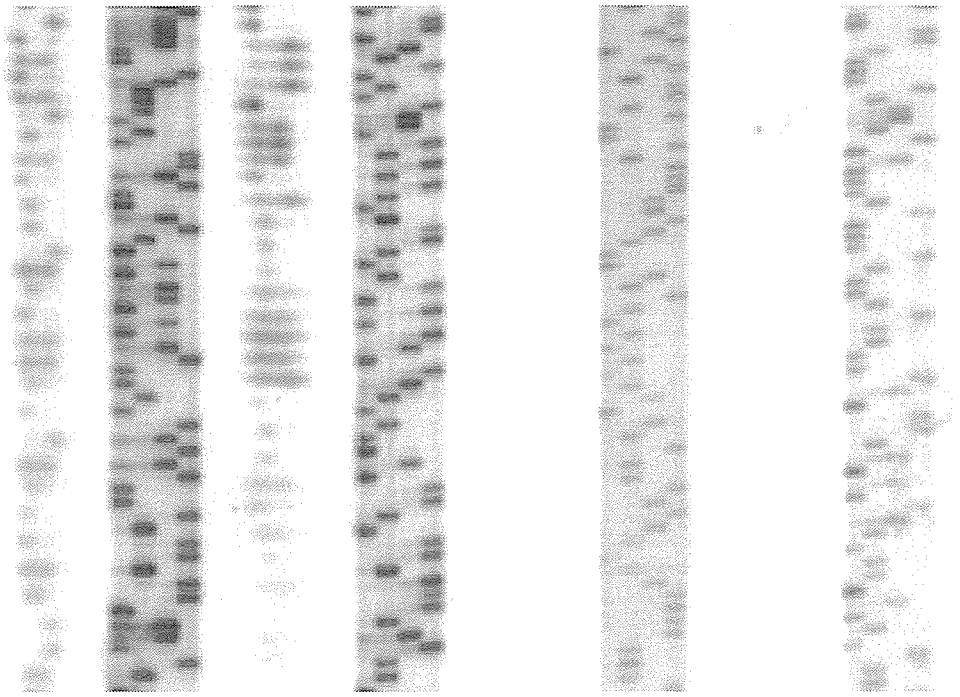


Figure 5.2-2a Example image 1 (bottom part of an autoradiograph).

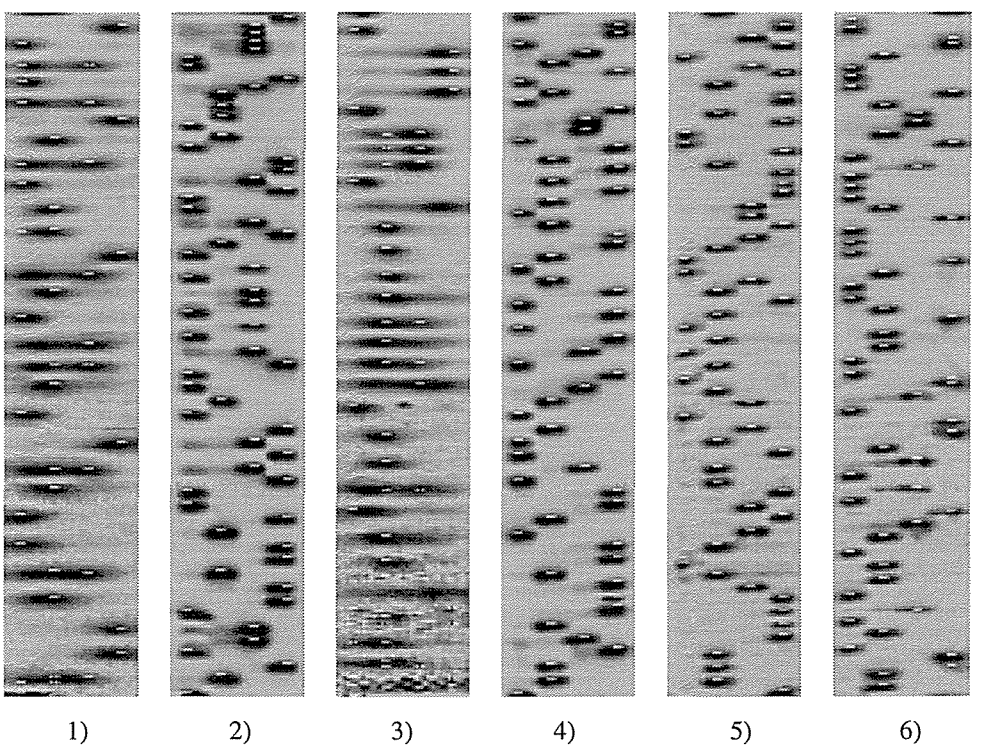


Figure 5.2-2b The results of image 1 (before checking).

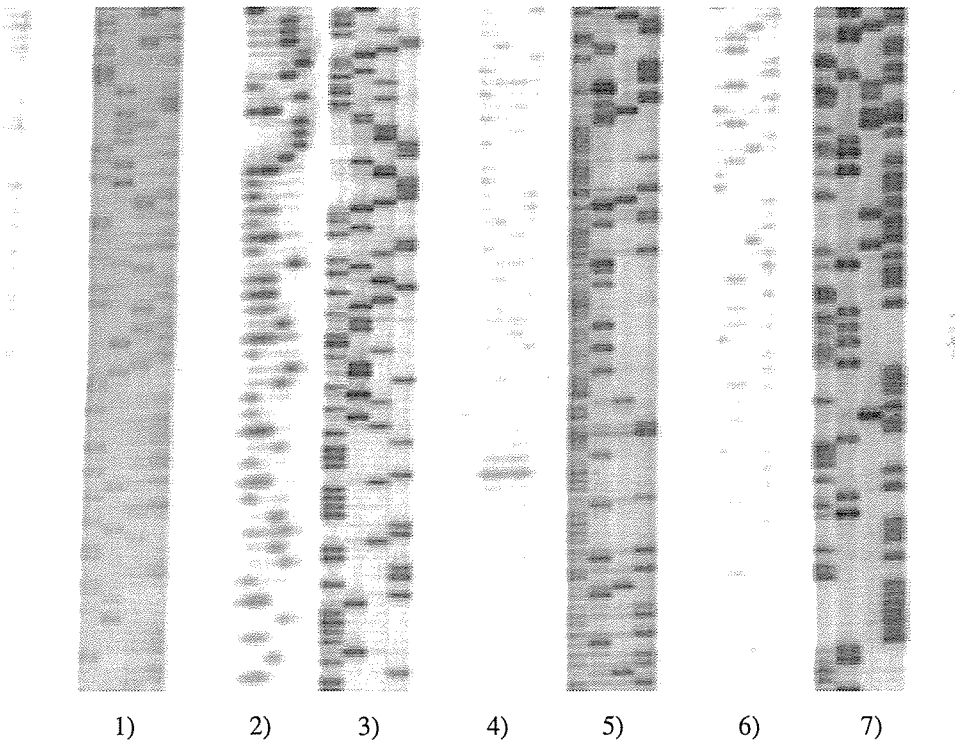


Figure 5.2-3a Example image 2.

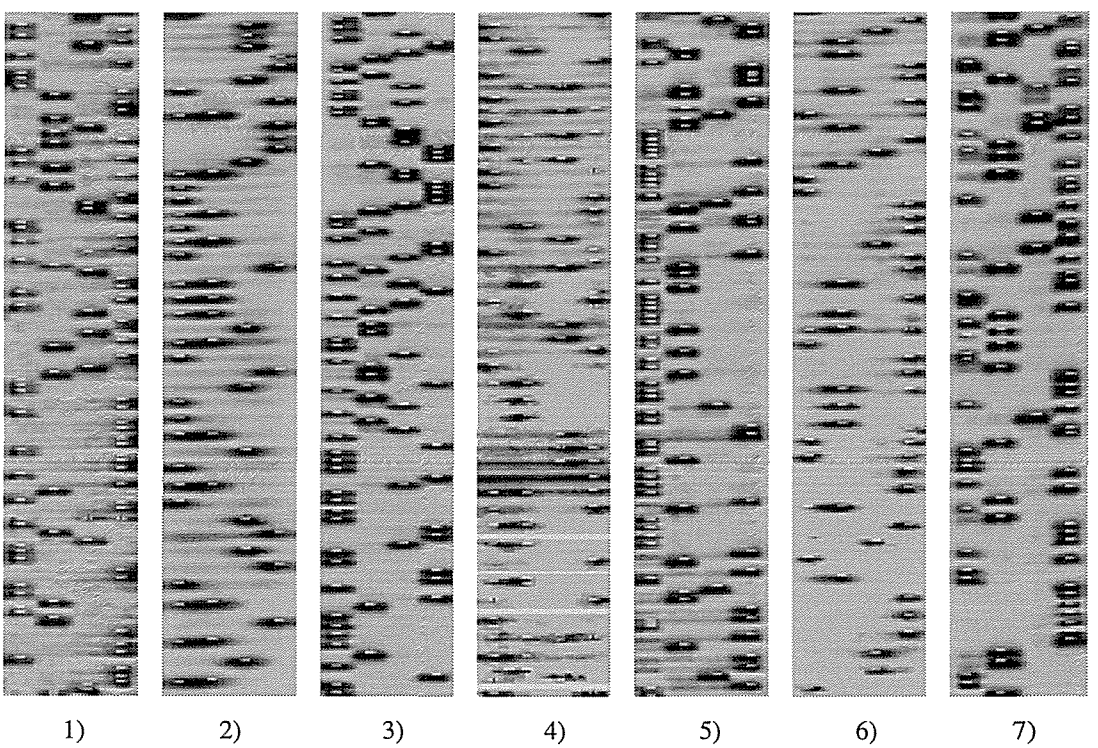


Figure 5.2-3b The results of image 2 (before checking).

Some subimages may not be sufficiently clear because of the conditions of the reactants running on gel or other factors, for example, subimage 1 and 5 in image 2. The result may be checked manually to obtain an acceptable sequence. Some bands may be combined together without sufficient information to be separated, for example, some bands on subimage 7 in image 2. In this case, checking by a geneticist is necessary to decide if bands are missing.

If the bands captured in a subimage are too close (the band density is too high), there is a greater chance of setting two (or more) bands in the same row, causing the accuracy to drop to a point where it may be unacceptable. The top part of a gel autoradiograph often has closer bands. It is suggested that a higher resolution image be obtained by zooming in. This will increase the accuracy and broaden the readable range of the autoradiograph. Figure 5.2-4 shows a subimage from the top part of a gel autoradiograph. Many uncertain bands are obtained from the closer bands (see Fig.5.2-4b). It is also hard to check and correct the detected bands because they are so close. Figure 5.2-4c shows the same subsequence after zooming in. There are fewer uncertain bands and the result (Fig. 5.2-4d) is easier to check and correct.

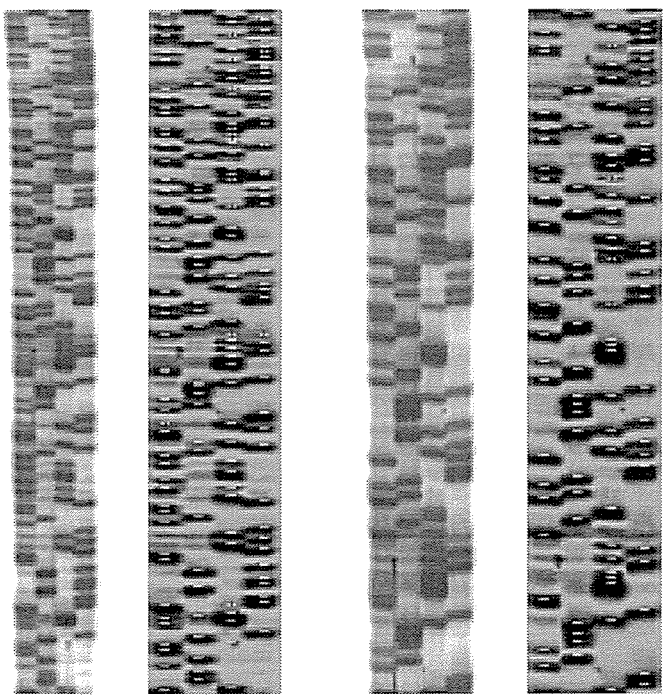


Figure 5.2-4 Zoom in for top part of a autoradiograph.

If part of a subimage is readable, the unreadable or unreliable part may be removed from the resultant sequence. The display may be zoomed in to help checking and

correcting false and missing bands. The top part of the subimage in Figure 5.2-5a is unreadable (see b). The result is then displayed by zooming in and then is corrected (see c). The unreadable and unreliable part is removed (see d).

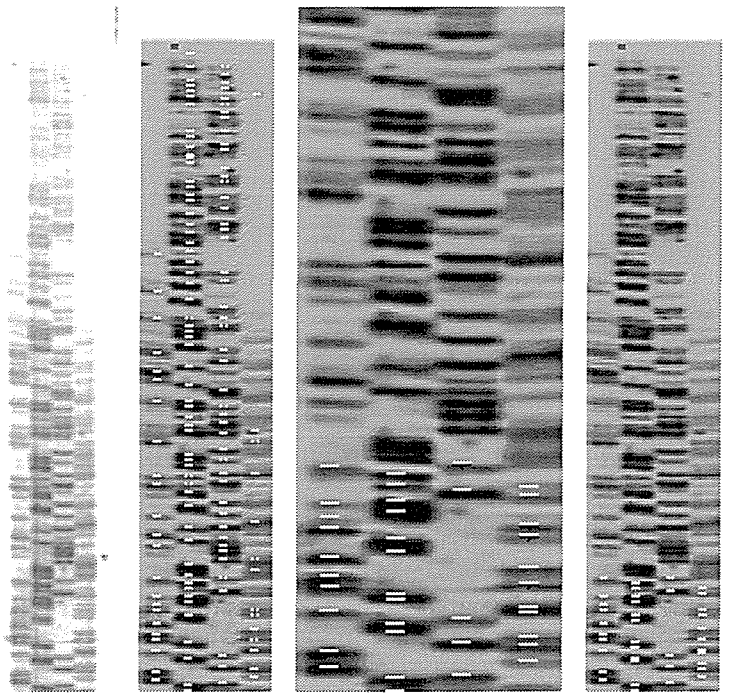


Figure 5.2-5 Remove unreadable and unreliable part by zooming in display.

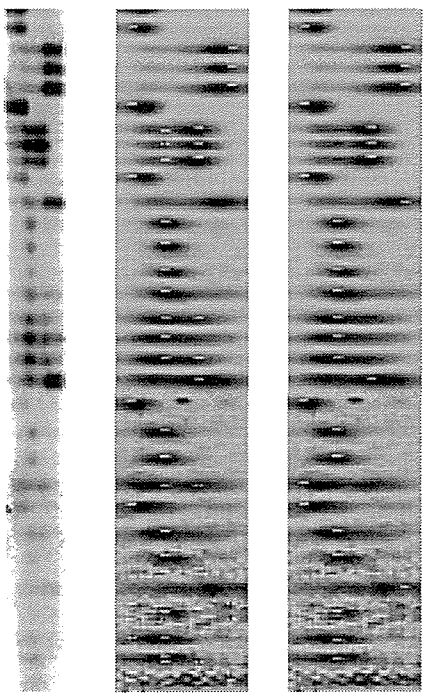


Figure 5.2-6 Sequence checking and correction on a half readable subimage.

Correction of the unreliable part of the sequence broadens the readable range of the autoradiograph. Figure 5.2-6 is a faint half readable subimage at bottom of a sequence. The result is corrected into an acceptable sequence. Figure 5.2-6c shows the corrected result with missing bands added and false bands deleted by consulting the original image.

5.3 Timing

Timing is carried out on a 33Hz 486 PC. For a 512×512 image captured from a gel autoradiograph, preprocessing before separation of the lane sets takes 90 seconds. Extracting and scanning each subimage takes 40 seconds. If there are 8 subimages in a single captured image, it takes 7 minutes for reading the 8 subsequences. This assumes that the user responds to Y/N checking without delay. Suppose 6 images are required to get the data from all readable sections of a gel autoradiograph, the image processing steps would take about 40 minutes to process the complete autoradiograph. However, this does not include the time required for the user to position and capture the image sections or the time for the user to check the resulting sequences. If manually subimage selection or geometry warping is required, it will take longer.

Chapter 6

Summary and Conclusions

In this thesis, the development of the DNA sequence reading system was described.

The genetic information of a living organism is encoded by the DNA contained within every living cell of that organism. DNA sequences are a representation of the genetic structure of DNA molecules. The generation and analysis of DNA sequence data, DNA sequencing, has played a significant role in the elucidation of biological systems. DNA sequence reading is a part of DNA sequencing.

Although several commercial packages are available for DNA sequence reading, these are either limited in what they do or are expensive and highly specialised. In this project, a system is developed for reading DNA sequences directly from DNA sequencing gel autoradiographs within a general purpose image processing system.

VIPS (Vision Image Processing System) is used as the software development environment which runs on PC under windows, Macintosh and Micro VAX under VMS. Chapter 2 in this thesis gave a brief survey of image processing concepts and introduced VIPS.

Since both hardware and techniques are still limited, applying image processing to a particular application relies strongly on checking intermediate results during the development process. The standard waterfall model of software development fits most of the stages in image processing software development. However, the feedback from implementation and system testing to detailed design is much stronger in image processing software development because various complicating factors may be introduced from a larger number of images. The DNA sequence reading software was developed based on the waterfall software development approach combined with exploratory programming. Chapter 3 described the software development model with the system development processes. In most of the cases, the individual lane sets in a captured image are able to be separated automatically. Manual processing was found to be necessary to handle the cases where the lane sets are too close or the lanes are severely warped.

The algorithms for each image processing module were given in Chapter 4. The following modules were necessary:

- Capture an image of several lane sets from a gel autoradiograph and normalise the background;
- Enhance the contrast of the image for different widths of lane sets from one image to another;
- Detect the centre lines of the gaps between the lane sets for automatic separation of the lane sets;
- Separate lane sets into subimages to be read individually;
- Extract boundaries of the lane set for geometry correction;
- Straighten the left side of the lane set and warp the right side into a standard width as geometry correction;
- Extract individual bands and detect the positions of the bands;
- Scan bands to determine the positions into the order of the DNA sequence and merge the subsequence into a longer sequence.

There are several limitations of this system. 1) For automatic processing, the system requires spaces between lane sets when the sequencing reactions are loaded into the gel. If no spaces are present, manual selection of a lane set and the lane set boundaries are necessary. 2) The number of lane sets in an image captured should be between 4 and 8. Spaces before the first lane set and after the last lane set are required for automatic separation of the lane sets and geometry correction. If the edges of the lane set are indistinct, automatic separation of lane set and geometry correction may be

unsuccessful, so manual selection of lane set edges is needed as well. 3) Manually checking of the detected bands is required to add missing bands or delete false bands to obtain an absolutely reliable sequence. The uncertain bands must be confirmed before joining into a longer sequence.

The system may achieve an accuracy of 98% or better if the bands are clear. If a lane set on the autoradiograph is indistinct or bands are too close it may reduce the accuracy, in extreme cases to the point where it is unreadable. For a 512×512 image captured from a gel autoradiograph, preprocessing takes 90 seconds, processing each subimage takes 40 seconds on a 33Hz 486 PC. If processing a 430×350 mm autoradiograph with 16 lane sets, assuming 6 images are required, it takes about 40 minutes.

For future work, a larger range of gel autoradiographs need to be tested and the image processing algorithms need to be refined further for varied images. The accuracy on unclear images may be improved for varied images. The gap line detection and boundary detection may be refined for very faint images and very oblique images. The human computer interface for manual checking may be refined to make it more friendly. Gel autoradiographs from different geneticists need to be tested before general use of the system.

References

[Bailey et al, 1984]: Bailey D.G., Hodgson R.M. and McNeill S.J., Local Filters in Digital Image Processing, 21st NZ national electronics conference, Christchurch, 1984, 95-100.

[Bailey, 1985]: Bailey D.G., Hardware and Software Developments for Applied Digital Image Processing, Ph.D. thesis, Christchurch, 1985.

[Bailey, 1988]: Bailey D.G., Machine Vision: a multi-disciplinary systems engineering problem, *Hybrid Image and Signal Processing*, SPIE Vol 939, 1988, 148-155.

[Bailey, 1991]: Bailey D.G., Raster Based Region Growing, 6th New Zealand image processing workshop, Lower Hutt, 1991, 21-26.

[Bailey, 1993]: Bailey D.G., Frequency Domain Self-filtering for Pattern Detection, First New Zealand Conference on Image & Vision Computing, Auckland, 1993.

[Bodmer, 1987]: Bodmer S.W., Introduction in "Nucleic Acid and Protein Sequence Analysis", edited by Bishop M.J. and Rawlings C.J., IRL Press, Oxford Washington DC, 1987.

[Castleman, 1979]: Castleman K.R., Digital Image Processing, Prentice Hall, Englewood Cliffs, 1979.

[Daivies, 1982]: Davies R.W., DNA Sequencing, in "Gel Electrophoresis of Nucleic Acids: a practical approach", edited by Rickwood D. and Hamer B.D., IRL Press, Oxford, 1982, 117-172.

[Elder and Southern, 1987]: Elder J.K., and Southern E.M., Automatic Reading of DNA Sequencing Gel Autoradiographs, in "Nucleic acid and protein sequence analysis", edited by Bishop M.J. and Rawlings C.J., IRL Press, Oxford Washington DC, 1987.

[Freeman, 1974]: Freeman H., Computer Processing of Line-Drawing Images, ACM Computing Survers, Vol 6, 1974, 57-97

[Gongalez and Wintz, 1987]: Gongalez R.C. and Wintz P., Digital Image Processing, second edition, Addison-Wesley Publishing Company, Reading Massachusetts, 1987.

[Hindley, 1983]: Hindley J., DNA Sequencing, Elsevier Biomedical Press, Amsterdam, 1983.

[Image Analysis Unit, 1992]: Image Analysis Unit, VIPS Reference Manual and Users Guide, version 4.1, Massey University, Palmerston North.

[Jain, 1989]: Jain A.K., Fundamentals of Digital Image Processing, Prentice Hall, Englewood Cliffs, 1989.

[Pratt, 1978]: Pratt W.K., Digital Image Processing, A wiley-interscience publication, New York, 1978.

[Sommerville, 1989]: Sommerville I., Software Engineering, Addison-Wesley Publishing Company, Wekingham, 1989.

Appendix I

Expressions of VIPS Commands

Moving window calculation :

$$p_{out}(x,y) = f(p_{in}(x+i,y+j), \quad \forall i \in \{-\tfrac{m}{2},..,\tfrac{m}{2}\}, j \in \{-\tfrac{n}{2},..,\tfrac{n}{2}\}) \tag{AI-1}$$

FILTER LINEAR *p_{in} p_{out} weights scale /ABSOLUTE* :

$$p_{out}(x,y) = \frac{1}{scale}\left|\sum_{i=-\frac{m}{2}}^{\frac{m}{2}}\sum_{j=-\frac{n}{2}}^{\frac{n}{2}} w(i,j)\,p_{in}(x+i,y+j)\right| \tag{AI-2}$$

FILTER LINEAR *$p_{in}p_{out}$ weights scale /POSITIVE* :

$$p_{out}(x,y) = \begin{cases} \dfrac{1}{scale}\displaystyle\sum_{i=-\frac{m}{2}}^{\frac{m}{2}}\sum_{j=-\frac{n}{2}}^{\frac{n}{2}} w(i,j)\,p_{in}(x+i,y+j) & \text{if} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{AI-3}$$

BOX AVERAGE *p_{in} p_{out} (m n)* :

$$p_{out}(x,y) = \frac{1}{m \times n}\sum_{i=-\frac{m}{2}}^{\frac{m}{2}}\sum_{j=-\frac{n}{2}}^{\frac{n}{2}} p_{in}(x+i,y+j) \tag{AI-4}$$

BOX MAXIMUM *pin pout (m n)* :

$$p_{out}(x,y) = Max(p_{in}(x+i,y+j), \quad \forall i \in \{-\tfrac{m}{2},..,\tfrac{m}{2}\}, j \in \{-\tfrac{n}{2},..,\tfrac{n}{2}\}) \tag{AI-5}$$

BOX MINIMUM *pin pout (m n)* :

$$p_{out}(x,y) = Min(p_{in}(x+i,y+j), \quad \forall i \in \{-\tfrac{m}{2},..,\tfrac{m}{2}\}, j \in \{-\tfrac{n}{2},..,\tfrac{n}{2}\}) \tag{AI-6}$$

BOX STRETCH *pin1/out pin2 (m n)* :

$$p_{out}(x,y) = \frac{p_{in}(x,y) - p_{\min}(x,y)}{p_{\max}(x,y) - p_{\min}(x,y)} \times 255 \tag{AI-7}$$

BOX ENHANCE *pin pout (m n)* :

$$p_{out}(x,y) = \begin{cases} p_{\max}(x,y) & \text{if} \quad p_{in}(x,y) > (p_{\max}(x,y) + p_{\min}(x,y))/2 \\ p_{\min}(x,y) & \text{otherwise} \end{cases} \tag{AI-8}$$

BOX EXTREME */MAX pin pout (m n)* :

$$p_{out}(x,y) = \begin{cases} 1 & \text{if } p_{in}(x,y) = p_{\max}(x,y), \\ 0 & \text{otherwise} \end{cases} \tag{AI-9}$$

ADD *pin1/out pin2 /saturate* :

$$p_{out}(x,y) = \begin{cases} p_{in1}(x,y) + p_{in2}(x,y) & \text{if} < 255 \\ 255 & \text{otherwise} \end{cases} \tag{AI-10}$$

ADD *pin1/out constant /saturate* :

$$p_{out}(x,y) = \begin{cases} p_{in1}(x,y) + const & \text{if} < 255 \\ 255 & \text{otherwise} \end{cases} \tag{AI-11}$$

SUBTRACT *pin1/out pin2 /SATURATE* :

$$p_{out}(x,y) = \begin{cases} p_{in1}(x,y) - p_{in2}(x,y) & \text{if} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{AI-12}$$

AND *pin1/out pin2 /BITWISE* :

$$p_{out}(x,y) = p_{in1}(x,y) \ \wedge \ p_{in2}(x,y) \tag{AI-13}$$

DIVIDE *pin1/out pin2 k /SATURATE* :

$$p_{out}(x,y)=\begin{cases} k\dfrac{p_{in1}(x,y)}{p_{in2}(x,y)} & \text{if} < 255 \\ 255 & \text{otherwise} \end{cases} \tag{AI-14}$$

THRESHOLD *pin/out th1 th2* :

$$p_{out}(x,y)=\begin{cases} 255 & \text{if } threshold1 \le P_{in}(x,y) \le threshold2 \\ 0 & \text{otherwise} \end{cases} \tag{AI-15}$$

EXPAND *pin1/out level1 level2* :

$$p_{out}(x,y)=\begin{cases} 0 & if \le 0 \\ \dfrac{p_{in}(x,y)-\text{level1}}{\text{level2}-\text{level1}}\times 255 & if > 0, < 255 \\ 255 & if \ge 255 \end{cases} \tag{AI-16}$$

INVERT *pin1/out /ADDITIVE* :

$$p_{out}(x,y)=255-p_{in}(x,y) \tag{AI-17}$$

DISTANCE *pin1/out /EIGHT* :

$$p_{out}(x,y)=k\|(x,y)\text{ - (nearest black pixel)}\| \tag{AI-18}$$

COPY *pin pout (i j)* :

$$p_{out}(x,y)=p_{in}(x+i,y+j) \tag{AI-19}$$

Appendix II

VIPS Programs for DNA Sequence Reading

Numbers on RHS refer to where the algorithm is described in the main body of the thesis. [4.1(2)] refers to chapter 4 section 1 DFD step 2.

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

DNA Sequence Reading module: (see section 3.5)

```
PROGRAM                                                    !seq.vip ---- Main program.
   WRITE /LINE /LINE /LINE /LINE
   WRITE "          ************************************" /LINE
   WRITE "          *  DNA  Sequence Reading System   *" /LINE
   WRITE "          ************************************" /LINE
   WRITE /LINE /LINE /LINE /LINE
   DECLARE PROGRAM dna_acq dna_cont dna_line dna_sepa dna_sub dna_boun dna_warp
   DECLARE PROGRAM dna_geom dna_band dna_scan dna_join dna_manu dna_bline1 dna_bline2
   DECLARE STRING seq_imvar seq_auto seq_ok seq_subok seq_lepos seq_getyp seq_exit
   DECLARE STRING seq_manu seq_correct dna_dir
   DECLARE LIST seq_lp seq_rp seq_lepol
   DECLARE INTEGER seq_n seq_left seq_right seq_bacexist
   DECLARE IMAGE (512 512) seq_i seq_a seq_b seq_bac

   LET dna_dir = "dna\"

   LOAD 'dna_dir'acq.vip dna_acq
```

```
    LOAD 'dna_dir'cont.vip dna_cont
    LOAD 'dna_dir'line.vip dna_line
    LOAD 'dna_dir'sepa.vip dna_sepa
    LOAD 'dna_dir'sub.vip dna_sub
    LOAD 'dna_dir'boun.vip dna_boun
    LOAD 'dna_dir'warp.vip dna_warp
    LOAD 'dna_dir'geom.vip dna_geom
    LOAD 'dna_dir'band.vip dna_band
    LOAD 'dna_dir'scan.vip dna_scan
    LOAD 'dna_dir'join.vip dna_join
    LOAD 'dna_dir'manu.vip dna_manu
    LOAD 'dna_dir'bline1.vip dna_bline1
    LOAD 'dna_dir'bline2.vip dna_bline2

    SET OUTPUT /OFF
    ROAM (0 0)(512 512)
    LET seq_bacexist = 0

    REPEAT
!Step 1: select image input type and input a DNA sequence image to be read
        LET seq_exit = "N"
        WHILE seq_exit = "N"
            RUN dna_acq seq_i seq_bacexist seq_exit seq_bac          !acquires an image
            IF seq_exit = "Y"
                DELETE dna_* seq_*
                SET OUTPUT /ON
                EXIT
            END
        END
!Step 2: enhance contrast of the DNA sequence image
        RUN dna_cont seq_i seq_a                                    !enhances contrast of the image
        WRITE /LINE
        INQUIRE " Separate lane sets automatically ? [Y]  " /ENTITY seq_auto
        WRITE /LINE
        IF seq_auto = "N"
!Step 3: manually process
            RUN dna_manu seq_i seq_a                                !manually separates subimage
        ELSE
!Step 4: automatically process
            RUN dna_line seq_a seq_b seq_lp seq_rp                  !detects gap lines
            FOR seq_n = 0  %LENGTH(seq_lp - 1)
                DECLARE IMAGE (1 1) seq_seta seq_seti
                RUN dna_sepa seq_n seq_lp seq_rp seq_a seq_seta seq_i seq_seti!separates subimage
                WRITE /LINE
                INQUIRE " Is the selected subimage ok ? [Y]  " /ENTITY seq_ok
                WRITE /LINE
                IF seq_ok = "N"
                    INQUIRE " Correct the subimage ? [N]  " /ENTITY seq_correct
                    IF seq_correct = "Y"                            !select subimage manually
                        LET seq_subok = "N"
                        WHILE seq_subok = "N"
DISPLAY seq_a (0 0)
                            WRITE /LINE " Select a subimage to be processed " /LINE
                            INQUIRE " Tick left_top and drag to right_bottom: " seq_lt seq_size
                            DELETE seq_seti seq_seta
                            DECLARE IMAGE seq_size seq_seti seq_seta
                            COPY seq_i seq_seti ((0 - %ROW(seq_lt)) (0 - %COL(seq_lt)))
                            COPY seq_a seq_seta ((0 - %ROW(seq_lt)) (0 - %COL(seq_lt)))
                            CLEAR
DISPLAY seq_seti seq_lt
                            WRITE /LINE
                            INQUIRE " Is the selected subimage ok ? [y] " /ENTITY seq_subok
```

```
                          WRITE /LINE
                       END
                       CLEAR
DISPLAY seq_seti (0 0)
                       RUN dna_sub seq_seta seq_seti               !read subimage
                    END
                 ELSE
                    CLEAR
DISPLAY seq_seti (0 0)
                    RUN dna_sub seq_seta seq_seti
                 END
                 DELETE seq_set*
              END
           END
           WRITE /LINE
           INQUIRE " Select any subimage? [N] " /ENTITY seq_manu
           IF seq_manu = "Y"
              RUN dna_manu seq_i seq_a                             !manually selects subimage
           END
           WRITE /LINE /LINE /LINE /LINE
   UNTIL 1 = 2
END
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Acquire Image module: (see section 4.1)

```
PROGRAM                                                   !acq.vip ---- Acquires images.
    DECLARE STRING acq_type acq_imfile acq_bacfile acq_imvar acq_exit acq_imin
    DECLARE IMAGE (512 512) acq_i acq_a acq_bac acq_b
    DECLARE INTEGER acq_back

    LET acq_exit = "N"
    LET acq_back = 0
    WRITE /LINE " Select Image Input: " /LINE /LINE
    WRITE "    Capture Image(C)  Load Image File(F)   Exit(E)   [C]_ "
    INQUIRE "  " /ENTITY acq_type /LINE /LINE
    WRITE /LINE /LINE
    IF acq_type = "F"                                     !input image file
       LET #3 = "F"                                       !#3=seq_exit
       LET acq_imin = "N"
       REPEAT
          INQUIRE " Input Image File Name: " /ENTITY acq_imfile
          IF %LENGTH(acq_imfile) > 0
             IF %EXIST('acq_imfile' /FILE) = 0
                WRITE " The image file does not exist" /LINE
             ELSE
                LET acq_imin = "Y"
             END
          END
       UNTIL acq_imin = "Y"
       LOAD 'acq_imfile' acq_i /pic
       WHILE acq_back = 0                                 !#2=seq_bac
          INQUIRE " Input Background File Name: " /ENTITY acq_bacfile
          IF %LENGTH(acq_bacfile) > 0
             IF %EXIST('acq_bacfile' /FILE) = 0
                WRITE " The background image file does not exist" /LINE
             ELSE
```

```
                LOAD 'acq_bacfile' acq_bac /pic
                LET acq_back = 1
            END
        END
    END
ELSE
    IF acq_type = "E"                                  !exit DNA Sequence Reading
        INQUIRE " Exit DNA Sequence Reading [N] ? " /ENTITY acq_exit
        WRITE /LINE /LINE
        IF acq_exit = "Y"
            LET #3 = "Y"                               !#3=seq_exit
        ELSE
            LET #3 = "N"
        END
        DELETE acq_*
        SET ERROR /INFO
        EXIT
    ELSE                                               !capture DNA sequence image
        SET CUR /off
        WRITE /LINE /LINE "Capture an image of DNA sequence autoradiograph " /LINE
        CAPTURE 0 (0 0) (512 512)                      !capture DNA image        [4.1(1)]
        GET acq_i (0 0) (512 512)
        IF #2 = 0
            WRITE  "Remove everything to capture an image of the background " /LINE
            CAPTURE 0 (0 0) (512 512)                  !capture background       [4.1(2)]
            GET acq_bac (0 0) (512 512)
            LET #2 = 1
            LET #4 = acq_bac
        ELSE
            LET acq_bac = #4
        END
        SET CUR /on
        LET #3 = "C"
    END
END
DISPLAY acq_i (0 0)
    WRITE /LINE /LINE
    BOX AVERAGE acq_i acq_a (2 3)                      !smooths noise            [4.1(3)]
DISPLAY acq_a (0 0)
    BOX AVERAGE acq_bac acq_b 15                       !smooths background       [4.1(4)]
    DIVIDE acq_a acq_b 240                             !background normalize     [4.1(5)]
DISPLAY acq_a
    LET #1 = acq_a                                     !#1=seq_i
    DELETE acq_*
END
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Enhance Contrast module: (see section 4.2)

```
PROGRAM                                                !cont.vip ---- Enhances contrast.
    DECLARE IMAGE (512 512) cont_d cont_e cont_f cont_g cont_t

DISPLAY #1 (0 0)                                       !#1=seq_i
    BOX AVERAGE #1 cont_d (15 3)                       !vertically smooth        [4.2(1)]
    BOX MAXIMUM cont_d cont_e (1 80)                   !horizontal maximum       [4.2(2)]
    BOX MINIMUM cont_d cont_f (400 1)                  !vertically minimum       [4.2(3)]
    SUBTRACT cont_e cont_f /SATURATE                   !the range image          [4.2(4)]
```

```
    LET cont_g = #1
    SUBTRACT cont_g cont_f /SATURATE                    !the proportion image      [4.2(5)]
    ADD cont_g 5 /SATURATE
    DIVIDE cont_g cont_e 200                            !contrast stretching       [4.2(6)]
DISPLAY cont_g
    LET #2 = cont_g                                     !#2=seq_a
    DELETE cont_*
END
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Detect gap lines module: (see section 4.3)

```
PROGRAM                                                 !line.vip ---- Detects gap lines.
    DECLARE IMAGE (512 512) line_a line_b
    DECLARE CHAIN line_c line_chn
    DECLARE LIST line_cp line_fm line_leh line_rih line_lp line_rp line_len line_len1
    DECLARE LIST  line_l line_r line_tll line_brl
    DECLARE VECTOR line_lep line_rip line_p
    DECLARE INTEGER line_ch_n line_ch_bn line_p_n line_pn line_h_n line_n line_sortn line_sortm
    DECLARE STRING line_ok

    LET line_leh = { }                                  !gets gap line left and right point lists
    LET line_rih = { }
    LET line_len1 = { }
DISPLAY #1 (0 0)                                        !#1=seq_a
    BOX AVERAGE #1 line_a (3 3)
    BOX MINIMUM line_a line_b (400 3)                   !vertical minimum          [4.3(1)]
DISPLAY line_b
    THRESHOLD line_b 200                                !segments gaps between lane sets  [4.3(2)]
DISPLAY line_b
    THIN line_b /BINARY                                 !thins gap centre line     [4.3(3)]
DISPLAY line_b
    CHAIN CODE line_b line_c                            !codes gap lines           [4.3(4)]
    CHAIN BRANCH line_c line_ch_bn
!   CHAIN SIZE line_c line_tll line_brl
    IF line_ch_bn = 0
       WRITE "no line found"                            !reports error message
    ELSE
       FOR line_ch_n = 1 line_ch_bn                     !detects key points of gap [4.3(5)]
           CHAIN EXTRACT line_c line_ch_n line_chn
           POINTS line_chn line_cp
           LET line_lep = %INDEX(line_cp 1)
           LET line_rip = %INDEX(line_cp 1)
           FOR line_p_n = 2 %LENGTH(line_cp)
               IF %COL(line_lep) > %COL(%INDEX(line_cp line_p_n))
                   LET line_lep = %INDEX(line_cp line_p_n)      !puts the leftest point to left list
               END
               IF %COL(line_rip) < %COL(%INDEX(line_cp line_p_n))
                   LET line_rip = %INDEX(line_cp line_p_n)      !puts the rightest point to right list
               END
           END
           LET line_leh = line_leh & {line_lep}
           LET line_rih = line_rih & {line_rip}
           LET line_len1 = line_len1 & {%COL(%INDEX(line_cp %LENGTH(line_cp))) -
                                        %COL(%INDEX(line_cp 1))}
       END
```

```
        LET line_l = line_leh                                 !sorts lists from left to right
        LET line_r = line_rih
        LET line_len = { }
        SORT line_leh /h                                      !sorts left point list
        SORT line_rih /h                                      !sorts right point list
        !SORT line_tll /HORIZONTAL
        !SORT line_brl /HORIZONTAL
        FOR line_sortn = 1 %LENGTH(line_leh)                  !sorts line length list
            LET line_sortm = 1
            REPEAT
                IF %INDEX(line_leh line_sortn) = %INDEX(line_l line_sortm)
                    LET line_len = line_len & {%INDEX(line_len1 line_sortm)}
                    LET line_ok = "Y"
                ELSE
                    LET line_sortm = line_sortm + 1
                    LET line_ok = "N"
                END
            UNTIL line_ok = "Y"
        END

        LET line_lp = {%INDEX(line_leh 1)}                    !deletes false points
        LET line_rp = {%INDEX(line_rih 1)}
        LET line_p = %INDEX(line_leh 1)
        LET line_pn = 1
        FOR line_h_n = 2 (line_ch_bn - 1)
            IF (%COL(%INDEX(line_leh line_h_n)) - %COL(line_p)) > 45
                LET line_lp = line_lp & {%INDEX(line_leh line_h_n)}
                LET line_rp = line_rp & {%INDEX(line_rih line_h_n)}
                LET line_p = %INDEX(line_leh line_h_n)
                LET line_pn = line_h_n
            ELSE
                IF (%COL(%INDEX(line_rih line_h_n)) - %COL(line_p)) > 45
                    LET line_lp = line_lp & {%INDEX(line_leh line_h_n)}
                    LET line_rp = line_rp & {%INDEX(line_rih line_h_n)}
                    LET line_p = %INDEX(line_leh line_h_n)
                LET line_pn = line_h_n
            ELSE
                IF %INDEX(line_len line_h_n) > %INDEX(line_len line_pn)
                    LET line_lp = line_lp & {%INDEX(line_leh line_h_n)}
                    LET line_rp = line_rp & {%INDEX(line_rih line_h_n)}
                    LET line_p = %INDEX(line_leh line_h_n)
                    LET line_pn = line_h_n
                END
            END
        END
    END

    LET #2 = line_b                                           !#2=seq_b
    LET #3 = line_lp                                          !#3=seq_lp
    LET #4 = line_rp                                          !#4=seq_rp
    END
    DELETE line_*
END
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Separate Subimage module: (see section 4.4)

```
PROGRAM                                                    !sepa.vip ---- Separates lane sets
    DECLARE INTEGER sepa_n sepa_le sepa_ri
    DECLARE STRING sepa_ok

    LET sepa_n = #1                                        !#1=seq_n
    REPEAT
       IF sepa_n = 0
          LET sepa_le = 0
          LET sepa_ri = %COL(%INDEX(#3  (sepa_n + 1)))
       ELSE
          IF sepa_n = %LENGTH(#2)                          !#2=seq_lp
             LET sepa_le = %COL(%INDEX(#2  sepa_n ))
             LET sepa_ri = (%COL(%SIZE(#4)) - 1)
          ELSE
             LET sepa_le = %COL(%INDEX(#2  sepa_n ))
             LET sepa_ri = %COL(%INDEX(#3  (sepa_n + 1)))  !#3=seq_rp
          END
       END
       IF (sepa_ri - sepa_le) > 40
          DELETE sepa_set* #5 #7
          DECLARE image ((%ROW(%SIZE(#4))) (sepa_ri - sepa_le)) sepa_seta sepa_seti #5 #7
          COPY #4 sepa_seta (0 (0 - sepa_le))              !#4=seq_a
          COPY #6 sepa_seti (0 (0 - sepa_le))              !#6=seq_i
          CLEAR
DISPLAY sepa_seta (0 sepa_le)
DISPLAY sepa_seti (0 0)

          LET #5 = sepa_seta                               !#5=seq_seta
          LET #7 = sepa_seti                               !#7=seq_seti
          LET sepa_ok = "Y"
       ELSE
          LET sepa_n = sepa_n + 1
       END
    UNTIL sepa_ok = "Y"
    LET #1 = sepa_n
    DELETE sepa_*
END
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Subimage Reading module: (see section 3.5)

```
PROGRAM                                                    !sub.vip ---- Processes subimage.
    DECLARE STRING sub_ok sub_giv sub_seq
    DECLARE IMAGE (%SIZE(#1)) sub_setg sub_setp
    DECLARE INTEGER sub_bn

    RUN dna_boun #1 sub_setg                               !#1=seq_seta
    RUN dna_warp #2 sub_setg sub_setp                      !#2=seq_seti

    REPEAT                                                 !geometry correction manually
       WRITE /LINE
       INQUIRE " Is the geometry corrected image ok ? [Y] "  /ENTITY sub_ok
       IF sub_ok = "N"
          INQUIRE " Give up the subimage ? [Y] "  /ENTITY sub_giv
          IF sub_giv = "N"
             RUN dna_geom sub_setp #2
          ELSE
```

```
            DELETE sub_*
            SET ERROR /INFO
            EXIT
         END
      ELSE
         LET sub_ok = "Y"
      END
   UNTIL sub_ok = "Y"

   DECLARE IMAGE (%SIZE(sub_setp)) sub_setc sub_setb
   RUN dna_band sub_setp sub_setc sub_setb sub_bn
   RUN dna_scan sub_setc sub_seq sub_bn sub_setb
   RUN dna_join sub_seq
   DELETE sub_*
END
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Extract Boundaries module: (see cection 4.5)

```
PROGRAM                                              !boun.vip --- Extracts boundaries of lane set.
   DECLARE IMAGE ((%ROW(%SIZE(#1))) (%COL(%SIZE(#1)))) boun_b boun_c boun_d boun_e
                                                        boun_f boun_g boun_gg
   DECLARE MASK boun_mask

   BOX AVERAGE #1 boun_b (40 3)                       !#1=seq_seta vertically smooth   [4.5(1)]
   SET MASK boun_mask {1 0 -1 1 0 -1 1 0 -1}
   FILTER LINEAR boun_b boun_c boun_mask 2.0 /ABSOLUTE !filters boundaries of lane set[4.5(2)]
   EXTEND boun_c
   BOX AVERAGE boun_c boun_d (30 3)                        !smooths boundaries
   BOX EXTREME /max boun_d boun_e (1 15)              !obtains the most changed edges   [4.5(3)]
   AND boun_e boun_d                                       !removes false edges        [4.5(4)]
   THRESHOLD boun_e 15
   LET boun_f = boun_b
   THRESHOLD boun_f 0 240
   DISTANCE boun_f                                         !distance transform         [4.5(5)]
   EXPAND boun_f
   INVERT boun_f
   AND boun_e boun_f                                       !removes centre edges       [4.5(6)]
   THRESHOLD boun_e 200
   LET boun_g = boun_f                                     !distinguish left and righ  [4.5(7)]
   THRESHOLD boun_g 0 254
   BOX AVERAGE boun_g boun_gg (51 21)
   COPY boun_gg boun_g (0 1)
   SUBTRACT boun_g boun_gg /off
   EXPAND boun_g 108 148
   BOX AVERAGE boun_g boun_gg (3 10)
   AND boun_gg boun_e                                 !boundaries with different intensities   [4.5(8)]
   LET #2 = boun_gg                                        !#2=sub_setg
   DELETE boun_*
END
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Warp Geometry module: (see section 4.6)

```
PROGRAM                                                    !warp.vip ---- Geometry warping.
   DECLARE IMAGE ((%ROW(%SIZE(#2))) (%COL(%SIZE(#2)))) warp_a warp_leg warp_leedge
   DECLARE IMAGE ((%ROW(%SIZE(#2))) (%COL(%SIZE(#2)))) warp_rig warp_riedge warp_i
   DECLARE IMAGE ((%ROW(%SIZE(#2))) 100) warp_ip
   DECLARE CHAIN warp_lechai warp_lecn warp_rich warp_ricn
   DECLARE LIST warp_lep warp_lep2 warp_p warp_rip warp_rip2 warp_newp
   DECLARE VECTOR warp_point
   DECLARE INTEGER warp_chbn warp_chn warp_pn warp_sn warp_t1 warp_t
   DECLARE REAL warp_shift warp_mean warp_sd

   LET warp_i = #1                                         !#1=seq_seti
   LET warp_leg = #2                                       !#2=seq_setg
DISP warp_i (0 200)
   CLEAR 0 (0 0) (512 200)
   THRESHOLD warp_leg 1 127                                !convert left boundary      [4.6(1)]
   BOX AVERAGE warp_leg warp_leedge (40 3)                 !smooth edges
   THRESHOLD warp_leedge 30
   THIN warp_leedge                                        !left edges
   CHAIN CODE warp_leedge warp_lechai
   CHAIN BRANCHES warp_lechai warp_chbn
   IF warp_chbn = 0
      write "no boundary found"
   ELSE
      FOR warp_chn = 1 warp_chbn
         CHAIN EXTRACT warp_lechai warp_chn warp_lecn       !get one left edge
         IF warp_chn = 1
            POINTS warp_lecn warp_lep                      !get first part of the key point list
         ELSE
            POINTS warp_lecn warp_lep2                     !get other parts of the point list
            IF (%COL(%INDEX(warp_lep2 1))- %COL(%INDEX(warp_lep 1))) < 15
                                                           !remove false points        [4.6(2)]
               IF (%COL(%INDEX(warp_lep 1))- %COL(%INDEX(warp_lep2 1))) < 15
                  IF %ROW(%INDEX(warp_lep2 1)) < %ROW(%INDEX(warp_lep
                     %LENGTH(warp_lep)))
                     IF (%ROW(%INDEX(warp_lep2 %LENGTH(warp_lep2)))
                           - %ROW(%INDEX(warp_lep2 1))) > (%ROW(%INDEX(warp_lep
                           %LENGTH(warp_lep)))- %ROW(%INDEX(warp_lep 1)))
                        LET warp_lep = warp_lep2
                     END
                  ELSE
                     ADD warp_lep warp_lep2 /END
                  END
               ELSE
                  IF (%ROW(%INDEX(warp_lep2 %LENGTH(warp_lep2)))
                        - %ROW(%INDEX(warp_lep2 1))) > (%ROW(%INDEX(warp_lep
                        %LENGTH(warp_lep)))- %ROW(%INDEX(warp_lep 1)))
                     LET warp_lep = warp_lep2
                  END
               END
            ELSE
               IF (%ROW(%INDEX(warp_lep2 %LENGTH(warp_lep2)))
                     - %ROW(%INDEX(warp_lep2 1))) > (%ROW(%INDEX(warp_lep
                     %LENGTH(warp_lep)))- %ROW(%INDEX(warp_lep 1)))
                  LET warp_lep = warp_lep2
               END
            END
         END
```

```
    END
END
IF %LENGTH(warp_lep) > 0
    IF %ROW(%INDEX(warp_lep 1)) > 0
        LET warp_p = warp_lep
        LET warp_lep = { }
        LET warp_lep = {(0 (%COL(%INDEX(warp_p 1))))}
        ADD warp_lep warp_p /END
    END
ELSE
    LET warp_lep = {(0 0) (511 0)}
END
IF %ROW(%INDEX(warp_lep %LENGTH(warp_lep))) < (%ROW(%SIZE(#1)) - 1)
    LET warp_lep = warp_lep & {(((%ROW(%SIZE(#1))) - 1) (%COL(%INDEX(warp_lep
                                %LENGTH(warp_lep))))))}
END
STRAIGHTEN warp_i warp_lep /v                          !straighten left side      [4.6(3)]

LET warp_rig = #2                                      !get right edge
THRESHOLD warp_rig 128 255                             !converts right boundary   [4.6(4)]
BOX AVERAGE warp_rig warp_riedge (40 3)
THRESHOLD warp_riedge 30
THIN warp_riedge                                       !right edges
CHAIN CODE warp_riedge warp_rich
CHAIN BRANCHES warp_rich warp_chbn
IF warp_chbn = 0
    WRITE "no boundary found"
ELSE
    FOR warp_chn = 1 warp_chbn
        CHAIN EXTRACT warp_rich warp_chn warp_ricn
        IF warp_chn = 1
            POINTS warp_ricn warp_rip                  !get first part of the key point list
        ELSE
            POINTS warp_ricn warp_rip2                 !get other part of the point list
            IF %ROW(%INDEX(warp_rip2 1)) < %ROW(%INDEX(warp_rip %LENGTH(warp_rip)))
                IF %LENGTH(warp_rip2) > %LENGTH(warp_rip)
                    LET warp_rip = warp_rip2
                END
            ELSE
                ADD warp_rip warp_rip2 /END            !join available parts of point list
            END
        END
    END
END

LET warp_newp = { }                                    !remove false points  [4.6(5)]
FOR warp_pn = 1 %LENGTH(warp_rip)
    FOR warp_sn = 1 %LENGTH(warp_lep)                  !get shift
        IF %ROW(%INDEX(warp_rip warp_pn)) < %ROW(%INDEX(warp_lep warp_sn))
            LET warp_t1 = %REAL((%COL(%INDEX(warp_lep warp_sn))
                                - %COL(%INDEX(warp_lep (warp_sn - 1)))))
            LET warp_t = warp_t1 / %REAL((%ROW(%INDEX(warp_lep warp_sn))
                                - %ROW(%INDEX(warp_lep (warp_sn - 1)))))
            LET warp_shift = warp_shift + warp_t * %REAL((%ROW(%INDEX(warp_rip warp_pn))
                                - %ROW(%INDEX(warp_lep (warp_sn - 1)))))
            LET warp_point = (%ROW(%INDEX(warp_rip warp_pn)) (%COL(%INDEX(warp_rip
                                warp_pn)) - %INTEGER(warp_shift)))
            ADD warp_newp {warp_point} /END
            LET warp_sn = %LENGTH(warp_lep) + 1
        ELSE
            LET warp_shift = %REAL(%COL(%INDEX(warp_lep warp_sn)))
            IF %ROW(%INDEX(warp_rip warp_pn)) = %ROW(%INDEX(warp_lep warp_sn))
```

```
                LET warp_point = (%ROW(%INDEX(warp_rip warp_pn))
                            (%COL(%INDEX(warp_rip warp_pn)) - %INTEGER(warp_shift)))
                ADD warp_newp {warp_point} /END
                LET warp_sn = %LENGTH(warp_lep) + 1
              END
            END
          END
        END
        IF %ROW(%INDEX(warp_newp 1)) > 0
          LET warp_p = warp_newp
          LET warp_newp = {}
          LET warp_newp = {(0 (%COL(%INDEX(warp_p 1))))}
          ADD warp_newp warp_p /END
        END
        IF %ROW(%INDEX(warp_newp %LENGTH(warp_newp))) < (%ROW(%SIZE(#1)) - 1)
          ADD warp_newp {(((%ROW(%SIZE(#1))) - 1) (%COL(%INDEX(warp_newp
                          %LENGTH(warp_newp)))))} /END
        END

        STATISTICS warp_i ,,,, warp_mean warp_sd
        TEST warp_ip %INT(((warp_mean - warp_sd) * 1.02))
        PARALLEL warp_i warp_ip warp_newp /v              !trapezoid warp right side  [4.6(6)]
        EXTEND warp_ip
DISP warp_ip (0 300)
        DELETE #3
        DECLARE IMAGE ((%ROW(%SIZE(#2))) 100) #3
        LET #3 = warp_ip                                   !#3=seq_setg
        DELETE warp_*
END
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Manual Correction module: (see section 3.4 Figure 3.4-3)

```
PROGRAM                                        !geom.vip ---- Manually geometry correction.
    DECLARE IMAGE geom_seta geom_setp
    DECLARE STRING geom_cor geom_res geom_le geom_lepos geom_ri geom_ripos geom_ok
    DECLARE INTEGER geom_n
    DECLARE REAL geom_mean geom_sd
    DECLARE LIST geom_list geom_lel geom_ril

    LET geom_cor = "Y"
    WHILE geom_cor = "Y"
      WRITE /LINE
      LET geom_seta = #1                               !#1=subsetp
      INQUIRE " Correct from the result ? [Y]  " /ENTITY geom_res
      IF geom_res = "N"
        LET geom_seta = #2                             !#2=subseti
        CLEAR 0 (0 300) (512 100)
DISP geom_seta (0 300)
      END

      REPEAT                                           !straighten left side manually
        INQUIRE " Correct left side ? [N]  " /ENTITY geom_le
        IF geom_le = "Y"
          INQUIRE " Tick points of left boundary to strait " /LINE geom_lepos
          IF %LENGTH(geom_lepos) > 0
            LET geom_list = {'geom_lepos'}
```

```
                LET geom_lel = { }
                FOR geom_n = 1 %LENGTH(geom_list)
                    ADD geom_lel {(%INDEX(geom_list geom_n) - (0 300))} /END
                END
                STRAIGHTEN geom_seta geom_lel /v
DISP geom_seta (0 300)
            END
        ELSE
            LET geom_le = "N"
        END
    UNTIL geom_le = "N"

    REPEAT                                              !straighten right side manually
        INQUIRE " Correct right side ? [N]  " /ENTITY geom_ri
        IF geom_ri = "Y"
            INQUIRE " Tick points of right boundary to warp " /LINE geom_ripos
            IF %LENGTH(geom_ripos) > 0
                LET geom_list = {'geom_ripos'}
                LET geom_ril = { }
                FOR geom_n = 1 %LENGTH(geom_list)
                    ADD geom_ril {(%INDEX(geom_list geom_n) - (0 300))} /END
                END
                STATISTICS geom_seta ,,,, geom_mean geom_sd
                TEST geom_setp %INT(((geom_mean - geom_sd) * 1.02))
                PARALLEL geom_seta geom_setp geom_ril /v
                EXTEND geom_setp
DISP geom_setp (0 300)
                LET geom_seta = geom_setp
            END
        ELSE
            LET geom_ri = "N"
        END
    UNTIL geom_ri = "N"

    INQUIRE " Correcte the subimage again? [N]  " /ENTITY geom_cor
  END
  LET #1 = geom_seta
  DEL geom_*
END
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Extract Bands module: (see section 4.7)

```
PROGRAM                                                 !band.vip --- Extracts bands.
    DECLARE IMAGE (512 100)  band_a band_b band_c band_d band_e band_f band_g
    DECLARE IMAGE (512 100)  band_h band_i band_ta band_tb
    DECLARE MASK band_mask
    DECLARE INTEGER band_wb
    DECLARE REAL band_mean band_sd

    BOX AVERAGE #1 band_a (2 5)                         !#1=seq_setp
    LET band_b = band_a
    AVERAGE /MINIMUM /ROW /FILL band_b band_c           !enhances contrast
    BOX MINIMUM band_c band_d (9 1)
    BOX MAXIMUM band_b band_e (21 1)
    SUBTRACT band_e band_d /sat
    SUBTRACT band_b band_d /sat
```

```
    DIVIDE band_b band_e
DISP band_b (0 300)
    LET band_ta = band_b
    BOX AVERAGE band_ta band_tb (5 3)                    !get band number for window size
    SUBTRACT band_ta band_tb /OFF /SAT
    STATISTICS band_ta ,,,, band_mean band_sd
    RUN dna_bline1 band_ta %INT(((band_mean - band_sd) * 1.03))
    BOX AVERAGE band_ta band_tb (1 11)
    STATISTICS band_tb ,,,, band_mean band_sd
    THRESHOLD band_tb 0 %INT(((band_mean - band_sd) * 0.97))
    BLOB band_tb ,, band_wb
    IF band_wb > 85                                      !reenhances contrast
        LET band_b = band_a
        AVERAGE /MINIMUM /ROW /FILL band_b band_c
        BOX MINIMUM band_c band_d (5 1)
        BOX MAXIMUM band_b band_e (21 1)
        SUBTRACT band_e band_d /sat
        SUBTRACT band_b band_d /sat
        DIVIDE band_b band_e
DISP band_b (0 300)
        IF band_wb > 130
            LET band_b = band_a
            AVERAGE /MINIMUM /ROW /FILL band_b band_c
            BOX MINIMUM band_c band_d (3 1)
            BOX MAXIMUM band_b band_e (21 1)
            SUBTRACT band_e band_d /sat
            SUBTRACT band_b band_d /sat
            DIVIDE band_b band_e
DISP band_b (0 300)
        END
    END
    BOX MAXIMUM band_b band_f (1 60)                     !remove background      [4.7(1)]
    BOX AVERAGE band_f band_d (9 69)
    SUBTRACT band_d band_b /sat
    IF band_wb < 85
    SUBTRACT band_d 100 /sat
    ELSE
        IF band_wb < 130
            SUBTRACT band_d 80 /sat
        ELSE
            SUBTRACT band_d 50 /sat
        END
    END
    INVERT band_d
    EXPAND band_d                                        !background subtracted
    RUN dna_bline1 band_d 255                            !keeps centre part of each lane  [4.7(2)]
    box ave band_c band_d (2 3)                          !smooth                 [4.7(3)]
    SET MASK band_mask {1 1 1 2 2 2 -3 -3 -3}
    TEST band_e
    FILTER LINEAR band_d band_e band_mask /POSITIVE      !filter bands           [4.7(4)]
    EXTEND band_e
    BOX AVERAGE band_e band_g
    BOX ENHANCE band_g band_h (3 13)                     !further enhances image
    BOX MAXIMUM band_h band_i (1 24)                     !long bands image       [4.7(5)]
    RUN dna_bline1 band_i 255
    THRESHOLD band_i 2 255                               !segment bands          [4.7(6)]
    RUN dna_bline2 band_i 0                              !thin bands             [4.7(7)]
    LET #2 = band_i
    LET #3 = band_b
    LET #4 = band_wb
    DELETE band_*
END
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Scan Bands module: (see section 4.8)

```
PROGRAM                                                          !scan.vip ---- Scans bands.
    DECLARE IMAGE ((%ROW(%SIZE(#1))) (%COL(%SIZE(#1)))) scan_c
    DECLARE INTEGER scan_len scan_n scan_k scan_dn scan_dm scan_d scan_un
    DECLARE INTEGER scan_len1 scan_len2 scan_len3 scan_resort
    DECLARE REAL scan_width
    DECLARE LIST scan_data scan_add_bands scan_del_bands scan_bands scan_uncer scan_off
    DECLARE STRING scan_long scan_string scan_seq scan_corr scan_zoom scan_ad
    DECLARE STRING scan_ok scan_name scan_addyes scan_delyes scan_offyes
    DECLARE VECTOR scan_pos1 scan_pos2 scan_pos scan_lt scan_offsize scan_add scan_del
    DECLARE CHAIN scan_chain scan_chaink
    DECLARE MASK scan_mask

    LET scan_data = {}                                           !scan bands
    LET scan_uncer = {(-1 -1)}
    LET scan_long = " Band is too wide!"
    LET scan_c = #1                                              !#1=sub_setc

    CHAIN CODE scan_c scan_chain                                 !scan band positions      [4.8(1)]
    CHAIN BRANCHES scan_chain scan_n
    FOR scan_k = 1 scan_n                                        !extract each band
        CHAIN EXTRACT scan_chain scan_k scan_chaink
        CHAIN SIZE scan_chaink scan_pos1 scan_pos2
        LET scan_pos = (scan_pos1 + scan_pos2 + (1 1)) / 2
        CHAIN LENGTH scan_chaink scan_len
        IF scan_k < 5
            LET scan_width = 4
        ELSE
            LET scan_width = %ROW(scan_pos) / scan_k
        END
        IF #3 > 100
            LET scan_len1 = %INTEGER((scan_width * 4.5))
            LET scan_len2 = %INTEGER((scan_width * 6.5))
            LET scan_len3 = %INTEGER((scan_width * 8.5))
        ELSE
            LET scan_len1 = %INTEGER((scan_width * 3.2))
            LET scan_len2 = %INTEGER((scan_width * 5.1))
            LET scan_len3 = %INTEGER((scan_width * 7.0))
        END
        IF scan_len < scan_len1                     !normal band
            ADD scan_data {scan_pos} /END
            DRAW LINE (scan_pos + (0 297)) (scan_pos + (0 303))    !draw band extracted for user
            FOR scan_un = 1 %LENGTH(scan_uncer)
                IF %ROW(%INDEX(scan_uncer scan_un)) = %ROW(scan_pos)
                    DRAW LINE (scan_pos + (-1 300)) (scan_pos + (1 300)) 0  !sign uncertain band
                END
                LET scan_uncer = {scan_pos}
            END
        ELSE
            IF scan_len < scan_len2
                ADD scan_data {(scan_pos - (%INT((scan_width / 2)) 0)) (scan_pos
                                + (%INT((scan_width / 2)) 0))} /END
                DRAW LINE (scan_pos + ((0 - (%INT((scan_width / 2)))) 297)) (scan_pos
                                + ((0 - (%INT((scan_width / 2)))) 303))
```

```
            DRAW LINE (scan_pos + (%INT((scan_width / 2)) 297)) (scan_pos
                                    + (%INT((scan_width / 2)) 303))
        ELSE
            IF scan_len < scan_len3
                ADD scan_data {(scan_pos - (%INT(scan_width) 0)) scan_pos (scan_pos
                                            + (%INT(scan_width) 0))} /END
                DRAW LINE (scan_pos + ((0 - (%INT(scan_width))) 297)) (scan_pos
                                    + ((0 - (%INT(scan_width))) 303))
                DRAW LINE (scan_pos + (0 297)) (scan_pos + (0 303))
                DRAW LINE (scan_pos + (%INT(scan_width) 297)) (scan_pos
                                    + (%INT(scan_width) 303))
            ELSE
                WRITE scan_long
            END             !end if len<=three
        END             !end if len<=two
    END         !end if len<=one
END         !end for
SORT scan_data /VERTICAL                                    !vertically sort positions  [4.8(2)]
SEQUENCE scan_data scan_string /VERTICAL                    !DNA sequencing             [4.8(3)]
WRITE /LINE scan_string /LINE

WRITE /LINE                                                 !check detected bands       [4.8(4)]
INQUIRE " Correct sequence ?  [N]  " /ENTITY scan_corr
IF scan_corr = "Y"
    INQUIRE " Zoom in ?  [N] " /ENTITY scan_zoom
    IF scan_zoom = "Y"
        ROAM (0 200) (256 256)
    END
END
WHILE scan_corr = "Y"
    LET scan_add_bands = { }
    LET scan_del_bands = { }
    LET scan_off = { }
    LET scan_bands = { }

    INQUIRE " Add bands(A)   Delete band(D)   Remove part(R)?  [N] " /ENTITY scan_ad
    WRITE /LINE
    WHILE scan_ad = "A"                                     !add bands
        INQUIRE  " Tick a band to be added: "  /ENTITY scan_addyes
        IF %LENGTH(scan_addyes) > 0
            LET scan_add = 'scan_addyes'
            IF %COLUMN(scan_add) < (25 + 300)
                LET scan_add = (%ROW(scan_add) 12)
            ELSE
                IF %COLUMN(scan_add) < (50 + 300)
                    LET scan_add = (%ROW(scan_add) 37)
                ELSE
                    IF %COLUMN(scan_add) < (75 + 300)
                        LET scan_add = (%ROW(scan_add) 62)
                    ELSE
                        IF %COLUMN(scan_add) < (100 + 300)
                            LET scan_add = (%ROW(scan_add) 87)
                        END
                    END
                END
            END
            DRAW LINE (scan_add + (0 297)) (scan_add + (0 303))
            ADD scan_add_bands {scan_add} /END
        ELSE
            LET scan_ad = "N"
        END
    END
```

```
ADD scan_data scan_add_bands /END                          !join added bands

WHILE scan_ad = "D"                                        !delete band
   INQUIRE " Tick a band to be deleted: "  /ENTITY scan_delyes
   IF %LENGTH(scan_delyes) > 0
      LET scan_del = 'scan_delyes'
      IF %COLUMN(scan_del) < (25 + 300)
         LET scan_del = (%ROW(scan_del) 12)
         IF %COLUMN(scan_del) < (50 + 300)
            LET scan_del = (%ROW(scan_del) 37)
         ELSE
            IF %COLUMN(scan_del) < (75 + 300)
               LET scan_del = (%ROW(scan_del) 62)
            ELSE
               IF %COLUMN(scan_del) < (100 + 300)
                  LET scan_del = (%ROW(scan_del) 87)
               END
            END
         END
      END
      DRAW LINE (scan_del + (0 297)) (scan_del + (0 303)) 0
      ADD scan_del_bands {scan_del} /END
   ELSE
      LET scan_ad = "N"
   END
END
IF %LENGTH(scan_del_bands) > 0                             !removed bands
   FOR scan_dn = 1 %LENGTH(scan_data)
      LET scan_d = 0
      FOR scan_dm = 1 %LENGTH(scan_del_bands)
         IF %INDEX(scan_data scan_dn) = %INDEX(scan_del_bands scan_dm)
            LET scan_d = 1
         END
      END
      IF scan_d < 1
         ADD scan_bands {%INDEX(scan_data scan_dn)} /END
      END
   END
   LET scan_data = scan_bands
END

WHILE scan_ad = "R"                                        !remove unreadable part
   INQUIRE " Tick left_top and drag to right_bottom of the unreadable part " scan_lt scan_offsize
   FOR scan_dn = 1 %LENGTH(scan_data)
      IF %ROW(%INDEX(scan_data scan_dn)) > %ROW(scan_lt)
         IF %ROW(%INDEX(scan_data scan_dn)) < (%ROW(scan_lt) + %ROW(scan_offsize ))
            ADD scan_off {%INDEX(scan_data scan_dn)} /END
         END
      END
   END
   INQUIRE " Remove another unreadable part ?  [N] " /ENTITY scan_offyes
   IF scan_offyes = "Y"
      LET scan_ad = "R"
   ELSE
      LET scan_ad = "N"
      IF %LENGTH(scan_off) > 0                             !delete removed bands
         DECLARE IMAGE scan_offsize scan_offim
         COPY #4 scan_offim ((0 - %ROW(scan_lt)) 0)
DISP scan_offim scan_lt
         DELETE scan_offim
         LET scan_bands = { }
         FOR scan_dn = 1 %LENGTH(scan_data)
```

```
                    LET scan_d = 0
                    FOR scan_dm = 1 %LENGTH(scan_off)
                        IF %INDEX(scan_data scan_dn) = %INDEX(scan_off scan_dm)
                            LET scan_d = 1
                        END
                    END
                    IF scan_d < 1
                        ADD scan_bands {%INDEX(scan_data scan_dn)} /END
                    END
                END
                LET scan_data = scan_bands
            END
        END
    END
    WRITE /line
    INQUIRE " Correct sequence ? [N] " /ENTITY scan_corr
END    !while scan_corr="Y"
ROAM (0 0) (512 512)

LET scan_resort = 1                                    !resequencing
IF %LENGTH(scan_add_bands) = 0                         !sign resort
    IF %LENGTH(scan_del_bands) = 0
        IF %LENGTH(scan_off) = 0
            LET scan_resort = 0
        END
    END
END
IF scan_resort > 0                                     !resequencing if sequence corrected
    SORT scan_data /VERTICAL
    SEQUENCE scan_data scan_seq /VERTICAL
    WRITE /LINE scan_seq /LINE /LINE
END

REPEAT                                                 !save sequence
    INQUIRE " Name of this subsequence:" /ENTITY scan_name
UNTIL %LENGTH( scan_name ) > 0
FILE /OPEN/WRITE 'dna_dir"scan_name'.dat
WRITE /FILE scan_seq
FILE /CLOSE 'dna_dir"scan_name'.dat

LET #2 = scan_seq
DEL scan_*
END


PROGRAM                                    !bline1.vip ---- Keeps centre of each lane.
    DECLARE IMAGE (512 17) subline
    TEST subline #2
    COPY subline #1 (0 -8)
    COPY subline #1 (0 17)
    COPY subline #1 (0 42)
    COPY subline #1 (0 67)
    COPY subline #1 (0 92)
    DELETE subline
END

PROGRAM                                    !bline2.vip ---- Keeps centre positions.
    DECLARE image (512 17) subline
    TEST subline #2
    COPY subline #1 (0 -5)
    COPY subline #1 (0 13)
    COPY subline #1 (0 20)
```

```
    COPY subline #1 (0 38)
    COPY subline #1 (0 45)
    COPY subline #1 (0 63)
    COPY subline #1 (0 70)
    COPY subline #1 (0 88)
    DELETE subline
END
```

&&

Manually Process module: (see section 3.5)

```
PROGRAM                                              !manu.vip ---- Selects a lane set manally.
    DECLARE VECTOR manu_lt manu_size
    DECLARE STRING manu_ok manu_sel
    DECLARE INTEGER manu_set

    LET manu_sel = "Y"
    REPEAT
    LET manu_ok = "N"
    LET manu_set = 0
    WHILE manu_ok = "N"
DISPLAY #1 (0 0)
        WRITE /LINE " Select a subimage to be processed " /LINE
        INQUIRE " Tick left_top and drag to right_bottom:" manu_lt manu_size
            IF manu_set = 1
                DELETE manu_seti manu_seta
            END
            DECLARE IMAGE manu_size manu_seti manu_seta
            LET manu_set = 1
            COPY #1 manu_seti ((0 - %ROW(manu_lt)) (0 - %COL(manu_lt)))        !#1=seqi
            COPY #2 manu_seta ((0 - %ROW(manu_lt)) (0 - %COL(manu_lt)))        !#2=seqa
            CLEAR
DISPLAY manu_seti manu_lt
            INQUIRE " Is the selected subimage ok ?  [Y]  " /ENTITY manu_ok
        END
        RUN dna_sub manu_seta manu_seti
        WRITE /LINE
        DELETE manu_seti manu_seta
        INQUIRE " Select another subimage ? [Y] " /ENTITY manu_sel
    UNTIL manu_sel = "N"
    DELETE manu_*
END
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Appendix III

C Programs of VIPS Commands developed

STRAIGTHEN.H:

```
enum { IPO_STRA_HORIZONTALLY = 1,
     IPO_STRA_VERTICALLY   = 2,
     IPO_STRA_WARP         = 4,
     IPO_STRA_BLACK        = 8,
     IPO_STRA_WHITE        = 16 };

ipv_status ipif_straighten( char *params );
ipv_status ipc_straighten( short options, ipv_image *image, ipv_list **vectors, long *position );

extern ipv_command ipcc_straighten;
```

STRAIGTHEN.C:

```
/*-----------------------------------------------------------------------------------------------------------------------
Format:    STRAIGHTEN image vectors position [options]
Parameters: image       (IMAGE)      <i> The input image to be straightened.
            vectors     (LIST)       <i> The vectors to be straightened horizontally or vertically.
            position    (INTEGER)    <i> The position (row/column) straightened line is set at.
            options     (OPTION) [/HORIZONTALLY/WARP]<i>The direction the image to be
                                                  straightened by and the mode to fill rest pixels.
                  /HORIZONTALLY      The image to be straightened horizontally.
                  /VERTICALLY  The image to be straightened vertically.
                  /WARP              Fill rest pixels with warping around.
                  /BLACK             Fill rest pixels with 0.
                  /WHITE             Fill rest pixels with 255.
-----------------------------------------------------------------------------------------------------------------------*/
#include <stddef.h>
```

```
#include "g_error.h"
#include "g_defin.h"
#include "u_option.h"
#include "u_parame.h"
#include "u_image.h"
#include "c_straig.h"

/*-----------------------------------------------------------------------------------------------------------------------------------------*/
long straight_pos;
long straight_pos_def = 0;

ipv_param ipcp_straight[3] =
  {{ IPVO_REQD, IPT_B_IMAGE, IPTM_B_IMAGE, NULL, NULL, 0 },
   { IPVO_REQD, IPT_LIST, IPTM_LIST, NULL, NULL, 0 },
   { 0, IPT_INTEGER, IPTM_INTEGER, &straight_pos, &straight_pos_def, 0 }};

ipv_com_variant ipcv_straight[1] =
  {{ 0, 0, (ipv_proc *) ipc_straighten, NULL, 3, ipcp_straight }};

ipv_command ipcc_straighten =
  { "STRAIGHTEN", "/HORIZONTALLY/VERTICALLY/WARP/BLACK/WHITE",
    IPO_STRA_VERTICALLY|IPO_STRA_WHITE,
    IPO_STRA_HORIZONTALLY|IPO_STRA_WARP,
    1, ipcv_straight };

#define WHERE ipcc_straighten.command
/*-----------------------------------------------------------------------------------------------------------------------------------------*/

ipv_status ipif_straighten( char *params ){
   ipv_status     stat;
   ipv_var_desc   image, vectors, position;        /* pointers to actual parameters*/
   long           def_position;                  /* Defined line on which straightened points will be */
   short          options;                        /* Direction and mode to fill rest pixels*/

   ipum_error(ipu_options( &options, params, "/HORIZONTALLY /VERTICALLY
                                        /WARP /BLACK /WHITE" ));
   if ((options & 0x3) == 0) options |= IPO_STRA_HORIZONTALLY;
   if ((options & 0x1c) == 0) options |= IPO_STRA_WARP;

   ipum_error(ipu_parameter( &params, 1, IPVO_REQD, &image,
                             IPT_B_IMAGE, IPTM_B_IMAGE ));

   ipum_error(ipu_parameter( &params, 2, IPVO_REQD, &vectors,
                             IPT_LIST, IPTM_LIST ));

   stat = ipu_parameter( &params, 3, 0, &position, IPT_INTEGER, IPTM_INTEGER );
   if (stat == IPMW_PARS_NOVAR) {
     def_position = 0;
     position.address.integer = &def_position;
   }
   else ipum_error(stat);

   return(ipc_straighten( options, image.address.image, vectors.address.list, position.address.integer ));
}

/*-----------------------------------------------------------------------------------------------------------------------------------------
Procedure:  ipic_straighten
Purpose:    Obtains a straitened image from a oblique image.
Parameters: ipv_image    *image      <i> The image to be straightened.
            ipv_list     *vectors    <i> The vectors to be straightened horizontally or vertically.
            long         position    <i> The line to be straightened at.
            short        options     <i> The direction the image to be straightened
                                         and the mode to fill rest pixels.
```

```
Return:  IPMS_
         IPMZ_
Errors:
Calls:   ipum_user_abort          Checks for control C.

Written:          24, Apr.1992    B.Fan   Add new command for DNA sequence.
Modified:         10, Oct.1992    B.Fan   For new VIPS verion.
-------------------------------------------------------------------------------------------------------------------*/
ipv_status ipc_straighten( short options, ipv_image *image, ipv_list **vectors, long *position )
{
  ipv_pixel       buffer[512];
  ipv_list        *p, *prep;
  long            i, j, diff, ndiff, total_diff, list_len, sort_count=0;
  long            image_size_row, image_size_col;
  float           tan;

  do {                                                              /*sort vectors*/
    list_len = 0;
    sort_count++;
    p = *vectors;
    while (p != NULL && p->next != NULL) {                          /*sort one vector*/
            if (p->type==IPT_REAL)
              ipum_error_3( IPME_PARS_WRONGTYP, "Element", "VECTOR", "REAL");
            else if (p->type!=IPT_VECTOR)
                    ipum_error_3( IPME_PARS_WRONGTYP, "Element", "VECTOR", "INTEGER");
             list_len++;
            if ((options & IPO_STRA_HORIZONTALLY)?(p->val.v.col <= p->next->val.v.col)
                  :(p->val.v.row<=p->next->val.v.row)){             /*point next*/
              prep = p;
              p = p->next;
            }
            else {
              prep->next = p->next;                                 /*change order with next*/
              p->next = prep->next;
              prep->next = p;
            }
    }
  } while(sort_count < list_len);
  if (p != NULL)
    if (options & IPO_STRA_HORIZONTALLY)
            p->val.v.col++;
    else p->val.v.row++;
  if (options & IPO_STRA_HORIZONTALLY && list_len>=1) {
    p = *vectors;
    image_size_row = image->size.row;
    while(p != NULL && p->next != NULL){                            /*loop for vector list*/
      if (p->val.v.col != p->next->val.v.col) {
            tan = (float)(p->next->val.v.row - p->val.v.row)/
                    (float)(p->next->val.v.col - p->val.v.col);     /*tan=row/column*/
            diff = p->val.v.row - *position;
            for ( j=p->val.v.col; j<p->next->val.v.col; j++ ){      /*restore from column 1 to column 2 */
              total_diff = diff + (int)((j-p->val.v.col) * tan);    /*get difference for each column*/
              if (total_diff > 0) {
                for (i = 0; i < total_diff; i++)
                        if (options & IPO_STRA_WARP)
                          buffer[i] = image->data.b[i][j];          /*store for warping*/
                for (i=total_diff; i<image_size_row; i++)
                        image->data.b[i-total_diff][j] = image->data.b[i][j]; /*shift pixels*/
                for (i=0; i<total_diff; i++)                        /*fill rest spaces*/
                        if (options & IPO_STRA_WARP)
                          image->data.b[image_size_row-total_diff+i][j] = buffer[i];
                        else if (options & IPO_STRA_BLACK)
```

```
                        image->data.b[image_size_row-total_diff+i][j] = 0;
                    else if (options & IPO_STRA_WHITE)
                        image->data.b[image_size_row-total_diff+i][j] = 255;
            }
            if (total_diff <0) {                                    /*opposite direction*/
              ndiff = -total_diff;
              for (i = 0; i <= ndiff; i++)
                    if (options & IPO_STRA_WARP)
                     buffer[i] = image->data.b[image_size_row-1-i][j];
              for (i = image_size_row-1-ndiff; i>0; i--)
                    image->data.b[i+ndiff][j] = image->data.b[i][j];
              for (i=0; i<=ndiff; i++)
                    if (options & IPO_STRA_WARP)
                     image->data.b[ndiff-i][j] = buffer[i];
                    else if (options & IPO_STRA_BLACK)
                        image->data.b[ndiff-i][j] = 0;
                    else if (options & IPO_STRA_WHITE)
                        image->data.b[ndiff-i][j] = 255;
            }   /*end if total_diff<0*/
          }   /*end for j*/
    }   /*end if ...col != ...col*/
    p = p->next;
    ipum_user_abort;
  }   /*end while*/
return(IPMS_);
}
else if (options & IPO_STRA_VERTICALLY && list_len>=1) {
  p = *vectors;
  image_size_col = image->size.col;
  while(p != NULL && p->next != NULL){                              /*loop for vector list*/
        if (p->val.v.row != p->next->val.v.row) {
          tan = (float)(p->next->val.v.col - p->val.v.col)/
                (float)(p->next->val.v.row - p->val.v.row);         /*tan=column/row*/
          diff = p->val.v.col - *position;
          for ( j=p->val.v.row; j<p->next->val.v.row; j++ ){   /*restore from column 1 to column 2 */
                total_diff = diff + (int)((j-p->val.v.row) * tan);      /*get difference for each column*/
                if (total_diff > 0) {
                  if (options & IPO_STRA_WARP)                  /*store pixels for WARP mode*/
                    for (i = 0; i < total_diff; i++)
                      buffer[i] = image->data.b[j][i];
                  for (i=total_diff; i<image_size_col; i++)
                    image->data.b[j][i-total_diff] = image->data.b[j][i];       /*shift pixels*/
                  for (i=0; i<total_diff; i++)                                  /*restore pixels*/
                    if (options & IPO_STRA_WARP)
                     image->data.b[j][image_size_col-total_diff+i] = buffer[i];
                    else if (options & IPO_STRA_BLACK)
                        image->data.b[j][image_size_col-total_diff+i] = 0;
                    else if (options & IPO_STRA_WHITE)
                        image->data.b[j][image_size_col-total_diff+i] = 255;
                }
                if (total_diff <0) {
                  ndiff = -total_diff;
                  if (options & IPO_STRA_WARP)
                    for (i = 0; i <= ndiff; i++)
                        buffer[i] = image->data.b[j][image_size_col-1-i];
                  for (i = image_size_col-1-ndiff; i>0; i--)
                    image->data.b[j][i+ndiff] = image->data.b[j][i];
                  for (i=0; i<=ndiff; i++)
                    if (options & IPO_STRA_WARP)
                     image->data.b[j][ndiff-i] = buffer[i];
                    else if (options & IPO_STRA_BLACK)
                        image->data.b[j][ndiff-i] = 0;
                    else if (options & IPO_STRA_WHITE)
```

```
                            image->data.b[j][ndiff-i] = 255;
                    } /*end if total_diff<0*/
                } /*end for j*/
            } /*end if ...row != ...row*/
            p = p->next;
            ipum_user_abort;
        } /*end while*/
    return(IPMS_);
    }
    else { ipum_error_1(IPME_COMM_INVPARAM, " At least two vectors required ");
            return(IPMS_); }
}
#undef WHERE
/*------------------------------------------------------------------------------------------------------------*/
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

PARALL.H:

```
enum { IPO_PARA_HORIZONTALLY = 1,
    IPO_PARA_VERTICALLY   = 2 };

ipv_status ipif_parallel( char *params );
ipv_status ipc_parallel( short mode, ipv_image *image1, ipv_image *image2,
                        ipv_list **vectors, long position, ipv_vector *size );

extern ipv_command ipcc_parallel;
```

PARALL.C:

```
/*------------------------------------------------------------------------------------------------------------
Format:         PARALLEL image1 image2 vectors [position size mode]
Parameters:     image1  (IMAGE)          <i> The input image to be paralleled.
                image2  (IMAGE)          <o> The output image paralleled.
                vectors (LIST)           <i> The vectors to be paralleled horizontally or vertically.
                position (INTEGER)[0]    <i>The first line to be paralleled on the input image.
                size    (VECTOR)[(512 100)/HORIZONTALLY or (100 512)/VERTICALLY]
                                         <i>The output image size.
                mode    (OPTION)[/HORIZONTALLY]<i>The direction the image to be paralleled by.
                        /HORIZONTALLY          The image to be paralleled horizontally.
                        /VERTICALLY            The image to be paralleled vertically.
------------------------------------------------------------------------------------------------------------*/
#include <stddef.h>
#include "g_error.h"
#include "g_defin.h"
#include "u_option.h"
#include "u_parame.h"
#include "u_image.h"
#include "c_parall.h"
#include <stdio.h>

/*------------------------------------------------------------------------------------------------------------*/
long        parallel_pos;
long        parallel_pos_def = 0;                              /*Defined position*/
ipv_vector      parallel_siz;
ipv_vector      parallel_siz_hdef = {100, 512};
ipv_vector      parallel_siz_vdef = {512, 100};

ipv_param ipcp_parallel_h[5] =
 {{ IPVO_REQD, IPT_B_IMAGE, IPTM_B_IMAGE, NULL, NULL, 0 },
```

```
  { IPVO_REQD|IPVO_DECL, IPT_B_IMAGE, IPTM_B_IMAGE, NULL, NULL, 0 },
  { IPVO_REQD, IPT_LIST, IPTM_LIST, NULL, NULL, 0 },
  { 0, IPT_INTEGER, IPTM_INTEGER, &parallel_pos, &parallel_pos_def, 0 },
  { 0, IPT_VECTOR, IPTM_VECTOR, &parallel_siz, &parallel_siz_hdef, 0 }};
ipv_param ipcp_parallel_v[5] =
 {{ IPVO_REQD, IPT_B_IMAGE, IPTM_B_IMAGE, NULL, NULL, 0 },
  { IPVO_REQD|IPVO_DECL, IPT_B_IMAGE, IPTM_B_IMAGE, NULL, NULL, 0 },
  { IPVO_REQD, IPT_LIST, IPTM_LIST, NULL, NULL, 0 },
  { 0, IPT_INTEGER, IPTM_INTEGER, &parallel_pos, &parallel_pos_def, 0 },
  { 0, IPT_VECTOR, IPTM_VECTOR, &parallel_siz, &parallel_siz_vdef, 0 }};

ipv_com_variant ipcv_parallel[2] =
 {{ IPO_PARA_VERTICALLY, IPO_PARA_HORIZONTALLY, (ipv_proc *) ipc_parallel,
        NULL, 5, ipcp_parallel_h },
  { IPO_PARA_HORIZONTALLY, IPO_PARA_VERTICALLY, (ipv_proc *) ipc_parallel,
        NULL, 5, ipcp_parallel_v }};

ipv_command ipcc_parallel =
 { "PARALLEL", "/HORIZONTALLY/VERTICALLY",
        IPO_PARA_VERTICALLY, IPO_PARA_HORIZONTALLY,
        2, ipcv_parallel };

#define WHERE ipcc_parallel.command
/*-------------------------------------------------------------------------------------------------------------------------------------------------------*/

ipv_status ipif_parallel( char *params ){
  ipv_status     stat1, stat2;
  ipv_var_desc   image1, image2, vectors, position, size;        /*pointers to actual parameters*/
  long           def_position;                                   /*Defined position*/
  short          mode;                                         /*The direction to be paralleled by.*/
  ipv_vector     def_size;

  ipum_error(ipu_options( &mode, params, "/HORIZONTALLY /VERTICALLY" ));
  if (mode == 0) mode = IPO_PARA_HORIZONTALLY;

  ipum_error(ipu_parameter( &params, 1, IPVO_REQD, &image1,
                            IPT_B_IMAGE, IPTM_B_IMAGE ));
  ipum_error(ipu_parameter( &params, 2, IPVO_REQD | IPVO_DECL, &image2,
                            IPT_B_IMAGE, IPTM_B_IMAGE ));
  ipum_error(ipu_parameter( &params, 3, IPVO_REQD, &vectors,
                            IPT_LIST, IPTM_LIST ));
  stat1 = ipu_parameter( &params, 4, 0, &position, IPT_INTEGER, IPTM_INTEGER );
  if (stat1 == IPMW_PARS_NOVAR) {
    def_position = 0;
    position.address.integer = &def_position;
  }
  else ipum_error(stat1);
  stat2 = ipu_parameter( &params, 5, 0, &size, IPT_VECTOR, IPTM_VECTOR );
  if (stat2 == IPMW_PARS_NOVAR) {
    if (mode == IPO_PARA_HORIZONTALLY) {
      def_size.row = 100;
      def_size.col = 512;
      size.address.vector = &def_size;
    }
    if (mode == IPO_PARA_VERTICALLY) {
      def_size.row = 512;
      def_size.col = 100;
      size.address.vector = &def_size;
    }
  }
  else ipum_error(stat2);
```

```
  return(ipc_parallel( mode, image1.address.image, image2.address.image, vectors.address.list,
                *position.address.integer, size.address.vector ));
}
/*-----------------------------------------------------------------------------------------------------------------------
Procedure: ipic_parallel
Purpose:   Obtains a paralleled image from a arbitrary image.
Parameters: ipv_image     *image1      <i> The image to be paralleled.
        ipv_image         *image2      <o> The output image paralleled.
        ipv_list          *vectors     <i> The vectors to be paralled of the arbitrary image.
        long              position     <i> The start line to be paralled on the input image.
        ipv_vector        *size        <i> The output image size.
        short             mode         <i> The mode to be paralled by.
Return: IPMS_
        IPMZ_
Errors:
Calls:  ipum_error              If images are different sizes.
        ipum_user_abort         Checks for control C.
Written:        12, Agu.1992    B.Fan   Add new command for paralleling DNA sequence.
Modified:       12, Oct.1992    B.Fan   For new VIPS verion.
-----------------------------------------------------------------------------------------------------------------------*/
ipv_status ipc_parallel( short mode, ipv_image *image1, ipv_image *image2,
                         ipv_list **vectors, long position, ipv_vector *size )
{
 ipv_list *p, *prep;
 long i1, i2, j, i2start, i2end, list_len, diff, sort_count=0;
 float tan, size1, rate;

 if (image2->size.row !=size->row || image2->size.col != size->col)
  ipum_error(ipu_change_image(image2, size));

 do {                                                          /*sort vectors*/
   list_len = 0;
   sort_count++;
   p = *vectors;
   while (p != NULL && p->next != NULL) {                      /*sort one vector*/
    if (p->type==IPT_REAL)
        ipum_error_3( IPME_PARS_WRONGTYP, "Element", "VECTOR", "REAL");
    else if (p->type!=IPT_VECTOR)
            ipum_error_3( IPME_PARS_WRONGTYP, "Element", "VECTOR", "INTEGER");
             list_len++;
        if ((mode == IPO_PARA_HORIZONTALLY)?(p->val.v.col<=p->next->val.v.col)
          :(p->val.v.row<=p->next->val.v.row)){                /*point to next vector*/
            prep = p;
            p = p->next;
        }
        else {
          prep->next = p->next;                                /*change order with next vector*/
          prep = p->next;
          p->next = prep->next;
          prep->next = p;
        }
   }
 } while(sort_count < list_len);
 if (p != NULL)                 /*add 1 more column/row to last vector for right edge of output image*/
  if (mode == IPO_PARA_HORIZONTALLY)
    p->val.v.col++;
  else p->val.v.row++;
 if (mode == IPO_PARA_HORIZONTALLY && list_len>=1) {
  p = *vectors;
  while(p != NULL && p->next != NULL){                         /*loop for vector list*/
    if (p->val.v.col != p->next->val.v.col) {
        tan = (float)(p->next->val.v.row - p->val.v.row)/
              (float)(p->next->val.v.col - p->val.v.col);      /*tan=row/column*/
```

```
          diff = p->val.v.row - position;
          for ( j=p->val.v.col; j<p->next->val.v.col; j++ ){       /*restore from column 1 to column 2 */
            i2start = 0;
            size1 = diff + (j-p->val.v.col)*tan;
            rate = image2->size.row/size1;
            for (i1=0; i1<=size1; i1++) {
              i2end=(long)(i1*rate+0.5);
              for (i2=i2start; (i2end<image2->size.row)?(i2<=i2end):(i2<i2end); i2++)  {
                   image2->data.b[i2][j] = image1->data.b[i1+position][j];  }
              i2start = ++i2end;
            }
          }  /*end for j*/
    }  /*end if p..!=next..*/
    p = p->next;
    ipum_user_abort;
  }  /*end while*/
 return(IPMS_);
 }
 else
 if (mode == IPO_PARA_VERTICALLY && list_len >= 1) {
   p = *vectors;
   while(p != NULL && p->next != NULL){                        /*loop for vector list*/
    if (p->val.v.row != p->next->val.v.row) {
          tan = (float)(p->next->val.v.col - p->val.v.col)/
              (float)(p->next->val.v.row - p->val.v.row);              /*tan=column/row*/
          diff = p->val.v.col - position;
          for ( j=p->val.v.row; j<p->next->val.v.row; j++ ){       /*restore from column 1 to column 2 */
           i2start = 0;
           size1 = diff + (j-p->val.v.row)*tan;
           rate = image2->size.col / size1;
           for (i1=0; i1<size1; i1++) {
             i2end=(long)(i1*rate+0.5);
             for (i2=i2start; (i2end<image2->size.col)?(i2<=i2end):(i2<i2end); i2++)
               image2->data.b[j][i2] = image1->data.b[j][i1+position];
             i2start = ++i2end;
           }
           ipum_user_abort;
          }  /*for j*/
    }  /*if p..!=next..*/
    p = p->next;
    ipum_user_abort;
  }  /*while*/
  return(IPMS_);
 }
 else ipum_error_1(IPME_COMM_INVPARAM, " At least two vectors required ");
 return(IPMS_);
}
#undef WHERE
/*-----------------------------------------------------------------------------------------------------------------------*/
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

SORT.H:

```
enum { IPO_SORT_HORIZONTALLY = 1,
     IPO_SORT_VERTICALLY   = 2 };

ipv_status ipif_sort( char *params );
ipv_status ipc_sort( short mode, ipv_list **list );

extern ipv_command ipcc_sort;
```

SORT.C:

```
/*-----------------------------------------------------------------------------------------------------------------------
Format:         SORT pre_list [mode]
Parameters:     pre_list  (LIST)          <i> The previous list to be inserted.
                mode      (OPTION) [/HORIZONTALLY] <i>The mode to be sorted.
                          /HORIZONTALLY     Sort horizontally.
                          /VERTICALLY       Sort vertically.
-----------------------------------------------------------------------------------------------------------------------*/
#include <stddef.h>
#include "g_error.h"
#include "g_defin.h"
#include "u_option.h"
#include "u_parame.h"
#include "u_image.h"
#include "c_sort.h"

/*-----------------------------------------------------------------------------------------------------------------------*/

ipv_param ipcp_sort[1] =
  {{ IPVO_REQD, IPT_B_IMAGE, IPTM_B_IMAGE, NULL, NULL, 0 }};

ipv_com_variant ipcv_sort[1] =
  {{ 0, 0, (ipv_proc *) ipc_sort, NULL, 1, ipcp_sort }};

ipv_command ipcc_sort =
  { "SORT", "/HORIZONTALLY/VERTICALLY", IPO_SORT_VERTICALLY,
        IPO_SORT_HORIZONTALLY,
        1, ipcv_sort };

#define WHERE ipcc_sort.command
/*-----------------------------------------------------------------------------------------------------------------------*/

ipv_status ipif_sort( char *params ){
  ipv_var_desc   list;                                    /*pointers to actual parameters*/
  short          mode;                                    /*sorting mode*/

  ipum_error(ipu_options( &mode, params, "/HORIZONTALLY /VERTICALLY" ));
  if (mode == 0) mode = IPO_SORT_HORIZONTALLY;

  ipum_error(ipu_parameter( &params, 1, IPVO_REQD, &list, IPT_LIST, IPTM_LIST ));

  return(ipc_sort( mode, list.address.list));
}
/*-----------------------------------------------------------------------------------------------------------------------
Procedure:      ipc_sort
Purpose:        Inserts data to the sorted data list and cancels invalid data.
Parameters:     ipv_list  *list     <io> The list to be inserted.
                short     mode      <i> The sorting mode.
Return:         IPMS_
Errors:         IPME_PARS_WRONGTYP
Calls:          ipum_error_3
Written:        30, Apr. 1992     B.Fan     Add new command for DNA sequence.
Modified:       10, Oct. 1992     B.Fan     For new VIPS verion.
-----------------------------------------------------------------------------------------------------------------------*/
ipv_status ipc_sort( short mode, ipv_list **list ) {
  ipv_list       *prep, *newp, *nextpp, *prenewp;
  char           comp;

  prep = (*list);                                         /*prep points to the head of previous list*/
  newp = NULL;                                            /*newp points to the head of the ne*/
```

```
  while(prep != NULL) {                                   /*loop for every iterm in previous list*/
    if (prep->type==IPT_INTEGER)
          ipum_error_3( IPME_PARS_WRONGTYP, "Element", "VECTOR", "INTEGER");
    if (prep->type==IPT_REAL)
          ipum_error_3( IPME_PARS_WRONGTYP, "Element", "VECTOR", "REAL");
    if (newp == NULL) {                                   /*if new_list empty*/
          newp = prep;                                    /*insert the item as head of new list*/
          prep = prep->next;
          newp->next = NULL;
          *list = newp;
     }
    else {
          comp=(mode==IPO_SORT_HORIZONTALLY)?(prep->val.v.col<=newp->val.v.col)
                  :(prep->val.v.row>=newp->val.v.row);
          if (comp) {                                     /*insert item as head of new list*/
               nextpp = prep->next;
               prep->next = newp;
               newp = prep;
               prep = nextpp;
               *list = newp;
            }
          else {
             comp=(mode==IPO_SORT_HORIZONTALLY)?(prep->val.v.col>newp->val.v.col)
                  :(prep->val.v.row<newp->val.v.row );    /*item inserted not as head of new list*/
             while( newp != NULL && comp) {               /*find the position to be inserted to*/
               prenewp = newp;
               newp = newp->next;
               if (newp!=NULL)
               comp=(mode==IPO_SORT_HORIZONTALLY)?(prep->val.v.col>newp->val.v.col)
                  :(prep->val.v.row<newp->val.v.row);
          }
             nextpp = prep->next;                         /*insert data*/
             prep->next = newp;
             prenewp->next = prep;
             prep = nextpp;
           }
    newp = (*list);
    }    /* else 1 */
    ipum_user_abort;

  }    /* while */
  return( IPMS_ );
}

#undef WHERE
/*-------------------------------------------------------------------------------------------------------*/
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

SEQUENCE.H:

```
enum { IPO_SEQU_HORIZONTALLY = 1,
     IPO_SEQU_VERTICALLY   = 2 };

ipv_status ipif_sequence( char *params );
ipv_status ipc_sequence( short mode, ipv_list **vectors, char **sequence, char **order,
                          long line_width );

extern ipv_command ipcc_sequence;
```

SEQUENCE.C:

```
/*-----------------------------------------------------------------------------------------------------------------------
Format:         SEQUENCE vectors sequence [order line_width mode]
Parameters:     vectors         (LIST)          <i> The vector list to get DNA sequence.
                sequence        (STRING)        <o> The DNA sequence.
                order           (STRING) [TCGA]<i>The order of bases.
                line_width      (INTEGER) [50]  <i>The output file line width.
                mode    (OPTION)[/HORIZONTALLY]<i>The sequencing direction.
                        /HORIZONTALLY           Sequence horizontally.
                        /VERTICALLY             Sequence vertically.
-----------------------------------------------------------------------------------------------------------------------*/
#include <stdlib.h>
#include <ctype.h>
#include "g_error.h"
#include "g_defin.h"
#include "u_option.h"
#include "u_parame.h"
#include "c_sequen.h"
#include "f_length.h"

/*-----------------------------------------------------------------------------------------------------------------------*/
char *sequence_ord;
char *sequence_ord_def = "TCGA";
long sequence_wid;
long sequence_wid_def = 50;

ipv_param ipcp_sequence[4] =
 {{ IPVO_REQD,IPT_LIST, IPTM_LIST, NULL, NULL, 0 },
  { IPVO_REQD|IPVO_DECL, IPT_STRING, IPTM_STRING, NULL, NULL, 0 },
  { 0, IPT_STRING, IPTM_STRING, &sequence_ord, &sequence_ord_def, sizeof( char ) },
  { 0, IPT_INTEGER, IPTM_INTEGER, &sequence_wid, &sequence_wid_def, sizeof( long ) }};

ipv_com_variant ipcv_sequence[1] =
 {{ 0, 0, (ipv_proc *) ipc_sequence, NULL, 4, ipcp_sequence }};

ipv_command ipcc_sequence =
 { "SEQUENCE", "/HORIZONTALLY/VERTICALLY",
  IPO_SEQU_VERTICALLY, IPO_SEQU_HORIZONTALLY,
  1, ipcv_sequence };

#define WHERE ipcc_sequence.command
/*-----------------------------------------------------------------------------------------------------------------------*/

ipv_status ipif_sequence( char *params ){
  ipv_status      stat1, stat2;
  ipv_var_desc    vectors, sequence, order, line_width;          /*pointers to actual parameters*/
  long            def_line_width;
  char            *def_order="TCGA";
  short           mode;

  ipum_error(ipu_options( &mode, params, "/HORIZONTALLY /VERTICALLY" ));
  if (mode == 0) mode = IPO_SEQU_HORIZONTALLY;

  ipum_error(ipu_parameter( &params, 1, IPVO_REQD, &vectors,
                                    IPT_LIST, IPTM_LIST ));
  ipum_error(ipu_parameter( &params, 2, IPVO_REQD | IPVO_DECL, &sequence,
                                    IPT_STRING, IPTM_STRING ));

  stat1 = ipu_parameter( &params, 3, 0, &order, IPT_STRING, IPTM_STRING );
  if (stat1 == IPMW_PARS_NOVAR)
```

```
    order.address.string = &def_order;
  else ipum_error(stat1);

  stat2 = ipu_parameter( &params, 4, 0, &line_width, IPT_INTEGER, IPTM_INTEGER );
  if (stat2 == IPMW_PARS_NOVAR) {
    def_line_width = 50;
    line_width.address.integer = &def_line_width;
   }
  else ipum_error(stat2);

  return(ipc_sequence( mode, vectors.address.list, sequence.address.string,
                order.address.string, *line_width.address.integer ));
}
/*-------------------------------------------------------------------------------------------------------------------------
Procedure: ipc_sequence
Purpose:   From vector list gets DNA sequence list.
Parameters: ipv_list      **vectors      <i> The vector list to get DNA sequence.
            char          **sequence     <o> The DNA sequence.
            char          **order        <i> The order of bases.
            long          line_width     <i> The line width of the output file.
            short         mode           <i> The sequencing direction.
Return:  IPMS_
Errors:
Calls:   ipum_error
         ipum_user_abort
Written:          4, May 1992    B.Fan          Add new command for DNA sequence.
Modified:         27, Jul  1993  D Bailey       New list length
-------------------------------------------------------------------------------------------------------------------------*/
ipv_status ipc_sequence( short mode, ipv_list **vectors, char **sequence, char **order,
                         long line_width ) {

  ipv_list      *pp;
  long          num, length, preposition, position1, position2;
  char          *orderlist, *element = '\0';

  ipum_error(ipf_length_list(&length,vectors));
  if (*sequence == NULL)
    free(*sequence);
  element = *sequence = malloc(length+2+length/5+length/line_width);
  if (element == NULL)
     ipum_error(IPME_SYST_INSUFMEM);
  *element = 0;
  orderlist = *order;
  num = 0;
  preposition = -1;
  for (pp = (*vectors); pp != NULL; pp = pp->next) {
     num++;
    position1 = (mode==IPO_SEQU_HORIZONTALLY)?(pp->val.v.row)
                  :(pp->val.v.col);                                  /*line of the position*/
    position2 = (mode==IPO_SEQU_HORIZONTALLY)?(pp->val.v.col):(pp->val.v.row);
     switch (position1) {                                            /*set the base*/
            case 12: if (position2 != preposition)                   /*the first lane*/
                        *element++ = toupper(orderlist[0]);
                     else *element++ = tolower(orderlist[0]); break;
            case 37: if (position2 != preposition)                   /*the second lane*/
                        *element++ = toupper(orderlist[1]);
                     else *element++ = tolower(orderlist[1]); break;
            case 62: if (position2 != preposition)                   /*the third lane*/
                        *element++ = toupper(orderlist[2]);
                     else *element++ = tolower(orderlist[2]); break;
            case 87: if (position2 != preposition)                   /*the fourth lane*/
                        *element++ = toupper(orderlist[3]);
                     else *element++ = tolower(orderlist[3]); break;
```

```
            default: ipum_error_1(IPME_COMM_INVPARAM, "Band in wrong position");
    }
    preposition = position2;
   if (num/(float)line_width == num/line_width)                     /*data file form*/
            *element++ = '\n';
    else
            if (num/10.0 == num/10 )  {
            *element++ = ' ';
            *element++ = ' ';
        }
    ipum_user_abort;
  }
  *element = '\0';
  return( IPMS_ );
}

#undef WHERE
/*------------------------------------------------------------------------------------------------------------*/
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

JOIN.H:

```
ipv_status ipif_join( char *params );
ipv_status ipc_join( char **seq1, char **seq2 );

extern ipv_command ipcc_join;
```

JOIN.C:

```
/*------------------------------------------------------------------------------------------------------------
Format:         JOIN seq1 seq2
Parameters:     seq1    (STRING)      <io>    The first part of the DNA sequence
                seq2    (STRING)      <i>     The seconde part of the DNA sequence
------------------------------------------------------------------------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include "g_error.h"
#include "g_defin.h"
#include "u_parame.h"
#include "c_join.h"

/*------------------------------------------------------------------------------------------------------------*/
ipv_param ipcp_join[2] =
 {{ IPVO_REQD, IPT_STRING, IPTM_STRING, NULL, NULL, 0 },
  { IPVO_REQD, IPT_STRING, IPTM_STRING, NULL, NULL, 0 }};

ipv_com_variant ipcv_join[1] =
  {{ 0, 0, (ipv_proc *) ipc_join, NULL, 3, ipcp_join }};

ipv_command ipcc_join =
  { "JOIN", 0, 0, 0, 1, ipcv_join };

#define WHERE ipcc_join.command
/*------------------------------------------------------------------------------------------------------------*/

ipv_status ipif_join( char *params ){
```

```
  ipv_var_desc seq1, seq2;                                    /*pointers to actual parameters*/

  ipum_error(ipu_parameter( &params, 1, IPVO_REQD, &seq1,
                                    IPT_STRING, IPTM_STRING ));
  ipum_error(ipu_parameter( &params, 2, IPVO_REQD, &seq2,
                                    IPT_STRING, IPTM_STRING ));

  return(ipc_join( seq1.address.string, seq2.address.string ));
}
/*-------------------------------------------------------------------------------------------------------------------
Procedure: ipc_join
Purpose:   Joins multiple parts of a DNA sequence together
Parameters:     char    **seq1  <io>    The first part of the DNA sequence
                char    **seq2  <i>     The seconde part of the DNA sequence
Return:         IPMS_
Errors:
Calls:          ipum_error
                ipum_user_abort
Written:        7, Oct.1992     B.Fan   Add new command for DNA sequence.
Modified:       12, Oct.1992    B.Fan   For new VIPS verion.
-------------------------------------------------------------------------------------------------------------------*/
ipv_status ipc_join( char **seq1, char **seq2 ) {

  long   seq1i, seq2i, seq1temp, seq1len, seq2len;
  long   i=0, num=0, width=0;
  char   *sequence;

  seq1len = strlen(*seq1);
  seq2len = strlen(*seq2);
  if ((sequence =(char *) malloc(seq1len+seq2len+3)) == NULL)
     ipum_error(IPME_SYST_INSUFMEM);
  strcpy(sequence, (*seq1));
  while(sequence[i]!='\n')                                    /*count output line width*/
    if (sequence[i++]!=' ')
          width++;
  for (seq1i=0; seq1i<=seq1len; seq1i++) {                    /*compare *seq1 and *seq2*/
    while(sequence[seq1i]==' '||sequence[seq1i]=='\n')        /*ignor ' ','\n' in seq1*/
          seq1i++;
    seq1temp=seq1i;
    for (seq2i=0; seq2i<seq2len; seq2i++) {                   /*compare each *seq2[i]*/
          while(((*seq2)[seq2i]==' '||(*seq2)[seq2i]=='\n'))  /*ignor ' ','\n' in seq2*/
            seq2i++;
          while(sequence[seq1temp]==' '||sequence[seq1temp]=='\n')
                                                              /*ignor ' ','\n' in seq1 when comparing*/
            seq1temp++;
          if (sequence[seq1temp] == (*seq2)[seq2i]) {
            seq1temp++;
            if (sequence[seq1temp]=='\0') {
              sequence[seq1temp++] = ' ';
              sequence[seq1temp] = ' ';
              strcpy(sequence+seq1temp-1, (*seq2)+seq2i+1);
              seq1i=seq1len;
               break;                                         /*compeleted comparing exit loop*/
            }
          }
          else break;                                         /*not match from next seq1[i]*/
    }
  }
  seq1temp = 0;
  i=0;                                                        /*tidy sequence*/
  while(sequence[i]!='\0') {
    if (sequence[i]!=' ' && sequence[i]!='\n') {
          sequence[seq1temp++] = sequence[i++];
```

```
            num++;
            if (num/(float)width == num/width)
              sequence[seq1temp++] = '\n';
            else
              if (num/10.0 == num/10 )  {
                sequence[seq1temp++] = ' ';
                sequence[seq1temp++] = ' ';
              }
       }
       else i++;
   }
   sequence[seq1temp] = '\0';
   free( *seq1 );
   *seq1 = sequence;
   ipum_user_abort;
   return( IPMS_ );
}

#undef WHERE
/*------------------------------------------------------------------------------------------------------------*/
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

MAIN.C:

```
#include "i_comman.h"
#include "u_initia.h"
#include "u_vips.h"
#include "sd_bits.h"
#include "c_parall.h"
#include "c_join.h"
#include "c_sequen.h"
#include "c_straig.h"
#include "c_points.h"

#define WHERE "main program"

main() {
   ipum_error( ipu_initialise() );
   ipum_error( ipu_init_mouse() );
   ipum_error( ipi_abort_init() );

   ipum_load_c( "SORT", ipif_sort );
   ipum_load_c( "SEQUENCE", ipif_sequence );
   ipum_load_c( "STRAIGHTEN", ipif_straighten );
   ipum_load_c( "PARALLEL", ipif_parallel );
   ipum_load_c( "JOIN", ipif_join );
   ipum_load_c( "POINTS", ipif_points );
   ipu_vips();
   return(IPMS_);
}
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&